МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Гордієнко Ю.Г., Кочура Ю.П.

ГЕНЕТИЧНІ АЛГОРИТМИ

Конспект лекцій

Навчальний посібник
для здобувачів ступеня магістра
за освітньою програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності 121 «Інженерія програмного забезпечення»
за освітньою програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
за освітньою програмою «Інформаційні управляючі системи та технології »
спеціальності 126 «Інформаційні системи та технології»

Електронне мережне навчальне видання

ЗАТВЕРДЖЕНО
на засіданні кафедри обчислювальної техніки
протокол № 10 від 25.05.2022

2022

Генетичні алгоритми

-

Лекція 01. Вступ

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Рекомендовані джерела - Книги

(деякі з них використовуються тут!)

Книги (класика):

Голланд (1992). Адаптація в природних і штучних системах: вступний аналіз із застосуванням до біології, управління та штучний інтелект. Преса MIT. **<-винахідник GA(!)**, **найбільша кількість цитувань для GA-публікації** від Google Scholar!

Мітчелл, М. (1998). Введення в генетичні алгоритми. Преса MIT. <-**класичний підручник**, **найбільша кількість цитати до ГА-підруч** від Google Scholar!

Книги (з кодами на github):

Вірсанський, Е. (2020). Практичні генетичні алгоритми з Python. Packt Publishing

Шеппард, К. (2019). Генетичні алгоритми з Python

Рекомендовані джерела – статті

(деякі з них використовуються тут!)

Холланд, (1992). Генетичні алгоритми. Scientific American, 267 (1), 66-73.**<-винахідник GA(!) <- Просто задля розваги! :)**

Каточ, С., Чаухан, С. С., Кумар, В. (2020). Огляд про генетичний алгоритм: минуле, теперішнє та майбутнє. Мультимедійні засоби та програми, 1-36.

Гарсіа-Мартінес, К., Родрігес, Ф. Дж., Лозано, М. (2018). Генетичні алгоритми, Довідник з евристики, 2018, стор. 431-464.

Зміст

- Рекомендовані джерела
- **Що таке генетичні алгоритми (GA)?**
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Що таке генетичні алгоритми?

Генетичні алгоритми(**GA**) є сімейством алгоритмів пошуку, натхненних принципами природної еволюції.

Імітація природного відбору та розмноженняGA можут виробляти високоякісні рішення для різних**проблеми**:

- **пошук,**
- **оптимізація,**
- **навчання.**

Аналогію з природною еволюцією допускає ГА**подолати деякі проблеми, складні для традиційних** алгоритми, особливо для випадків з:

- **велика кількість параметрів**і
- **складні математичні уявлення**.

Теорія ГА - еволюція Дарвіна

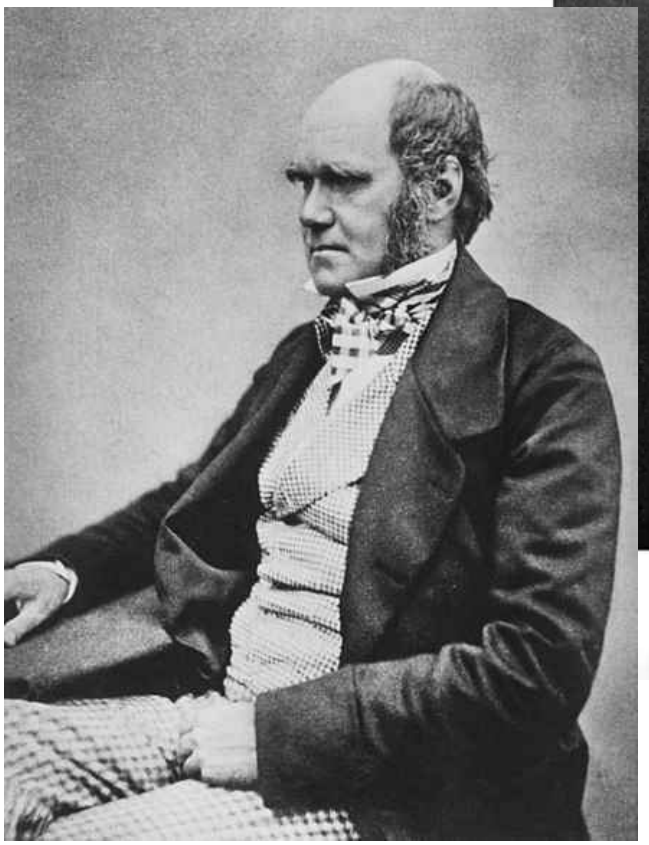
GA реалізують а спрощена версія

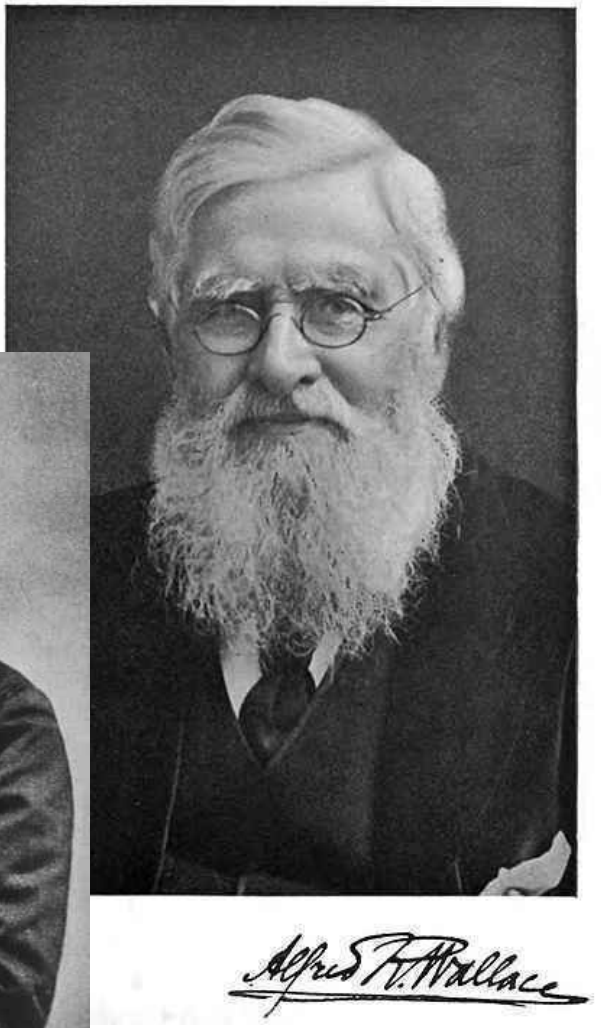
Дарвінівський природний

еволюція.

Принципи дарвінівської еволюції:

- **Варіація**
- **Спадщина**
- **Вибір**





Теорія ГА - еволюція Дарвіна

Варіація:

The**риси**(**атрибути**) окремих зразків

належність до населення**може змінюватися**.

В результаті,**екземпляри відрізняються**один від одного

певною мірою,

наприклад, у:

- їх**поведінка**або
- їх**зовнішній вигляд**.

Теорія ГА - еволюція Дарвіна

Спадщина:

Деякі**риси**є послідовно**пройшли**на від зразки**своїм нащадкам**.

Як результат,**нащадки схожі на своїх батьків** більше**ніж**вони схожі**неспоріднені зразки**.

Теорія ГА - еволюція Дарвіна

Вибір:

Популяціїтипово**боротьба**для ресурсів в межах свого середовища.

The**зразки**з**риси**які**краще**

адаптованийна навколишнє середовище:

- **буде успішнішим**при виживанні, і
- **дасть більше потомства**до наступного покоління.

Теорія ГА - дарвінівська еволюція

Резюме:

Еволюція**підтримує**популяція особин **зразки**що**змінюватися**один від одного.

Ті, хто є**краще адаптовані**до свого середовища мають а**більше шансів вижити**, розведення та**проходження**їхні риси**до наступне покоління**.

Таким чином, з поколіннями,**види стають більш адаптовані**до їх**навколишнє середовище**і до **виклики**представлені їм.

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- **GA Аналогія з IT**
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Аналогія GA з IT

GA повинні знайти**оптимальне рішення**для проблеми.

Дарвінівська еволюція підтримує популяцію **окремі екземпляри**, **АЛЕ(!)**...GAs підтримують популяцію**кандидат рішення**(**особи**), для цієї проблеми. The**особи** ітеративно оцінюються та використовуються створити нове покоління**особи**. Ті, хто є **краще**при вирішенні цієї проблеми є а **більше** шанс бути відібраним і проходити їх якості для наступного покоління**особи**. Таким чином ... з поколіннями ...**особи**стало краще при вирішенні проблеми.

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- **Компоненти ГА**
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Аналогія GA з IT -

Основні компоненти

- Генотип

- Населення

- Фітнес функція

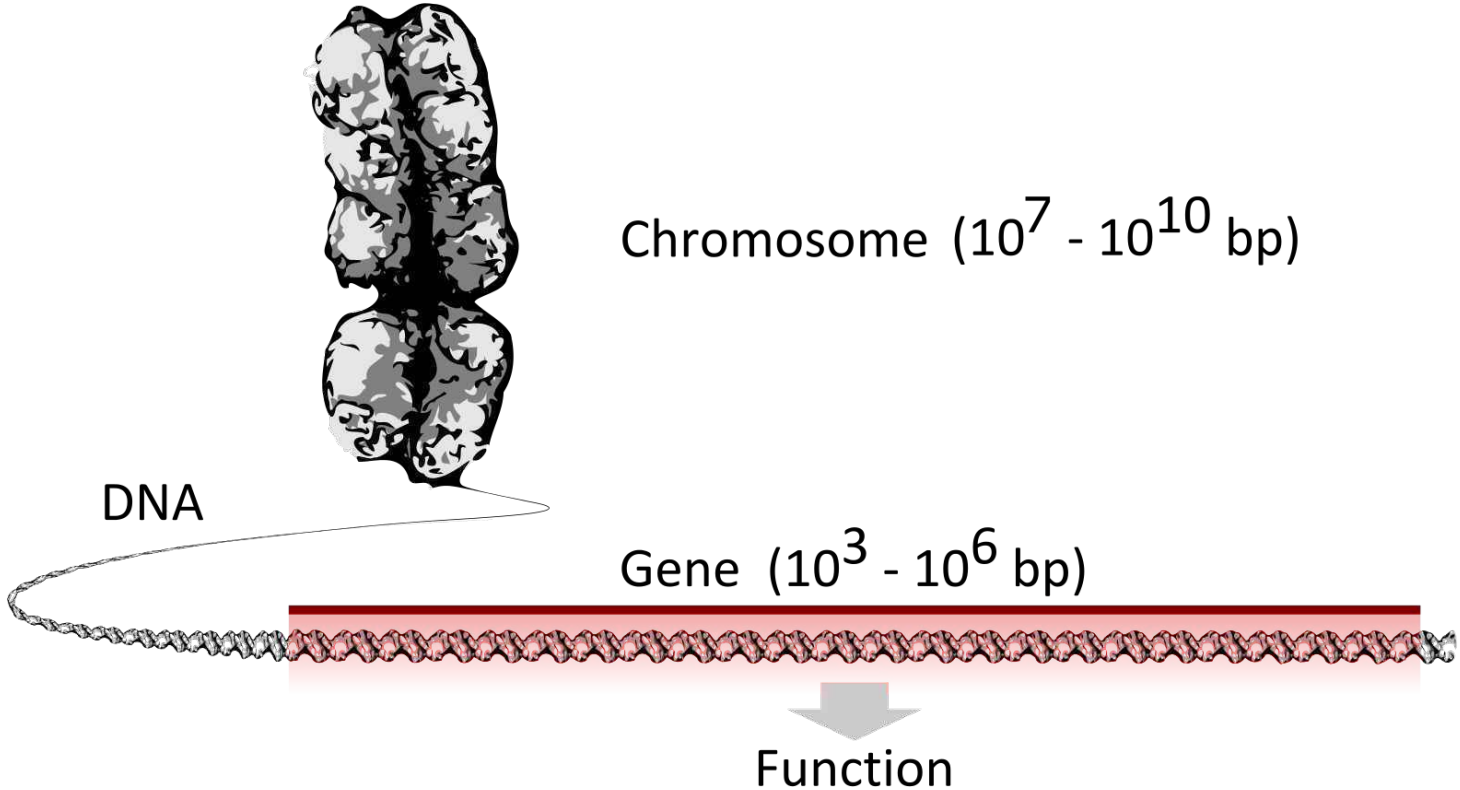
- Вибір

- Кросовер

- Мутація

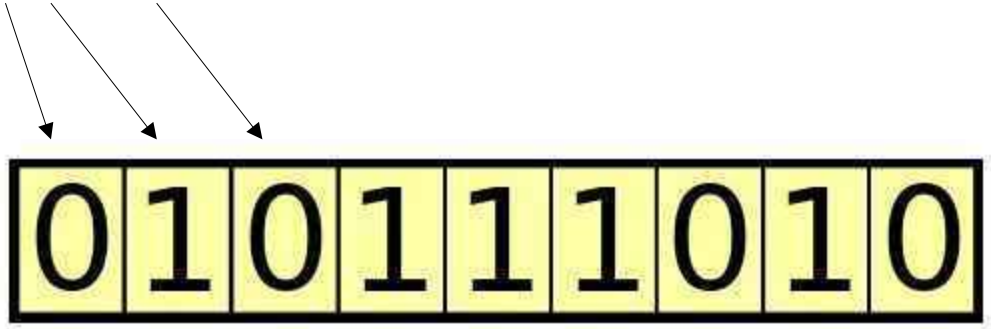
Аналогія GA з IT - Основні компоненти -Генотип

- **У біології: генотип**є колекцією**гени**які згруповані в **хромосоми**. Якщо два екземпляри розмножуються для створення потомства, кожен**хромосома**з потомство буде**нести а**суміш генів**в**ід обох



Аналогія GA з IT - Основні компоненти -Генотип

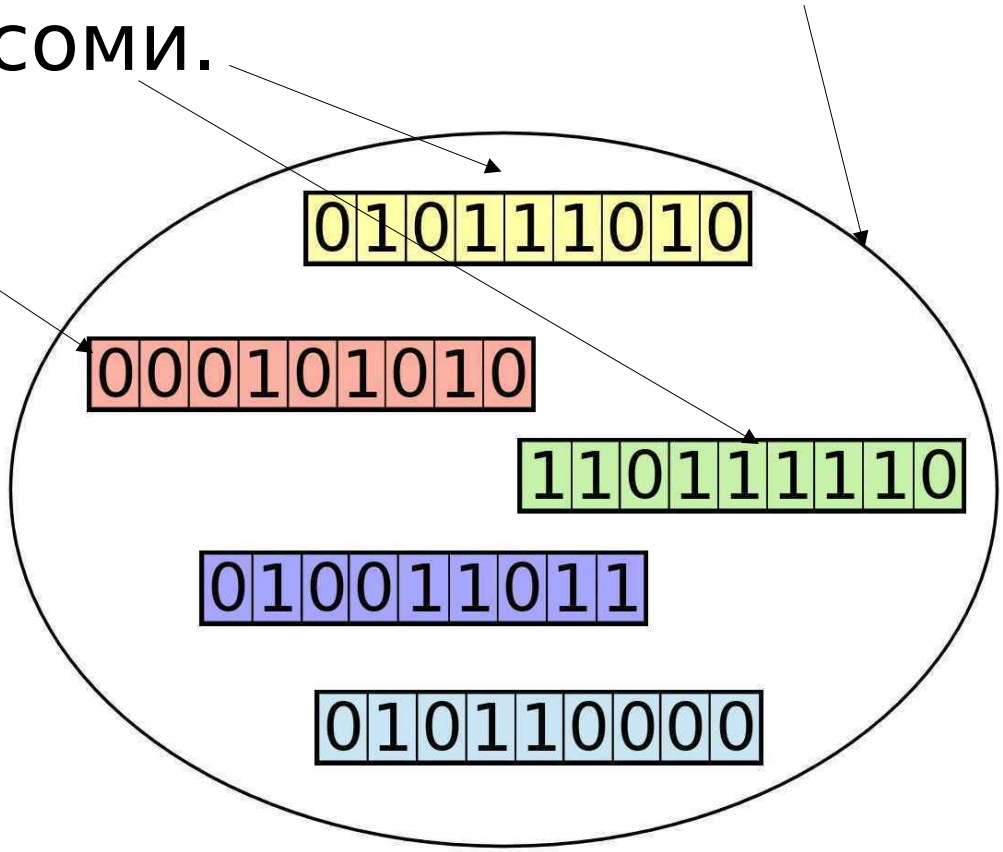
- **У цьому**(GAs):
- кожен**індивідуальний**представлений**'IT-хромосом**
що можна виразити як а**двійковий рядок**, де **кожен бітявляє собою єдине**ген**.



Основні компоненти -Населення

- GA завжди підтримує а**населення**осіб -> а**коле**
варіантів вирішення проблеми. Особина ->
хромосома, популяція -> сукупність
хромосоми.

Населення
представляє
поточнийпокоління
і
розвивається з часом
коли струм
покоління замінено
новим.



Основні компоненти -Фітнес функція

На кожній ітерації GA особи є **оцінено**від а **функція фітнесу**(також називається цільова функція). Це та функція, якої ми прагнемо оптимізувати або проблему, яку ми намагаємося вирішити.

Особи, які досягли а**краще**показник фітнесу **краще**рішень і, швидше за все, такими будуть **обраний для відтворення**і бути**представлений** наступне покоління.

З часом,**якість**рішень**покращує**, **фітнес**значення **збільшити.**Процес**може зупинитися**як тільки рішення є**знайдено**з**задовільний**значення фітнесу

Основні компоненти -Вибір

Вибірпроцес звик**визначити**до кого з особин популяції потрапить**відтворюватиі**

створити потомствощо сформує наступний покоління.

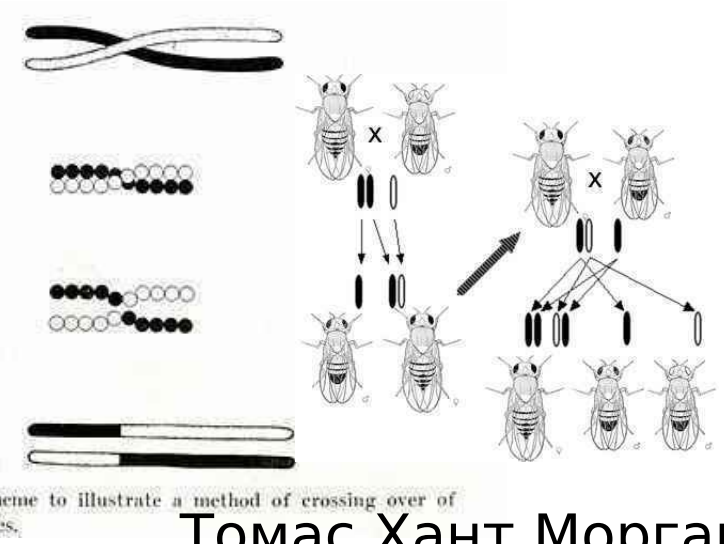
Це базується на фітнес-оцінці окремих осіб. Ті з **вище**оцінки є**швидше за все**бути

обраний і пасїхній генетичний матеріал до наступного покоління.

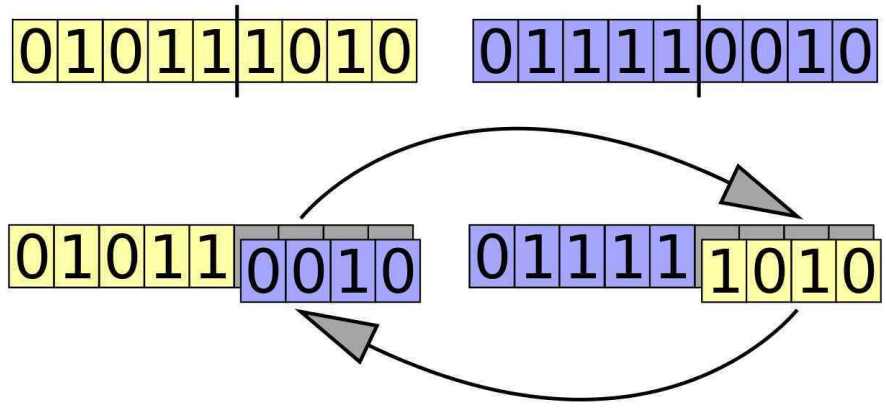
Фізичні особи с**низька тренованість**значення все ще можна вибрати, але**з меншою ймовірністю**. Таким чином, їх генетичний матеріал не виключається повністю.

Основні компоненти -Кросовер

Для створення пари нових особин зазвичай вибирають двох батьків із поточного покоління, а частини їхніх хромосом**поміняти місцями**(**кросовер** або**рекомбінація**) для створення двох нових хромос що представляє потомство.



Томас Хант Морган
(Нобелівська премія - 1933) ілюстрація переправа (1916)

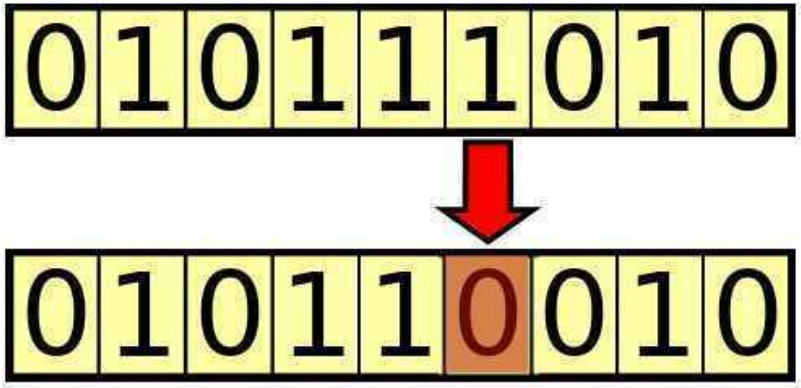


IT GA- версія
кросинговеру (рекомбінації)

Основні компоненти -Мутація

Мета**мутація**(як оператор) періодично і випадково**оновити населення**,**вводити нові зразки**в хромосоми, і**заохочувати пошук**у незвіданих областях простору рішення.

А**мутація**може бути як а**випадкова зміна**в гені, наприклад, перевернути біт у двійковому рядку.



Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- **Основна гіпотеза GA**
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Основна гіпотеза GA

The**будівельна гіпотеза**->в**оптимальний**вирішення проблеми є**зібрані**малої забудови**блоки**, і як ми приносимо**більше**цих будівельних блоків разом, ми отримуємо **ближче**до цього**оптимальний**рішення.

Особи в популяції з**бажана будівля** **блоки**ідентифікуються за їх**кращі бали**.

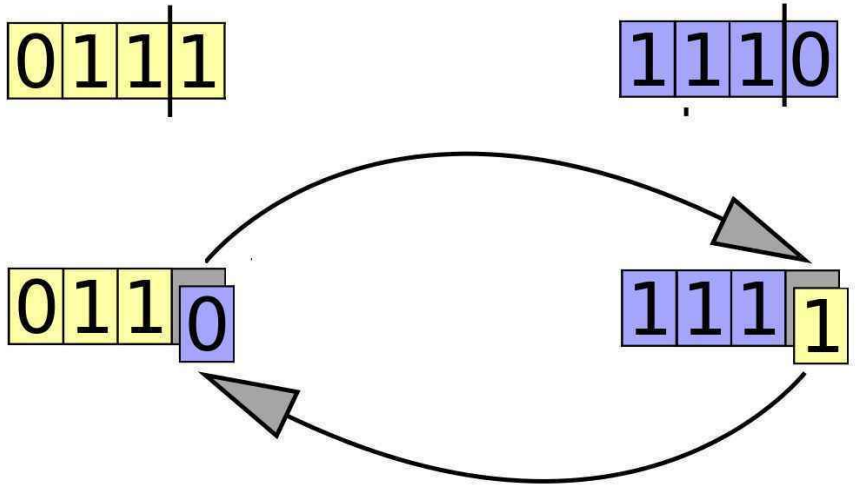
The**повторний вибір/кросовер**призведе до кращих індивідів, які передають ці будівельні блоки наступним поколінням, одночасно, можливо, поєднуючи їх з іншими успішні будівельні блоки.

Це створює**генетичний тиск**, таким чином**направляючий**Населенн**назустріч**мати більше людей із будівельними блоками**що**

Основна гіпотеза -**приклад**

Ми маємо сукупність 4-значних двійкових рядків. мета: що знайти рядок із**найбільший**можлива сума цифр. початок: Цифра що з’являється в будь-якому з 4 позицій рядка, буде

бути хорошим будівельним блоком.



Поступовий алгоритм визначить рішення, які містят ці будівельні блоки, і об’єднає їх. Кожен**новий** покоління матиме**більше**осіб із значеннями 1 у різних позиції, що в кінцевому підсумку призводить до рядка 1111, який

Теорема Холланда про схему

Схема це **візерунок** (або шаблон), які можуть бути
знаходяться в хромосомах.

Він представляє (як а **регулярний вираз** із символа
підстановки) підмножина хромосом, які мають
певна схожість між ними. приклад:
якщо набір хромосом представлено
двійковими рядками довжиною 4, схема
1*01 представляє всі ці хромосоми
які мають 1 у крайній лівій позиції, 01 у дв
крайніх правих позиціях і 1 або 0 у
другий зліва, оскільки *

Джон Генрі Голланд

(**2 лютого**, 1929 – 9 серпня 2015)

"Він є**засновник комплексного системного підходу**. Зокрема, він розробив генетичні алгоритми та навчальний класифікатор системи».

SANTA FE INSTITUTE

Він був членом Опікунської ради та наукової ради Інституту Санта-Фе та членом th **Всесвітній економічний форум** . Отримав 1961 рік**Медаль Луї Е. Леві** з Інституту Франкліна та**Стипендія МакАртура**(неофіційно відомий як «Genius Grant») у 1992 році.

THE FRANKLIN INSTITUTE

WORLD ECONOMIC FORUM

Теорема Холланда про схему

Для кожної схеми можна призначити дві метрики:

порядок:

Кількість фіксованих цифр (не символи підстановки!)

The following table provides several examples of four-digit binary schemata and their measurements:

Schema	Order	Defining Length
1101	4	3
1*01	3	3
*101	3	2
*1*1	2	2
**01	2	1
1***	1	0
****	0	0

Each chromosome in the population corresponds to multiple schemata in the same way that a given string matches regular expressions. The chromosome **1101**, for example,

Теорема Холланда про схему

Основна теорема GA: Частота схем
низький порядок,
коротка визначальна довжина, і**фізична підготовлен**
вище середньогозростає в геометричній прогресії
наступні покоління. Іншими словами:**менше,простіше** будівельні блоки, які представляють атрибути **які роблять рішення кращим**стане **все більше присутній**в популяції як ГА прогресує.

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- **Відмінності між GA та традиційними алгоритмами**
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Відмінності GAs з традиційних алгоритмів

Розрізнення основних характеристик ГА їх від традиційних алгоритмів:

- Підтримання а**населення**розчинів

- Використовуючи а**генетичне представлення**з рішення

- Використовуючи результат а**фітнес**функція

- Виставка а**імовірнісний**поведінка

Відмінності GAs з традиційних алгоритмів -

Підтримка популяції рішень

GA **діє над населенням** кандидатів рішень (окремих осіб), а не одного кандидата.

GA **працює з набором** осіб, які утворюють нинішнє покоління. **Кожна ітерація** ГА **створює** наступне покоління **встановити** фізичних осіб. На відмін від більшості інших пошукових алгоритмів **підтримувати єдин рішення** і ітеративно змінювати його в пошуках найкраще рішення.

приклад: The **алгоритм градієнтного спуску** (дуже широко використовується в ML/DL) ітераційно **працює зі струмом** рішення (переміщує його в напрямку найкрутішого спуску, визначається негативним градієнтом функції).

Відмінності GAs з традиційних алгоритмів - Генетичне представлення розчинів

Традиційні алгоритми: діють безпосередньо на кандидата рішення,

GA: оперувати своїми представленнями (або кодуванням), часто називають **хромосоми**. приклад: хромосома - це фіксований двійковий рядок.

The **генетичні операції** використовуються для хромосом

- **Кросовер** відбувається заміна частин хромосом між двома батьками.
- **Мутація** змінює частини хромосоми. Побічний ефект: ГА є **не в курсі** того, що представляють роблять хромосоми **не інтерпретувати** їх.

Відмінності GAs з традиційних алгоритмів - Результат фітнес-функції

Фітнес функція(FF) представляє (оцінити). проблема, яку ми хотіли б вирішити. Мета GA: щоб знайти осіб, які дають **найвищий бал** коли цей FF розраховується для них.

Традиційні алгоритми: **використовувати похідні**або будь-який інша інформація, що стосується FF. GA: **тільки** розглянути**значення**отримані FF. Це дозволяє використовувати FF, які важко або неможливо математично диференціювати.

Відмінності GAs з традиційних алгоритмів -

Імовірнісна поведінка

Традиційні алгоритми: є**детермінований**.

GAs : правила є**імовірнісний**.

приклад : при підборі особин, які будуть використовуватися створити наступне покоління,**ймовірність**вибору даної особи**збільшується з**індивіда**фітнес**, але у цьому виборі все ще є елемент випадковості. **Мутація** керується ймовірністю, зазвичай вносить зміни в випадкове розташування в хромосомі. **Кросовер**може також мати ймовірнісний елемент. Незважаючи на імовірнісни характер,**GA не є випадковим**; натомість це **використовує випадковий**аспект, щоб спрямувати пошук на області в простір пошуку, де є кращі шанси покращитися результати.

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- **Переваги ГА**
- Обмеження GA
- Коли використовувати GA

Переваги ГА

- **Глобальний** можливість оптимізації
- Вирішення проблем із а**складні** математичне представлення
- Вирішення проблем, які **відсутність** математичний представництво
- **Стійкість** до шуму
- Підтримка для **паралелізм** і поширюється обробки
- Придатність для **безперервне навчання**

Переваги GA -

Можливість глобальної оптимізації

Традиційні алгоритми

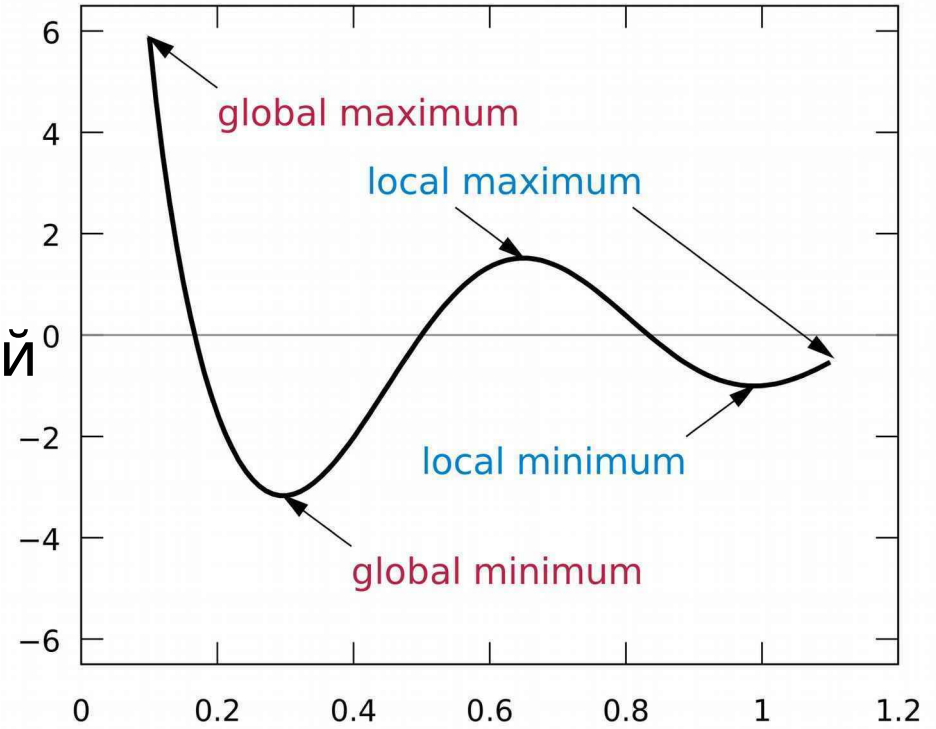
(на основі градієнта) :

може застрягти в локальному максимумі, а не знайти глобальний

->

оскільки поблизу локального максимуму, будь-яка незначна зміна погіршиться

рахунок.



GA: більше шансів знайти глобальний максимум через:

1 - використання а**населення**варіантів рішень,

2 -**кросовер**і**мутація**це, у багатьох випадках, призвед до варіантів рішень, які є**далекий**від попередніх.

Це є**правда якщо**ми підтримуємо**різноманітність**населення і

Переваги GA -

Комплексні проблеми

GAs**потрібен тільки вихід**FF для кожної окрем
людини та не стосуються інших аспектів FF
такі як похідні.

- Ось чому GAs**може бути ефективним**для проблем з
- **складні математичні уявлення**або
- функції, які є**важко**або неможливо

диференціювати,

- проблеми з а**велика кількість параметрів**,
- проблеми з а**поєднання типів параметрів**

(поєднання неперервного і дискретного
параметри).

Переваги GA - Проблем без Математичне представлення

Припустимо, що оцінка FF базується на людській думці

приклад:

знайти найпривабливішу палітру кольорів для веб-сайту.

рішення:

- спробувати**різні поєднання кольорів**і запитайте користувачів оцінити привабливість сайту;

- застосовувати GA для пошуку**найкращий результат поєднання**використовуючи цю оцінку на основі думки як результат фітнес-функції.

GA зробить це,**незважаючи на те, що FF має БЕЗ математичного представлення**і є**НЕМАЄ способу обчислити рахунок**безпосередньо з заданої колірної комбінації.

Переваги GA –

Стійкість до шуму

Є деякі проблеми**шумна поведінка**:

-навіть для**аналогічний вхід**значення параметрів,

в**вихід**значення може бути**дещо інше**кожен

час, який вимірюється.

Приклади :

-дані надходять із виходів датчиків, або

-Оцінка FF базується на людській думці.

Шумна поведінкаможе**руїна**багато**традиційний** алгоритми, але**GAs**загалом**пружний**до нього, завдяки до**повторюваній**операція**повторне складання**і **переоцінка**особи.

Переваги GA -

Паралелізм

GA за своїм визначенням є**готовий до розпаралелювання** і**розподілена обробка**.

- **FF**є**самостійно**розрахована на**кожен** індивідуальний, що означає**всі особи**в популяц може бути**оцінюється одночасно**.

- Генетичні операції**вибір**,**кросовер**, і **мутація** кожен може бути виконаний**одночасно**на особинах і парах особин у популяції.

Ось чому**GA є природними кандидатами**для

Переваги GA -

Безперервне навчання

В природі,**еволюція ніколи не зупиняється**.

Але це сумнівно ... :) ... подивіться навколо.

Як екологічний**умови змінюються**,
населення адаптуєтьсяїм.

Так само**ГА можуть працювати безперервно**у колись мінливе середовище, і в будь-який момент часу, **найкраще поточне рішення може бути отримано та використано.**

Але як щодо часу?

Щоб GA була ефективною,**зміни**в середовищі **потрібно бути повільним**по відношенню до покоління**швидкість обороту**пошуку на основі GA.

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- **Обмеження GA**
- Коли використовувати GA

Обмеження GA

- Необхідність спеціальних визначень

- Необхідність гіперпараметричного налаштування

- Обчислювально інтенсивні операції

- Ризик передчасної конвергенції

- Немає гарантованого рішення

Обмеження GA -

Спеціальні визначення

- Щоб застосувати GA до даної проблеми, нам потрібно створити відповідне представлення для GA і **визначити**
  - **FF**і**хромосома** структура,
  - **генетичні оператори** (відбір, схрещування та мутація), які будуть працювати для цієї проблеми.
- **Це складний і трудомісткий процес!**

- **АЛЕ ... GA вже застосовано незліченні різні типи проблем**, і багато з цих визначень стандартизовано.
- В інших лекціях будуть представлені деякі типи реальних проблем, які можна вирішити за допомогою GA.

Обмеження GA -

Гіперпараметрична настройка

Поведінка GA контролюється набором **гіперпараметри**, наприклад**чисельність населен**

швидкість мутаціїі т.д.

Застосовуючи GA до проблеми,

точних правил немає (!)

для здійснення цих виборів.

однак,**це є**вірно також для ... майже всіх традиційних алгоритмів пошуку та оптимізації

Провівши кілька власних експериментів, ви зможе зробити розумний вибір для цих значень.

Обмеження GA -

Обчислювально інтенсивний

Операції

Працює (потенційно**великий і дуже великий**) населення та**повторюваніприрода** ГА може бути **обчислювально інтенсивний**, так добре як **трудомісткий**до досягнення хорошого результату.

Їх можна полегшити:

- а**гарний вибір**гіперпараметрів,
- впровадження**паралельна обробка**,
- і**кешування**проміжні результати (у деяких випадків).

Обмеження GA - Ризик передчасної конвергенції

Якщо придатність однієї особини є **набагато вище** ніж решта населення, можливо **достатньо дубльований**, що це **бере на себе** все населення.

Це може призвести до отримання GA **передчасно застряг** **локальний екстремум**, **замість** знаходження **глобальний**

Щоб цього не сталося, важливо **підтримувати різноманітність** з населення.

Обмеження GA - Немає гарантованого рішення

- Використання GA робить **не гарантія** що глобальний екстремум для розглянутої проблеми **буде знайдено**

- Однак це майже **вірно для ... будь-якого традиційно** алгоритм пошуку та оптимізації, якщо це не аналітичне рішення конкретного типу задачі.

- Як правило, GA, якщо використовуються належним чином, відомі **надавати хороші рішення** в межах а **розумний проміжок часу**.

Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- **Коли використовувати GA**

Випадки використання GA

- ГА є **найкраще підходить для** наступні типи проблеми:
-
- з **складні** математичне представлення
- з **ні** математичне представлення
- за участю а **шумний** навколишнє середовище
- залучення середовища, яке **змінюється з часом**

Лекція 1 - ДЕМО А - Вступ до генетичних алгоритмів

(довга версія) на основі (С) роботи Еяля Вірсанського

У цій лекції ми представляємо **DEAP — потужну та гнучку структуру еволюційних обчислень**, здатну вирішувати реальні проблеми з використанням **генетичних алгоритмів (GA)**.

Короткий зміст:

- вступ,
- установка,
- основні модулі: *creator* і *toolbox*,
- компоненти, необхідні для робочого процесу GA,
- найпростіший приклад, *проблема OneMax*, так звана «Hello World» генетичних алгоритмів.

До кінця цієї лекції ви будете знати:

- структура DEAP та її модулі,
- поняття creator та toolbox у структурі DEAP,
- найпростіший приклад GA,
- як створити рішення GA за допомогою структури DEAP,
- як використовувати вбудовані алгоритми структури DEAP для створення стислого коду
- як вирішити проблему OneMax за допомогою GA, закодованого за допомогою DEAP framework,
- як експериментувати з різними налаштуваннями GA та інтерпретувати відмінності в результатах.

Installation and import of libraries

In these and other lectures, we will use various Python packages:

- NumPy
- Matplotlib
- Seaborn

They are already pre-installed in Colab. Let's import them by the following code.

```
# Import all necessary standard libraries
import random
import numpy
import matplotlib.pyplot as plt
import seaborn as sns
```

Install DEAP by *pip* with the following code:

```
# Install DEAP
!pip install deap
```

```
Collecting deap
  Downloading https://files.pythonhosted.org/packages/0a/eb/2bd0a32e3ce757fb2
     |████████████████████████████████| 163kB 8.7MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-package
Installing collected packages: deap
Successfully installed deap-1.3.1
```

```
# Import DEAP
from deap import base
from deap import creator
from deap import tools
from deap import algorithms
```

Example: OneMax problem

Constants

```
# Let's declare constants that set the parameters for the problem and control the

# problem constants:
ONE_MAX_LENGTH = 100  # length of bit string to be optimized

# GA constants:
POPULATION_SIZE = 200
P_CROSSOVER = 0.9  # probability for crossover
P_MUTATION = 0.1   # probability for mutating an individual
MAX_GENERATIONS = 50
```

Reproducibility of Results

One important aspect of the GA is the use of probability, which introduces a random element to the behavior of the algorithm.

However, **for reproducibility of results**, when experimenting with the code, we may want to be able to run the same experiment several times and get repeatable results.

To accomplish this, we set the random function seed to a constant number of some value, as

```
# set the random seed:
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
```

Toolbox class

The **Toolbox** class is used as a container for functions (or operators), and enables us to create new operators by aliasing and customizing existing functions.

```
toolbox = base.Toolbox()
```

```
# For example, suppose we have a function, multiply() , defined as follows:
def multiply(a, b):
  return a*b

# Using toolbox, we can now create a new operator, incrementByFive(),
# which customizes the sumOfTwo() function as follows:
toolbox.register("MultiplyBy", multiply, b=5)

# examples:
A = toolbox.MultiplyBy(10)
print('toolbox.MultiplyBy(10) =', A)
B = multiply(10,5)
print('multiply(10,5) =', B)
```

```
toolbox.MultiplyBy(10) = 50
multiply(10,5) = 50
```

Let's create the *zeroOrOne* operator, which customizes the *random.randomint(a, b)* function.

This function normally returns a random integer *N* such that $a \le N \le b$.

By fixing the two arguments, *a* and *b*, to the values 0 and 1 the *zeroOrOne* operator will randomly return either the value 0 or the value 1 when called later in the code.

```
# create an operator that randomly returns 0 or 1:
toolbox.register("zeroOrOne", random.randint, 0, 1)

# examples:
A = toolbox.zeroOrOne()
print('zeroOrOne =', A)
B = toolbox.zeroOrOne()
print('zeroOrOne =', B)
C = toolbox.zeroOrOne()
print('zeroOrOne =', C)
```

```
D = toolbox.zeroOrOne()
print('zeroOrOne =', D)
```

```
zeroOrOne = 0
zeroOrOne = 0
zeroOrOne = 1
zeroOrOne = 0
```

Fitness class

Next, we need to create the *Fitness* class. Since we only have one objective here—the sum of digits—and our goal is to maximize it, we choose the FitnessMax strategy, using a weights tuple with a single positive weight, as shown in the following code.

```
# define a single objective, maximizing fitness strategy:
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
```

```
A = base.Fitness.weights
print(A)
```

```
None
```

In DEAP, the *Individual* class is used to represent each of the population's individuals. This class is created with the help of the creator tool. In our case, list serves as the base class, which is used as the individual's chromosome. The class is augmented with the fitness attribute, initialized to the *FitnessMax* class that we defined earlier

```
# create the Individual class based on list:
creator.create("Individual", list, fitness=creator.FitnessMax)
#creator.create("Individual", array.array, typecode='b', fitness=creator.FitnessMa
```

Next, register the *individualCreator* operator, which creates an instance of the *Individual* class, filled up with random values of either 0 or 1 . This is done by customizing the previously defined *zeroOrOne* operator.

Since the objects generated by the *zeroOrOne* operator are integers with random values of either 0 or 1 , the resulting *individualCreator* operator will fill an *Individual* instance with 100 randomly generated values of 0 or 1.

```
# create the individual operator to fill up an Individual instance:
toolbox.register("individualCreator",   # Register the individualCreator operator,
                 tools.initRepeat,      # The initRepeat operator is customized he
                 creator.Individual,    # The container type (Individual) in which
                 toolbox.zeroOrOne,     # The function used to generate objects (=
                 ONE_MAX_LENGTH)        # The number of objects we want to generat
```

Register the *populationCreator* operator that creates a list of individuals.

```
# create the population operator to generate a list of individuals:
toolbox.register("populationCreator",   # Register the populationCreator operator,
                 tools.initRepeat,      # The initRepeat operator is customized he
                 list,                  # The container type (list) in which the r
                 toolbox.individualCreator) # The function used to generate objects
```

Define the function *oneMaxFitness* that computes the number of 1s in the individual.

```
# fitness calculation:
# compute the number of '1's in the individual
def oneMaxFitness(individual):
    return sum(individual),  # return a tuple,
                             # fitness values in DEAP are represented as tuples,
                             # and therefore a comma needs to follow when a single
```

Define the *evaluate* operator as an alias to the *oneMaxfitness()* function we defined earlier.

```
# create the evaluate alias for calculating the fitness (by a DEAP convention)
toolbox.register("evaluate", oneMaxFitness)
```

Genetic operators

The genetic operators are typically created by aliasing existing functions from the tools module and setting the argument values as needed.

Note: The *mutFlipBit* function iterates over all the attributes of the individual, a list with values of 1s and 0s in our case, and for each attribute will use the argument value (*indpb* parameter) as the probability of flipping (applying the not operator to) the attribute value. This value is independent of the mutation probability, which is set by the *P_MUTATION* constant that we defined earlier and has not yet been used. The mutation probability serves to decide if the *mutFlipBit* function is called for a given individual in the population.

```
# genetic operators:

# Tournament selection with tournament size of 3:
toolbox.register("select", tools.selTournament, tournsize=3)

# Single-point crossover:
toolbox.register("mate", tools.cxOnePoint)

# Flip-bit mutation:
# indpb: Independent probability for each attribute to be flipped
toolbox.register("mutate", tools.mutFlipBit, indpb=1.0/ONE_MAX_LENGTH)
```

GA workflow

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)
generationCounter = 0
```

Long version

```
# calculate fitness tuple for each individual in the population:
fitnessValues = list(map(toolbox.evaluate, population)) # use map() to apply the ev
for individual, fitnessValue in zip(population, fitnessValues):
  individual.fitness.values = fitnessValue
```

```
# extract the first value out of each fitness for gathering statistics:
fitnessValues = [individual.fitness.values[0] for individual in population]
```

```
# initialize statistics accumulators:
maxFitnessValues = []
meanFitnessValues = []
```

```
# main evolutionary loop:
# stop if max fitness value reached the known max value
# OR if number of generations exceeded the preset value:
while max(fitnessValues) < ONE_MAX_LENGTH and generationCounter < MAX_GENERATIONS:
  # update counter:
  generationCounter = generationCounter + 1

  # apply the selection operator, to select the next generation's individuals:
  offspring = toolbox.select(population, len(population))
  # clone the selected individuals:
  offspring = list(map(toolbox.clone, offspring))

  # apply the crossover operator to pairs of offspring:
  for child1, child2 in zip(offspring[::2], offspring[1::2]):
    if random.random() < P_CROSSOVER:
      toolbox.mate(child1, child2)
      del child1.fitness.values
      del child2.fitness.values

      for mutant in offspring:
        if random.random() < P_MUTATION:
          toolbox.mutate(mutant)
          del mutant.fitness.values

  # calculate fitness for the individuals with no previous calculated fitness valu
  freshIndividuals = [ind for ind in offspring if not ind.fitness.valid]
```

```
freshFitnessValues = list(map(toolbox.evaluate, freshIndividuals))
for individual, fitnessValue in zip(freshIndividuals, freshFitnessValues):
  individual.fitness.values = fitnessValue

# replace the current population with the offspring:
population[:] = offspring

# collect fitnessValues into a list, update statistics and print:
fitnessValues = [ind.fitness.values[0] for ind in population]

maxFitness = max(fitnessValues)
meanFitness = sum(fitnessValues) / len(population)
maxFitnessValues.append(maxFitness)
meanFitnessValues.append(meanFitness)
print("- Generation {}: Max Fitness = {}, Avg Fitness = {}".format(generationCou

# find and print best individual:
best_index = fitnessValues.index(max(fitnessValues))
print("Best Individual = ", *population[best_index], "\n")
```

```
- Generation 1: Max Fitness = 62.0, Avg Fitness = 52.59
Best Individual =  0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1

- Generation 2: Max Fitness = 64.0, Avg Fitness = 55.205
Best Individual =  0 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0 1

- Generation 3: Max Fitness = 67.0, Avg Fitness = 56.88
Best Individual =  1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0

- Generation 4: Max Fitness = 71.0, Avg Fitness = 58.425
Best Individual =  1 1 1 0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 1

- Generation 5: Max Fitness = 69.0, Avg Fitness = 59.77
Best Individual =  0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1

- Generation 6: Max Fitness = 73.0, Avg Fitness = 61.53
Best Individual =  1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1

- Generation 7: Max Fitness = 73.0, Avg Fitness = 62.525
Best Individual =  1 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1

- Generation 8: Max Fitness = 74.0, Avg Fitness = 63.23
Best Individual =  1 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1

- Generation 9: Max Fitness = 74.0, Avg Fitness = 63.76
Best Individual =  1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1

- Generation 10: Max Fitness = 74.0, Avg Fitness = 64.165
Best Individual =  1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 1 1 1 1

- Generation 11: Max Fitness = 75.0, Avg Fitness = 64.23
Best Individual =  1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1

- Generation 12: Max Fitness = 75.0, Avg Fitness = 64.83
Best Individual =  1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1

- Generation 13: Max Fitness = 78.0, Avg Fitness = 65.225
Best Individual =  1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1
```

```
- Generation 14: Max Fitness = 80.0, Avg Fitness = 65.355
Best Individual =  1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1

- Generation 15: Max Fitness = 75.0, Avg Fitness = 65.87
Best Individual =  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1

- Generation 16: Max Fitness = 76.0, Avg Fitness = 65.705
Best Individual =  0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 1

- Generation 17: Max Fitness = 75.0, Avg Fitness = 65.845
Best Individual =  1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1

- Generation 18: Max Fitness = 79.0, Avg Fitness = 66.02
Best Individual =  1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1

- Generation 19: Max Fitness = 80.0, Avg Fitness = 66.44
Best Individual =  1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1
```

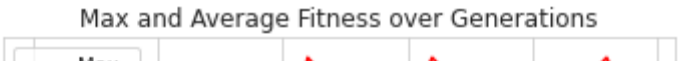
You should get the following output:

-- Generation 1: Max Fitness = 62.0, Avg Fitness = 52.59 Best Individual = 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 0

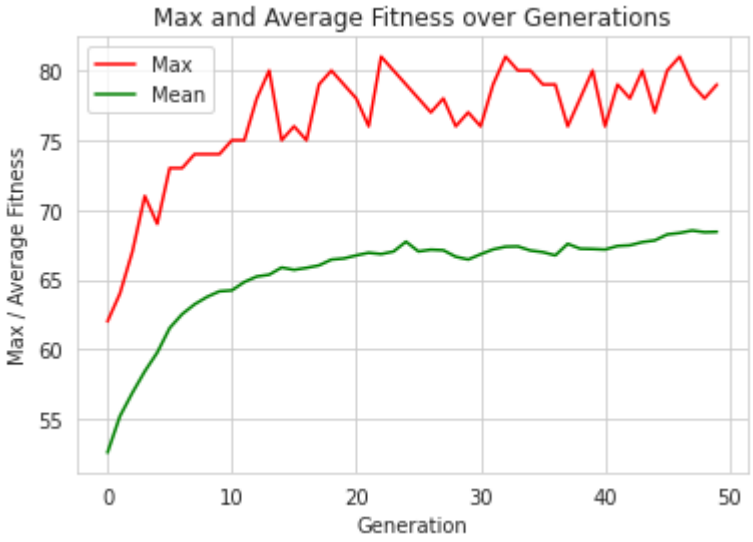
...

-- Generation 50: Max Fitness = 79.0, Avg Fitness = 68.43 Best Individual = 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1

```
# Plot statistics:
sns.set_style("whitegrid")
plt.plot(maxFitnessValues, color='red', label='Max')
plt.plot(meanFitnessValues, color='green', label='Mean')
plt.xlabel('Generation')
plt.ylabel('Max / Average Fitness')
plt.title('Max and Average Fitness over Generations')
plt.legend()
plt.show()
```



You should get the following output:



Лекція 1 – ДЕМО В – Вступ до генетичних алгоритмів

(коротка версія реалізації коду) на основі (С) роботи Еяля Вірсанського

У цій лекції ми представляємо **DEAP — потужну та гнучку структуру еволюційних обчислень**, здатну вирішувати реальні проблеми з використанням **генетичних алгоритмів (GA)**.

Короткий зміст:

- вступ,
- встановлення,
- основні модулі: *creator* і *toolbox*,
- компоненти, необхідні для робочого процесу GA,
- найпростіший приклад, *проблема OneMax*, так звана «Hello World» генетичних алгоритмів.

До кінця цієї лекції ви будете знати:

- структура DEAP та її модулі,
- поняття creator та toolbox у структурі DEAP,
- найпростіший приклад GA,
- як створити рішення GA за допомогою структури DEAP,
- як використовувати вбудовані алгоритми структури DEAP для створення стислого коду
- як вирішити проблему OneMax за допомогою GA, закодованого за допомогою DEAP framework,
- як експериментувати з різними налаштуваннями GA та інтерпретувати відмінності в результатах.

Installation and import of libraries

In these and other lectures, we will use various Python packages:

- NumPy
- Matplotlib
- Seaborn

They are already pre-installed in Colab. Let's import them by the following code.

```
# Import all necessary standard libraries
import random
import numpy
import matplotlib.pyplot as plt
import seaborn as sns
```

Install DEAP by *pip* with the following code:

```
# Install DEAP
!pip install deap
```

```
Collecting deap
  Downloading https://files.pythonhosted.org/packages/0a/eb/2bd0a32e3ce757fb2
     |████████████████████████████████| 163kB 8.2MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-package
Installing collected packages: deap
Successfully installed deap-1.3.1
```

```
# Import DEAP
from deap import base
from deap import creator
from deap import tools
from deap import algorithms
```

Example: OneMax problem

Constants

```
# Let's declare constants that set the parameters for the problem and control the

# problem constants:
ONE_MAX_LENGTH = 100  # length of bit string to be optimized

# GA constants:
POPULATION_SIZE = 200
P_CROSSOVER = 0.9  # probability for crossover
P_MUTATION = 0.1   # probability for mutating an individual
MAX_GENERATIONS = 50
```

Reproducibility of Results

One important aspect of the GA is the use of probability, which introduces a random element to the behavior of the algorithm.

However, **for reproducibility of results**, when experimenting with the code, we may want to be able to run the same experiment several times and get repeatable results.

To accomplish this, we set the random function seed to a constant number of some value, as shown in the following code:

```
# set the random seed:
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
```

Toolbox class

The **Toolbox** class is used as a container for functions (or operators), and enables us to create new operators by aliasing and customizing existing functions.

```
toolbox = base.Toolbox()
```

```
# For example, suppose we have a function, multiply() , defined as follows:
def multiply(a, b):
  return a*b

# Using toolbox, we can now create a new operator, incrementByFive(),
# which customizes the sumOfTwo() function as follows:
toolbox.register("MultiplyBy", multiply, b=5)

# examples:
A = toolbox.MultiplyBy(10)
print('toolbox.MultiplyBy(10) =', A)
B = multiply(10,5)
print('multiply(10,5) =', B)
```

```
    toolbox.MultiplyBy(10) = 50
    multiply(10,5) = 50
```

Let's create the *zeroOrOne* operator, which customizes the *random.randomint(a, b)* function.

This function normally returns a random integer *N* such that $a \le N \le b$.

By fixing the two arguments, *a* and *b*, to the values 0 and 1 the *zeroOrOne* operator will randomly return either the value 0 or the value 1 when called later in the code.

```
# create an operator that randomly returns 0 or 1:
toolbox.register("zeroOrOne", random.randint, 0, 1)

# examples:
A = toolbox.zeroOrOne()
print('zeroOrOne =', A)
B = toolbox.zeroOrOne()
```

```
print('zeroOrOne =', B)
C = toolbox.zeroOrOne()
print('zeroOrOne =', C)
D = toolbox.zeroOrOne()
print('zeroOrOne =', D)
```

```
zeroOrOne = 0
zeroOrOne = 0
zeroOrOne = 1
zeroOrOne = 0
```

Fitness class

Next, we need to create the *Fitness* class. Since we only have one objective here—the sum of digits—and our goal is to maximize it, we choose the FitnessMax strategy, using a weights tuple with a single positive weight, as shown in the following code.

```
# define a single objective, maximizing fitness strategy:
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
```

```
A = base.Fitness.weights
print(A)
```

```
None
```

In DEAP, the *Individual* class is used to represent each of the population's individuals. This class is created with the help of the creator tool. In our case, list serves as the base class, which is used as the individual's chromosome. The class is augmented with the fitness attribute, initialized to the *FitnessMax* class that we defined earlier

```
# create the Individual class based on list:
creator.create("Individual", list, fitness=creator.FitnessMax)
#creator.create("Individual", array.array, typecode='b', fitness=creator.FitnessMax
```

Next, register the *individualCreator* operator, which creates an instance of the *Individual* class, filled up with random values of either 0 or 1 . This is done by customizing the previously defined *zeroOrOne* operator.

Since the objects generated by the *zeroOrOne* operator are integers with random values of either 0 or 1 , the resulting *individualCreator* operator will fill an *Individual* instance with 100 randomly generated values of 0 or 1.

```
# create the individual operator to fill up an Individual instance:
toolbox.register("individualCreator",   # Register the individualCreator operator,
                 tools.initRepeat,      # The initRepeat operator is customized he
                 creator.Individual,    # The container type (Individual) in which
```

```
                 toolbox.zeroOrOne,     # The function used to generate objects (=
                 ONE_MAX_LENGTH)        # The number of objects we want to generat
```

Register the *populationCreator* operator that creates a list of individuals.

```
# create the population operator to generate a list of individuals:
toolbox.register("populationCreator",   # Register the populationCreator operator,
                 tools.initRepeat,      # The initRepeat operator is customized he
                 list,                  # The container type (list) in which the r
                 toolbox.individualCreator) # The function used to generate object
```

Define the function *oneMaxFitness* that computes the number of 1s in the individual.

```
# fitness calculation:
# compute the number of '1's in the individual
def oneMaxFitness(individual):
    return sum(individual),  # return a tuple,
                             # fitness values in DEAP are represented as tuples,
                             # and therefore a comma needs to follow when a single
```

Define the *evaluate* operator as an alias to the *oneMaxfitness()* function we defined earlier.

```
# create the evaluate alias for calculating the fitness (by a DEAP convention)
toolbox.register("evaluate", oneMaxFitness)
```

Genetic operators

The genetic operators are typically created by aliasing existing functions from the tools module and setting the argument values as needed.

Note: The *mutFlipBit* function iterates over all the attributes of the individual, a list with values of 1s and 0s in our case, and for each attribute will use the argument value (*indpb* parameter) as the probability of flipping (applying the not operator to) the attribute value. This value is independent of the mutation probability, which is set by the *P_MUTATION* constant that we defined earlier and has not yet been used. The mutation probability serves to decide if the *mutFlipBit* function is called for a given individual in the population.

```
# genetic operators:

# Tournament selection with tournament size of 3:
toolbox.register("select", tools.selTournament, tournsize=3)

# Single-point crossover:
toolbox.register("mate", tools.cxOnePoint)
```

```
# Flip-bit mutation:
# indpb: Independent probability for each attribute to be flipped
toolbox.register("mutate", tools.mutFlipBit, indpb=1.0/ONE_MAX_LENGTH)
```

GA workflow

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)
```

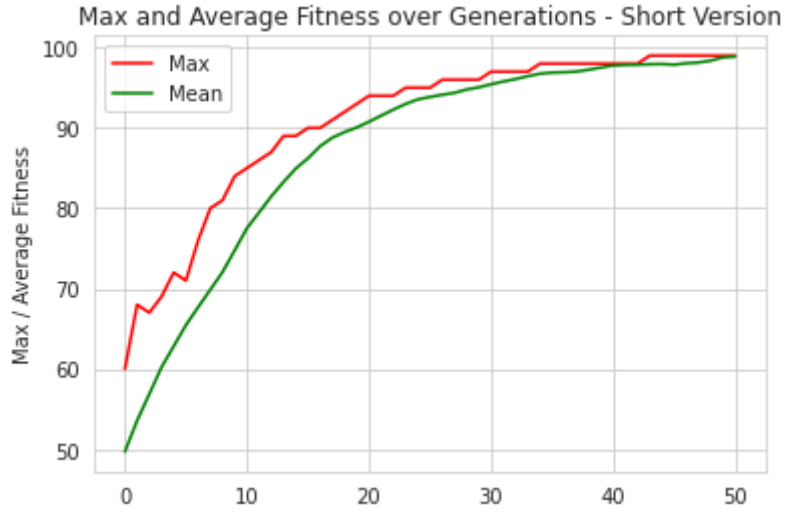
Short version

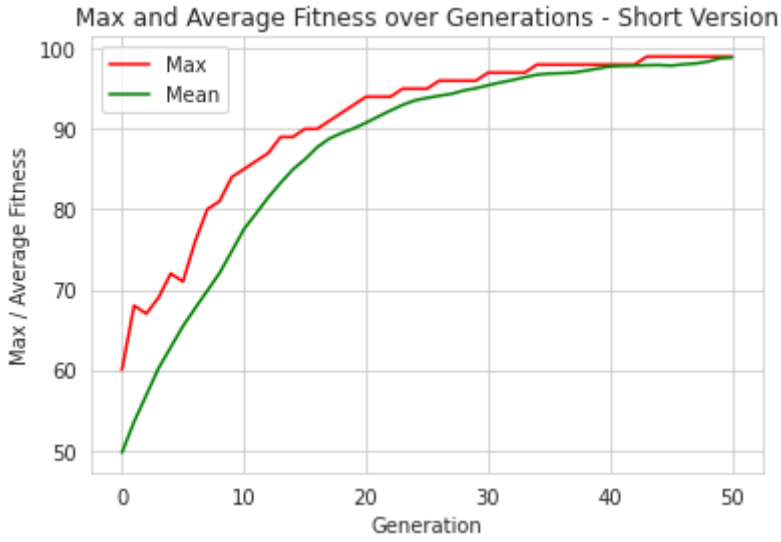
```
# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# perform the Genetic Algorithm flow:
population, logbook = algorithms.eaSimple(population, toolbox, cxpb=P_CROSSOVER, mu
                                  stats=stats, verbose=True)

# Genetic Algorithm is done - extract statistics:
maxFitnessValues, meanFitnessValues = logbook.select("max", "avg")
```

```
gen     nevals  max     avg
0       200     60      49.705
1       190     68      53.56
2       175     67      56.87
3       179     69      60.21
4       175     72      62.825
5       184     71      65.45
6       178     76      67.68
7       187     80      69.865
8       189     81      72.055
9       184     84      74.765
10      185     85      77.515
11      181     86      79.485
12      190     87      81.49
13      181     89      83.27
14      184     89      84.94
15      189     90      86.22
16      176     90      87.725
17      176     91      88.79
18      182     92      89.485
19      185     93      90.065
20      182     94      90.765
21      170     94      91.535
22      179     94      92.28
23      178     95      92.985
24      181     95      93.545
25      189     95      93.855
26      174     96      94.125
```

```
27      179     96      94.36
28      186     96      94.78
29      185     96      95.055
30      185     97      95.43
31      186     97      95.775
32      187     97      96.075
33      179     97      96.435
34      176     98      96.745
35      187     98      96.885
36      186     98      96.93
37      190     98      97.015
38      175     98      97.245
39      171     98      97.515
40      179     98      97.78
41      188     98      97.845
42      188     98      97.87
43      178     99      97.925
44      174     99      97.95
45      176     99      97.87
46      185     99      98.04
47      184     99      98.14
48      184     99      98.37
49      187     99      98.79
50      185     99      98.885
```

```
# Plot statistics:
sns.set_style("whitegrid")
plt.plot(maxFitnessValues, color='red', label='Max')
plt.plot(meanFitnessValues, color='green', label='Mean')
plt.xlabel('Generation')
plt.ylabel('Max / Average Fitness')
plt.title('Max and Average Fitness over Generations - Short Version')
plt.legend()
plt.show()
```





You should get the following output:

Основи еволюційних обчислень

-

Основи еволюційних обчислень

-

Лекція 02. Огляд

(за мотивами Алана Тюрінга, Холланда, Халеда Рашида, Бена роботи Філіпса, Еяля Вірсанського та ін.)

. . . на попередній лекції... Зміст

- Рекомендовані джерела
- Що таке генетичні алгоритми (GA)?
- GA Аналогія з IT
- Компоненти ГА
- Основна гіпотеза GA
- Відмінності між GA та традиційними алгоритмами
- Переваги ГА
- Обмеження GA
- Коли використовувати GA

Зміст — це лекція

- **Рекомендовані джерела**
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням

Рекомендовані джерела - Книги

(те саме, що для GA!)

Книги (класика):

Голландія, JH (1992). Адаптація в природних і штучних системах: вступний аналіз із застосуванням до біології, управління та штучний інтелект. Преса MIT. **<-винахідник GA(!)**, **найбільша кількість цитувань для GA-публікації** від Google Scholar!

Мітчелл, M. (1998). Введення в генетичні алгоритми. Преса MIT. <-**класичний підручник**, **найбільша кількість цитати до ГА-підруч** від Google Scholar!

Книги (з кодами на github):

Вірсанський, E. (2020). Практичні генетичні алгоритми з Python. Packt Publishing

Шеппард, K. (2019). Генетичні алгоритми з Python (скоро опубліковано).

Рекомендовані джерела - папери

(те саме, що для GA!)

Голландія, JH (1992). Генетичні алгоритми. Scientific American, 267 (1), 66-73. **<-винахідник GA(!) <- Просто задля розваги! :)**

Каточ, С., Чаухан, С. С., Кумар, В. (2020). Огляд про генетичний алгоритм: минуле, теперішнє та майбутнє. Мультимедійні засоби та програми, 1-36.

Гарсіа-Мартінес, К., Родрігес, Ф. Дж., Лозано, М. (2018). Генетичні алгоритми. Довідник з евристики, 2018, стор. 431-464.

Зміст

- Рекомендовані джерела
- **Що таке еволюційні обчислення (ЕС)**
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням

Що таке еволюційні обчислення (ЕС)

ЕВОЛЮЦІЯ		**ІТ**
		–
Навколишнє середовище	⟷	проблема
Індивідуальний	⟷	Кандидат Рішення (Індивідуальна)
Фітнес	⟷	Якість

Фітнес - шанси на виживання і відтворення

якість - шанс для посів нові рішення

ЩоЄК — Метафора (природа-IT)

А**населення**з**особи**існує в середовищі с **обмежений**ресурси.

Конкуренціядля тих**ресурси**причини**вибір**з тих **монтер**осо які є**краще адаптовані**до навколишнього середовища.

Ці**особи**виступають в якості**насіння**для**нове покоління**з осіб через деякі**варіаційні операції** (наприклад, GA як рекомбінація та мутація).

Нові особини мають свої**оцінено придатність**і змагатися (можливо, також з батьками)**для виживання**.

Природний відбірвикликає а**підвищення фізичної форми**з

Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- **Історія ЄС**
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням

ЄС — Історія — Засновники

1948, Тюрінг:

«генетичний або еволюційний пошук»

1962, Бремерман

оптимізація через еволюцію та рекомбінацію

1964, Рехенберг

стратегії еволюції

1965, Л. Фогель, Оуенс і Волш

еволюційне програмування

1975 рік, Голландія

генетичні алгоритми

1992, Коза

генетичне програмування

ЄС —Історія — Громада

1985 рік:

перша міжнар**конференції**(ICGA)

1990 рік:

перша міжнар**конференції в Європі** (PPSN)

1993 рік:

перший науковий ЕК**журнал**(MIT Press)

1997 рік:

запуск Європейського ЕС**Дослідницька мережа**EvoNet

ЄС — Історія — ЗАРАЗ!

- 3 **майор** конференції ЄС
  + 10 **маленький** споріднені

- 3 наук **основні журнали ЄС**

- **750-1000 паперів** опубліковано в 2003 році

- численні додатки

- численні консалтингові та науково-дослідні фірми

ЄС —Історія — Уроки

Природа завжди була джерелом натхнення для інженерів та вчених

Найкращий засіб вирішення проблем, відомий у природі:

- мозок (людини).який створив «колесо, Нью-Йорк, війни тощо» (за путівником автостопом Дугласа Адамса)
- механізм еволюціїякий створив людський мозок (за книгою Дарвіна «Походження видів»)

Відповідь 1 - нейрокомп'ютинг

Відповідь 2 - еволюційні обчислення

ЄС —Поточні потреби

Розробляємо, аналізуємо, застосовуємо
вирішення проблемметоди (алгоритми)
є центральною темою
з математики та інформатики.

чому

- **час**для ретельного аналізу проблеми**зменшується**

- **Складність**поточних проблем**збільшується**

Резюме:

Надійне вирішення проблемпотрібна техніка!

Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- **Типи задач для ЕК**
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням

ЄС —Типи задач

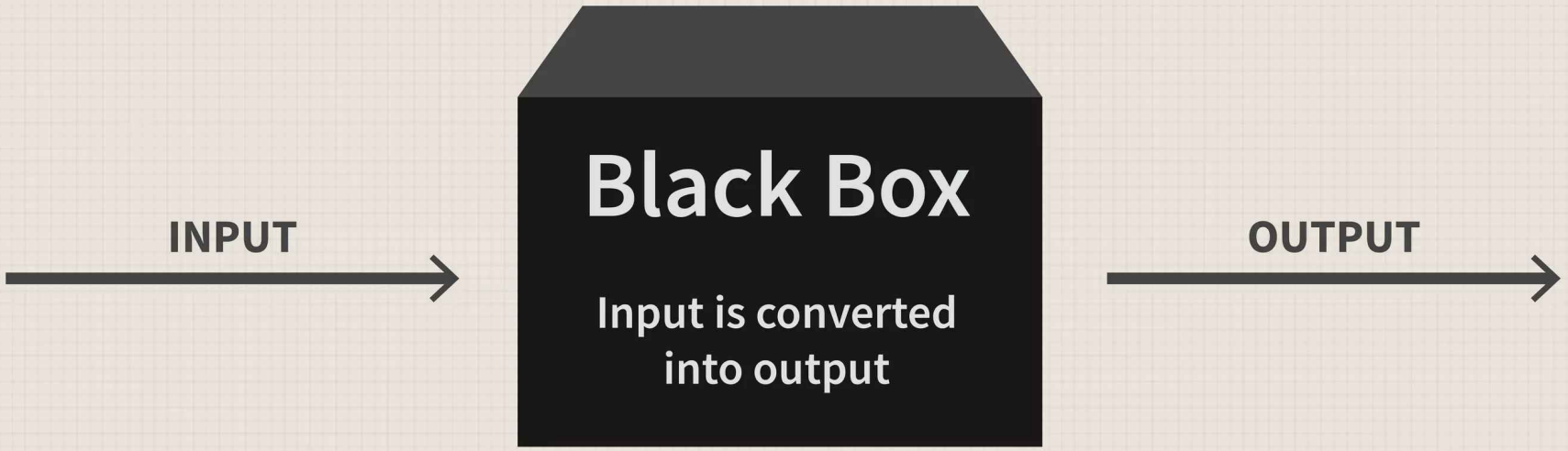
Ми маємо
модель, входи та виходи
нашої системи
і шукайте різні сутності:

- оптимізація,

- моделювання,

- моделювання.

Типи проблем –Оптимізація



Модель відома!

Investopedia **Введення?**

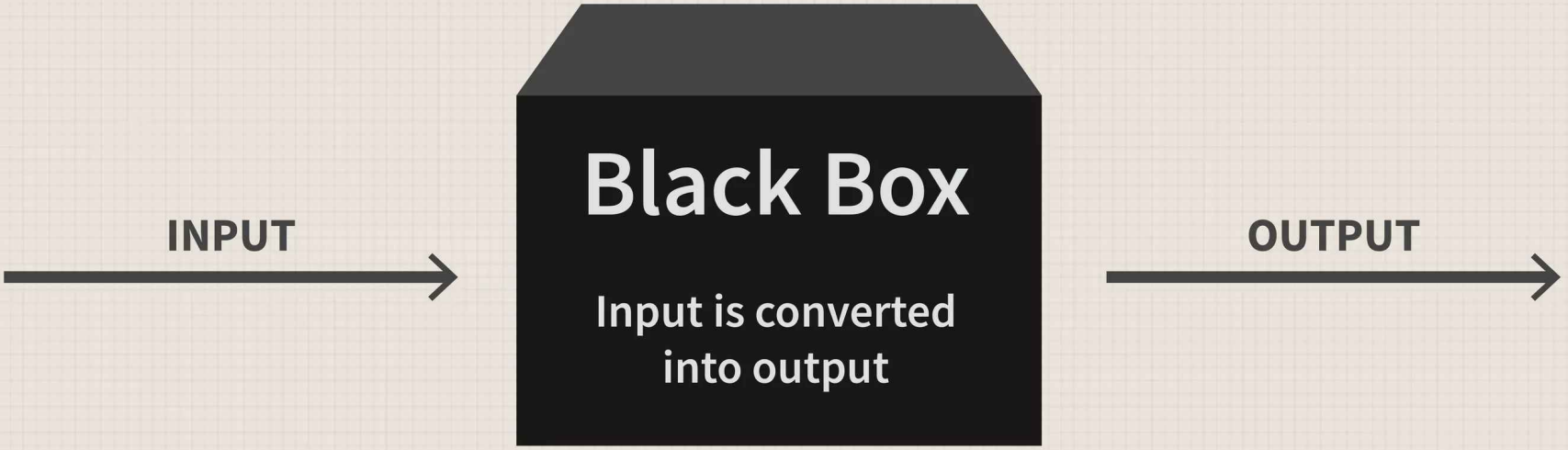
ми**шукати вхідні дані**для досягнення визначеної мети, наприклад:

- розклад КПІ (rozklad.kpi.ua - фантастика!),

- специфікації конструкції програмного/апаратного забезпечення,

- тощо

Типи проблем –Моделювання



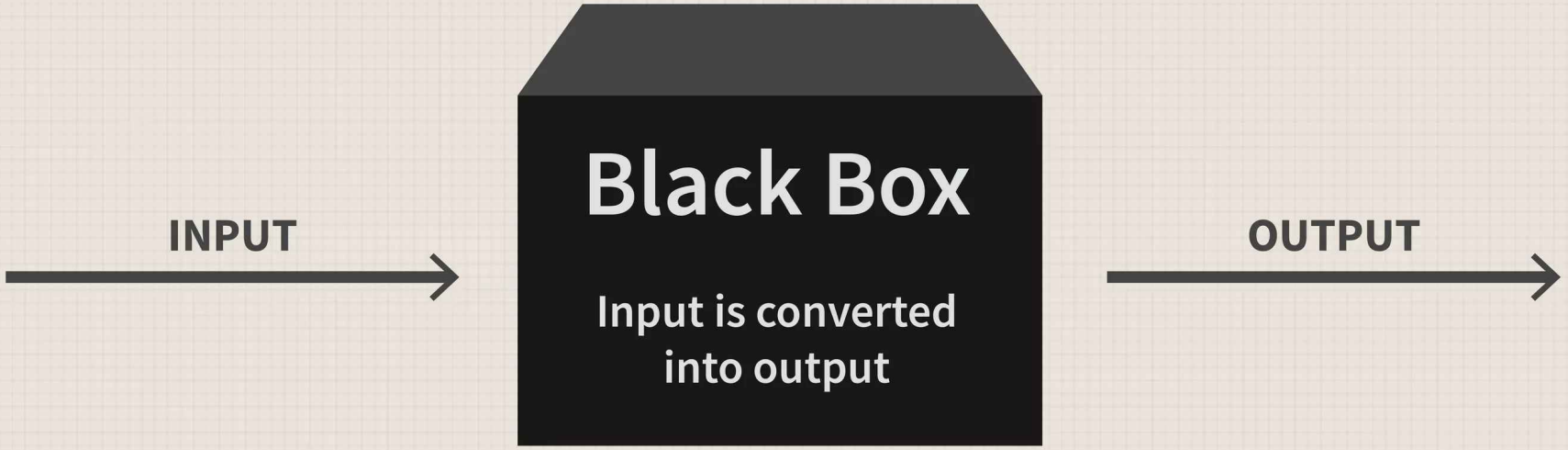
Вхід відомий! **Модель?** **Вихід відомий!**

Investopedia

Модельповинен доставити**правильний вихід**для кожного**відомий вхід**, наприклад:

- моделі машинного навчання,
- моделі глибокого навчання.

Типи проблем –Симуляція



Вхід відомий! Модель відома! **Вихід?**

Investopedia

Це звикло**досліджувати сценарії**в**розвивається**динамічне середовище:

- еволюційна економіка,
- геополітика,
- військове планування,

Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- **Що таке еволюційний алгоритм (ЕА)**
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням

Знову:

ЩоЄК— Метафора (природа-ІТ)

ЕВОЛЮЦІЯ		**ІТ**
		–
Навколишнє середовище	⟷	проблема
Індивідуальний	⟷	Кандидат Рішення (Індивідуальна)
Фітнес	⟷	Якість

Фітнес - шанси на виживання і відтворення

якість - шанс для посів нові рішення

Що**Еволюційні алгоритми (EA)**—

Метафора (природа-IT)

EAs — це категорія**«створити та протестувати"**алгоритми.

Вони стохастичні,**популяційний**алгоритми.

Варіація(генетична?)**оператори**(рекомбінація та мутація)**створити**необхідне**різноманітність**і тим самим сприяти**новизна**.

Вибір**зменшує(!) різноманітність**
і
дієяк**якість штовхання сили**.

EA —Історія та види

Різні типи EA асоціюються з різними уявлення:

- **Двійковий**рядки : Генетичні алгоритми (**GA**)
- **Справжня**-значні вектори : Стратегії еволюції (**ES**)
- **Кінцевий стан**Машини: еволюційне програмування (**EP**)
- LISP**дерева**: Генетичне програмування (**GP**)

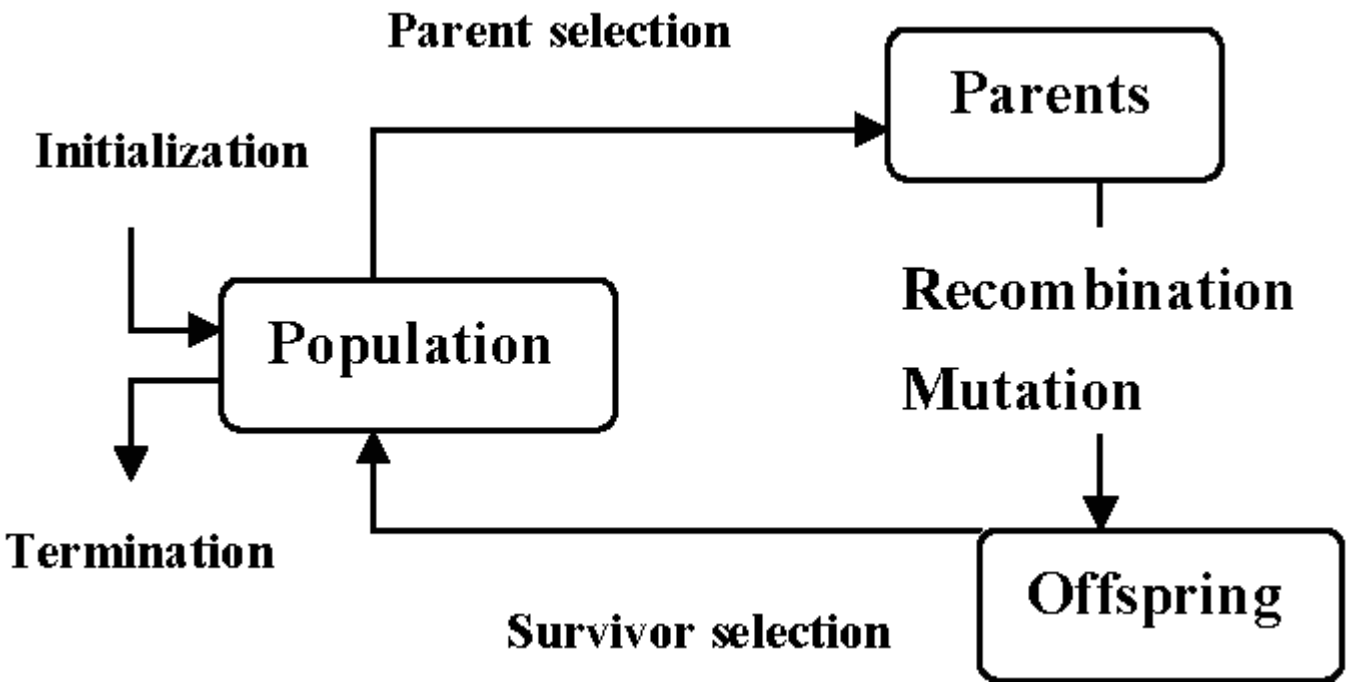
- Ці відмінності здебільшого не мають значення, найкраща стратегія
- вибрати**представництво**пасувати**проблема**
- вибрати**оператори варіації**пасувати**представництво**

- **Вибір**оператори**використовуйте тільки фітнес**
- і так
- є**незалежно від представництва**.

Зміст

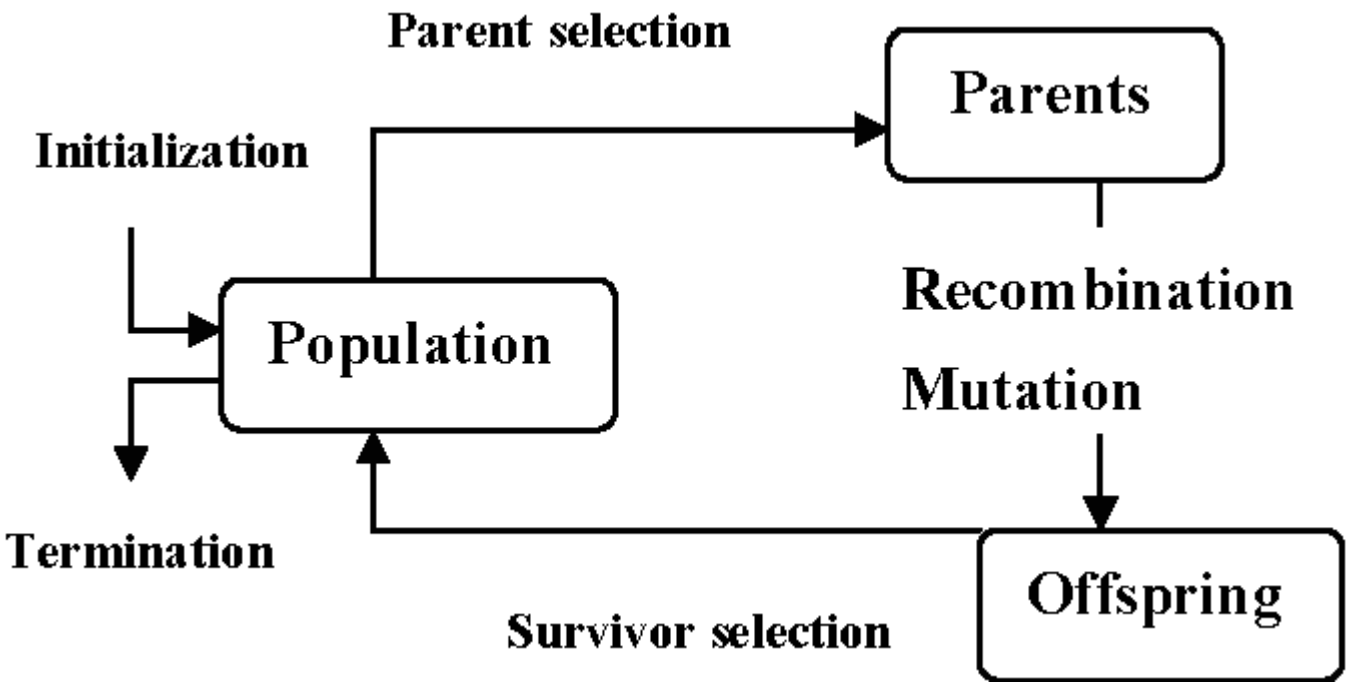
- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням

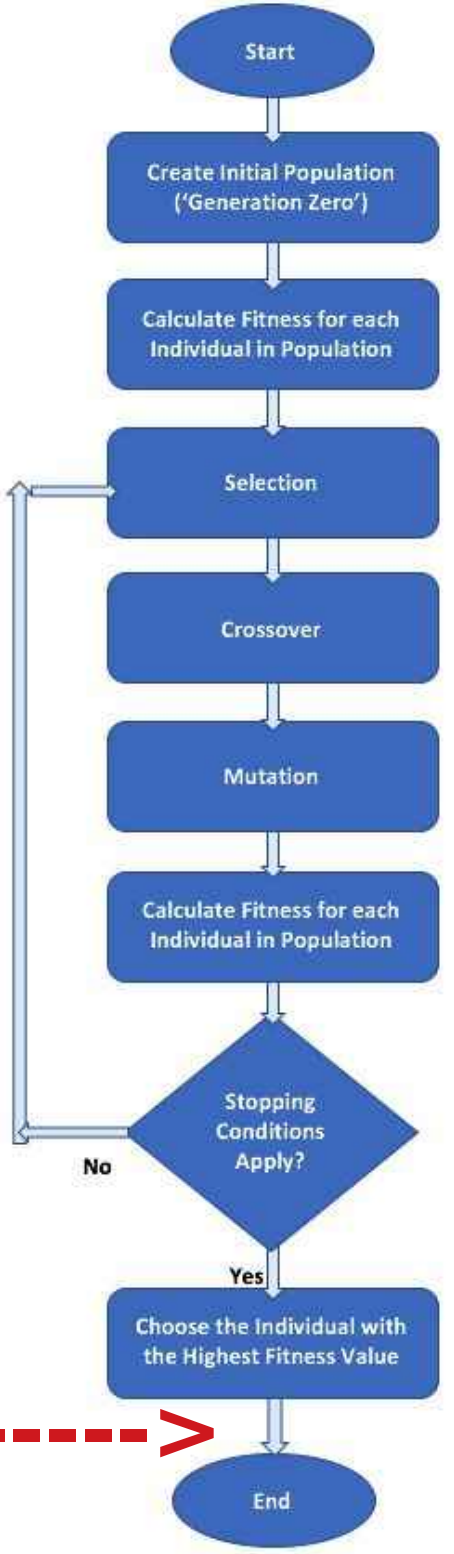
EA —Загальна схема...



. . . і...

EA — Загальна схема...





. . . і робочий процес ---------->

EA — Робочий процес —Термінологія

Кандидати рішень (**особи**) існує в**фенотип простір**.

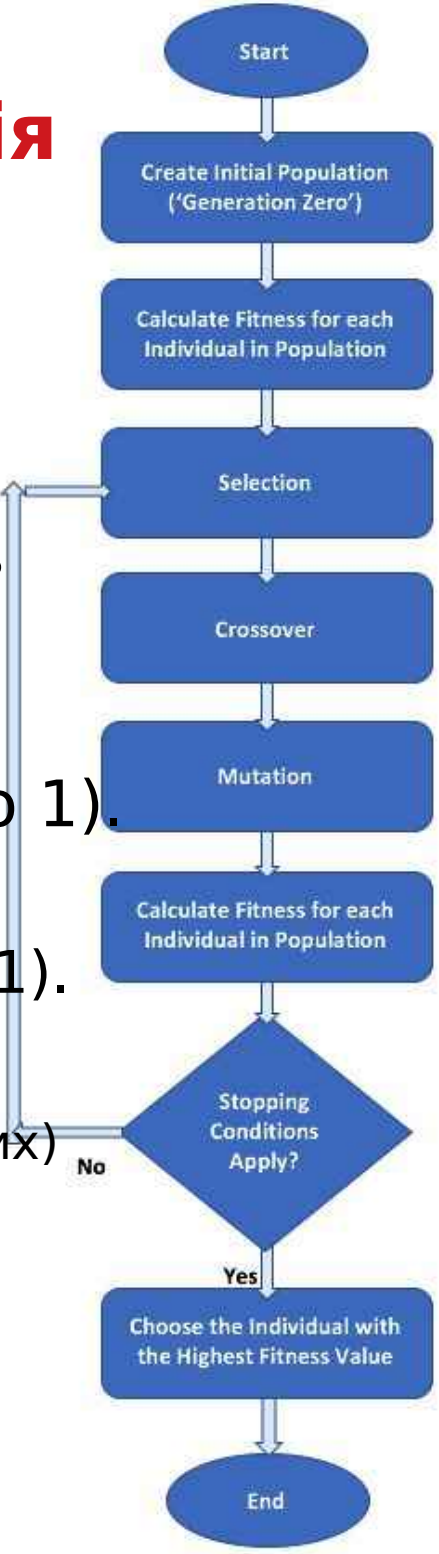
Вони є**закодований**в**хромосоми**, які існують в **простір генотипу**.

Кодування: фенотип->генотип (не завжди 1 до 1).

Розшифровка: генотип->фенотип (має бути 1 до 1).

Хромосомимістять**гени**, які знаходяться в (зазвичай фіксованих) положеннях, які називаються**локуси**(співати.**локус**) і мають а значення (алель).

Щоб знайти**глобальний оптимум**кожне можливе рішення має бути представленим у просторі генотипу!



EA — Робочий процес —Населення

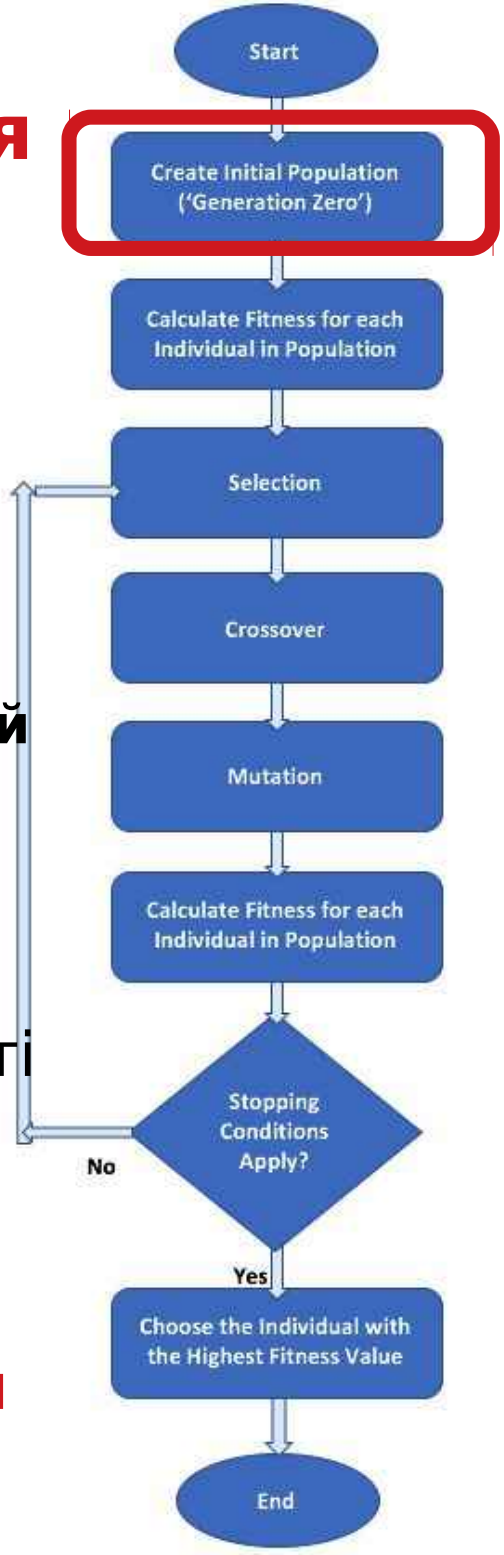
Має (представлення)**можливі рішення.**

Зазвичай має а**фіксований розмір**і є а**мульти-набір генотипи.**

Деякі складні EA також стверджують а**просторовий структура**на населення, наприклад, сітка.

Оператори виборупрацювати з**все населення**тобто репродуктивні ймовірності відносяться до поточного покоління.

різноманітністьнаселення відноситься до кількості **різні фітнес/фенотипи/генотипи** присутній (зверніть увагу, не те саме).



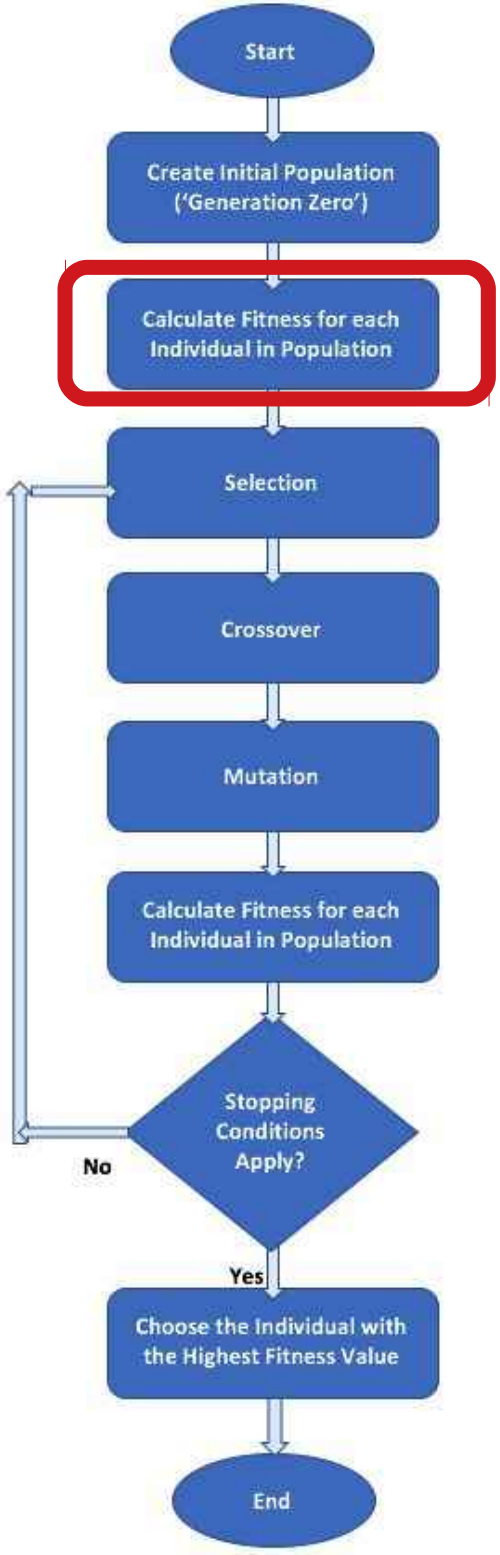
EA — Робочий процес —Фітнес

Представляє вимоги, що **населення** повинен**адаптуватися**деяким**критерії**люблю **якість**функція або**об'єктивний**функція.

ПризначаєОдинокий**реальна вартість**d придатність для кожного **фенотип**який утворює**основа для вибір.**

Отже**більше різноманітності**(різні значення) **кращий.**

Типово**фітнес**передбачається**розгорнутий, але**...деякі проблеми можна сформулювати як **мінімізація**проблеми.



EA — Робочий процес —Вибір

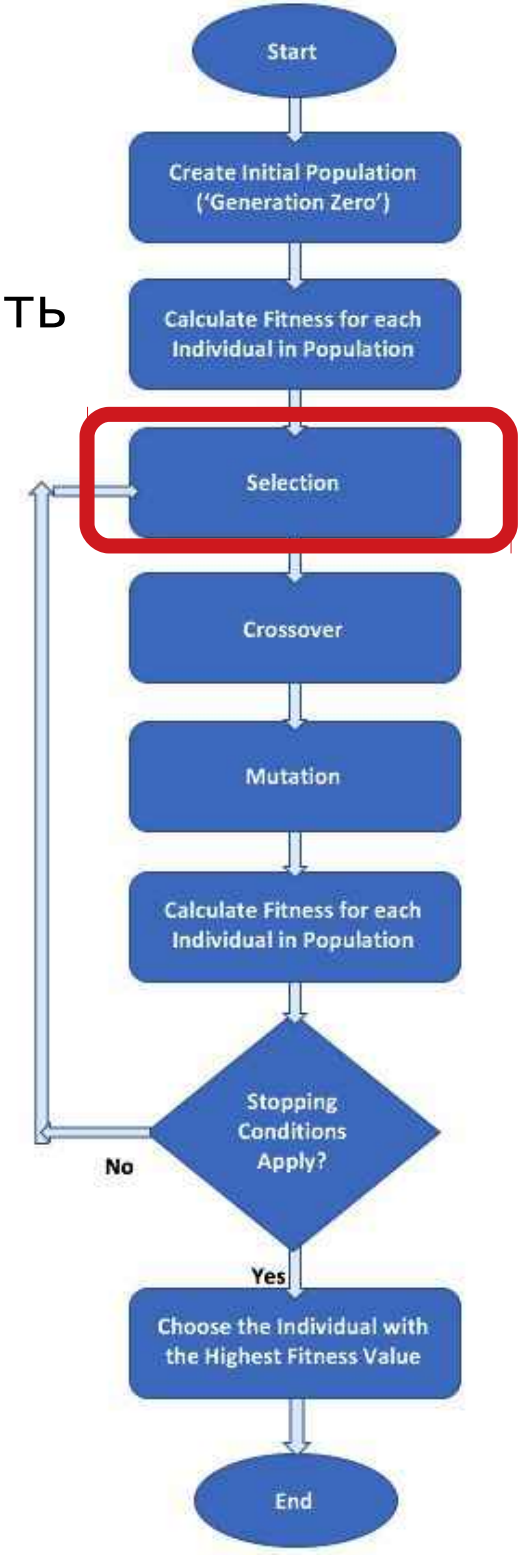
Призначаєзмінна**ймовірності**осіб, які виступають батьками залежно від їх фізичної підготовки.

Зазвичай**імовірнісний:**

вищеякісні рішення**більше**швидше за все стане батьки ніж**нижче**якість, але ... ні гарантований.

Навіть**найгірше**у поточній популяції зазвичай має **ненульова ймовірність**стати батьком.

Це**стохастичний**природи може**допомога**втекти від**локальні оптимуми**!



EA — Робочий процес —

Варіаційні оператори

The**Головна мета**є

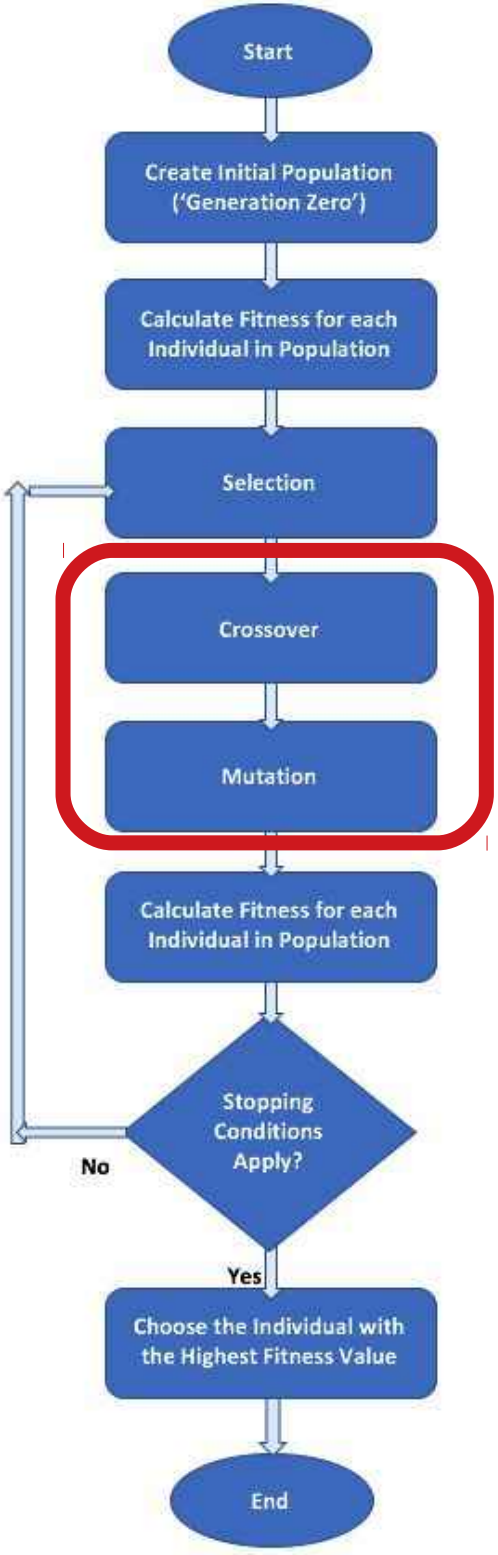
до**генерувати**новий**варіанти рішень**.

Зазвичай поділяють на типи відповідно до їх **арності** (кількість входів):

- arity = 1 -> оператори мутації
- арність > 1 -> оператори рекомбінації
- arity = 2 -> оператори кросоверу

Відносне значення рекомбінації і мутація зараз інтенсивно обговорюється, але більшість EA використовують обидва.

Вибір конкретних операторів варіації є



Робочий процес -Варіаційні оператори

- Кросовер

Кросоверабо**Рекомбінація**

Зливаєтьсяінформація від**батьки**в **потомство**.

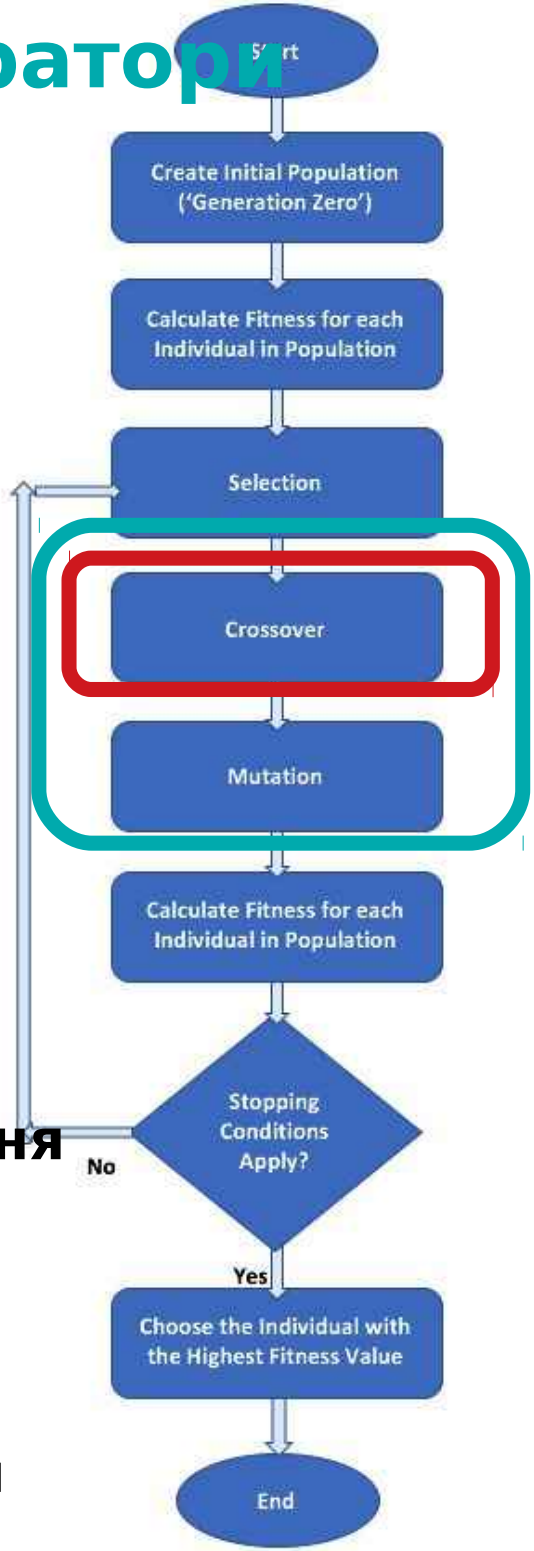
Вибіряку інформацію потрібно об’єднати **стохастичний**.

більшість**потомство**може бути**гірше**або**те саме**як в**батьки**.

Гіпотеза: деякі можуть бути**краще**за**комбінування елементів**генотипів, що**привести до добра**риси.

Метафора з природи:

це було**успішно використовується**селекціонерами рослин і худоби!



Робочий процес -Варіаційні оператори-

Мутація

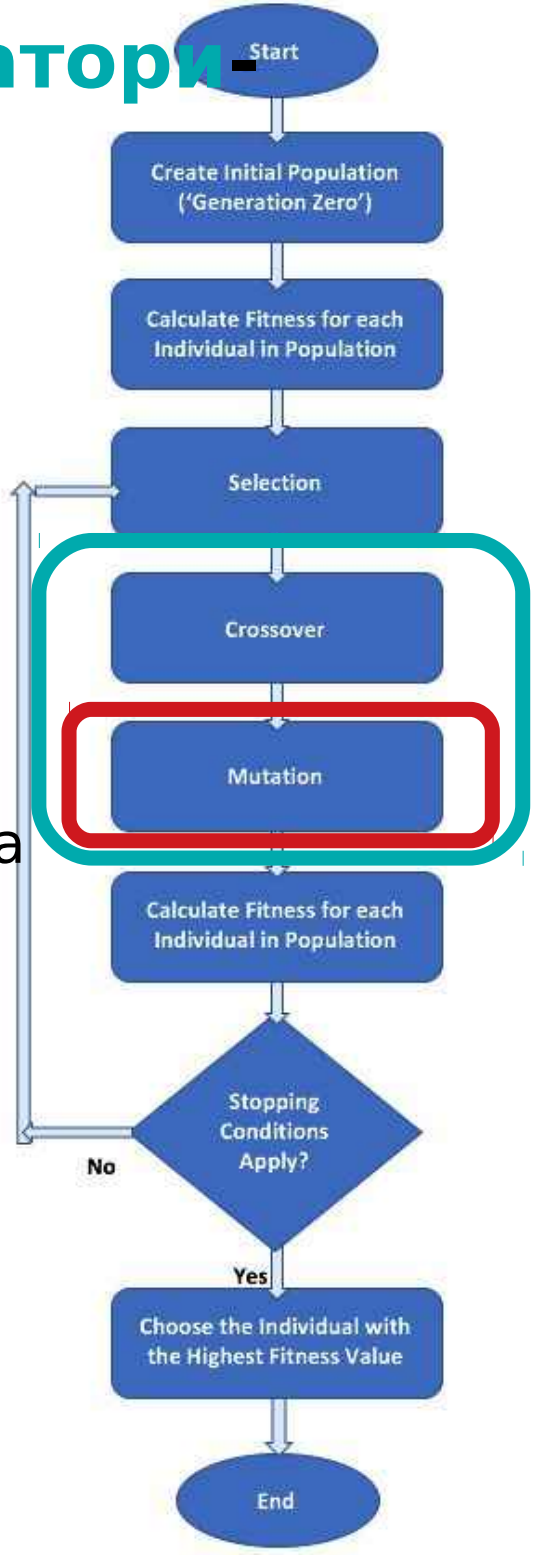
Діє на один генотип і**доставляє інший**.

Елемент**випадковість**є важливим і відрізняє його від інших унарних евристичних операторів.

Це залежить від репрезентації та діалекту:

- **Двійкові GA**–фоновий оператор, відповідальний за збереження та впровадження різноманітності,
- **EP**для FSM/безперервних змінних – тільки пошук оператор,
- **GP**–майже не використовується.

Може**гарантія**зв'язність простору пошуку і отже**конвергенція**докази.



EA — Робочий процес —Пуск/Зупинка

старт

Ініціалізаціязазвичай робиться при**випадковий.**

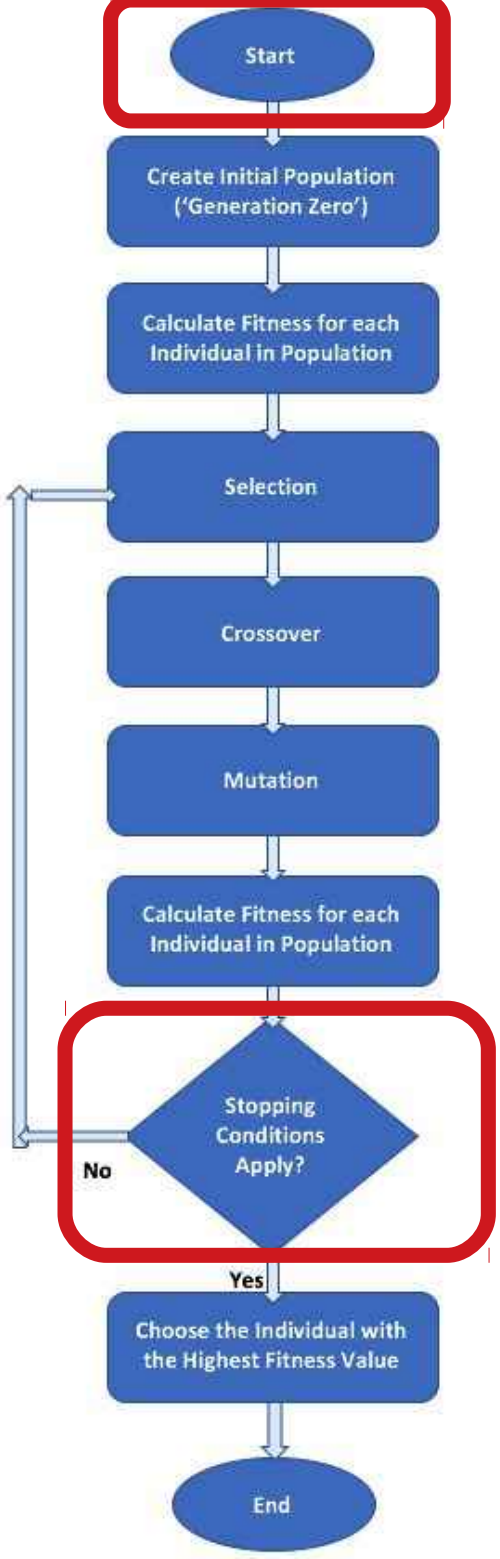
Воно повинно бути рівномірним і змішаним по можливості значення алелей.

Він може включати**існуючі**рішення, або використовуйте problemspecific**евристики**, до “**насіння**” населення (догляд треба брати!)

СТОП

Умова розірванняперевірено кожне покоління:

- деякі**планується**(відомо/припускається)**фітнес**,
- якийсь максимально допустимий**кількість поколінь**,
- деякі**мінімум**рівень**різноманітність**,
- деякі уточнювали**кількість поколінь без** фітнес**поліпшення**.



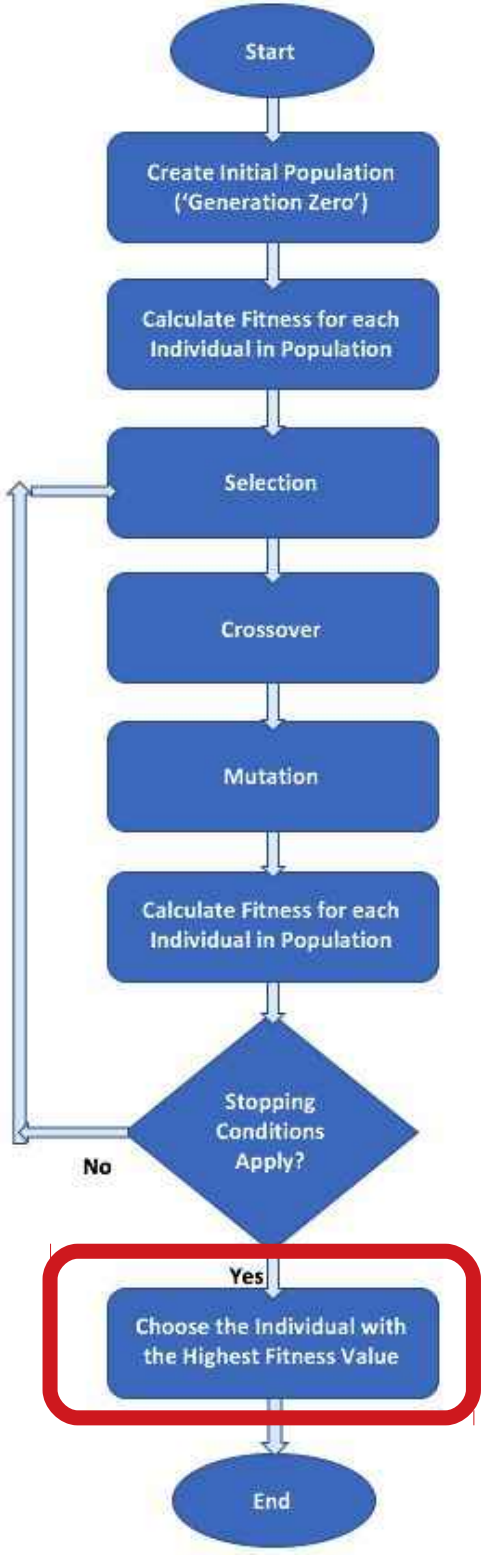
EA — Робочий процес —Кінець

Виберіть

в**індивідуальний**

з

в**найвища пристосованість**значення.



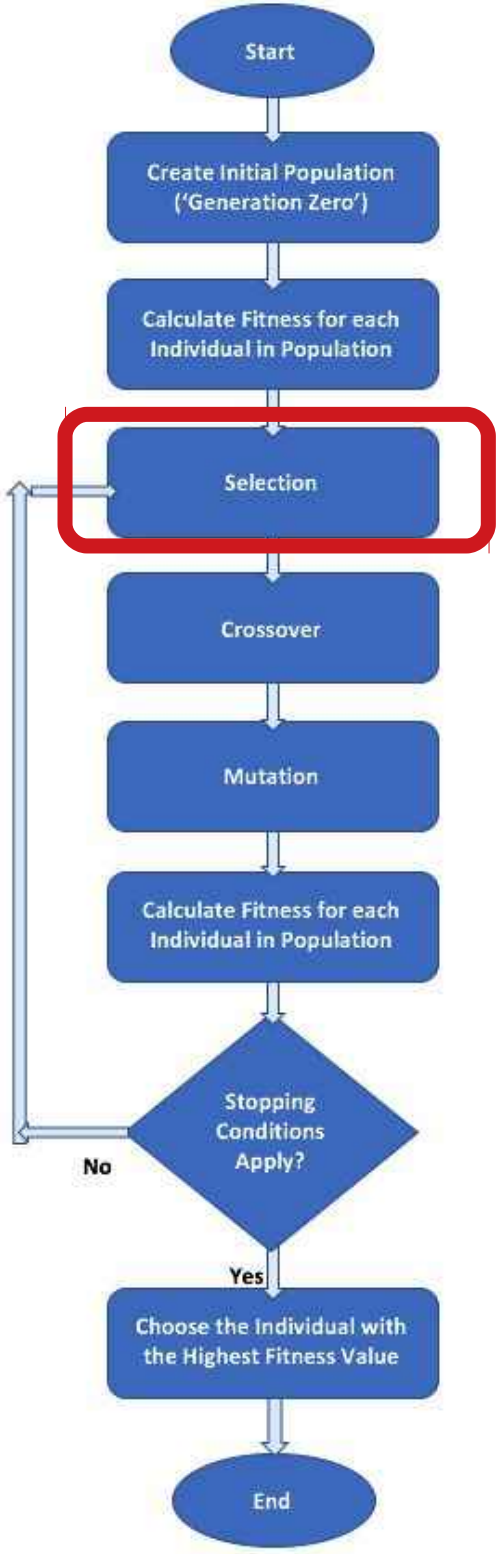
Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- **Відбір — подробиці зараз**
- Кросовер
- Мутація
- ЕА з реальним кодуванням

EA — Робочий процес —

Методи відбору

- Вибір колеса рулетки (пропорційний вибір фітнесу — FPS)
- Стохастична універсальна вибірка (SUS)
- Відбір на основі рангу
- Вибір турніру



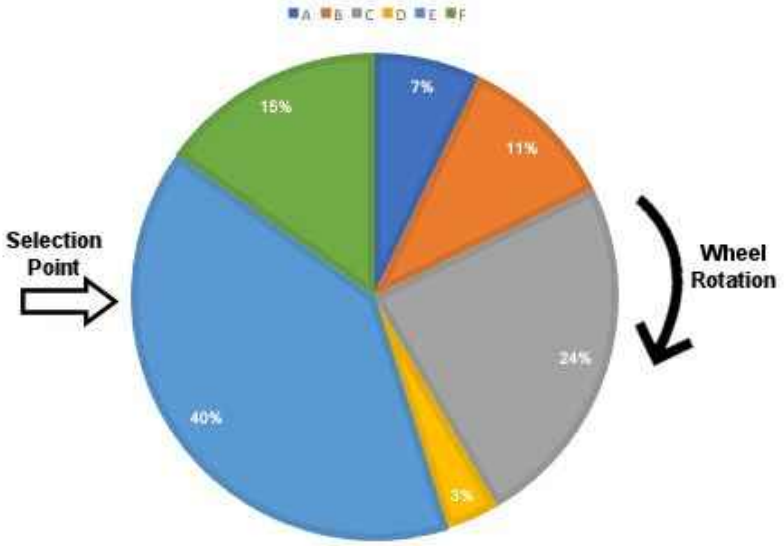
Робочий процес —

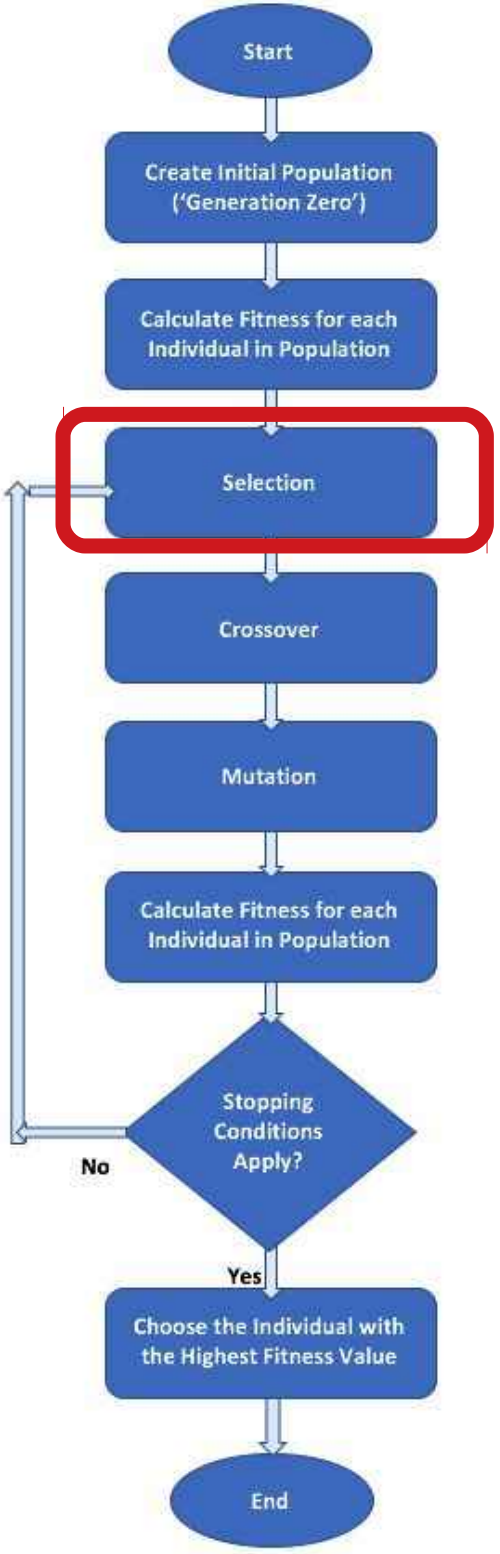
Вибір колеса рулетки

fitness value. This is comparable to using a roulette wheel in a casino and assigning each individual a portion of the wheel proportional to its fitness value. When the wheel is turned, the odds of each individual being selected are proportional to the size of the portion of the wheel that it occupies.

For example, suppose we have a population of six individuals with fitness values as shown in the following table. The relative portion of the roulette wheel dedicated to each individual is calculated based on these fitness values:

Individual	Fitness	Relative portion
A	8	7%
B	12	11%
C	27	24%
D	4	3%
E	45	40%
F	17	15%





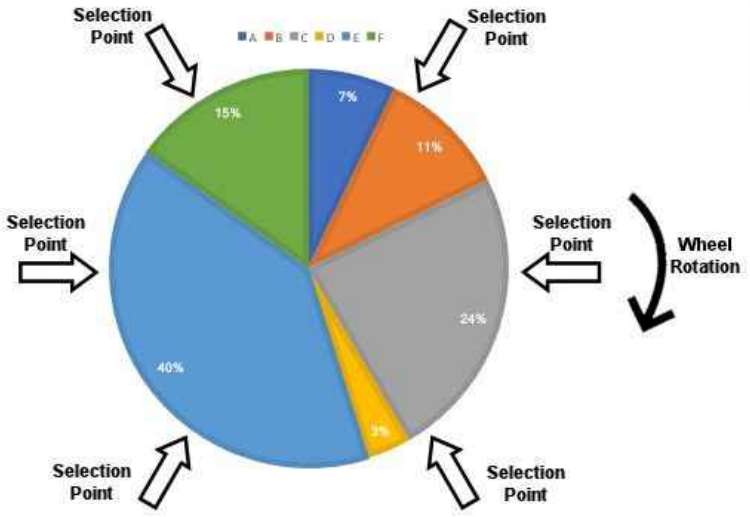
Робочий процес —

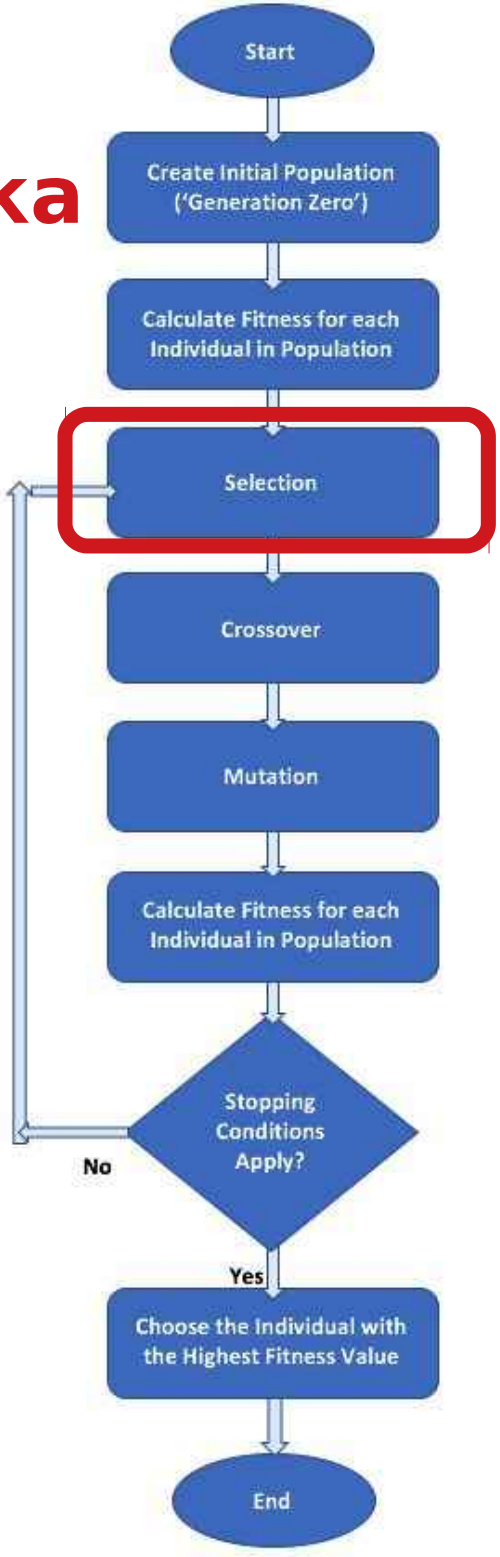
Стохастична універсальна вибірка

fitness value. This is comparable to using a roulette wheel in a casino and assigning each individual a portion of the wheel proportional to its fitness value. When the wheel is turned, the odds of each individual being selected are proportional to the size of the portion of the wheel that it occupies.

For example, suppose we have a population of six individuals with fitness values as shown in the following table. The relative portion of the roulette wheel dedicated to each individual is calculated based on these fitness values:

Individual	Fitness	Relative portion
A	8	7%
B	12	11%
C	27	24%
D	4	3%
E	45	40%
F	17	15%





Робочий процес —

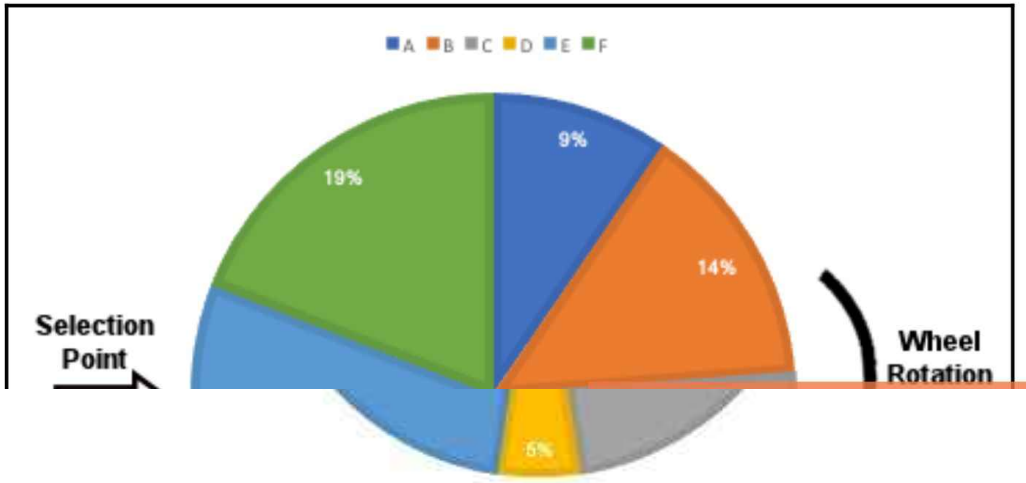
Відбір на основі рангу

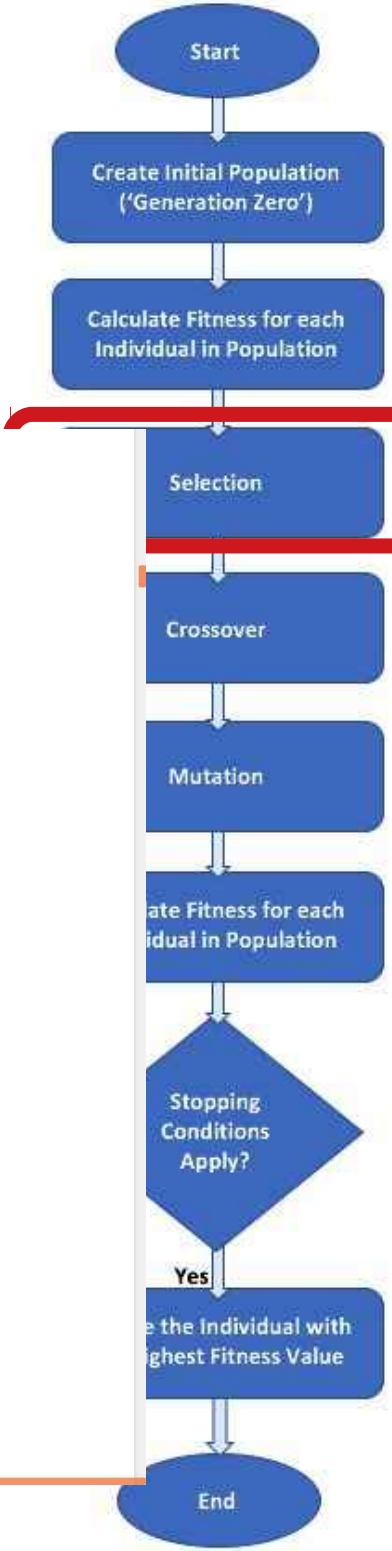
The**фітнес**використовуєтьсяо**сортувати**особи: кожен особі надається а**ранг**для свого**положення**і

our example is six, the highest-ranking individual gets the rank value of 6, the next one gets the rank value of 5, and so on. The relative portion of the roulette wheel dedicated to each individual is now calculated based on these rank values instead of using the fitness values:

Individual	Fitness	Rank	Relative portion
A	8	2	9%
B	12	3	14%
C	27	5	24%
D	4	1	5%
E	45	6	29%
F	17	4	19%

The matching roulette wheel is depicted in the following diagram:





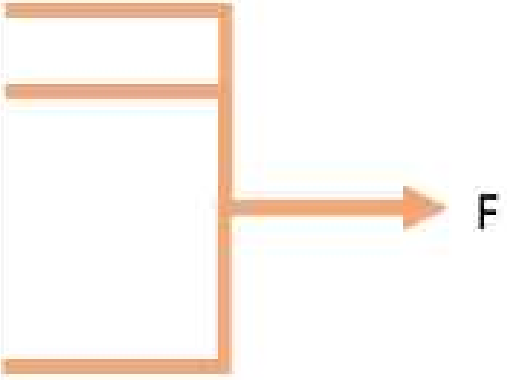
Робочий процес —

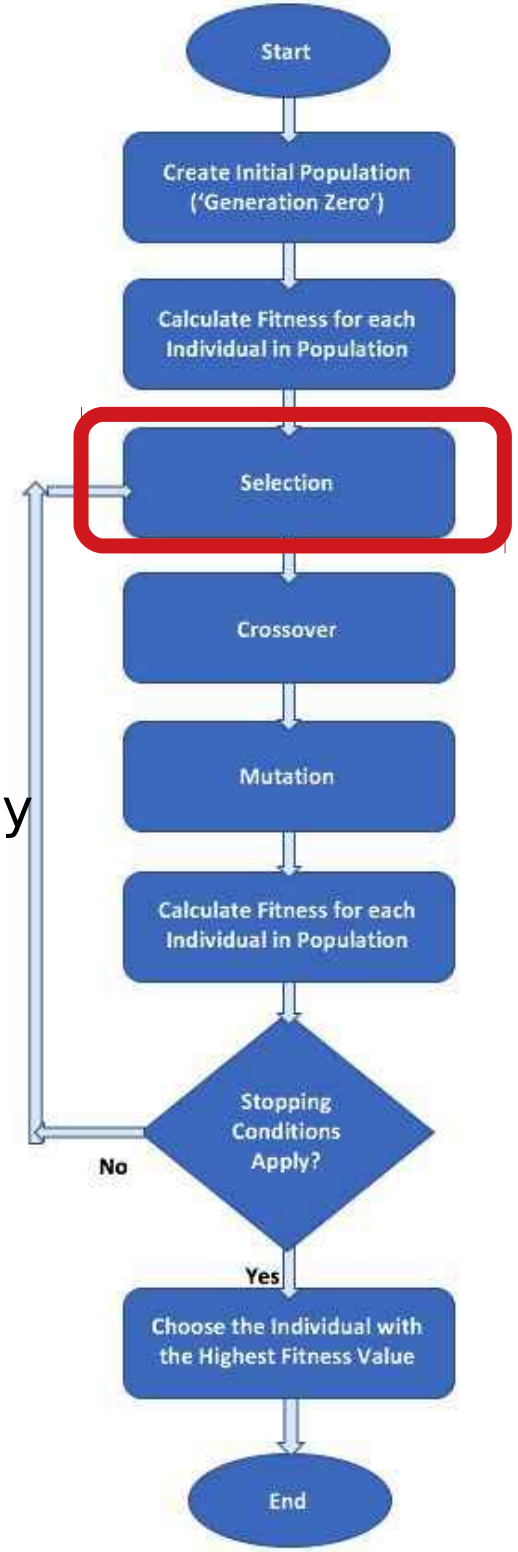
Відбір турніру

У кожному раунді турніру відбірний метод два або більше**особи**є випадковим чином **зібрано**від населення, а в**один**з в**найвищий**фітнес оцінка**виграє**і отримує **вибрано**.

Відповідним чином називається кількість осіб, які беруть участь у кожному відбірковому раунді турніру (три на цьому малюнку). The**більший** **розмір турніру** розмір турніру,**вище**шанси що**найкращий**особи будуть**бути обраним**.

Individual	Fitness
A	8
B	12
C	27
D	4
E	45
F	17





Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- **Кросовер — подробиці зараз**
- Мутація
- ЕА з реальним кодуванням

Робочий процес -Варіаційні оператори

- Кросовер

Кросоверабо**Рекомбінація**

Зливаєтьсяінформація від**батьки**в **потомство**.

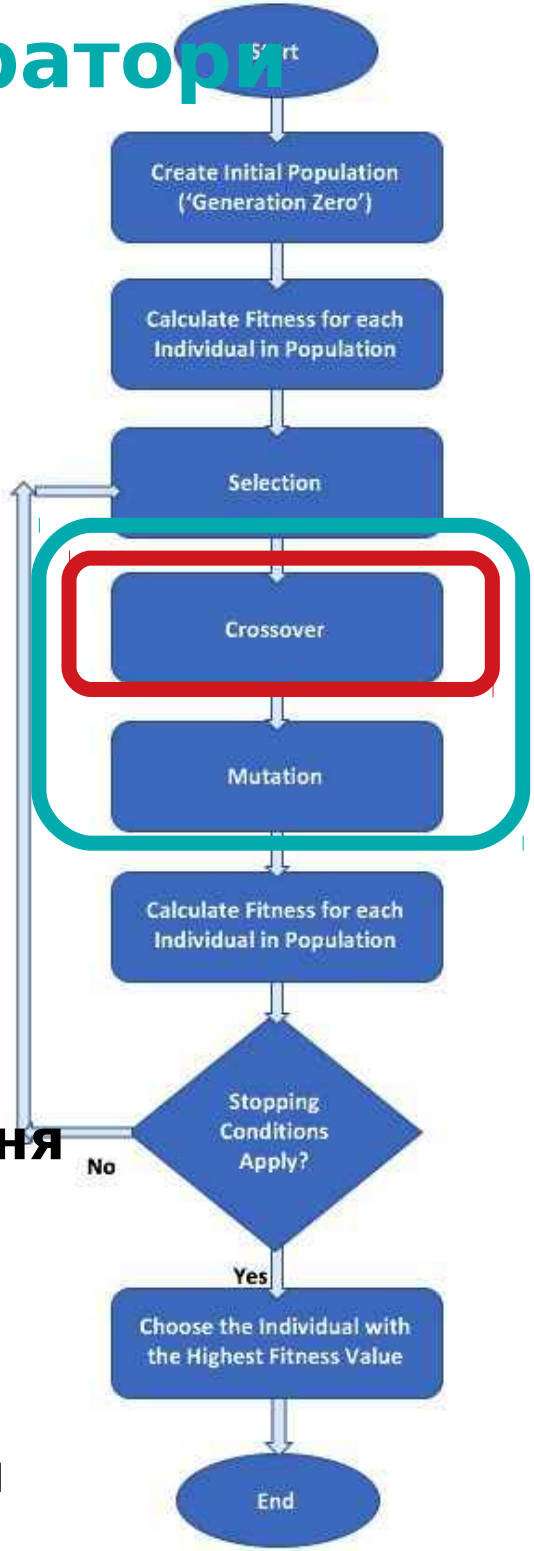
Вибіряку інформацію потрібно об’єднати **стохастичний**.

більшість**потомство**може бути**гірше**або**те саме**як в**батьки**.

Гіпотеза: деякі можуть бути**краще**за**комбінування елементів**генотипів, що**привести до добра**риси.

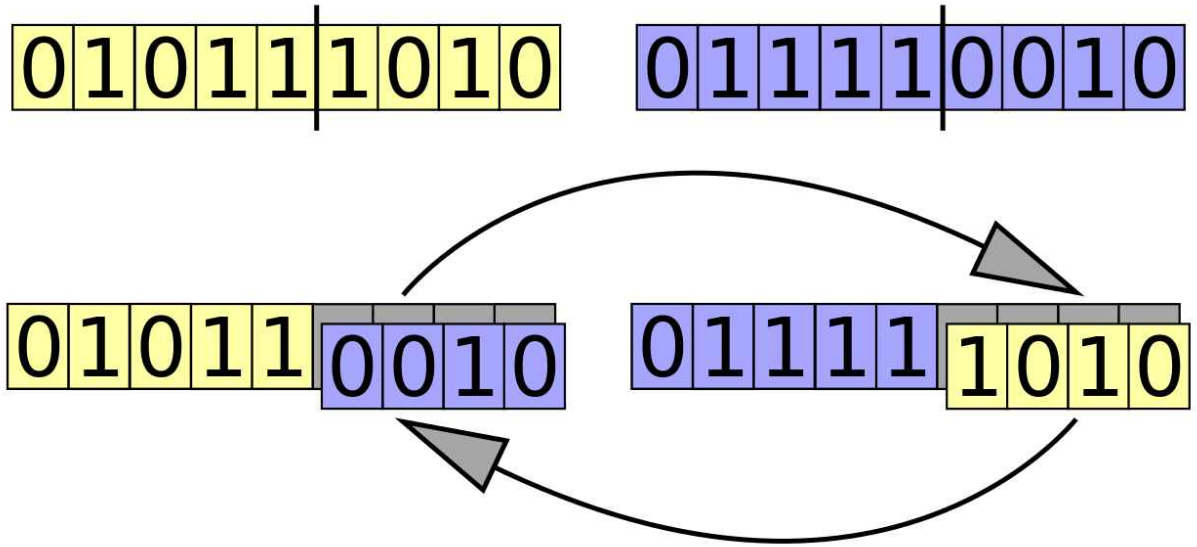
Метафора з природи:

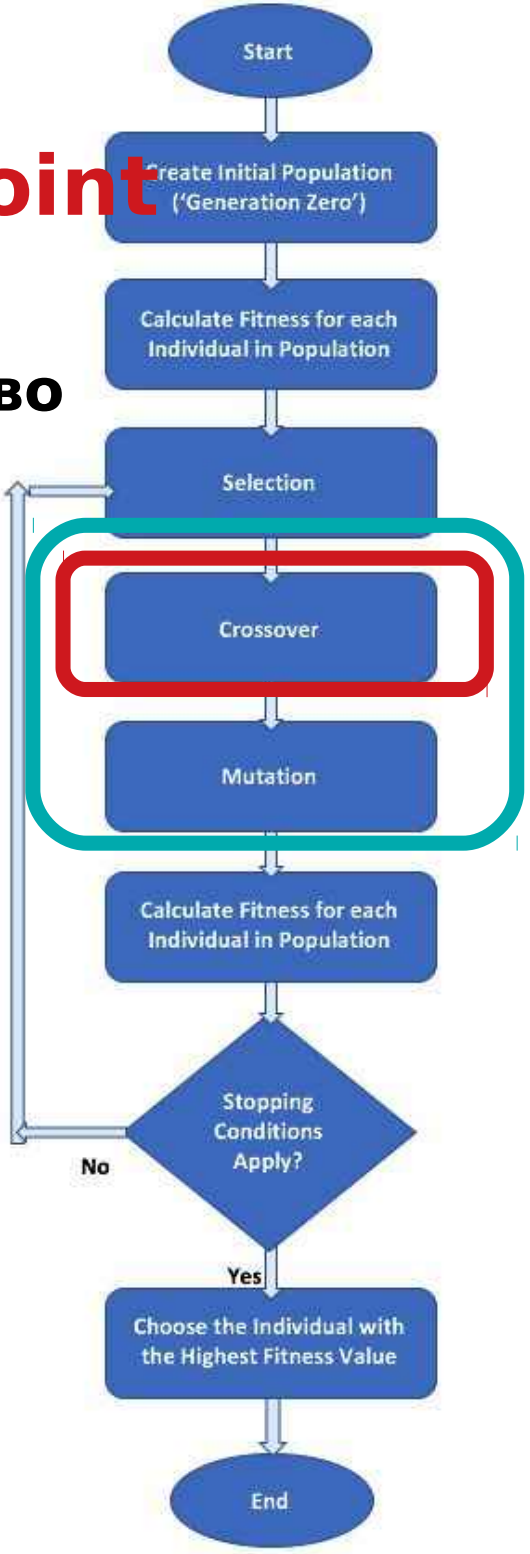
це було**успішно використовується**селекціонерами рослин і худоби!



Робочий процес -Варіаційні оператори- Кросовер - Sinlge-point

The**точка перетину**(або**точка зрізу**) на хромосомах обох батьків є**вибрано випадково** .Гени праворуч від цієї точки міняються між двома батьківськими хромосомами. В результаті ми**отримати двох нащадків**, де кожен їх**нести**деяка генетична інформація**від обидва**батьки.



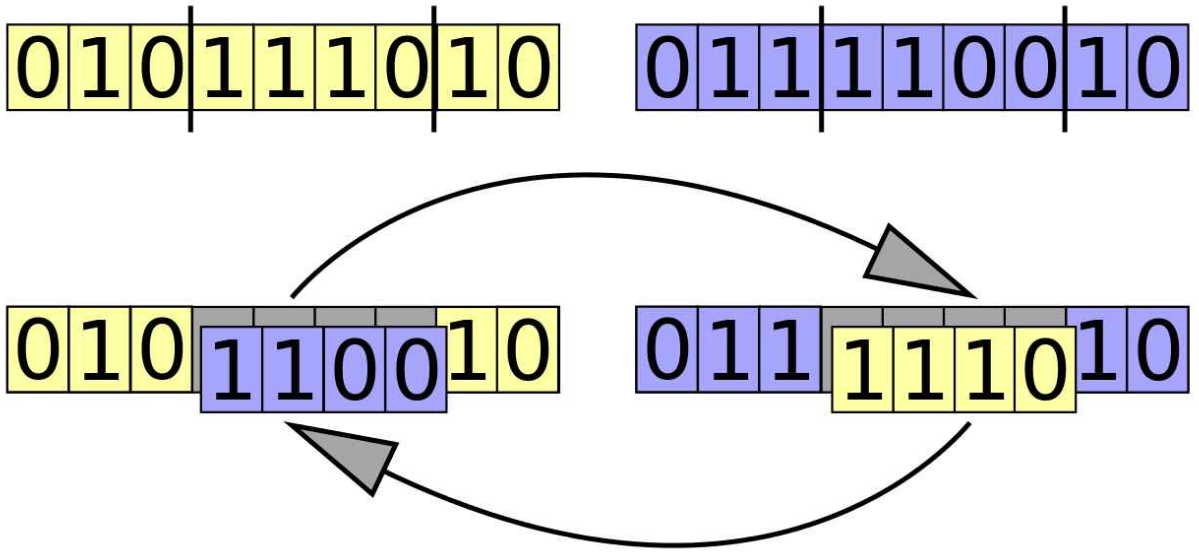


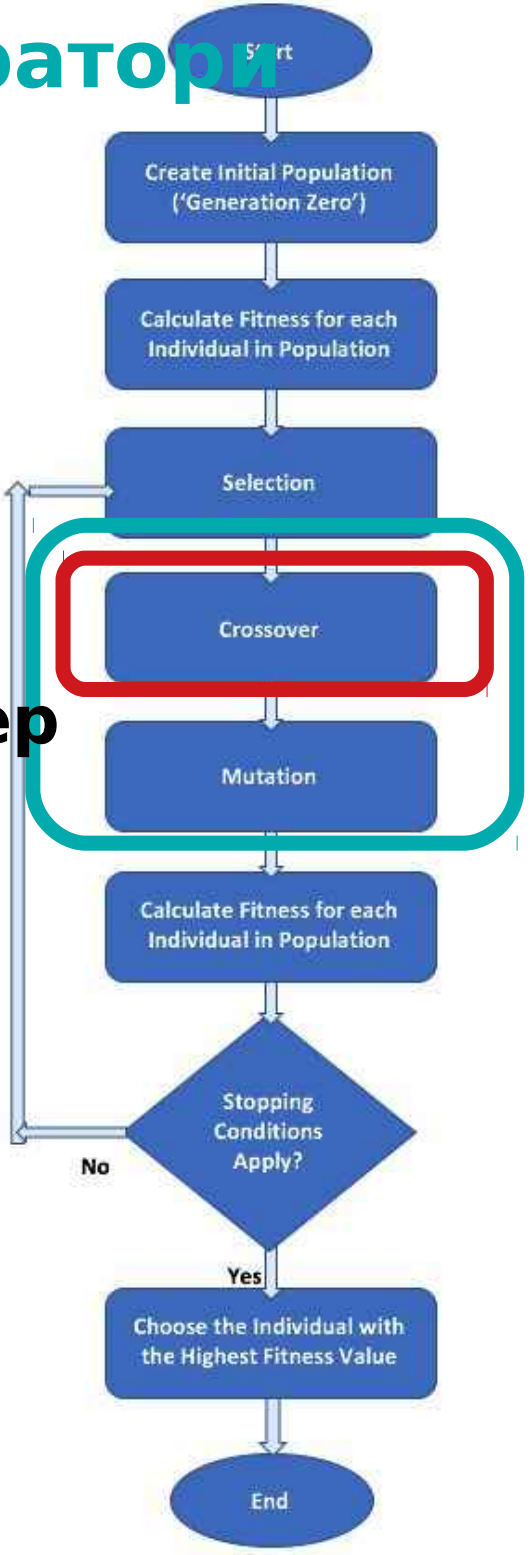
Робочий процес - Варіаційні оператори

- Кросовер K точка

Наприклад, при 2-точковому кросинговері вибирають 2 точки на хромосомах обох батьків випадковим чином. Гени, що знаходяться між цими точками, обмінюються між двома батьками хромосоми.

Узагальненням цього методу є **кросовер kpoint**, де **k** представляє позитив і



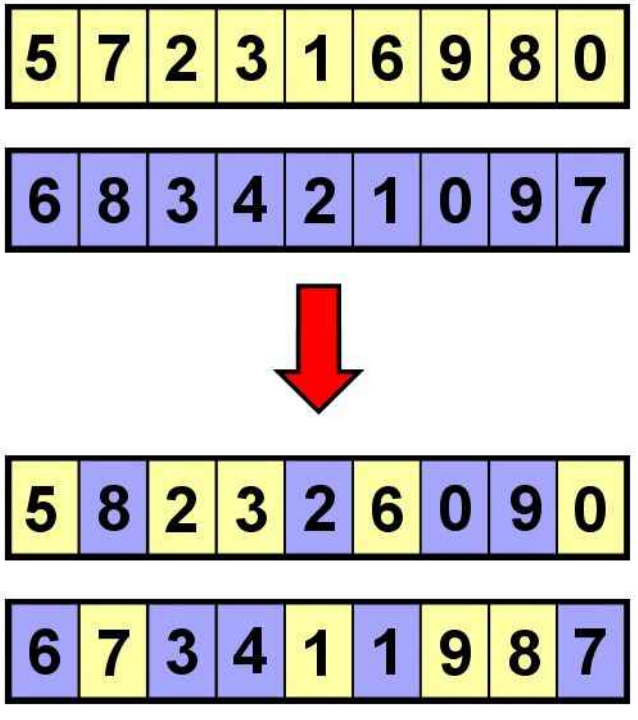


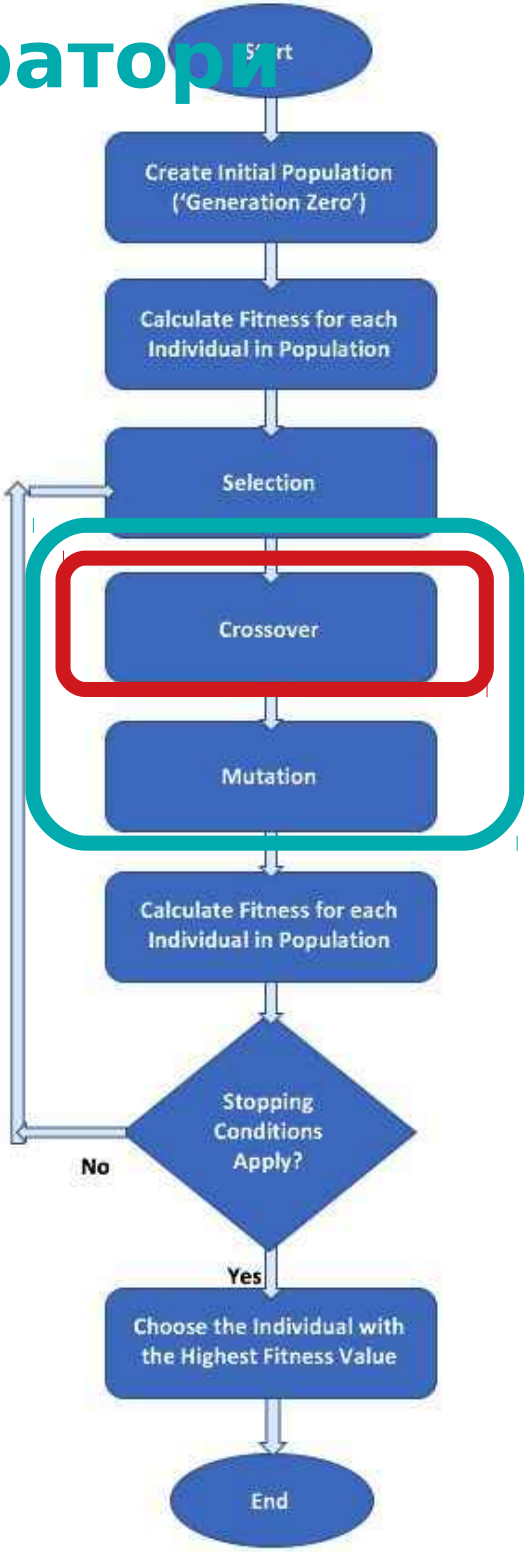
Робочий процес -Варіаційні оператори

- Кросовер - Уніформа

Кожен ген є**самостійно визначається**за **випадковим чином**вибір одного з батьків. Якщо випадковий розподіл становить 50%, кожен із батьків має однакові шанси вплинути на потомство.

ПРИМІТКА: нижче,**на основі цілих чисел**хромосоми показані, але це так**те саме для двійкового**одні.

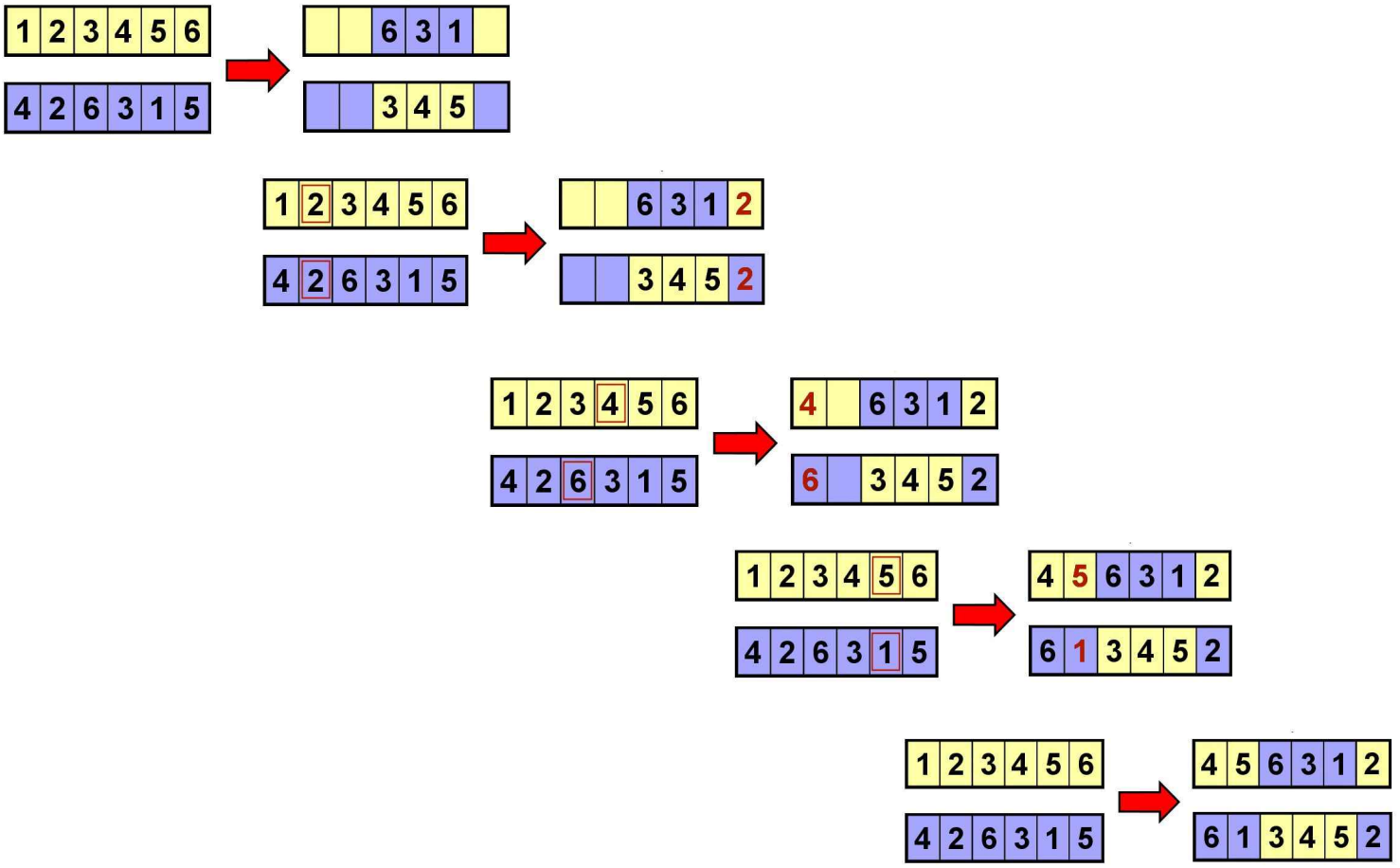


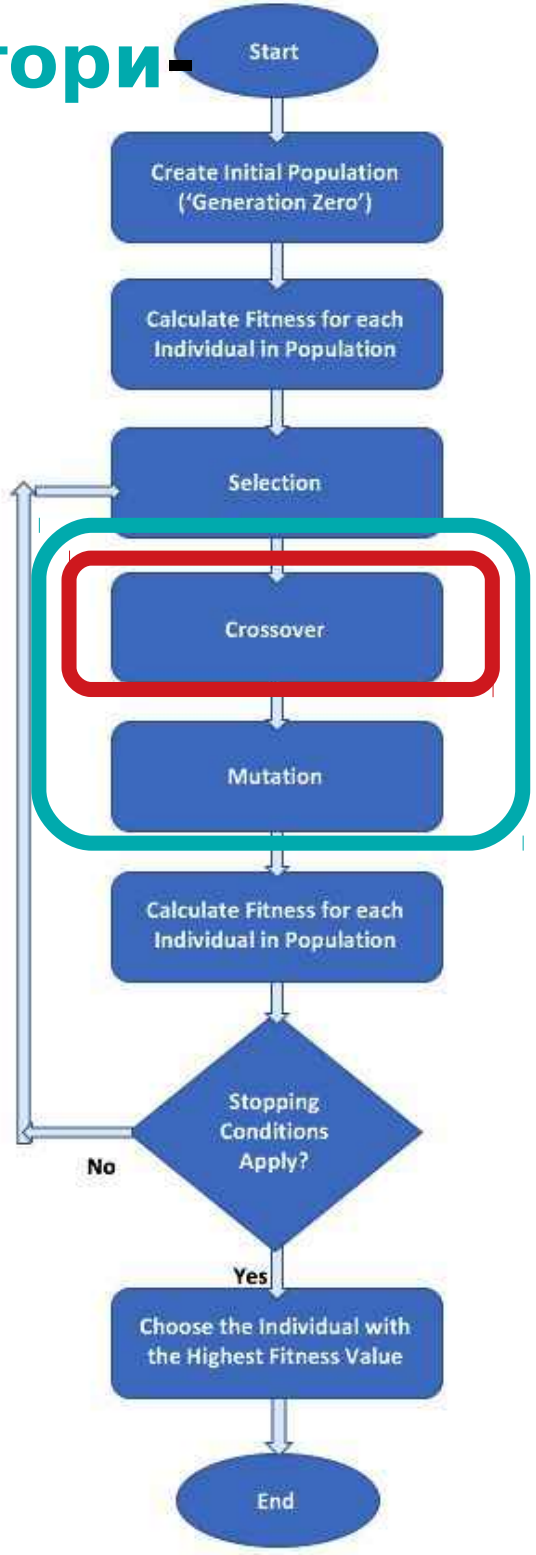


Робочий процес -Варіаційні оператори-

Кросовер — упорядковані списки

The**замовив кросовер**(OX1) метод прагне зберегти відносний порядок батьківського генів якомога більше.





Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- **Мутація — подробиці зараз**
- ЕА з реальним кодуванням

Робочий процес -Варіаційні оператори-

Мутація

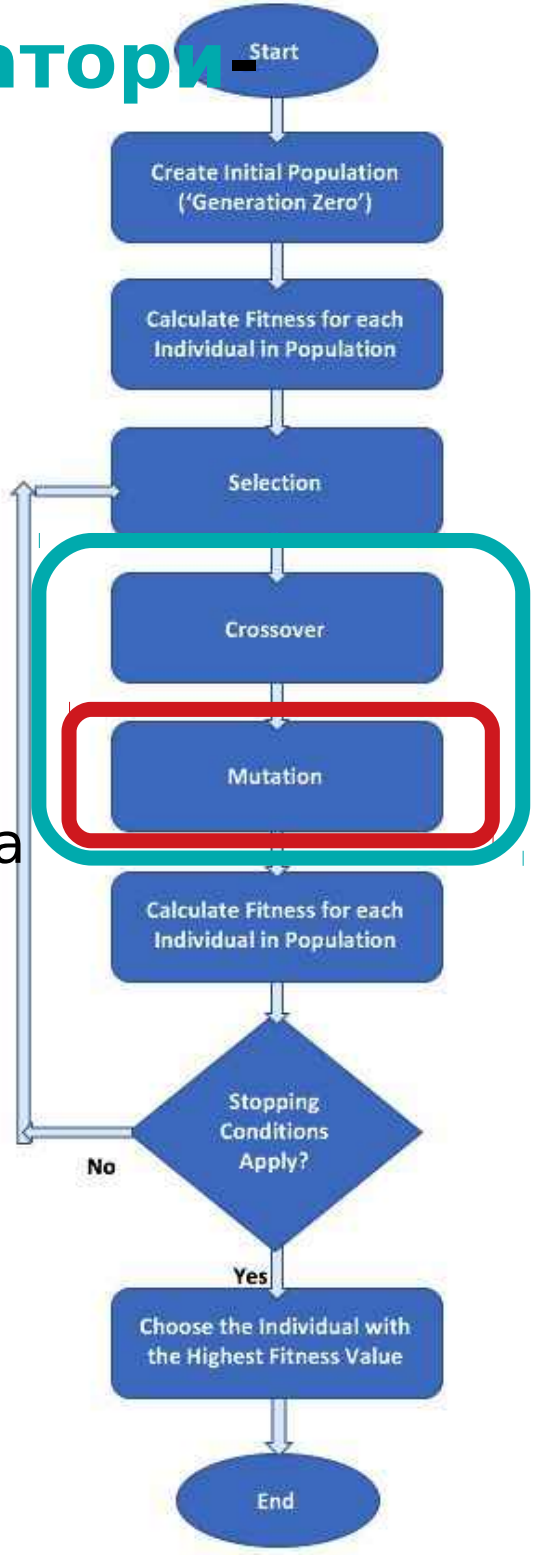
Діє на один генотип і**доставляє інший**.

Елемент**випадковість**є важливим і відрізняє його від інших унарних евристичних операторів.

Це залежить від репрезентації та діалекту:

- **Двійкові GA**–фоновий оператор, відповідальний за збереження та впровадження різноманітності,
- **EP**для FSM/безперервних змінних – тільки пошук оператор,
- **GP**–майже не використовується.

Може**гарантія**зв'язність простору пошуку і отже**конвергенція**докази.

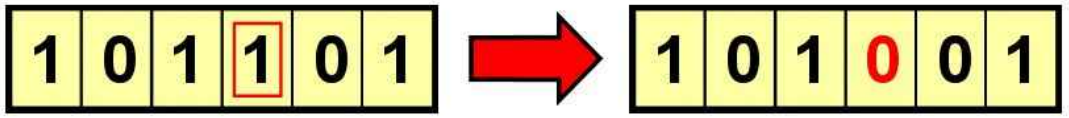


Робочий процес —Мутація - фліп-біт

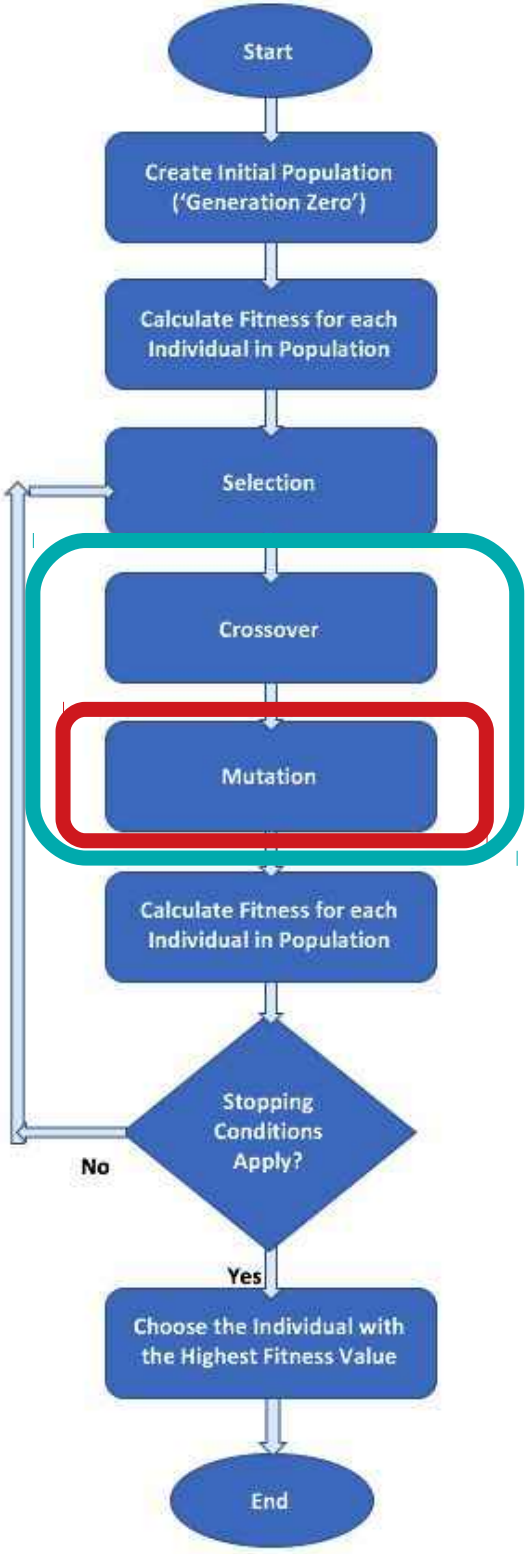
Для бінарної хромосоми,

1 генєвипадковим чином

вибраної його цінність**перевернуто**(доповнюється).



Це можна поширити на кілька випадкових генів бути перевернутим замість одного.

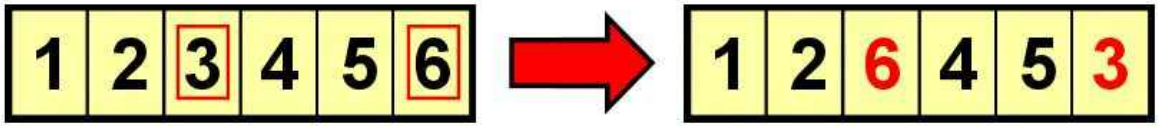


Робочий процес —Мутація - Swap

Для двійкових або цілих хромосом,

2 гениєвипадково вибрані

і їх значення**поміняні місцями**.

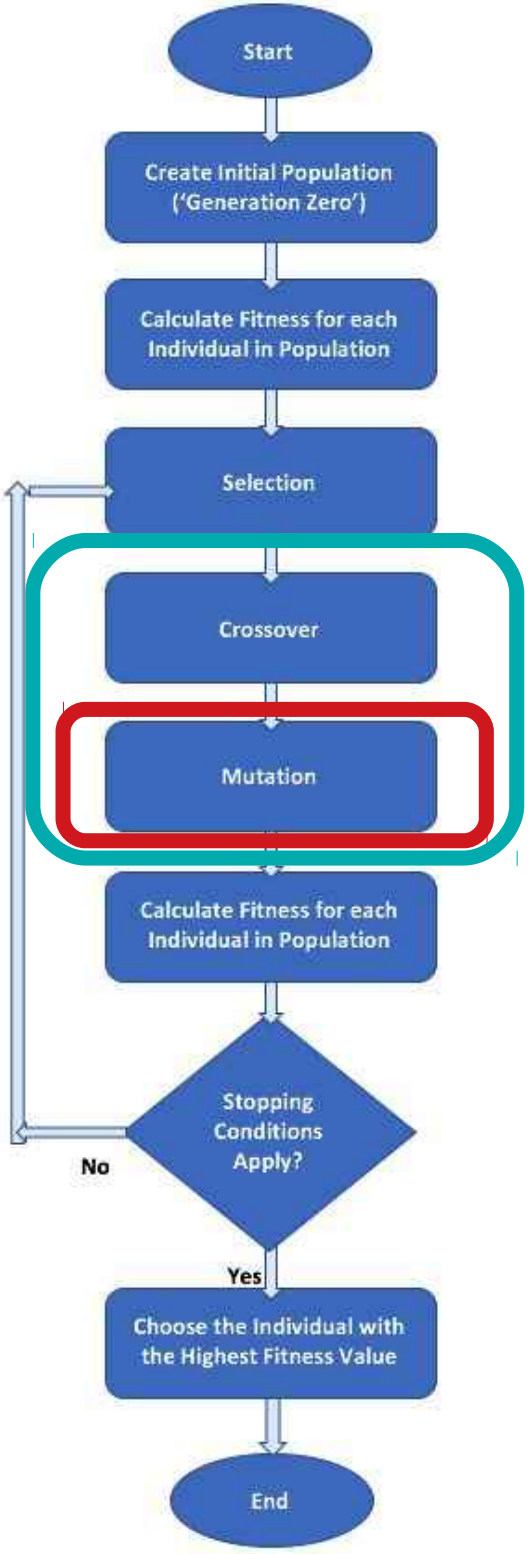


Це операція мутації**підходить** для

хромосоми впорядкованих списків,

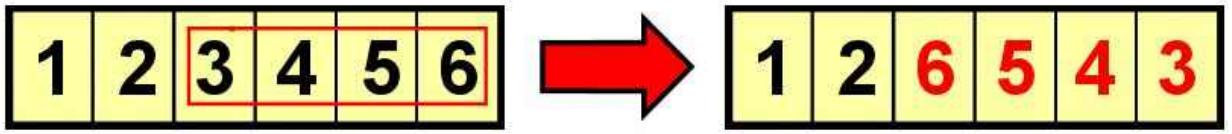
як**нова хромосома**

доси**несеоднакові гени**як оригінальний.

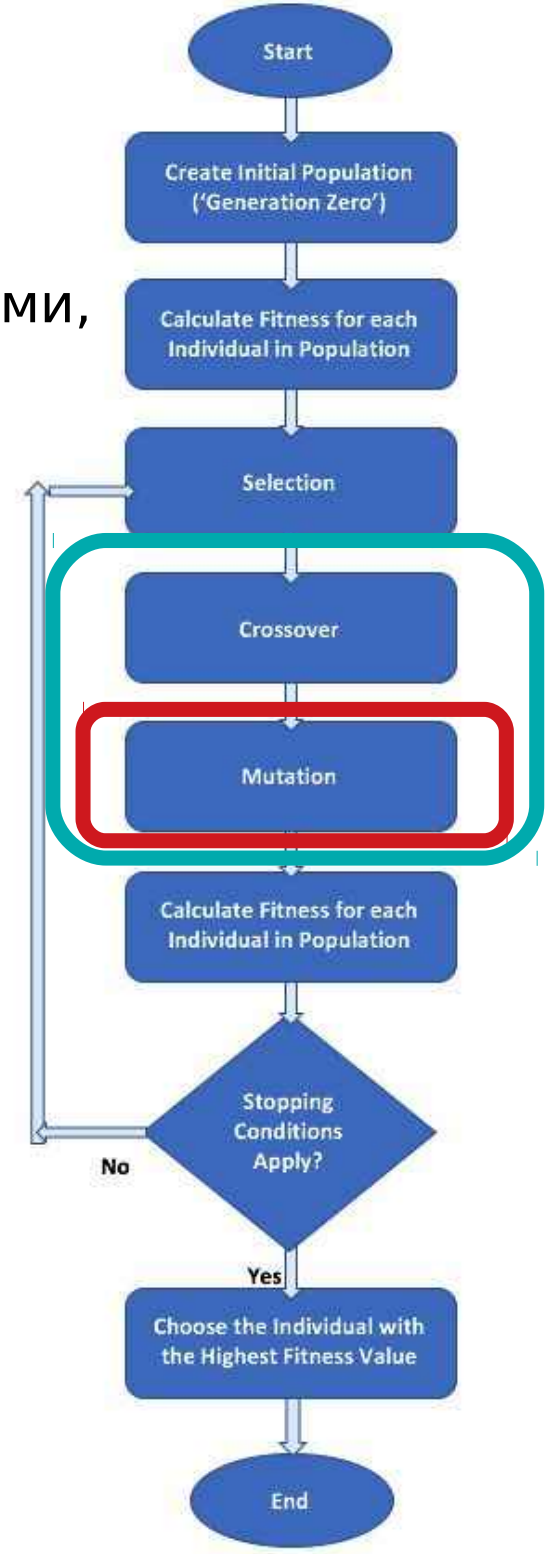


Робочий процес —Мутація - Інверсія

Для**двійковий**або**на основі цілих чисел**хромосоми, а**випадкова послідовність**генів відбирається і **порядок**генів у цій послідовності**перевернутий**.

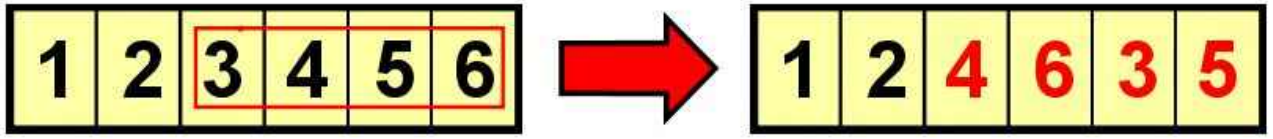


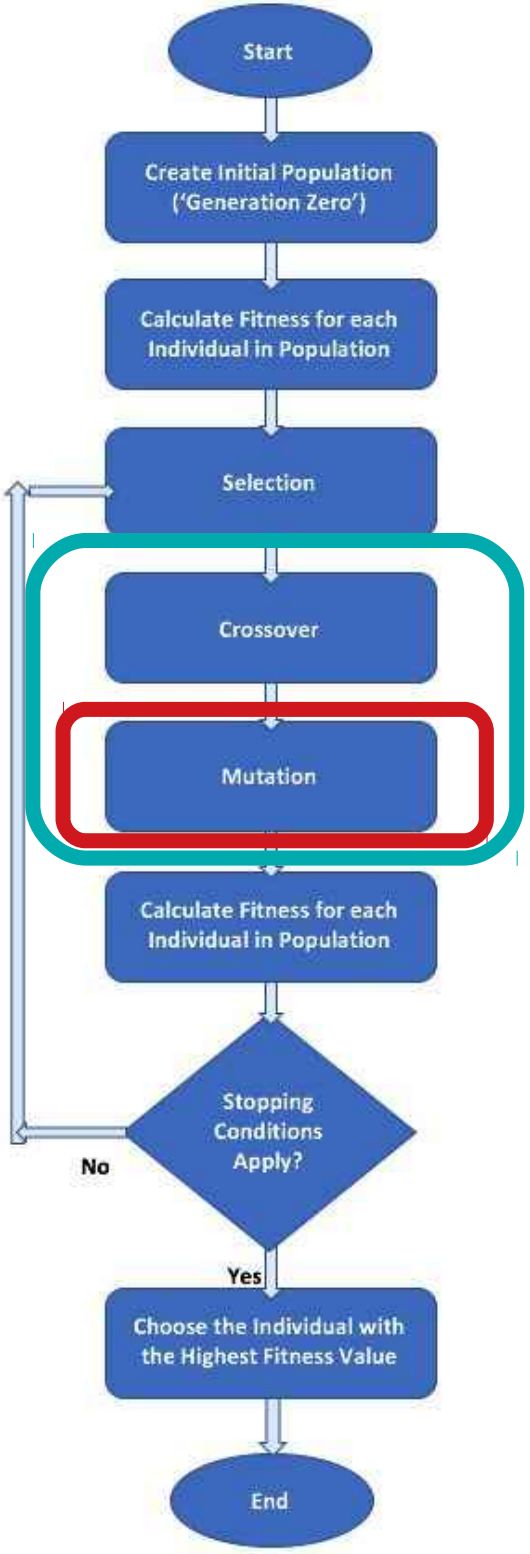
Подібний до**мутація обміну**, в**інверсійна мутація**операція є**підходить** для хромосом**упорядковані списки**.



Робочий процес —Мутація - Scramble

Для**двійковий**або**на основі цілих чисел**хромосоми, а **випадкова послідовність**генів відбирається і і**порядок**генів у цій послідовності є**перетасувався**(або**скремблював**).





Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- **ЕА з реальним кодуванням**

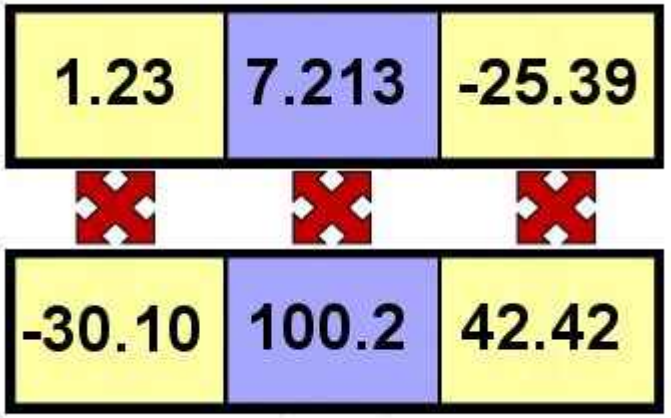
Робочий процес -Варіаційні оператори-

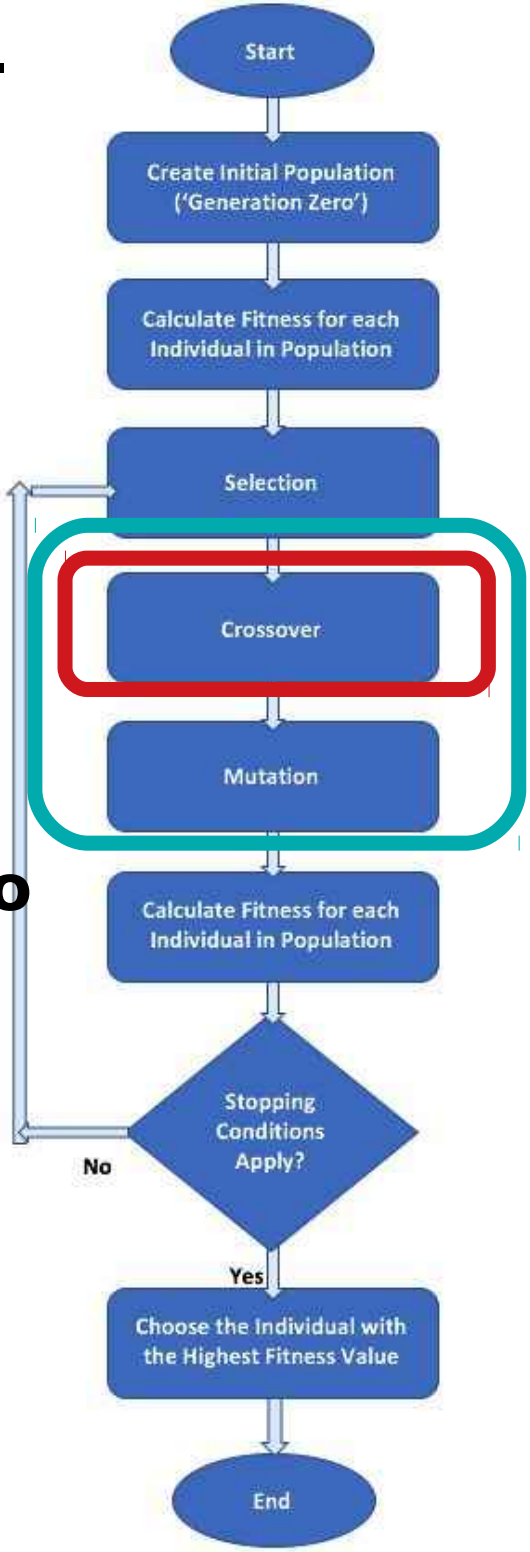
Реальне кодування

The**вибір**методи працюватимуть так само **оскільки вони залежать лише від фітнесу**з особи, а не їхнє представництво.

алев**кросовер**і**мутація**методи будуть **не підходяты**і тому спеціалізованим потрібно бути використаним.

Їх слід застосовувати**окремо для кожного розміру**масиву, який утворює реальну закодована хромосома.





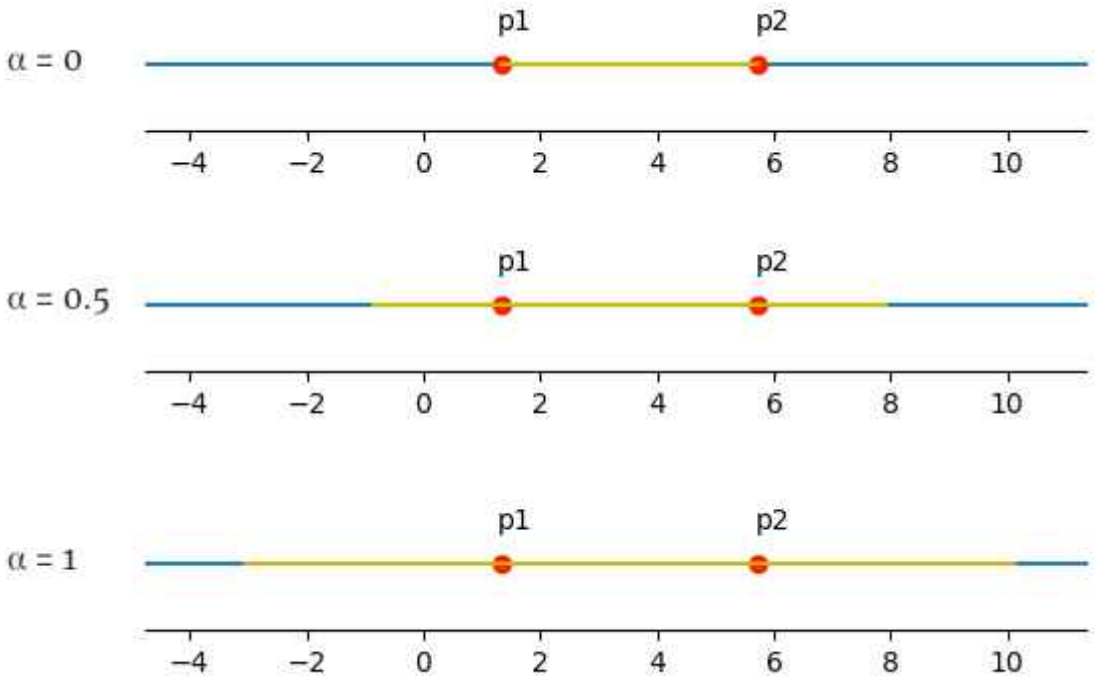
Робочий процес -Варіаційні оператори

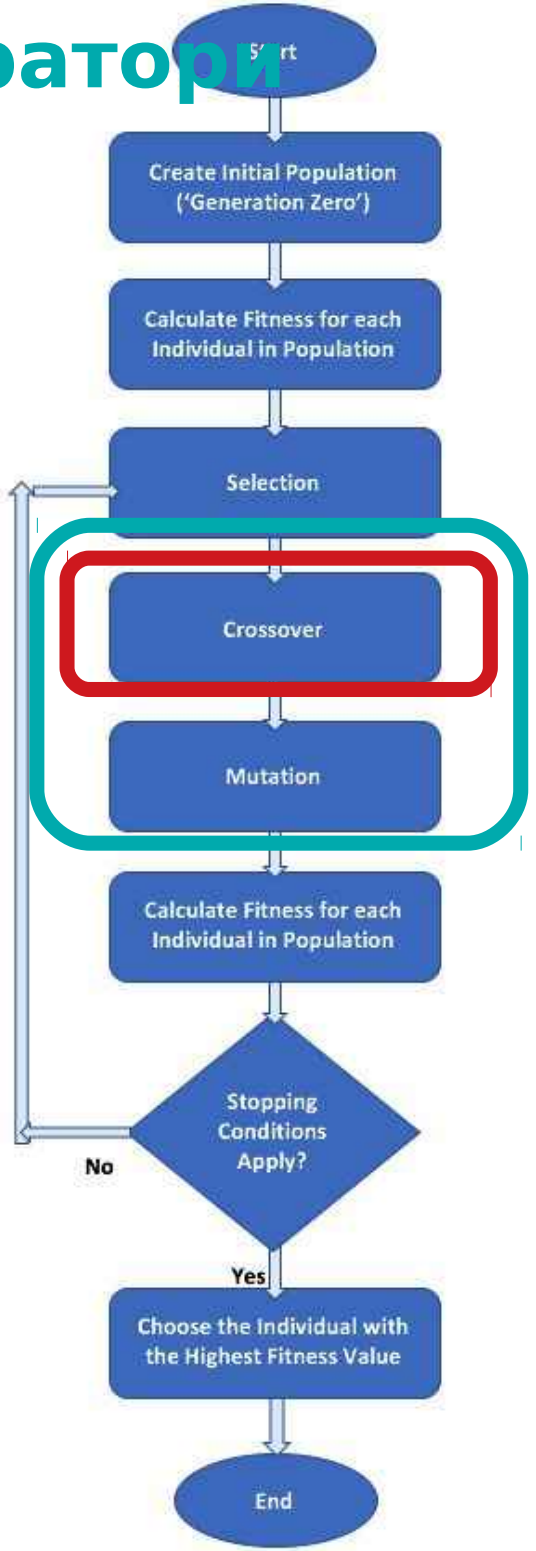
- Real-coded — Blend Crossover

Змішаний кросовер(**BLX**) - кожен**потомство**є випадковим чином**вибрано**з інтервалу, створеного його батьківські значення за деякими формулами:

$$[parent_1 - \alpha(parent_2 - parent_1),\ \ parent_2 + \alpha(parent_2 - parent_1)]$$

Параметр α є константою, значення якої лежить між 0 і 1. При більших значеннях α,



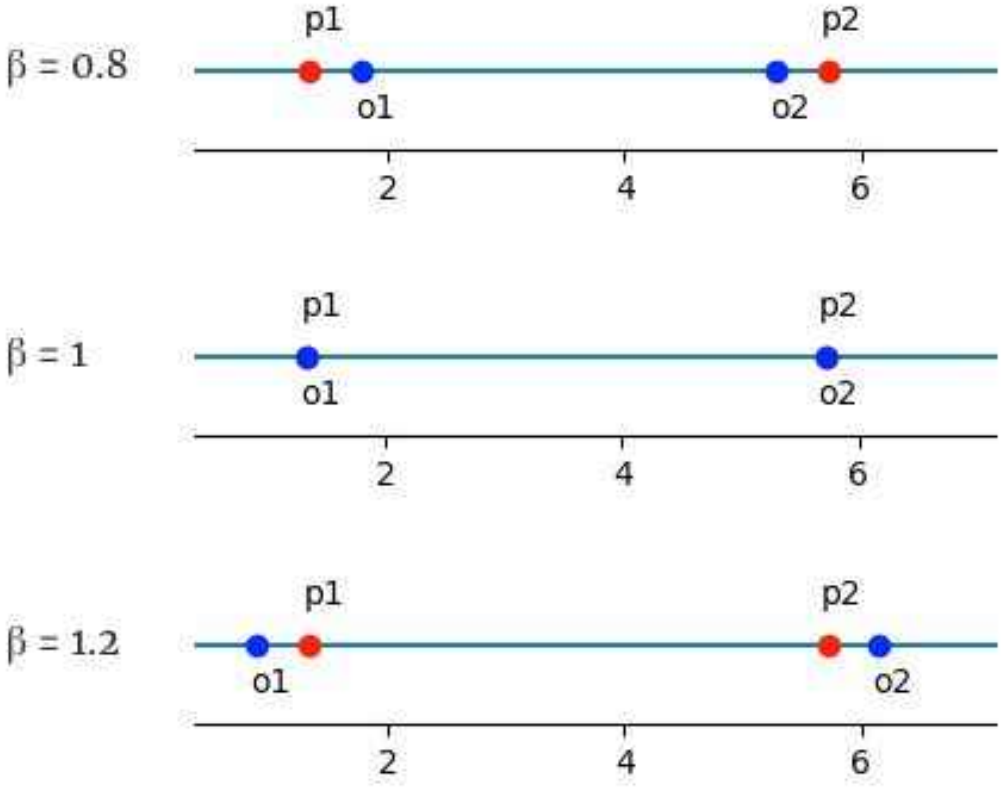


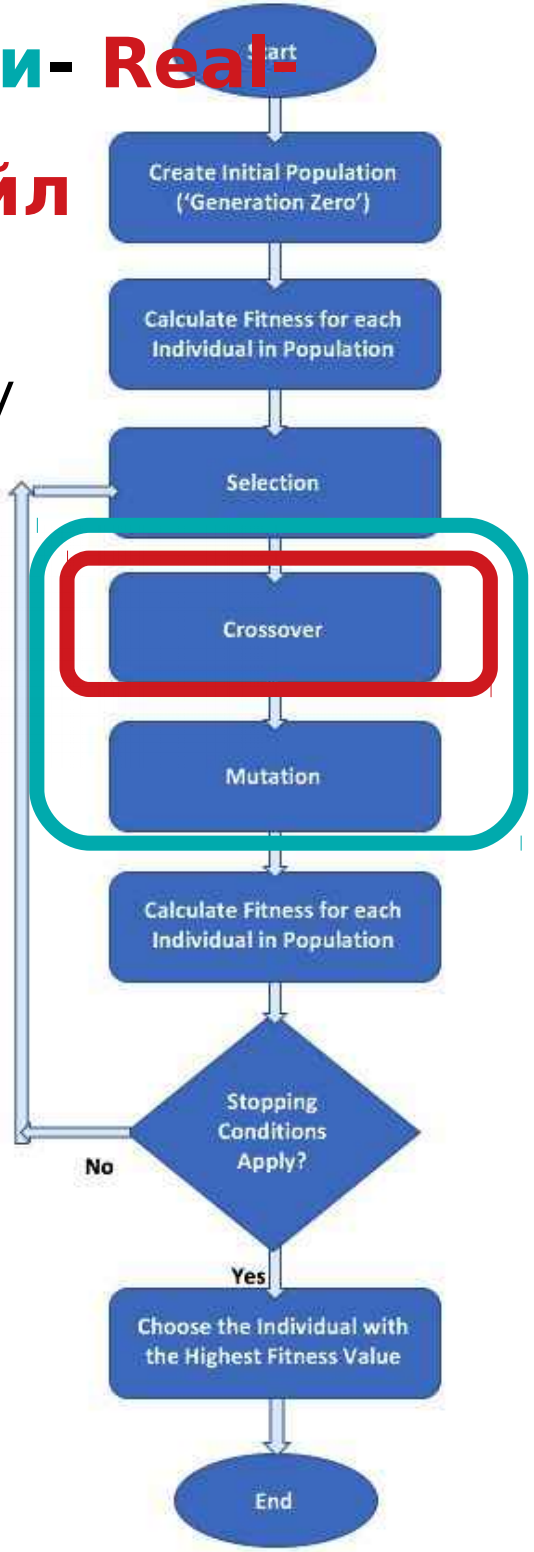
Робочий процес -Варіаційні оператори- Real-coded — симульований двійковий файл

Імітований бінарний кросовер(**SBX**) - кожен нащадок випадковим чином вибирається з інтервалу створені його батьківськими значеннями за формулою:

$$offspring_1 = \frac{1}{2}[(1+\beta)parent_1 + (1-\beta)parent_2]$$

$$offspring_2 = \frac{1}{2}[(1-\beta)parent_1 + (1+\beta)parent_2]$$





Робочий процес -Варіаційні оператори- Real-coded — симульований двійковий файл

У попередніх випадках**середній**значення два**потомство**становить 3,525, тобто**дорівнює**в **середній**значення двох**батьки**. **Ми повинні зберегти схожість між ними нащадків і батьків.**

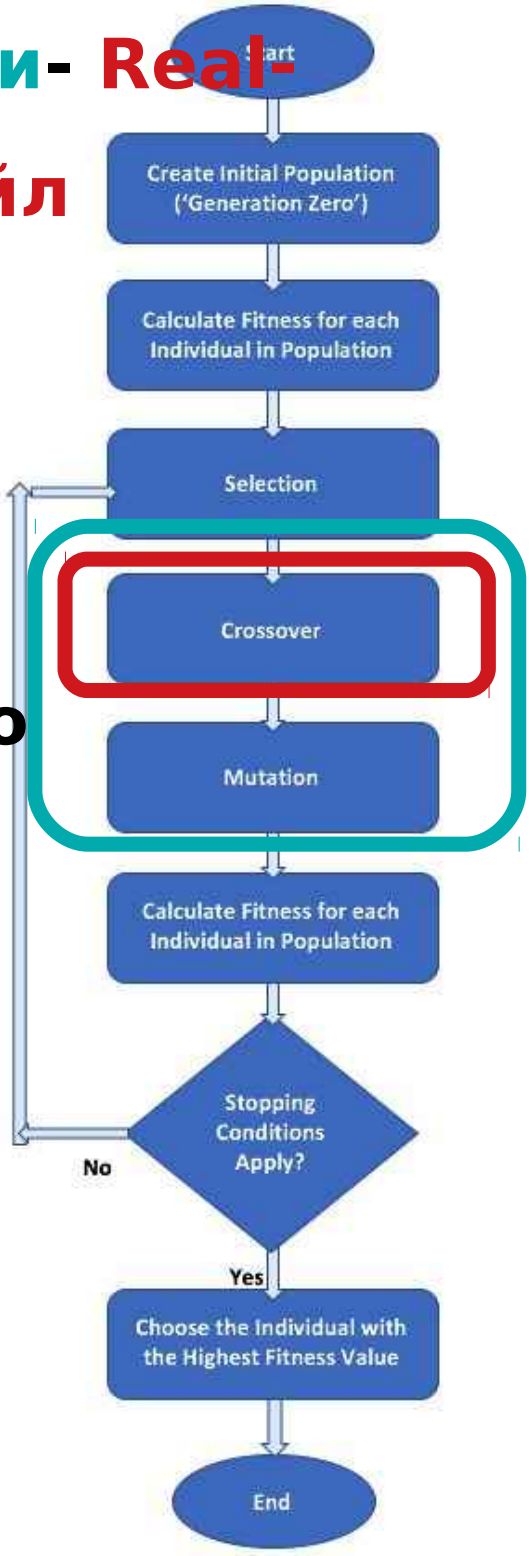
Для цього ймовірність**β**має бути**набагато вище**для цінностей**біля 1**, де приплід є**подібні**до батьків. Тому значення β розраховується за допомогою **інший випадковий**величина, позначена u, рівномірно розподілена ов [0, 1]:

u <=0,5

$$\beta = (2u)^{\frac{1}{\eta+1}}$$

u > 0,5

$$\beta = \left[\frac{1}{2(1-u)}\right]^{\frac{1}{\eta+1}}$$

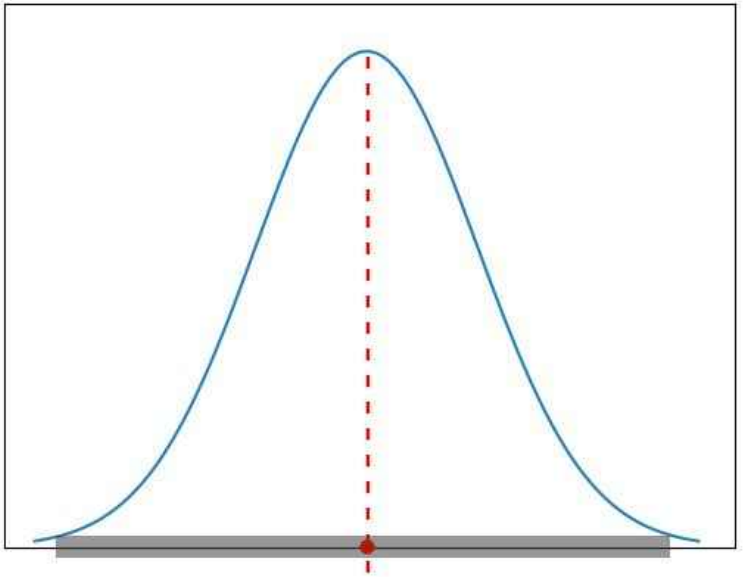


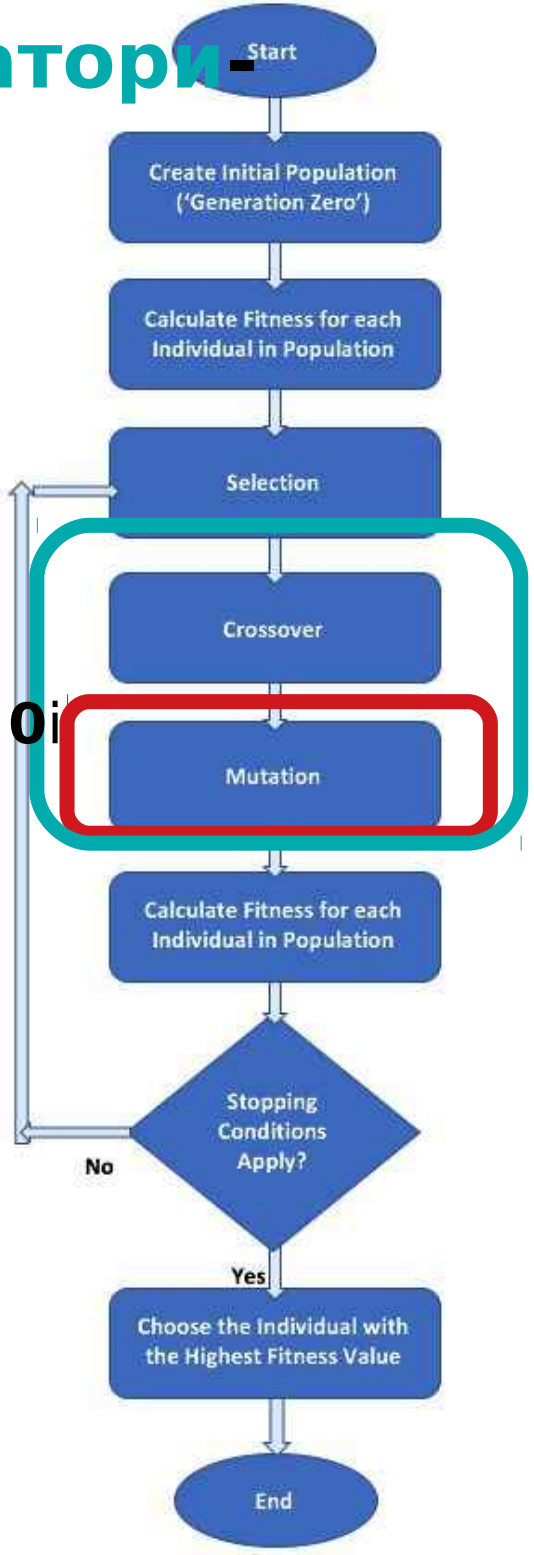
Робочий процес -Варіаційні оператори-

Real-coded — Справжня мутація

Інший підхід полягає в тому, щоб**генерувати випадкове дійсне** номер, який проживає**в околицях с** **оригінальний**індивідуальний.

приклад :**нормально розподілені**(або Гауса) **мутація**->а**випадкове число**генерується за допомогою а**нормальний розподіл**з **середнє = 0**і деякі**заздалегідь визначений стандарт відхилення**.



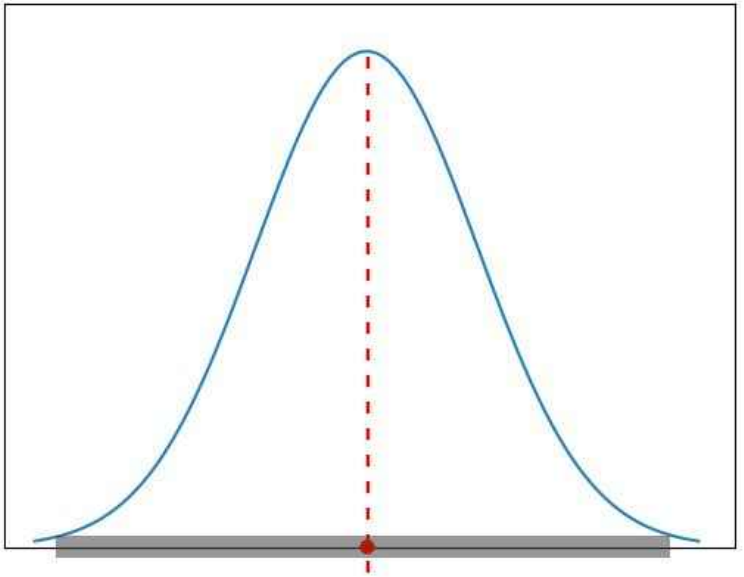


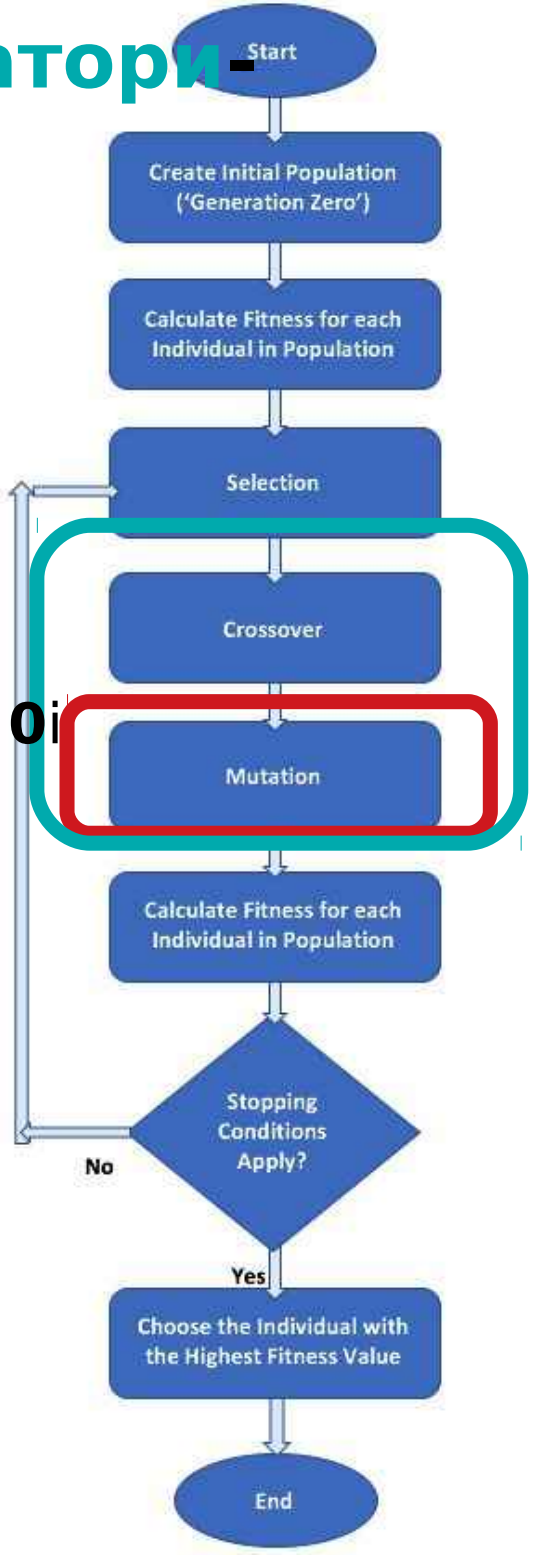
Робочий процес -Варіаційні оператори-

Real-coded — Справжня мутація

Інший підхід полягає в тому, щоб**генерувати випадкове дійсне** номер, який проживає**в околицях с** **оригінальний**індивідуальний.

приклад :**нормально розподілені**(або Гауса) **мутація**->а**випадкове число**генерується за допомогою а**нормальний розподіл**з **середнє = 0**і деякі**заздалегідь визначений стандарт відхилення**.





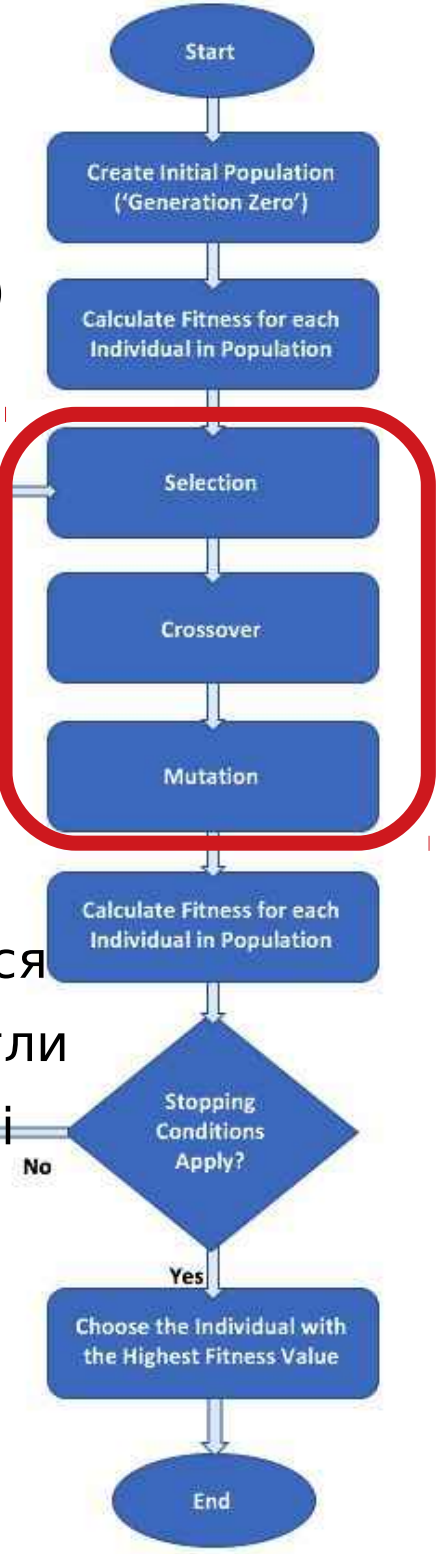
Зміст

- Рекомендовані джерела
- Що таке еволюційні обчислення (ЕС)
- Історія ЄС
- Типи задач для ЕК
- Що таке еволюційний алгоритм (ЕА)
- Робочий процес ЕА
- Вибір
- Кросовер
- Мутація
- ЕА з реальним кодуванням
- **Елітність, Нічінг, Спільне використання**

Робочий процес -

Стратегія елітарності

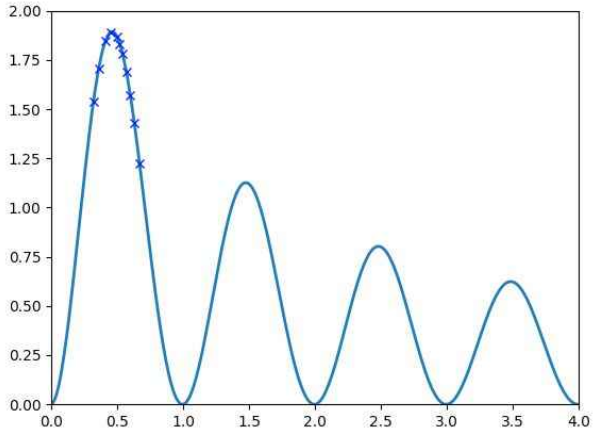
Ми хочемо гарантувати, що**найкраща людина(и)** завжди досягати наступного покоління, ми можемо застосувати**необов'язковий**[стратегія елітарності](). Це означає, що**зверху особи**(де попередньо визначеним параметром)**дублюються в наступне покоління**перед нами**заповнити решту**з наявних місць с**потомство**які створюються **за допомогою відбору, кросинговеру та мутації**. The **еліта**особи, які були продубльовані, все ще залишаються **підходять для відбору**обробляти, щоб вони все ще могли **використовувати як батьків**нових осіб. Елітарність іноді може мати значне значення**позитивний вплив**на продуктивність алгоритму, оскільки він уникає потенціалу**трата часу**необхідні для**повторно відкриваючи добро**рішення, які були втрачені.

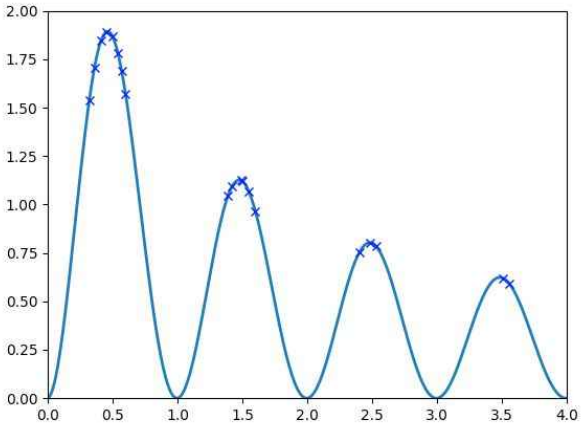


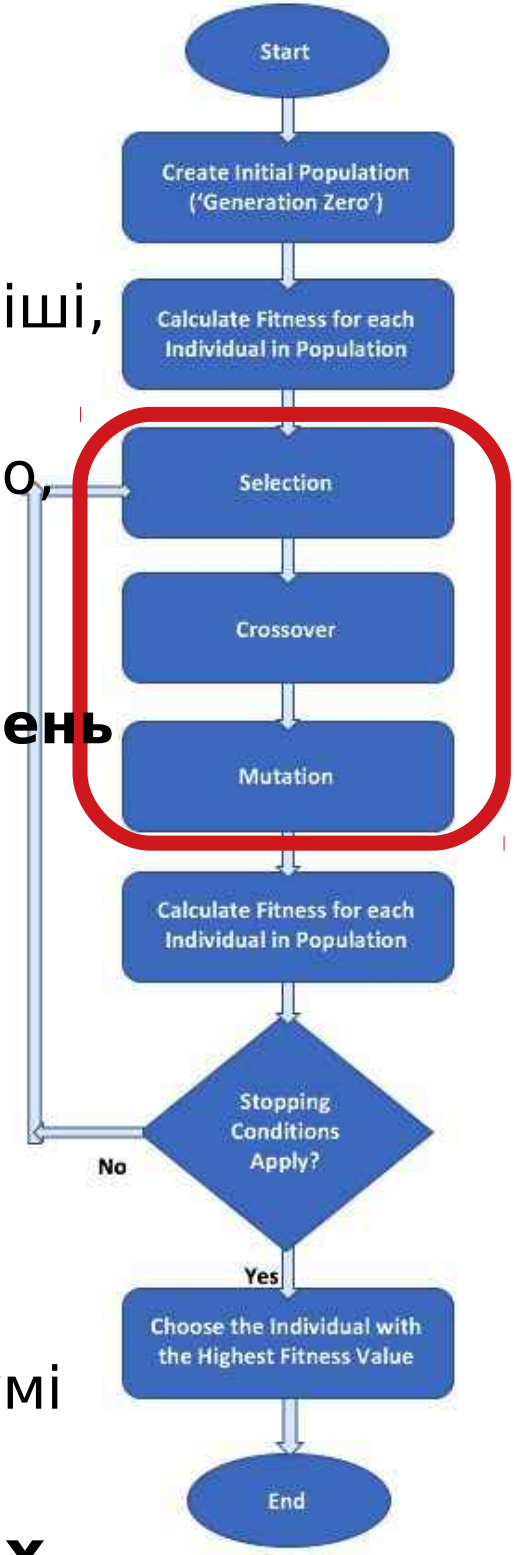
Робочий процес -

Ніші та обмін

Коли кілька різних видів співіснують в одній ніші, усі вони конкурують за ті самі ресурси, і а **тенденція**полягає в тому, щоб**пошук**для нового, незаселений**ніші**і заселити їх. Це можна використовувати для підтримки різноманітності популяції та для**знайти кілька оптимальних рішень ->кілька ніш**.







Для цього ми повинні**пропонувати ресурси**в сумі **пропорційно висоті ніші**за**обмін фітнесом**залежало**на відстані до інших**.

Основи еволюційних обчислень

-

Основи еволюційного обчислення

-

Лекція 03. ЕС для машинного навчання

— Вибір функції

(за мотивами Алана Тюрінга, Голланда, Халеда Рашида, Бена роботи Філіпса, Еяля Вірсанського та ін.)

Зміст

- **Рекомендовані джерела**
- ЕА (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення ЕА (GA).
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення ЕА (GA).
- Резюме

Рекомендовані джерела — Книги

Книги (наукові):

Гайон І., Ганн С., Нікравеш М. та Заде Л.А. (Ред.). (2008). Вилучення ознак: основи та застосування (Том 207). Спрингер.

Донг, Г. та Лю, Х. (Ред.). (2018). Розробка функцій для машинне навчання та аналіз даних. CRC Press.

Книги (з кодами на github):

Соледад Галлі (2020). Розробка функцій Python Кулінарна книга. Packt Publishing

Аліса Чжен і Аманда Казарі (2018). Розробка функцій для машинного навчання. О'Райлі)

Рекомендовані джерела -

Документи та набори даних

Проблема регресії (F1RP):

Брейман, Лео (1996). Машинне навчання 24, сторінки 123-140. Фрідман, Джером Х. (1991) Багатовимірні сплайни адаптивної регресії. Літопис с Статистика 19 (1), сторінки 1-67.

Проблема класифікації

Набір даних UCI Zoo (http://archive.ics.uci.edu/ml/datasets/Zoo)

Ейбе Франк і Стефан Крамер. Ансамблі вкладених дихотомій для мультикласу проблеми. ICML. 2004 рік.

Хуан Лю, Хіроші Мотода та Лей Ю. Вибір функцій за допомогою Selective Відбір проб. ICML. 2002 рік.

Зміст

- Рекомендовані джерела
- **EA (GA) для вибору функцій — чому?**
- Типи проблем для вибору функції:
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення EA (GA).
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення EA (GA).
- Резюме

Еволюційні обчислення (ЕС) — для вибору функції — чому?

Контрольоване навчання:

робочий процес :**модель**отримує набір**входи**, дзвонив**особливості**, і відображає їх у набір**виходи**.

Успенський : інформація, описана в**особливості**є**корисний для** визначення вартості відповід**виходи**.

Здоровий глузд :**більше**інформацію, яку ми можемо використовувати як вхідні дані,**краще** наші шанси правильно передбачити результат(и).

Реальність : у багатьох випадках**навпаки**...якщо деякі особливості м використання є**не має значення**або**надлишковий**, наслідком може бути (іноді значний)**зниження точності**моделей.

Тому нам потрібно**вибір функції**: в**процес**з**вибираючи**найбільший**корисний набір функцій**із весь набір функцій для**досягти кращого**рішення.

ЕС для вибору функції —

Переваги

- Зменшення похибок (втраченої функції) моделі

- Підвищення точності моделі

- Час навчання моделей коротший.

- Навчені моделі простіші та легші для інтерпретації.

- Навчені отримані моделі, ймовірно, забезпечать краще **узагальнення**, тобто вони працюють краще з новими вхідними даними, які відрізняються від дані, які були використані для навчання.

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- **Типи проблем для вибору функції:**
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення EA (GA).
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення EA (GA).
- Резюме

ЕС для вибору функції — Типи задач

ЕС (GA) можна ефективно застосувати до класичних проблем машинного навчання під керівництвом:

– **регресія**(використання проблеми регресії Фрідмана-1)

і

– **класифікація**(використання класифікації тварин набору даних UCI)

для

– **вибір функції**

або

– **зменшення розмірності**

з метою:

– **зменшення MSE**

або

– **збільшення середнього точність**.

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- **Регресія: проблема Фрідмана-1**
  - Класичне рішення
  - Рішення EA (GA).
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення EA (GA).
- Резюме

ЕС для вибору функції —Приклад: Фрідман-1регресія Проблема (F1RP

F1RPбув описаний Фрідманом (1991) і Брейманом (1996).
входи:n_featuresнезалежні змінні, рівномірно розподілені на інтервал [0,1], лише 5 із них **n_features**фактично використовуються.
Виходи: створюються за формулою:

$$y(x_0, x_1, x_2, x_3, x_4) = 10 \cdot sin(\pi \cdot x_0 \cdot x_1) + 20(x_2 - 0.5)^2 + 10x_3 + 5x_4 + noise \cdot N(0,1)$$

Останнім компонентом у формулі є випадково згенерований шум. Шум нормально розподіляється і множиться на постійний шум, який визначає його рівень.

Різні реалізації в мовах програмування: **Python**:
make_friedman1()функція в scikit-learn (**sklearn**) бібліотека
P:**friedman1()**функція в**mlbench**бібліотека

Чому F1RP корисний для нас?

Брейман, Лео (1996). Машинне навчання 24, сторінки 123-140. Фрідман, Джером Х. (1991) Багатовимірні сплайни адаптивної регресії. Літопис с

EC для вибору функції — приклад:

чому F1RP корисний для нас?

Якщо **n_features = 15**, ми отримаємо набір даних з оригіналом **5** вхідні змінні (або функції), які були використані для створення **значення** за формулою і **10** функції, які абсолютно не мають відношення до результату.

чому: **F1RP** звик **тест** різні **регресія** моделі щодо наявності **шум** і нерелевантні функції в наборі даних.

приклад :

Цілься: перевірте EC (GA) як а **вибір функції** механізм. **робочий проце** використання **make_friedman1()** для створення набору даних із 15 функціями та використання GA для пошуку **підмножина** функцій, які забезпечує в **найкращий** продуктивність.

Гіпотеза: EC (GA) вибере перші 5 функцій і відкине решту, припускаючи, що точність моделі краща, коли лише відповідні функції використовуються як вхідні дані.

EC (GA) роль: The **функція фітнесу** (**FF**) використовуватиме а **регресійна модель** що кожного потенціалу **рішення** – а **підмножина функції** використовувати – буде бути **навчений** використовуючи набір даних, що містить **тільки вибране** особливості.

ЕС для вибору функції — приклад:

Індивідуальне представництво ЕС (G

Ан**індивідуальне рішення**(генотип) має вказувати який**особливості є вибрано**і які випадають:

- Кожен**індивідуальне рішення**це**список двійкових**значення

- кожен**запис**у списку (0 або 1) є**одна з особливостей**в наборі даних:
  - 1 - відповідна ознака**БУВ****вибрано**,
  - 0 - функція має**НІ**був**вибрано**.

Це дуже схоже на задачу ранця 0-1 з Lab01.

ВАЖЛИВО:

Кожен 0в індивідуальному рішенні**засоби**

->

падіннявідповідний**особливості**даних**колонка**з набору даних.

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- **Регресія: проблема Фрідмана-1**
  - **Класичне рішення**
  - Рішення EA (GA).
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення EA (GA).
- Резюме

ЕС для вибору функції — приклад: F1RP — Класичне рішення

1) Створити в **набір даних** за формулою Фрідмана використовуючи **make_friedman1()** – функція в scikit-learn (**sklearn**) бібліотека.

2) Розділити дані на дві підмножини – а **навчання** набір і а **перевірка** встановити – використовуючи **model_selection.train_test_split()** функції в scikit-learn.

3) Створити регресія **модель**...можна використовувати різні...**Градієнт Boosting Regressor (GBR)** у цьому прикладі.

4) Визначити продуктивність використаної моделі регресії для набору вибрані функції за **getMSE()** **функція-метрика***.

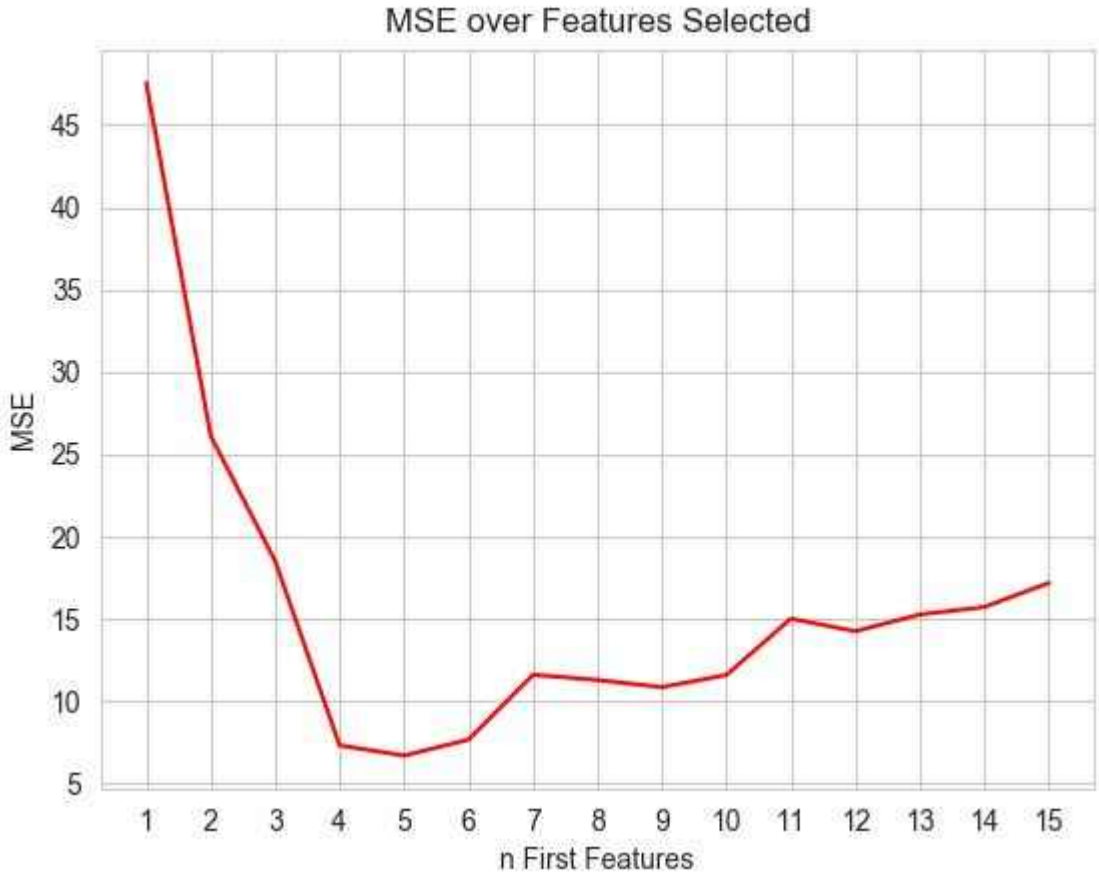
5) Тоді в **новий** навчання **підмножина** (з **лише вибрані функції**!) використовує для навчання моделі під час нової перевірки **підмножина**-щоб оцінити його.

*) **Середня квадратична помилка (MSE)**=середній квадрат різниці між прогнозовани значеннями моделі та фактичними значеннями. А **нижче** значення цього

EC для вибору функцій — Приклад: F1F — Класичне рішення —Результати...

Оскільки ми додаємо перші 5 функцій одну за одною, продуктивність покращує. Однак пізніше кожна додаткова функція погіршується продуктивність моделі:

1 перші ознаки: оцінка = 47,553993 2 перші особливості: оцінка = 26,121143 3 перші особливості: оцінка = 18,509415 4 перші особливості: оцінка = 7,322589 5 перші особливості: оцінка = 6,702669 6 перші особливості: оцінка = 7,677197 7 перші особливості: оцінка = 11,614536 8 перші функції: оцінка = 11,294010 9 перших характеристик: оцінка = 10,858028 10 перших характеристик: оцінка = 11,602919 11 перших особливостей: оцінка = 15,017591 12 перших функцій: оцінка = 14,258221 13 перших функцій: оцінка = 15,274851 14 перших функцій: оцінка = 15,726690 15 перших характеристики: бал = 17,187479



EC для вибору функцій — Приклад: F1RP — Класичне рішення —ДЕМО...

Спробуйте відтворити ці результати:

1 перші ознаки: оцінка = 47,553993 2 перші особливості: оцінка = 26,121143 3 перші особливості: оцінка = 18,509415 4 перші особливості: оцінка = 7,322589 5 перші особливості: оцінка = 6,702669 6 перші особливості: оцінка = 7,677197 7 перші особливості: оцінка = 11,614536 8 перші функції: оцінка = 11,294010 9 перших характеристик: оцінка = 10,858028 10 перших характеристик: оцінка = 11,602919 11 перших особливостей: оцінка = 15,017591 12 перших функцій: оцінка = 14,258221 13 перших функцій: оцінка = 15,274851 14 перших функцій: оцінка = 15,726690 15 перших характеристики: бал = 17,187479

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- **Регресія: проблема Фрідмана-1**
  - Класичне рішення
  - **Рішення EA (GA).**
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення EA (GA).
- Резюме

EC для вибору функції — приклад: F1RP —Розчин EC (GA).

The**відмінності**від**класичний**рішення:

1)**Хромосоми**-двійкові списки вибраних функцій

2)**Фітнес-функція (FF)**-повертає модель регресії**MSE**

3)**Вибір**
- **турнір**відбір з турніром**розмір 2**
- **елітарність**, де**зал слави (HOF)**учасники – діючі**найкращий** особи – є**завжди проходили недоторканими**до наступного покоління

4)**Еволюційні (генетичні) оператори**
- **кросовер**
і
- **мутація**оператори

які спеціалізуються на хромосомах з бінарним списком

EC для вибору функції — Приклад: F1RP — Рішення EC (GA) — Результат

Після 30 поколінь EC (GA):

Найкраща особа = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Найкращий фітнес = 6,70266891046328 7

Що це означає?

Найкращий MSE (близько 6,7) забезпечують перші п'ять функцій.

ВАЖЛИВО:
EA (GA) робить **ніяких припущень** про набір ознак.
EA (GA) робить **не знати** про перше чи останнє особливості. EA (GA) просто шукав **найкраща можлива підмножина** особливостей.

ЕС для вибору функції — приклад: F1RP — Рішення ЕС (GA) —ДЕМО

Спробуйте відтворити ці результати:

Після 30 поколінь ЕС (GA):

Найкраща особа = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Найкращий фітнес = 6,702668910463287

Що це означає?

Найкращий MSE (близько 6,7) забезпечують перші п'ять функцій.

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення EA (GA).
- **Класифікація: проблема тварин**
  - Класичне рішення
  - Рішення EA (GA).
- Резюме

ЕС для вибору функції —приклад:

Проблема класифікації тварин

Це класичний приклад проблеми класифікації.

о).

ЕС для вибору функції — приклад:

Класифікація тварин —Набір даних

Загальна інформація про набір даних:

Проста база даних, що містить**17 Атрибути з логічними значенням** "**типу**" видається атрибутом**клас**атрибут. Ось розбивка того, які тварини належать до якого типу: (Мені дивно, що такі є**2 екземпляри "жаба"**і**одна з "дівчинок"**!)

Клас № -- Набір тварин:

1 -- (41) трубкозуб, антилопа, ведмідь, кабан, буйвол, теля, зубчастий птах, гепард, олень, дельфін, слон, кажан, жираф, дівчина, коза, горила, хом'як, заєць, леопард, лев, рись, ноутбук, кріт, мангуст, опосум, орикс, качконіс, тхір, поні, морська свиня, пума, кицька, єнот, північний олень, тюлень, тюлень, білка, вампір, полівка, валлабі, вовк

2 -- (20) курка, ворона, голуб, качка, фламінго, чайка, яструб, ківі, жайворонок, страус, папуга, пінгвін, фазан, нанду, скиммер, поморник, горобець, лебідь, гриф, крапивник

3 -- (5) морська змія, повільний черв'як, черепаха, туатара

4 -- (13) окунь, короп, сом, головень, собачка, пікша, оселедець, щука, піранья, морський коник, морський язик, скат, тунець

5 -- (4) жаба, жаба, тритон, жаба

6 -- (8) блоха, комар, медоносна бджола, кімнатна муха, сонечко, міль, терміт, оса

7 -- (10) молюски, краби, раки, омари, восьминіги, скорпіони, морські оси, слимаки, морські зірки,

ЕС для вибору функції — приклад:

Класифікація тварин —Набір даних

Атрибут (Характеристика) Інформація:

1. назва тварини: унікальна для кожного екземпляра
2. волосся: Boolean
3. пір'я: логічний
4. яйця: Boolean
5. молоко: логічний
6. бортовий: Boolean
7. водний: Boolean
8. хижак: Булев
9. зубчастий: булевий
10. магістраль: логічний
11. дихає: логічний
12. отруйний: Булев
13. ласти: булеві
14. ноги: числові (набір значень: {0,2,4,5,6,8})
15. хвіст: логічний
16. вітчизняні: Булев
17. catsize: логічний

ЕС для вибору функції — приклад: Класифікація тварин —проблема

Походження: це класичний приклад проблеми класифікації, де вхідні функції потрібно відобразити у двох або більше категоріях/мітках.

Вхідні дані: ознаки 2-17 (волосся, пір’я, плавники тощо), переважно ознаки логічними (значення 1 або 0), що означає наявність або відсутність певний атрибут, такий як волосся, плавники тощо.

Примітка : 1-а особливість -**назва тварини**-просто надати нам трохи інформація та**участі не бере**у навчанні.

Виходи: остання функція –**типу**–представляє 7 категорій. Наприклад, тип 5 представляє категорію з: жаба, тритон і жаба.

Цілься:**поїзд**а**модель класифікації**на цьому наборі даних с**особливості 2-17** (волосся, пір'я, плавники і так далі) до**передбачити**значення**функція 18** (тварина**типу**).

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення EA (GA).
- **Класифікація: проблема тварин**
  - **Класичне рішення**
  - Рішення EA (GA).
- Резюме

ЕС для вибору ознак — Приклад:

Класифікація тварин — Класичний спос

1) Навантаження UCI-Зоопарк **набір даних** за стандартом **read_csv** функція.

2) Розділити дані в **введення** функції (перші 16 стовпців, що залишилися) і результуючі **вихід** категорія (останній стовпець). Тоді замість того, щоб розділит дані в **1 тренування** встановити і **1 тест** набір, як і в попередньому розділі, який ми використовуємо **k-кратна перехресна перевірка** -> Дані розділені на **k** рівні частини і модель оцінюється **k** разів:

(k-1) частини для **навчання** і **1** частина, що залишилася для тестування (або **перевірка**).

3) Створити класифікація **модель** ...можна використовувати різні моделі...

Класифікатор дерева рішень (DCT) в цьому прикладі.

4) Визначити продуктивність використаної моделі регресії для набору вибраних ознак **getMeanAccuracy()** **функція-метрика** *.

*) **Точність** – частина справ, які були класифіковані правильно. А **вище** значення цього вимірювання вказує **краща продуктивність** моделі.

EC для вибору ознак — Приклад: Класифікація — Класичний спосіб — ДЕМ

Після навчання/тестування:

модель - DTC-класифікатор

5-кратна перехресна перевірка

всі 16 функцій

точність класифікації становила близько **91%**.

Спробуйте відтворити ці результати:

Усі вибрані функції:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Точність = 0,9099999999999999

Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення EA (GA).
- **Класифікація: проблема тварин**
  - Класичне рішення
  - **Рішення EA (GA).**
- Резюме

ЕС для вибору функції — приклад: Класифікація —Розчин ЕС (GA).

The**відмінності**від**класичний**рішення:

1)**Хромосоми**-двійкові списки вибраних функцій

2)**Фітнес-функція (FF)**-повертає моделі**середня точність**

3)**Вибір**
- **турнір**відбір з турніром**розмір 2**
- **елітарність**, де**зал слави (HOF)**учасники – діючі**найкращий** особи – є**завжди проходили недоторканими**до наступного покоління

4)**Еволюційні (генетичні) оператори**
- **кросовер**
і
- **мутація**оператори

які спеціалізуються на хромосомах з бінарним списком

EC для вибору функції — приклад:

Класифікація — EC (GA) —Результати

Після 50 поколінь EC (GA) і HOF розмір 5:

Найкращі рішення:
0 : [0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] придатність = 0,964 точність = 0,97 особливості = 6 1 : [0, 1 , 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0] придатність = 0,963 точність = 0,97 особливості = 7 2 : [1, 1, 0, 1, 1 , 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] придатність = 0,963 точність = 0,97 особливості = 7 3 : [0, 1, 0, 1, 1, 1, 0, 0 , 1, 0, 0, 1, 0, 0, 1, 0] придатність = 0,963 точність = 0,97 особливості = 7 4 : [0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0 , 1, 0, 0, 0, 1] придатність = 0,963 точність = 0,97 особливості = 7

The**топове рішення**це набір**6 особливості**, які є такими:

пір'я, молоко, в повітрі, хребет, плавники, хвіст

Вибравши ці особливості з 16 наведених у наборі даних:

1 - ми**зменшили розмірність**задачі, 2 - ми також **покращений**наша модель**точність**від 91% до 97%.

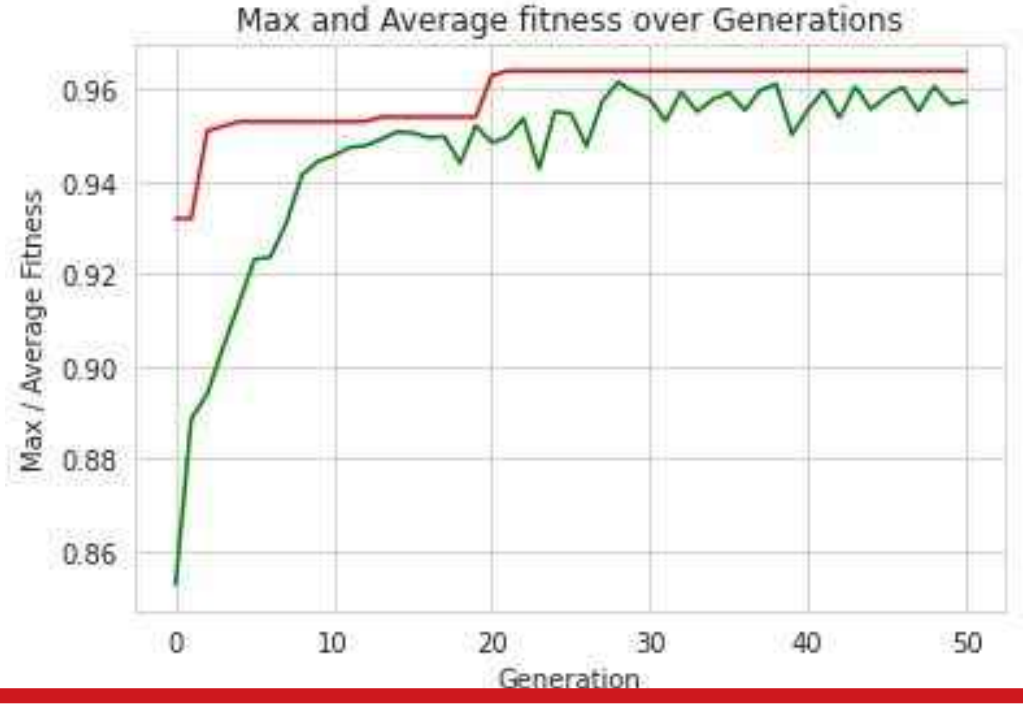
ВАЖЛИВО: Це є**не дуже великий**збільшення абсолюту**точність**, **АЛЕ**а **чудово (ПОТРІЙНО!) скорочення**з**частота помилок**від 9% до 3% – а

ЕС для вибору функції — приклад: Класифікація — ЕС (GA) —ДЕМО

Спробуйте відтворити ці результати:

Найкращі рішення:

0 : [0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] придатність = 0,964 точність = 0,97 особливості = 6 1 : [0, 1 , 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0] придатність = 0,963 точність = 0,97 особливості = 7 2 : [1, 1, 0, 1, 1 , 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] придатність = 0,963 точність = 0,97 особливості = 7 3 : [0, 1, 0, 1, 1, 1, 0, 0 , 1, 0, 0, 1, 0, 0, 1, 0] придатність = 0,963 точність = 0,97 особливості = 7 4 : [0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0 , 1, 0, 0, 0, 1] придатність = 0,963 точність = 0,97 особливості = 7



Зміст

- Рекомендовані джерела
- EA (GA) для вибору функцій — чому?
- Типи проблем для вибору функції:
- Регресія: проблема Фрідмана-1
  - Класичне рішення
  - Рішення EA (GA).
- Класифікація: проблема тварин
  - Класичне рішення
  - Рішення EA (GA).
- **Резюме**

ЕС для вибору функції — Резюме

ЕС (GA) можна ефективно застосувати до класичних проблем машинного навчання під керівництвом:

– **регресія**(використання проблеми регресії Фрідмана-1)

і

– **класифікація**(використання класифікації тварин набору даних UCI)

для

– **вибір функції**

або

– **зменшення розмірності**

з метою:

– **зменшення**MSE

або

– **збільшення середнього**точність.

Основи еволюційних обчислень

-

Основи еволюційного обчислення

-

Лекція 04. ЕС для машинного навчання

— Гіперпараметрична настройка

(за мотивами Холланда, Халеда Рашида, Бена Філіпса, Еяла Вірсанського та ін. праці)

Зміст

- **Рекомендовані джерела**
- EA (GA) для налаштування гіперпараметрів — чому?
- Типи проблем для вибору ознак
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - Гіперпараметрична настройка
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- Резюме

Рекомендовані джерела — Книги

Книги (наукові):

Гудфеллоу І., Бенгіо Ю., Курвіль А. та Бенгіо Ю. (2016). Глибоке навчання. Кембридж: MIT Press

Цитовано в 23692 джерелах.

Книги (з кодами на github):

Алан Фонтейн (2018). Освоєння інтелектуальної аналітики scikit-learn і TensorFlow. Packt Publishing.

Танай Агравал (2021). Оптимізація гіперпараметрів у машинному навчанні: зробіть своє машинне навчання глибоким. Ефективніші моделі навчання. Apress

Рекомендовані джерела -

Документи та набори даних

Приклад проблеми та набору даних

Набір даних UCI Wine(https://archive.ics.uci.edu/ml/datasets/wine)

С. Аберхард, Д. Куманс і О. де Вел,
Порівняння класифікаторів у параметрах високої розмірності, технічні
Представник № 92-02, (1992), кафедра комп'ютерних наук і кафедра математики і статистики, Університет Джеймса Кука Північного Квінсленда.

Дані були використані для порівняння різних класифікаторів.
(RDA: 100%, QDA 99,4%, LDA 98,9%, 1NN 96,1% (дані z-перетворення))

(Усі результати з використанням техніки «пропускання одного»)

Біленко Михайло(**Керівник відділу ШІ та досліджень Яндекс**) і Сугато Басу і Реймонд Дж. Муні.
Інтеграція обмежень і метричного навчання в напів-контрольована кластеризація
ICML. 2004 рік.

Камаль Алі та Майкл Дж. Паццані.
Зменшення помилок завдяки множинному навчанню
Описи
Машинне навчання, 24. 1996

Зміст

- Рекомендовані джерела
- **EA (GA) для налаштування гіперпараметрів — чому?**
- Типи проблем для вибору ознак
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - Гіперпараметрична настройка
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- Резюме

Еволюційні обчислення (ЕС) — для налаштування гіперпараметрів — чому?

Контрольоване навчання:

робочий процес :**модель**отримує набір**входи**, дзвонив**особливості**, і відображає їх у набір**виходи**.

Успенський : інформація, описана в**особливості**є**корисний для** визначення вартості відповід**виходи**. **Модель** : навчання є **коригування**(або налаштування).**внутрішні параметри**моделі для виробництва**бажані результати**у відповідь на**задані входи**. Для цього кожен тип моделі навчання під керівництвом супроводжується а **алгоритм навчання**що**ітеративно коригує**його внутрішній**параметри** під час фази навчання (або навчання).

Реальність :**АЛЕ**...більшість моделей мають інший набір**гіперпараметри**які встановлені**раніше**навчання, і вони впливають на спосіб навчання! **Зазвичай:** гіперпараметрів є деякі**значення за замовчуванням**що вступить в силу якщо ми спеціально не встановимо їх і**вони є не оптимальний!**

Тому нам потрібно**налаштування гіперпараметрів!**

ЕС для налаштування гіперпараметрів —

Вигоди та накладні витрати

Переваги:

-Зменшення похибок (втраченої функції) моделі

-Підвищення точності моделі

-Час навчання моделей коротший.

Накладні витрати:

-Можлива кількість комбінацій гіперпараметрів може бути дуже-дуже величезним.

-Пошук найкращих комбінацій гіперпараметрів (налаштування гіперпараметрів) займає значну кількість часу.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування гіперпараметрів — чому?
- **Типи проблем для вибору ознак**
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - Гіперпараметрична настройка
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- Резюме

ЕС для налаштування гіперпараметрів —

Тип проблеми - Класифікація

ЕС (GA) можна ефективно застосувати до класичних проблем машинного навчання під наглядом:

– **класифікація**(використання класифікації набору даних UCI Wine)

для

– **налаштування гіперпараметрів**

з метою:

– **зменшення** **MSE**

або

– **збільшення середнього** **точність**.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування гіперпараметрів — чому?
- Типи проблем для вибору ознак
- **Приклад задачі класифікації**
  - **Набір даних UCI Wine**
  - Гіперпараметрична настройка
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- Резюме

ЕС для налаштування гіперпараметрів —

Приклад: Проблема класифікації вина

Це класичний приклад проблеми класифікації.

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Wine Data Set

Download: Data Folder, Data Set Description

Abstract: Using chemical analysis determine the origin of wines

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1602802

Source:

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno,
16147 Genoa, Italy.

Набір даних UCI Wine (https://archive.ics.uci.edu/ml/datasets/wine)

ЕС для налаштування гіперпараметрів — Класифікація вина —Набір даних

Загальна інформація про набір даних:

Ці дані є результатами хімічного аналізу вин, вирощених там же регіон в Італії, але походить від**3 різних сорти**.

Аналіз визначив кількість**13 складових**знайдено в кожному з 3 види вина.

- У контексті класифікації це добре поставлена проблема з "добре поводився" класові структури.

- Хороший набір даних для першого тестування нового класифікатора, але не дуже складний.

-

EC для налаштування гіперпараметрів — Класифікація вина — Набір даних

Інформація про атрибут (функцію):

1) Алкоголь

2) Яблучна кислота

3) Зола

4) Лужність золи

5) Магній

6) Загальні феноли

7) Флавоноїди

8) Нефлавоноїдні феноли

9) Проантоціани

10) Інтенсивність кольору

11) Відтінок

12) OD280/OD315 розбавлених вин

13) Пролін

Ідентифікатор класу: Один (0-й) атрибут є ідентифікатором класу (1,2,3)

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування гіперпараметрів — чому?
- Типи проблем для вибору ознак
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - **Робочий процес і налаштування гіперпараметрів**
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- Резюме

ЕС для налаштування гіперпараметрів — Класифікація вина — проблема

Походження: це **класичний** приклад **проблема класифікації**, де вхідні функції повинні бути зіставлені **3 категорії**/мітки.

Вхідні дані: всі характеристики (властивості вина) є **безперервний**.

Виходи: одна особливість – **клас** – представляє 3 категорії (**культивари**).

Цілься: **поїзда** **модель класифікації** на цьому наборі даних с **13 особливостей** до **передбачити** значення **функція 0** (**культивар**).

Класифікація вина —

робочий процес

1) Навантаження вино UCI **набір даних** за стандартом **read_csv** функція (з url 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data').

2) Розділити дані в **введення** характеристики (перші 13 стовпців, що залишилися) і результуючі **вихід** категорія (перший стовпець). Тоді замість розділення даних на **1 тренування** встановити і **1 тест** set, як ми робили в попередньому прикладі, ми використовуємо **k-кратна перехресна перевірка** -> Дані розділені на **k** рівні частини і модель оцінюється **k** разів:

(k-1) частини для **навчання** і **1** частина, що залишилася для тестування (або **перевірка**).

3) Створити класифікація **модель** ...можна використовувати різні моделі... **AdaBoostClassifier** в цьому прикладі.

4) Визначити продуктивність використаної моделі регресії для набору вибрано **гіперпараметри** за **точність** **метрика***.

*) **Точність** – частина справ, які були класифіковані правильно. А **вище** значення цього вимірювання вказує **краща продуктивність** моделі.

Класифікація вина —

Гіперпараметрична настройка

Розглянемо детальніше цей етап:

3) Створитикласифікація**модель**...можна використовувати різні моделі...

AdaBoostClassifierв цьому прикладі.

The **адаптивний алгоритм підвищення** (**AdaBoost**) це потужна модель ML, яка **комбайнив** **виходи** множинних **екземпляри** простого алгоритму ML (слабкий навчається) із використанням зваженої суми. AdaBoost додає екземпляри слабкого учня під час процесу навчання, кожен з яких є налаштовано для покращення раніше неправильно класифікованих вхідних даних. Ми будемо використовувати sklearn бібліотечна реалізація AdaboostClassifier з деякими гіперпараметрами:

Name	Type	Description	Default Value
`n_estimators`	int	The maximum number of estimators	50
`learning_rate`	float	Can be used to shrink the contribution of each classifier	1
`algorithm`	{'SAMME', 'SAMME.R'}	'SAMME.R' – uses a real boosting algorithm, 'SAMME' – uses a discrete boosting algorithm	2

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування гіперпараметрів — чому?
- Типи проблем для вибору ознак
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - Гіперпараметрична настройка
- **Класичні рішення**
  - **ДЕМО 1 - Значення за замовчуванням**
  - **DEMO 2 – Розширений пошук у сітці**
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- Резюме

Класифікація вина —Класичний спосіб

Почнемо з двох класичних підходів:

-**значення за замовчуванням**гіперпараметрів моделі:

{'**алгоритм**': 'SAMME.R', '**швидкість_навчання**':**1.0**, '**n_estimators**':**50**,
'random_state': 42},

-пошук по сітціз**найкращі цінності**гіперпараметрів моделі:

Алгоритм—2 можливі значення «SAMME» і «SAMME.R»,

швидкість_навчання-10 значень із логарифмічним інтервалом
між 0,01 (10^{-2}) і 1 (10^{0}),

n_estimators->10 значень, лінійно розташованих між 10 і 100,

Всього: 200 = (10×10×2) різних комбінацій параметрів сітки.

Класифікація вина —

Класичний спосіб — ДЕМО 1 і 2

Результати:

ДЕМО 1- Значення за замовчуванням: Значення гіперпараметра класифікатора за замовчуванням:

{'algorithm': 'SAMME.R', 'base_estimator': Немає, 'learning_rate': 1,0, 'n_estimators': 50, 'random_state': 42} Оцінка (зі значеннями за замовчуванням) = 0,645714285714285 7

Час, що минув = 0,416749238967895 5

ДЕМО 2- Після gridSearch:

Найкращі параметри: {'algorithm': 'SAMME.R', 'learning_rate': 0,3593813663804626, 'n_estimators': 70}

Оцінка (після gridSearch): 0,9325842696629213

Час, що минув = 74,51628732681274

Спробуйте відтворити ці результати!

Класифікація вина —

Класичний спосіб — ДЕМО 1 і 2

Після навчання/тестування – див**test.gridTest()**функція в DEMO-коді:
- модель є**AdaBoostClassifier**-класифікатор

-5-кратна перехресна перевірка

Результати:

ДЕМО 1- Значення за замовчуванням:

Значення гіперпараметрів моделі:

{'algorithm': 'SAMME.R', 'learning_rate':**1.0**, 'n_estimators':**50**,
'random_state': 42}

Точність: 0,65%

Час, що минув =0,42секунд

ДЕМО 2- Після gridSearch: Найкращі значення гіперпараметрів:

{'algorithm': 'SAMME.R', 'learning_rate':**0,359**, 'n_estimators':**70**}

Точність: 0,93%

Час, що минув =74секунд Спробуйте відтворити ці результати!

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування гіперпараметрів — чому?
- Типи проблем для вибору ознак
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - Гіперпараметрична настройка
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- **Рішення EA (GA).**
  - **ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA**
  - **ДЕМО 4 — Direct GA**
- Резюме

Класифікація вина —EC (GA) Способи

Відмінність від класичних способів

The**відмінності**від**класичний**рішення:

1)**Хромосоми**—**неоднорідний**набори вибраних**значення**

гіперпараметри:

-**n_estimators значень**-список**10 цілих чисел**

-швидкість_навчання-ряд 10**плаває,**

-алгоритм-список**2 струни**

2)**Фітнес-функція (FF)**-повертає моделі**середня точність**

3)**Вибір**

- **турнір**відбір з турніром**розмір 2**

- елітарність, де**зал слави (HOF)**учасники – діючі**найкращий** особи –

є**завжди проходили недоторканими**до наступного покоління

4)**Еволюційні (генетичні) оператори**

- **кросовер**і

- мутаціяоператори

які спеціалізуються на хромосомах

Класифікація вина — EC (GA) Способи — Сітка та Direct

Можливі підходи на основі EC-GA:

-**Пошук у сітці на основі GA:**
шукатисередвспочатку вибрано200 комбінацій сітоктільки,

-**прямий**GA**:**
шукати безпосередньовесь простір параметрів**, де**
кожен гіперпараметр може бути представлений
як змінна, яка бере участь у пошуку,
іхромосомаможе бути апоєднаннязвсеці змінні.

Класифікація вина — Шляхи ЕС (GA).

3.Grid-пошук на основі GA — ДЕМО 3

3)**Пошук у сітці на основі GA:**

****

- - - Розвивайтеся в 200 можливих комбінаціях ---

	ген	nevals	сер	хв	макс	станд
0	20		0.708427	0.117978	0.910112	0.265992
1	13		0.865169	0.662921	0.926966	0.0717915
2	15		0.887921	0.646067	0.926966	0.0571676
3	12		0.896348	0.679775	0.926966	0.0526256
4	16		0.918539	0.88764	0.926966	0.0110233
5	9		0.911517	0.730337	0.926966	0.0425958

Найкраща особа: {'n_estimators': 60, 'learning_rate': 0,5994842503189409
'algorithm': 'SAMME.R'} з
відповідністю: 0,926966292134831
Час, що минув = 24,287983655929565

Спробуйте відтворити ці результати!

Класифікація вина — Шляхи ЕС (GA).

3.Grid-пошук на основі GA — ДЕМО 3

3)**Пошук у сітці на основі GA:**

шукати**серед**в**спочатку вибрано**200 комбінацій сіток**тільки**

GA-параметри:

популяція_size=20,

gene_mutation_prob=0,30,

розмір_турніру=2,

кількість поколінь=5

Результати:

Значення гіперпараметрів моделі:

{'algorithm': 'SAMME.R', 'learning_rate':**0,5995**, 'n_estimators': 6**0**,

'random_state': 42}

Точність: 0,93%

Час, що минув =24секунд протягом 6 поколінь (пор. зgridSearch:7

сек, але це займаєтільки 2покоління -8 секунд!- для досягнення максимальної точност

Спробуйте відтворити ці результати!

Класифікація вина —EC (GA) Ways 3.

Грід-пошук на основі GA — ДЕМО 3

Висновки:

Пошук у сітці за допомогою GA може знайти той самий найкращий результат

(знайдено класичним пошуком),

але в а**У 6 разів (!) швидше**–приблизно 12 секунд (2 покоління).

АЛЕ...в реальних ситуаціях:

-**набори даних**багато**більший**,

-**моделі**більше**складні**, і

-**сітки гіперпараметрів**є**більший**!

-Ось чому**вичерпний****класичний****пошук по сітці**може бути**непомірно довго**,

тоді як**керований GA****пошук по сітці**може досягти**хороші результати**

в межах а**розумний строк**.

АЛЕ тут...GA обмежуються підмножиною гіперпараметрів значення, які визначаються сіткою.

Давайте шукати за межами сітки підмножини попередньо визначених значень?

Класифікація вина —EC (GA) Способи

4.Direct GA — DEMO 4

4)**Прямий**GA**:**
шукати безпосередньо**весь простір параметрів, де**
кожен гіперпараметр може бути представлений
як змінна, яка бере участь у пошуку,
і**хромосома**може бути а**поєднання**з**все**ці змінні.

Нам потрібно представити кожен гіперпараметр як число з плаваючою комою,
незалежно від його фактичного типу:
-**n_estimators**-спочатку ан**ціле число**—він буде представлений а**плавати**
значення в**діапазон [1, 100]**,
-**швидкість_навчання**-вже а**плавати**, тому конвертація не потрібна — вона буде
прив'язаний до**діапазон [0,01, 1,0]**,
-алгоритм-мати одне з двох**рядок**значення, 'SAMME' або 'SAMME.R' -
це і буде представлено а**плавати**число в діапазоні**[0, 1]**.

Класифікація вина — Шляхи ЕС (GA).

4.Direct GA — DEMO 4

4)**Direct GA -Результати:**

****

ген	макс	середнє
0	20	0,92127	0,841024
1	14	0,943651	0,900603
2	13	0,943651	0,912841
3	14	0,943651	0,922476
4	15	0,949206	0,929468
5	13	0,949206	0,938563

Час, що минув = 46,6222867675781

- Найкраще рішення:

params = 'n_estimators'= 69, 'learning_rate'=0,628, 'algorithm'=SAMME.R

Точність = 0,94921

Спробуйте відтворити ці результати!

Класифікація вина — Шляхи ЕС (GA).

4.Direct GA — DEMO 4

4)**Прямий GA з GA-параметрами:**
популяція_size=20,
gene_mutation_prob=0,50,
ймовірність кросинговеру = 0,90,
розмір_турніру=2,
кількість поколінь=5
hall_of_fame_size=5

Результати:

Значення гіперпараметрів моделі:
{'алгоритм': '**CAMME.P**', 'learning_rate':**0,628**, 'n_estimators':**69**,
'random_state': 42}
Точність: 0,95%

Час, що минув =46секунд протягом 6 поколінь (пор. зgridSearch:74 се
це займаєтільки 2покоління -16 секунд!- до > Точність сітки GA)

Спробуйте відтворити ці результати!

Класифікація вина —EC (GA) Способи

4.Direct GA — DEMO 4

Висновки:

-Direct GA може знайти**краща точність 95%**
ніж класичний (65-93%) і пошук у сітці, керований GA (93%),

-і в**У 4-5 разів (!) швидше**(8 секунд для 2 поколінь), ніж класичний
той самий час для пошуку в сітці за допомогою GA.

ПРИМІТКА:в**найкращий**значення гіперпараметрів (для n_estimators і learning_rate).**за межами сітки**цінності!

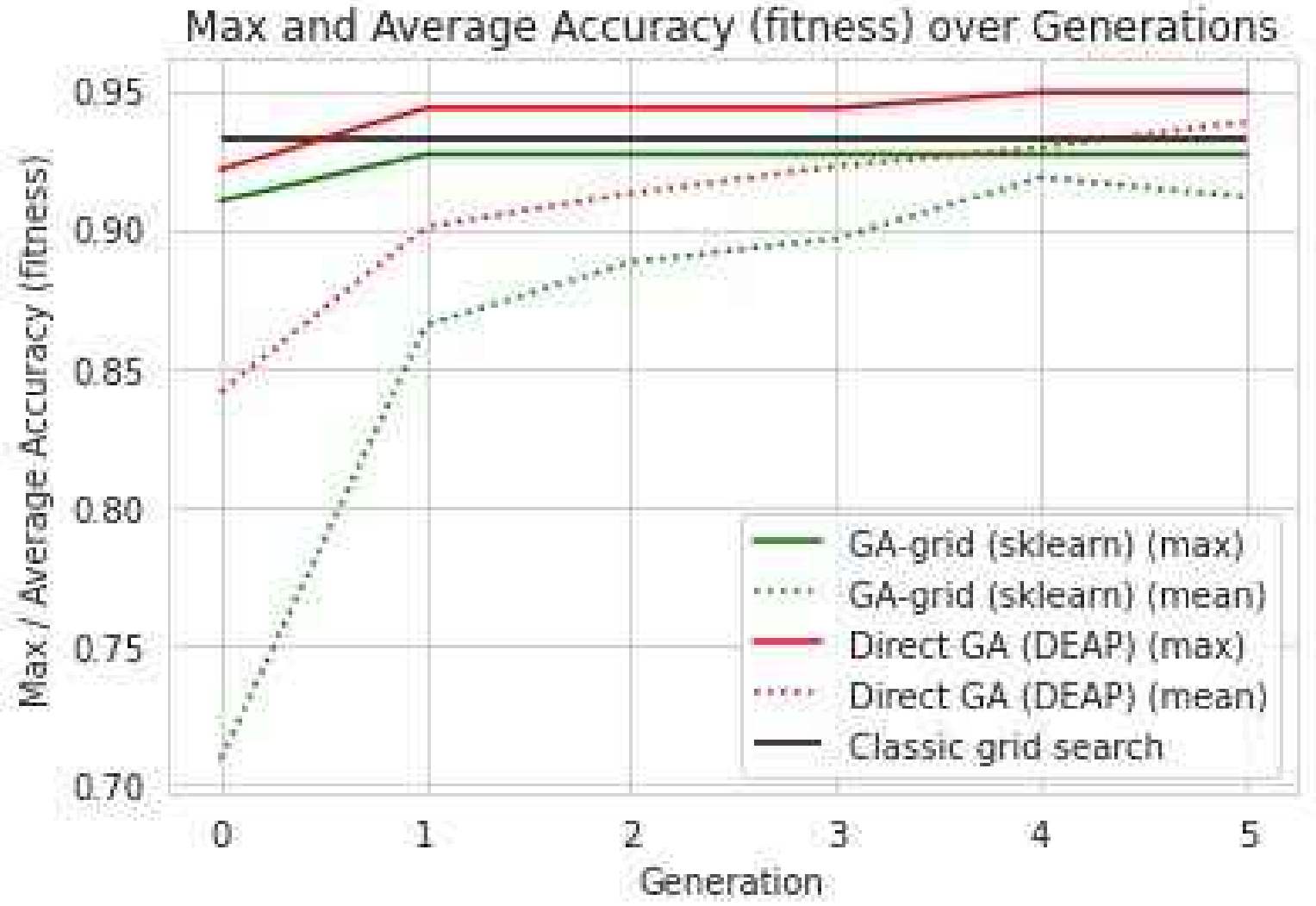
АЛЕ ... знову! ...в реальних ситуаціях:
-**набори даних**багато**більший**,
-**моделі**більше**складні**, і
-**сітки гіперпараметрів**є**більший**!

●Ось чому**вичерпний****класичний****пошук по сітці**може бути**непомірно довго**,
тоді як**керований GA****пошук по сітці**може досягти**хороші результати**
в межах а**розумний строк**.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування гіперпараметрів — чому?
- Типи проблем для вибору ознак
- Приклад задачі класифікації
  - Набір даних UCI Wine
  - Гіперпараметрична настройка
- Класичні рішення
  - ДЕМО 1 - Значення за замовчуванням
  - DEMO 2 - Розширений пошук у сітці
- Рішення EA (GA).
  - ДЕМОНСТРАЦІЯ 3 — Пошук на базі GA
  - ДЕМО 4 — Direct GA
- **Резюме**

ЕС для вибору ознак — Класифікація — Порівняльний сюжет



Спробуйте відтворити ці результати!

ЕС для вибору функції — Резюме

ЕС (GA) можна ефективно застосувати до класичних проблем машинного навчання під керівництвом:

– **регресія**(використання проблеми регресії Фрідмана-1)

і

– **класифікація**(використання класифікації тварин набору даних UCI)

для

– **вибір функції**

або

– **зменшення розмірності**

з метою:

– **зменшення MSE**

або

– **збільшення середнього точність**.

Основи еволюційних обчислень

-

Основи еволюційного обчислення

-

Лекція 05. ЕС для нейронних мереж — Архітектура та гіперпараметри - Тюнінг

(на основі праць Varoquaux, Grobler, Rasheed, Phillips, Вірсанського та ін.)

Зміст

- **Рекомендовані джерела**
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

Рекомендовані джерела — Книги

Книги (наукові):

Гудфеллоу І., Бенгіо Ю., Курвіль А. та Бенгіо Ю. (2016). Глибоке навчання. Кембридж: MIT Press

Цитовано в 23692 джерелах.

Книги (з кодами на github):

Алан Фонтейн (2018). Освоєння інтелектуальної аналітики scikit-learn і TensorFlow. Packt Publishing.

Танай Агравал (2021). Оптимізація гіперпараметрів у машинному навчанні: зробіть своє машинне навчання глибоким. Ефективніші моделі навчання. Apress

Рекомендовані джерела -

Документи та набори даних

Приклад проблеми та набору даних

Набір даних UCI Wine(https://archive.ics.uci.edu/ml/datasets/wine)
С. Аберхард, Д. Куманс і О. де Вел,
Порівняння класифікаторів у параметрах високої розмірності,
Представник № 92-02, (**1992 рік**), кафедра комп’ютерних наук і кафедра математики та статистики Університету Джеймса Кука Північного Квінсленда.

Набір даних UCI Iris(https://archive.ics.uci.edu/ml/datasets/iris) Фішер Р.А
Використання кількох вимірювань у таксономічних задачах,
Євгеніка, 7, частина II, 179-188 (**1936 рік**); також у «Внески до мат
Статистика» (Джон Вайлі, Нью-Йорк, 1950).

Набір даних UCI щодо раку молочної залози
(https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)) WN Street, WH Wolberg і OL Mangasarian,
пухлини молочної залози
Imaging: Science and Technology, том 1905, 861-870, Сан-Хосе, Каліфорнія,

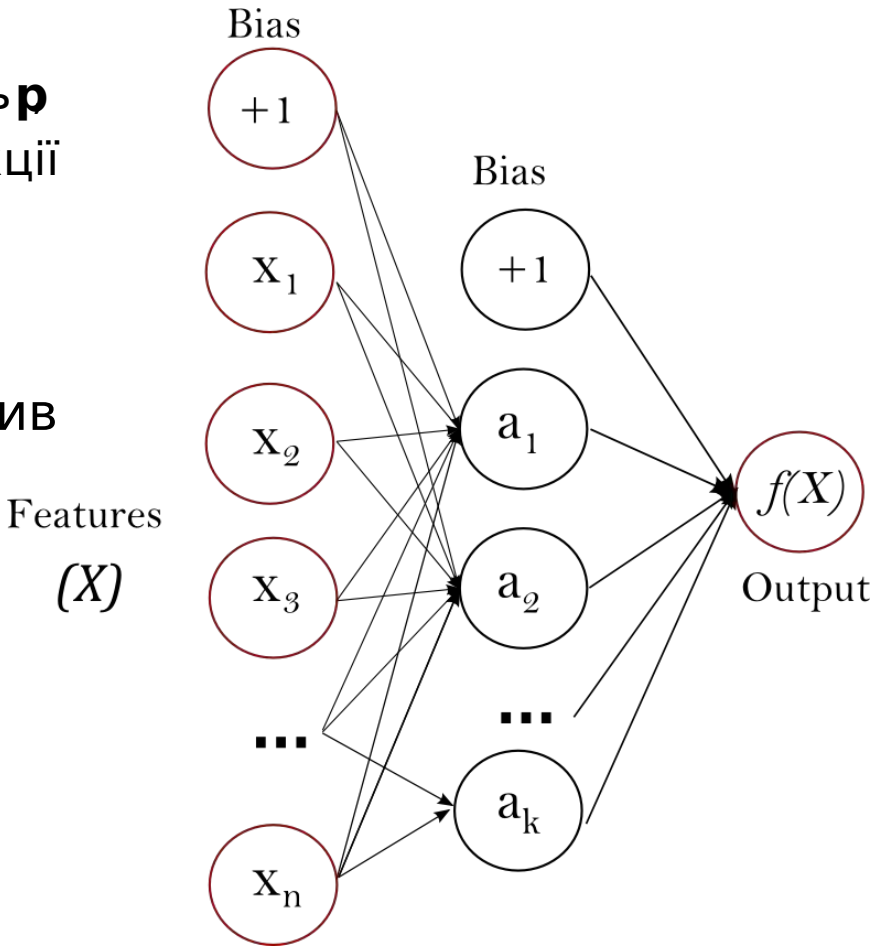
Зміст

- Рекомендовані джерела
- **EA (GA) для налаштування нейронної мережі (NN).**
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн гіперпараметрів
- Резюме

Еволюційні обчислення (EC) — MLP/ нейронна мережа (NN) — вступ

Багатошаровий персептрон (MLP) це контрольований алгоритм навчання (**штучна нейронна мережа - NN**), який вивчає функцію $f(X)$, навчаючись на наборі даних. Дано набір ознак $X = x_1, x_2, ..., x_n$ і ціль y, він може вивчати апроксиматор нелінійної функції для класифікації або регресії.

Між вхідним і вихідним шарами може бути один або **більше нелінійних шарів**, дзвонив **приховані шари**.



Матриця ваги W_i за деяким індексом i представляє вагові коефіцієнти між шаром i і шаром $i+1$. Упередженість за індексом i представляє упередження b_i значення, додані до шару $i+1$.

MLP використовує **зворотне поширення** для навчання. Це може **розрізняти не лінійно** роздільні дані.

NN Tuning —

Що таке об’єкти налаштування?

Контрольоване навчання:

робочий процес :**модель (NNтут)**отримує набір**входи**, дзвонив **особливості**і відображає їх у набір**виходи**.

Успенський : інформація, описана в**особливості**є**корисний для** визначення вартості відповід**виходи**. **Модель** : навчання є **коригування**(або налаштування).**внутрішні параметри (вагиу шар NN тут)**моделі для виробництва**бажані результати** у відповідь на**задані входи**. Кожен тип моделі навчання під наглядом є у супроводі а**лгоритм навчання**що**ітеративно коригує**його внутрішній**параметри (вагиу шарах NN тут)**під час навчання. І більшість моделей**(NNтут)**мати структуру (**Архітектура NNтут: шари, блоки та зв'язки між ними**) +**гіперпараметри** (швидкість навчання, ...), які встановлені**раніше**навчання, і вони впливають на нього!

Зазвичай: ЕК можна застосувати для пошуку оптимальних: а)**ваги**, б **гіперпараметри**(як у попередній лекції для ML), с)**архітектура**.

ВАЖЛИВО:Налаштування ваги за допомогою ЕСНЕ розглядаєтьсятут, оскіль виконується градієнтними методами.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- **Типи налаштування NN**
- Приклад проблеми класифікації: набір даних + робочий процес
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

NN Tuning —
Які види тюнінгу?

Типи налаштування ... для різних частин NN:

-**внутрішні параметри:**

вагив НН - є**НЕ розглядається**тут, бо його виконує градієнтні методи;

-зовнішні параметри:

1) NN**архітектура**(шари та вузли в шарах тут)

+ вплив різних ...

- RANDOM_SEEDs,

- **набори даних**,

- **МАКСИМАЛЬНА кількість шарів**.

2) NN**гіперпараметри**(швидкість навчання, функція активації, оптимізація розв’язувач і регулярізація тут),

3) NN**архітектура +**NN**гіперпараметри**.

ЕС для налаштування гіперпараметрів —

Вигоди та накладні витрати

Переваги:

-Зменшення похибок (втраченої функції) моделі
-Підвищення точності моделі
- Час навчання моделей коротший.

Накладні витрати:

- Можливе **номер** з **Архітектури NN** і **Гіперпараметр NN** комбінації можуть бути дуже-дуже **величезний**.
- Пошук найкращого **Архітектури NN** і **NN гіперпараметр** комбінації (гіперпараметрична настройка) приймає **значний** суми **час**.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- **Приклад задачі класифікації**
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ЕС для налаштування гіперпараметрів —

Тип проблеми - Класифікація

ЕС (GA) можна ефективно застосувати до класичних проблем машинного навчання під наглядом:

- **класифікація**(використання класифікації набору даних UCI Wine)

для

- **NN настройка**

з метою:

- **зменшення** **MSE**

або

- **збільшення середнього** **точність**.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- **Проблема класифікації:** **Набір даних** **+ Робочий процес**
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ЕС для налаштування гіперпараметрів —

Приклад: Проблема класифікації вина

Це класичний приклад проблеми класифікації.

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Wine Data Set

Download: Data Folder, Data Set Description

Abstract: Using chemical analysis determine the origin of wines

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1602802

Source:

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno,
16147 Genoa, Italy.

Набір даних UCI Wine (https://archive.ics.uci.edu/ml/datasets/wine)

ЕС для налаштування гіперпараметрів — Класифікація вина —Набір даних

Загальна інформація про набір даних:

Ці дані є результатами хімічного аналізу вин, вирощених там же регіон в Італії, але походить від**3 різних сорти**.

Аналіз визначив кількість**13 складових**знайдено в кожному з 3 види вина.

- У контексті класифікації це добре поставлена проблема з "добре поводився" класові структури.

- Хороший набір даних для першого тестування нового класифікатора, але не дуже складний.

-

ЕС для налаштування гіперпараметрів — Класифікація вина — Набір даних

Інформація про атрибут (функцію):

1) Алкоголь

2) Яблучна кислота

3) Зола

4) Лужність золи

5) Магній

6) Загальні феноли

7) Флавоноїди

8) Нефлавоноїдні феноли

9) Проантоціани

10) Інтенсивність кольору

11) Відтінок

12) OD280/OD315 розбавлених вин

13) Пролін

Ідентифікатор класу: Один (0-й) атрибут є ідентифікатором класу (1,2,3)

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- **Проблема класифікації: Dataset +** **робочий процес**
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ЕС для налаштування гіперпараметрів — Класифікація вина —робочий процес

Походження: це**класичний**приклад**проблема класифікації**, де вхідні функції повинні бути зіставлені**3 категорії**/мітки.

Вхідні дані: всі характеристики (властивості вина) є**безперервний**.

Виходи: одна особливість –**клас**– представляє 3 категорії (**культивари**).

Цілься:**поїзда****модель класифікації**на цьому наборі даних с**13 особливостей** до**передбачити**значення**функція 0**(**культивар**).

NN Tuning —

Класифікація вина —робочий процес

1) Навантаженнявино UCI**набір даних**за стандартом**read_csv**функція (з url 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data').

2) Розділитидані в**введення**характеристики (перші 13 стовпців, що залишилися) і результуючі**вихід**категорія (перший стовпець). Тоді замість розділення даних на**1 тренування**встановити і**1 тест**set, як ми робили в попередньому прикладі, ми використовуємо**k-кратна перехресна перевірка**дані -> розділені на**k**рівні частини і модель оцінюється**k**разів:

(k-1)частини для**навчання**і**1**частина, що залишилася для тестування (або**перевірка**).

3) Створитикласифікація**модель**...можна використовувати різні моделі...

Багатошаровий персептрон (MLP)в цьому прикладі.

4) Визначити продуктивністьвикористаної моделі регресії для набору вибрано**гіперпараметри**за**точність****метрика***.

*)**Точність**–частина справ, які були класифіковані правильно. А**вище**значення цього вимірювання вказує**краща продуктивність**моделі.

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- **ДЕМО – Частина 1: Рішення для налаштування архітектури NN**
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ДЕМО — Частина 1: Налаштування архітектури NN — Шари та вузли

Ми обмежуємо NN до **4** прихованих шарів, хромосома буде:

[n_1, n_2, n_3, n_4]

тут, **n_i** позначає кількість вузлів у шарі **i**, від 1 до 4. Щоб контролювати кількість прихованих шарів у NN, деякий **n_i** може бути 0 або <0 ... це означає -> **більше немає шарів** буде додано до НН.

Приклад деяких хромосом:

[10, 20, **-5**, 15] -> кортеж (10, 20) оскільки **-5** завершує підрахунок шарів

[10, **0**, -5, 15] -> кортеж (10,) оскільки **0** завершує підрахунок шарів

[10, 20, 5, **-15**] -> кортеж (10, 20, 5), оскільки **-15** закінчує відлік.

[10, 20, 5, 15] > кортеж (10, 20, 5, 15).

ДЕМО — Частина 1: Налаштування архітектури N

— Шари та вузли

Ми обмежуємо NN до **4** прихованих шарів, хромосома буде:

[n_1, n_2, n_3, n_4]

тут, **n_i** позначає кількість вузлів у шарі **i**, від 1 до 4. Щоб контролювати кількість прихованих шарів у NN, деякий **n_i** може бути 0 або <0 ... це означає -> **більше немає шарів** буде додано до НН.

Приклад деяких хромосом:

[**10**, 20, -5, 15] -> кортеж (10, 20), оскільки -5 завершує підрахунок шарів.

[**10**, 0, -5, 15] -> кортеж (10,), оскільки 0 закінчує підрахунок шарів. [**10**, 20, 5, -15] -> кортеж (10, 20, 5), оскільки -15 завершує підрахунок.

[**10**, 20, 5, 15] > кортеж (10, 20, 5, 15).

до **гарантія** що є **хоча б 1 прихований шар**, **1-й** параметр (**10** ось **завжди >0**.

The **інші параметри шару** може мати **різні розподіли** близько 0 ... чому ... контролювати свої шанси стати лідером кінцеві параметри.

ДЕМО -Частина 1:

Рішення для налаштування архітектури NN

Результати:

ДЕМО 1— Значення гіперпараметрів MLP за замовчуванням.

```
**************************
gen     nevals  max             avg
0       20      0.769841        0.284063
1       17      0.769841        0.473413
2       15      0.769841        0.606905
3       16      0.769841        0.659238
4       17      0.769841        0.673444
5       14      0.769841        0.703746
6       17      0.769841        0.739619
7       14      0.769841        0.70954
8       16      0.769841        0.686921
9       17      0.769841        0.689833
10      15      0.769841        0.680286
Time Elapsed =  82.1906521320343
Best solution is:  'hidden_layer_sizes'=(13, 4, 7)
Accuracy = 0.76984
```

Спробуйте відтворити ці результати!

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- **ДЕМО – Частина 1: Рішення для налаштування архітектури NN**
  - **Випадкове насіння**
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення

Що таке вплив...

- **Випадкове насіння**
- Набір даних (вино, ірис, рак молочної залози)
- Макс. номер шару NN

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення —Різні випадкові насіння?

Результати для різних RANDOM_SEED:

```
# dataset = 'wine'
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.769841  0.284063
1   17      0.769841  0.473413
2   15      0.769841  0.606905
3   16      0.769841  0.659238
4   17      0.769841  0.673444
5   14      0.769841  0.703746
6   17      0.769841  0.739619
7   14      0.769841  0.70954
8   16      0.769841  0.686921
9   17      0.769841  0.689833
10  15      0.769841  0.680286
Best solution is:  'hidden_layer_sizes'=(13, 4, 7)
Accuracy =  0.76984
```

Спробуйте відтворити ці результати!

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення —Різні випадкові насіння?

Результати для різних RANDOM_SEED:

```
# dataset = 'wine'
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.769841  0.284063
1   17      0.769841  0.473413
2   15      0.769841  0.606905
3   16      0.769841  0.659238
4   17      0.769841  0.673444
5   14      0.769841  0.703746
6   17      0.769841  0.739619
7   14      0.769841  0.70954
8   16      0.769841  0.686921
9   17      0.769841  0.689833
10  15      0.769841  0.680286
Best solution is:  'hidden_layer_sizes'=(13, 4, 7)
Accuracy =  0.76984
```

```
# dataset = 'wine'
# RANDOM_SEED = 666

**************************
gen nevals  max       avg
0   20      0.647937  0.31354
1   17      0.647937  0.41869
2   15      0.647937  0.478095
3   16      0.647937  0.418651
4   17      0.647937  0.503325
5   12      0.647937  0.492421
6   17      0.647937  0.435524
7   16      0.647937  0.503032
8   16      0.647937  0.466016
9   16      0.647937  0.51246
10  17      0.647937  0.572524
Time Elapsed =  93.69340062141418
Best solution is:  'hidden_layer_sizes'=(14, 3, 4, 4)
Accuracy = 0.64794
```

Спробуйте відтворити ці результати!

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення —Різні випадкові насіння?

Результати для різних RANDOM_SEED:

```
# dataset = 'wine'
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.769841  0.284063
1   17      0.769841  0.473413
2   15      0.769841  0.606905
3   16      0.769841  0.659238
4   17      0.769841  0.673444
5   14      0.769841  0.703746
6   17      0.769841  0.739619
7   14      0.769841  0.70954
8   16      0.769841  0.686921
9   17      0.769841  0.689833
10  15      0.769841  0.680286
Best solution is:  'hidden_layer_sizes'=(13, 4, 7)
Accuracy =  0.76984
```

```
# dataset = 'wine'
# RANDOM_SEED = 666

**************************
gen nevals  max       avg
0   20      0.647937  0.31354
1   17      0.647937  0.41869
2   15      0.647937  0.478095
3   16      0.647937  0.418651
4   17      0.647937  0.503325
5   12      0.647937  0.492421
6   17      0.647937  0.435524
7   16      0.647937  0.503032
8   16      0.647937  0.466016
9   16      0.647937  0.51246
10  17      0.647937  0.572524
Time Elapsed =  93.69340062141418
Best solution is:  'hidden_layer_sizes'=(14, 3, 4, 4)
Accuracy = 0.64794
```

```
# dataset = 'wine'
# RANDOM_SEED = 1042
**************************
gen nevals  max       avg
0   20      0.520159  0.289151
1   15      0.520159  0.322095
2   12      0.520159  0.42246
3   17      0.541587  0.419079
4   14      0.541587  0.41527
5   17      0.541587  0.471929
6   15      0.541587  0.457198
7   16      0.541587  0.472865
8   17      0.541587  0.493143
9   16      0.541587  0.45669
10  13      0.541587  0.488968
Time Elapsed =  84.34443235397339
Best solution is:  'hidden_layer_sizes'=(9, 9, 5)
Accuracy = 0.54159
```

Спробуйте відтворити ці результати!

ДЕМО – Частина 1: Рішення для налаштування архітектури NN —Різні випадкові насіння - резюме

Резюме для різних RANDOM_SEED:

Для різних RANDOM_SEED ми можемо отримати NN з дуже різними:

продуктивність (точність),

- **кількість вузлів у шарах,**
- **кількість шарів.**

The**можлива причина**є в**стохастичний**(т. зв**неградієнтні**) спосіб**зміна параметра** під час еволюції.

Є певна ймовірність того усі ці**моделі**для різних RANDOM_SEED може**досяжність**різні **місцевий**(+ глобальний) the**максимум**значення**функція фітнесу**(**точність** тут).

Спробуйте відтворити ці результати!

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- **ДЕМО – Частина 1: Рішення для налаштування архітектури NN**
  - Випадкове насіння
  - **Набір даних (вино, ірис, рак молочної залози)**
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення

Що таке вплив...

- Випадкове насіння
- **Набір даних (вино, ірис, рак молочної залози)**
- Макс. номер шару NN

Приклад налаштування NN:

Набір даних Wine

Це класичний приклад проблеми класифікації.

UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Wine Data Set

Download: Data Folder, Data Set Description

Abstract: Using chemical analysis determine the origin of wines

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1602802

Source:

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno,
16147 Genoa, Italy.

Набір даних UCI Wine (https://archive.ics.uci.edu/ml/datasets/wine)

Приклад налаштування NN:

Набір даних Iris

Це класичний приклад проблеми класифікації.

UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Iris Data Set

Download: Data Folder, Data Set Description

Abstract: Famous database; from Fisher, 1936

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	3847949

Набір даних UCI Iris (https://archive.ics.uci.edu/ml/datasets/iris)

Приклад налаштування NN:

Набір даних про рак молочної залози

Це класичний приклад проблеми класифікації.

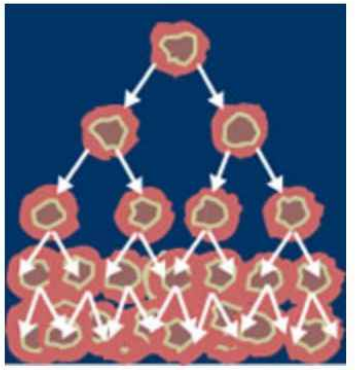
UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Breast Cancer Wisconsin (Diagnostic) Data Set

Download: Data Folder, Data Set Description

Abstract: Diagnostic Wisconsin Breast Cancer Database



Data Set Characteristics:	Multivariate	**Number of Instances:**	569	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	32	**Date Donated**	1995-11-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1444676

Набір даних UCI щодо раку молочної залози

(https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення —Різні набори даних?

Результати для різних RANDOM_SEED:

```
### wine
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.769841  0.284063
1   17      0.769841  0.473413
2   15      0.769841  0.606905
3   16      0.769841  0.659238
4   17      0.769841  0.673444
5   14      0.769841  0.703746
6   17      0.769841  0.739619
7   14      0.769841  0.70954
8   16      0.769841  0.686921
9   17      0.769841  0.689833
10  15      0.769841  0.680286
- Best solution is:  'hidden_layer_sizes'=(13, 4, 7) , accuracy =  0.7698412698412699
```

Спробуйте відтворити ці результати!

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення —Різні набори даних?

Результати для різних RANDOM_SEED:

```
### wine
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.769841  0.284063
1   17      0.769841  0.473413
2   15      0.769841  0.606905
3   16      0.769841  0.659238
4   17      0.769841  0.673444
5   14      0.769841  0.703746
6   17      0.769841  0.739619
7   14      0.769841  0.70954
8   16      0.769841  0.686921
9   17      0.769841  0.689833
10  15      0.769841  0.680286
- Best solution is:  'hidden_layer_sizes'=(13, 4, 7) , accuracy =  0.7698412698412699
```

```
### iris
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.666667  0.416333
1   17      0.693333  0.487
2   15      0.76      0.537333
3   14      0.76      0.550667
4   17      0.76      0.568333
5   17      0.76      0.653667
6   14      0.76      0.589333
7   15      0.76      0.618
8   16      0.866667  0.616667
9   16      0.866667  0.666333
10  16      0.866667  0.722667
- Best solution is:  'hidden_layer_sizes'=(15, 5, 8) , accuracy =  0.8666
```

Спробуйте відтворити ці результати!

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення —Різні набори даних?

Результати для різних RANDOM_SEED:

```
### wine
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.769841  0.284063
1   17      0.769841  0.473413
2   15      0.769841  0.606905
3   16      0.769841  0.659238
4   17      0.769841  0.673444
5   14      0.769841  0.703746
6   17      0.769841  0.739619
7   14      0.769841  0.70954
8   16      0.769841  0.686921
9   17      0.769841  0.689833
10  15      0.769841  0.680286
- Best solution is:  'hidden_layer_sizes'=(13, 4, 7) , accuracy =  0.7698412698412699
```

```
### iris
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.666667  0.416333
1   17      0.693333  0.487
2   15      0.76      0.537333
3   14      0.76      0.550667
4   17      0.76      0.568333
5   17      0.76      0.653667
6   14      0.76      0.589333
7   15      0.76      0.618
8   16      0.866667  0.616667
9   16      0.866667  0.666333
10  16      0.866667  0.722667
- Best solution is:  'hidden_layer_sizes'=(15, 5, 8) , accuracy =  0.8666
```

```
### breast_cancer
# RANDOM_SEED = 42

gen nevals  max       avg
0   20      0.927946  0.808865
1   15      0.929669  0.889953
2   15      0.929669  0.893562
3   15      0.929669  0.893683
4   16      0.934932  0.839395
5   17      0.934932  0.912204
6   14      0.934932  0.895351
7   16      0.934932  0.908839
8   17      0.934932  0.900869
9   16      0.934932  0.845574
10  15      0.934932  0.900429
- Best solution is:  'hidden_layer_sizes'=(15, 8, 10, 4) , accuracy =  0.9349
```

Спробуйте відтворити ці результати!

ДЕМО – Частина 1: Рішення для налаштування архітектури NN —Різні набори даних - резюме

Резюме для різних наборів даних:

Знову...для різних наборів даних ми можемо отримати NN з дуже різними:

продуктивність (точність),

- **кількість вузлів у шарах,**
- **кількість шарів.**

The**можлива причина**є

тут більш очевидно:

- **різні функції,**
- **різне число**особливостей,
- їх**різний внесок** до функції пристосованості (точність).

Спробуйте відтворити ці результати!

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- **ДЕМО – Частина 1: Рішення для налаштування архітектури NN**
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - **Макс. номер шару NN**
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ДЕМО – Частина 1: Налаштування архітектури NN

Рішення

Що таке вплив...

- Випадкове насіння
- Набір даних (вино, ірис, рак молочної залози)
- **Макс. номер шару NN**

ДЕМО – Частина 1: Рішення для налаштування архітектури NN —Різний номер шару MAX?

Результати для різних MAX Layer Number:

АЛЕ

. . .

спробуйте це як самостійне навчання

. . .

якщо хочеш! :)

Спробуйте відтворити ці результати!

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- **ДЕМО – Частина 2: Налаштування гіперпараметрів NN**
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- Резюме

ДЕМО - Частина 2:NN Гіперпараметричне налаштування

Для попереднього 1) NN Architecture Tuning

ми використали гіперпараметри за замовчуванням.

АЛЕ...з попередньої лекції ... налаштування різних гіперпараметрів може підвищення продуктивності класифікатора.

Q:Чи можемо ми тут використати налаштування гіперпараметрів?**A:**Так.

Від sklearn реалізації MLP ми можемо використовувати численні настроювані гіперпараметри:

Name	Type	Description	Default value
`activation`	`{'tanh', 'relu', 'logistic'}`	Activation function for the hidden layers	`'relu'`
`solver`	`{'sgd', 'adam', 'lbfgs'}`	The solver for weight optimization	`'adam'`
`alpha`	`float`	Regularization term parameter	`0.0001`
`learning_rate`	`{'constant', 'invscaling', 'adaptive'}`	Learning rate schedule for weight updates	`'constant'`

ДЕМО - Частина 2: NN Гіперпараметричне налаштування

Як і в попередніх демонстраціях лекцій,
представлення хромосоми з плаваючою комою дозволяє нам об’єднувати різні типи гіперпараметрів у процес оптимізації на основі GA.

активація-одне з трьох значень: tanh, relu, або логістичний.

Цього можна досягти, представивши його як число з плаваючою точкою в діапазоні [0, 2,99] .
Щоб перетворити float в одне з вищезгаданих рядкових значень, нам потрібно застосувати floor() до нього, яка дасть 0, 1 або 2.
Потім замінюємо 0 -> tanh, 1 -> relu, і 2 -> логістичний.

розв'язувач-одне з 3 значень: sgd, Адам, або lbfgs.

Подобається для **активація**: його можна представити за допомогою числа з плаваючою точкою в діапазоні [0, 2,99].

альфа-вже float, перетворення не потрібне.
Його буде прив’язано до діапазону [0,0001, 2,0].

швидкість_навчання-одне з 3 значень: постійний, invscaling, адаптивний. Подобається для **активація**: його можна представити за допомогою числа з плаваючою точкою в діапазоні [0, 2,99].

ДЕМО - Частина 2:NN Гіперпараметричне налаштування

Результати:

ДЕМО 2- Найкраща архітектура NN з DEMO 1.
HIDDEN_LAYER_SIZES =[13, 4, 7]

```
**************************
gen     nevals  max             avg
0       20      0.946667        0.362
1       15      0.946667        0.599667
2       16      0.946667        0.864333
3       16      0.946667        0.927333
4       17      0.946667        0.944667
5       14      0.946667        0.887
6       15      0.946667        0.944667
7       14      0.946667        0.946
8       16      0.946667        0.907667
9       15      0.946667        0.945
10      17      0.946667        0.946
Time Elapsed =  92.3740484714508
Best solution is:  'activation'='logistic'
 'solver'='lbfgs'
 'alpha'=0.17139833879055075
 'learning_rate'='invscaling'
Accuracy = 0.94667
```

Спробуйте відтворити ці результати!

Зміст

- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- **ДЕМО – Частина 3: Архітектура NN + Рішення для налаштування гіперпараметрів NN**
- Резюме

ДЕМО - Частина 3: Гібрид: NN Architecture +

NN Гіперпараметричне налаштування

Перші 4 діапазони для налаштування NN Archictecture

– > по одному для кожного прихованого шару.

Наступні 4 діапазони

- > представляють додаткові 4 гіперпараметри.

```
# 'hidden_layer_sizes': first four values
# 'activation':    0..2.99
# 'solver':        0..2.99
# 'alpha':    0.0001..2.0
# 'learning_rate': 0..2.99
BOUNDS_LOW =  [ 5,  -5, -10, -20, 0,     0,     0.0001, 0    ]
BOUNDS_HIGH = [15,  10,  10,  10, 2.999, 2.999, 2.0,    2.999]
```

ДЕМО - Частина 3: Гібрид: NN Architecture +

NN Гіперпараметричне налаштування

Вхідні межі:

```
# boundaries for all parameters:
# 'hidden_layer_sizes': first four values
# 'activation': ['tanh', 'relu', 'logistic'] -> 0, 1, 2
# 'solver': ['sgd', 'adam', 'lbfgs'] -> 0, 1, 2
# 'alpha': float in the range of [0.0001, 2.0],
# 'learning_rate': ['constant', 'invscaling', 'adaptive'] -> 0, 1, 2
BOUNDS_LOW =  [ 5,  -5, -10, -20, 0,     0,     0.0001, 0    ]
BOUNDS_HIGH = [15,  10,  10,  10, 2.999, 2.999, 2.0,    2.999]
```

ДЕМО - Частина 3: Гібрид: NN Architecture + NN Гіперпараметричне налаштування

Результати:

```
**************************
gen     nevals  max     avg
0       20      0.94    0.448
1       16      0.94    0.633
2       15      0.94    0.737667
3       16      0.946667        0.842
4       17      0.946667        0.889667
5       15      0.946667        0.937667
6       16      0.946667        0.939
7       16      0.946667        0.875
8       16      0.946667        0.876333
9       14      0.946667        0.942333
10      16      0.946667        0.902667
Time Elapsed =  83.84976434707642
Best solution is:  'hidden_layer_sizes'=(8, 7)
 'activation'='relu'
 'solver'='lbfgs'
 'alpha'=0.563775972907702
 'learning_rate'='adaptive'
Accuracy = 0.94667
```

Спробуйте відтворити ці результати!

ДЕМО - Частина 3: Гібрид: NN Architecture + NN Гіперпараметричне налаштування

Результати -> Порівняйте ТІЛЬКИ з NN Hyper

```
**************************
gen     nevals  max     avg
0       20      0.94    0.448
1       16      0.94    0.633
2       15      0.94    0.737667
3       16      0.946667        0.842
4       17      0.946667        0.889667
5       15      0.946667        0.937667
6       16      0.946667        0.939
7       16      0.946667        0.875
8       16      0.946667        0.876333
9       14      0.946667        0.942333
10      16      0.946667        0.902667
Time Elapsed =  83.84976434707642
Best solution is:  'hidden_layer_sizes'=(8, 7)
 'activation'='relu'
 'solver'='lbfgs'
 'alpha'=0.563775972907702
 'learning_rate'='adaptive'
Accuracy = 0.94667
```

```
**************************
gen     nevals  max             avg
0       20      0.946667        0.362
1       15      0.946667        0.599667
2       16      0.946667        0.864333
3       16      0.946667        0.927333
4       17      0.946667        0.944667
5       14      0.946667        0.887
6       15      0.946667        0.944667
7       14      0.946667        0.946
8       16      0.946667        0.907667
9       15      0.946667        0.945
10      17      0.946667        0.946
Time Elapsed =  92.3740484714508
Best solution is:  'activation'='logistic'
 'solver'='lbfgs'
 'alpha'=0.17139833879055075
 'learning_rate'='invscaling'
Accuracy = 0.94667
```

Смішні відмінності в параметрах найкращого рішення ... :)

Спробуйте відтворити ці результати!

Зміст

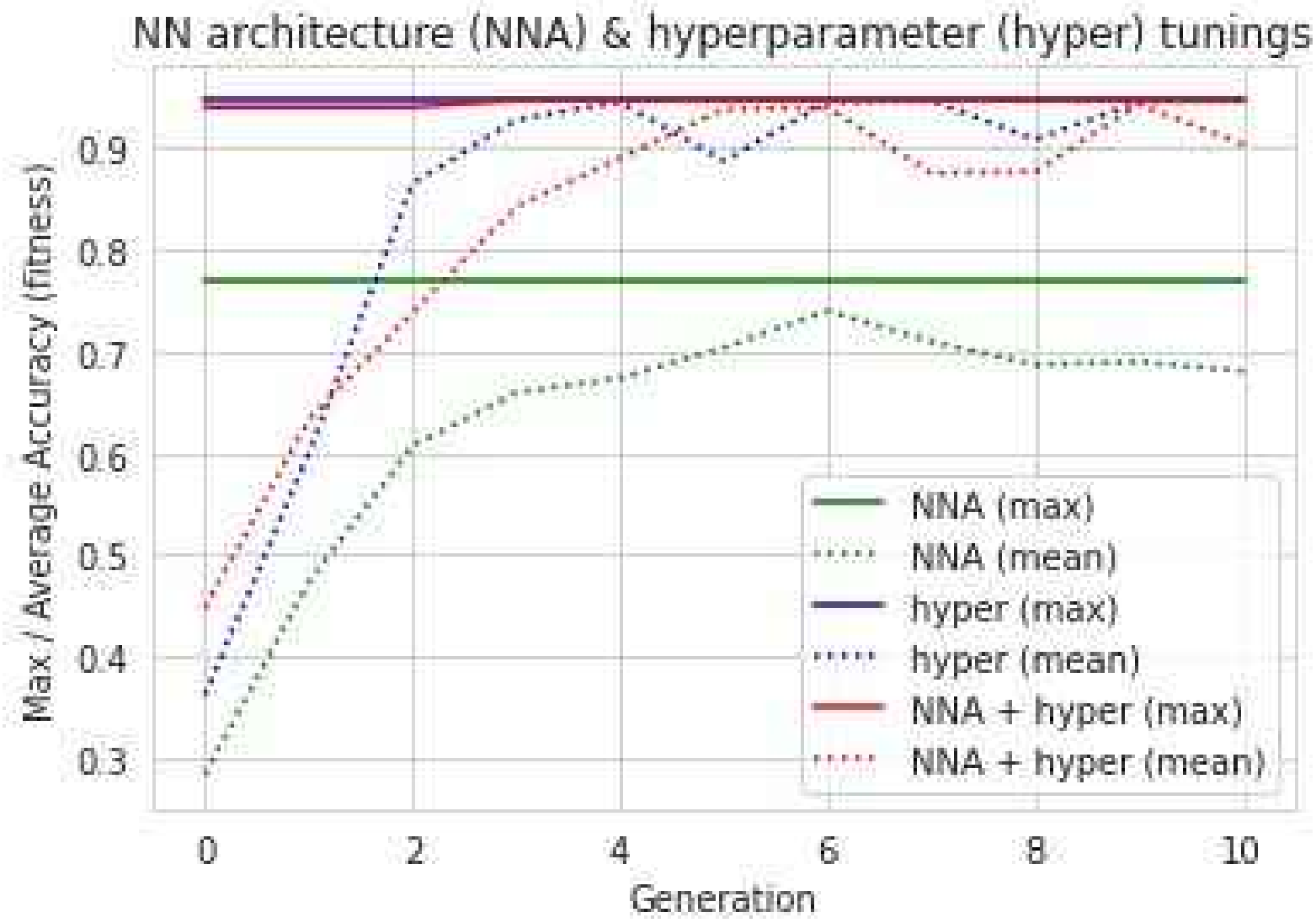
- Рекомендовані джерела
- EA (GA) для налаштування нейронної мережі (NN).
- Типи налаштування NN
- Приклад проблеми класифікації: набір даних + робочий процес
- ДЕМО – Частина 1: Рішення для налаштування архітектури NN
  - Випадкове насіння
  - Набір даних (вино, ірис, рак молочної залози)
  - Макс. номер шару NN
- ДЕМО – Частина 2: Рішення для налаштування гіперпараметрів NN
- ДЕМО – Частина 3: Архітектура NN + Рішення для налаштуванн

гіперпараметрів

- **Резюме**

EC для архітектури NN + гіперпараметр N

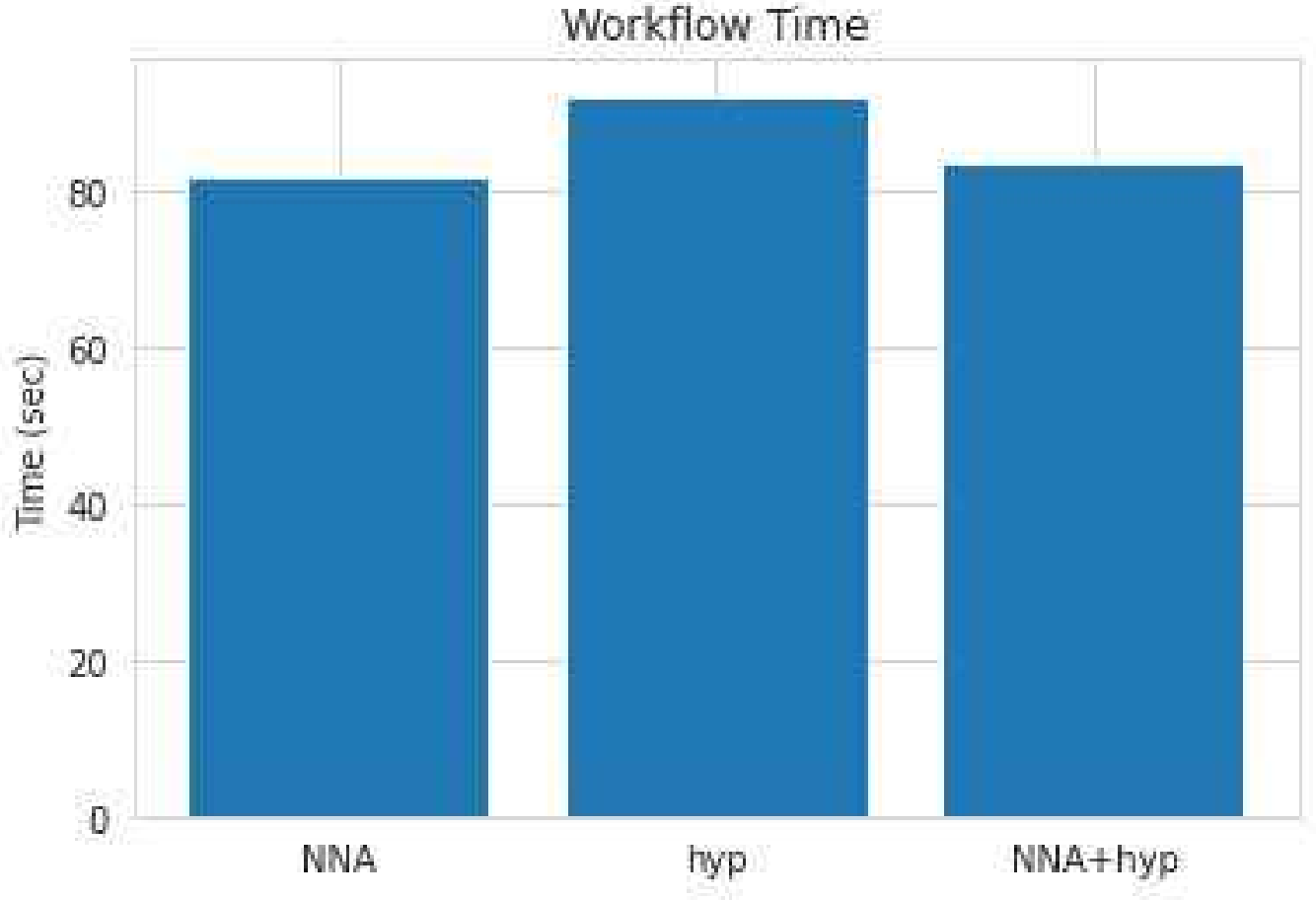
Тюнінг —Порівняльний сюжет — Точність



Спробуйте відтворити ці результати!

EC для архітектури NN + гіперпараметр N

Тюнінг — Порівняльний сюжет — Час



Неважливо ... в такій постановці задачі ... але ...

Спробуйте відтворити ці результати!

ЕС для вибору функції — Резюме

ЕС (GA) можна ефективно застосувати до класичних проблем машинного навчання під керівництвом:

– **регресія**(використання проблеми регресії Фрідмана-1)

і

– **класифікація**(використання класифікації тварин набору даних UCI)

для

– **вибір функції**

або

– **зменшення розмірності**

з метою:

– **зменшення MSE**

або

– **збільшення середнього точність**.

Основи еволюційних обчислень (EA).

Лекція 6 - Платформа OpenAI Gym

на основі (C) роботи OpenAI, Heaton, Moore, Varoquaux, Grobler, Wirsansky

Короткий зміст:

- Платформа OpenAI Gym
- *Проблеми навчання з підкріпленням (RL):*
  - MountainCar-v0,
  - MountainCarContinuous-v0,
  - CartPole-v1
  - ...
- Функції візуалізації сценаріїв ігор Gym у Colab.

Установка та імпорт бібліотек

Бібліотека для підтримки алгоритмів RL

Gym — це інструментарій для розробки та порівняння алгоритмів навчання з підкріпленням.

Він підтримує навчання агентів усьому: від ходьби до ігор, таких як Pong або Pinball.

```
! pip install gym
```

```
Requirement already satisfied: gym in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyglet<=1.5.0,>=1.4.0 in /usr/local/lib/pythor
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cloudpickle<1.7.0,>=1.2.0 in /usr/local/lib/py
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packag
```

Бібліотеки для візуалізації середовищ OpenAI Gym у Colab

Дії, які виконуються в Gym (грі, у яку грає ваш агент), можна візуалізувати навіть у Colab. У цьому розділі міститься інформація про те, як створити відео в Colab, яке показує епізод гри, у яку грає ваш агент.

```
%%time
!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
```

```
CPU times: user 28.9 ms, sys: 18.1 ms, total: 47 ms
Wall time: 11.6 s
```

```
%%time
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
```

```
Collecting setuptools
  Downloading https://files.pythonhosted.org/packages/60/6a/dd9533a
     |████████████████████████████████| 788kB 7.3MB/s
ERROR: datascience 0.10.6 has requirement folium==0.2.1, but you'll
Installing collected packages: setuptools
  Found existing installation: setuptools 54.0.0
    Uninstalling setuptools-54.0.0:
      Successfully uninstalled setuptools-54.0.0
Successfully installed setuptools-54.1.1
CPU times: user 73.7 ms, sys: 44.5 ms, total: 118 ms
```

IMPORTANT: you should restart runtime!

Part 1.Introduction to the OpenAI Gym

Gym - Advanages and Limitations

OpenAI Gym aims to provide an easy-to-setup general-intelligence benchmark with a wide variety of differentt environments. The goal is to standardize how environments are defined in AI research publications so that published research becomes more easily reproducible. The project claims to provide the user with a simple interface.

OpenAI gym is pip-installed onto your local machine. There are a few significant limitations to be aware of:

- developers can only use Gym with Python (as of June 2017).
- OpenAI Gym can not directly render animated games in Google CoLab.

Because OpenAI Gym requires a graphics display, the only way to display Gym in Google CoLab is an embedded video. The presentation of OpenAI Gym game animations in Google CoLab is discussed later in this module.

Gym - Leaderboard

The OpenAI Gym does have a leaderboard, similar to Kaggle; however, the OpenAI Gym's leaderboard is much more informal compared to Kaggle. The user's local machine performs all scoring. As a result, the OpenAI gym's leaderboard is strictly an "honor's system." The leaderboard is maintained the following GitHub repository:

- OpenAI Gym Leaderboard

If you submit a score, you are required to provide a writeup with sufficient instructions to reproduce your result. A video of your results is suggested, but not required.

Gym - Environments

The centerpiece of Gym is the environment, which defines the "game" in which your reinforcement algorithm will compete. An environment does not need to be a game; however, it describes the following game-like features:

- **action space**: What actions can we take on the environment, at each step/episode, to alter the environment.
- **observation space**: What is the current state of the portion of the environment that we can observe. Usually, we can see the entire environment.

Gym - Basic Termonology

- **Agent** - The machine learning program or model that controls the actions. Step - One round of issuing actions that affect the observation space.
- **Episode** - A collection of steps that terminates when the agent fails to meet the environment's objective, or the episode reaches the maximum number of allowed steps.
- **Render** - Gym can render one frame for display after each episode.
- **Reward** - A positive reinforcement that can occur at the end of each episode, after the agent acts.
- **Nondeterministic** - For some environments, randomness is a factor in deciding what effects actions have on reward and changes to the observation space.

It is important to note that many of the gym environments specify that they are not nondeterministic even though they make use of random numbers to process actions. It is generally agreed upon (based on the gym GitHub issue tracker) that nondeterministic property

means that a deterministic environment will still behave randomly even when given consistent seed value. The seed method of an environment can be used by the program to seed the random number generator for the environment.

Environment - Attributes

The Gym library allows us to query some of these attributes from environments. I created the following function to query gym environments.

```
import gym

def query_environment(name):
  env = gym.make(name)
  spec = gym.spec(name)
  print(f"Action Space: {env.action_space}")
  print(f"Observation Space: {env.observation_space}")
  print(f"Max Episode Steps: {spec.max_episode_steps}")
  print(f"Nondeterministic: {spec.nondeterministic}")
  print(f"Reward Range: {env.reward_range}")
  print(f"Reward Threshold: {spec.reward_threshold}")
```

Environment - Examples:

- MountainCar-v0,
- MountainCarContinuous-v0,
- CartPole-v1
- ...

MountainCar-v0

We will begin by looking at the MountainCar-v0 environment, which challenges an underpowered car to escape the valley between two mountains. The following code describes the Mountian Car environment.

```
query_environment("MountainCar-v0")
```

```
Action Space: Discrete(3)
Observation Space: Box(-1.2000000476837158, 0.6000000238418579, (2,), float32
Max Episode Steps: 200
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: -110.0
```

Actions

There are three distinct actions that can be taken:

- accelerate forward,
- decelerate,
- accelerate backwards.

Observation space

The observation space contains two continuous (floating point) values, as evident by the box object.

The observation space contains:

- the position and
- velocity of the car.

The car has 200 steps to escape for each episode.

Reward: The mountian car recieves NO incremental reward. The only reward for the car is given when it escapes the valley.

MountainCarContinuous-v0

There is also a continuous variant of the mountain car. This version does not simply have the motor on or off. For the continuous car the action space is a single floating point number that specifies how much forward or backward force is being applied.

```
query_environment("MountainCarContinuous-v0")
```

```
Action Space: Box(-1.0, 1.0, (1,), float32)
Observation Space: Box(-1.2000000476837158, 0.6000000238418579, (2,), float32
Max Episode Steps: 999
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 90.0
```

Note: ignore the warning above, it is a relativly inconsequential bug in OpenAI Gym.

CartPole-v1

The CartPole-v1 environment challenges the agent to move a cart while keeping a pole balanced.

Observation space

The environment has an observation space of 4 continuous numbers:

- Cart Position
- Cart Velocity
- Pole Angle
- Pole Velocity At Tip

Actions

To achieve this goal, the agent can take the following actions:

- Push cart to the left
- Push cart to the right

```
query_environment("CartPole-v1")
```

```
Action Space: Discrete(2)
Observation Space: Box(-3.4028234663852886e+38, 3.4028234663852886e+38, (4,),
Max Episode Steps: 500
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 475.0
```

Breakout-v0

Atari games, like breakout can use an observation space that is either equal to the size of the Atari screen (210x160) or even use the RAM memory of the Atari (128 bytes) to determine the state of the game. Yes thats bytes, not kilobytes!

```
query_environment("Breakout-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (210, 160, 3), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

Breakout-ram-v0

```
query_environment("Breakout-ram-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (128,), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

Atlantis-v0

```
query_environment("Atlantis-v0")
```

```
     Action Space: Discrete(4)
     Observation Space: Box(0, 255, (210, 160, 3), uint8)
     Max Episode Steps: 10000
     Nondeterministic: False
     Reward Range: (-inf, inf)
     Reward Threshold: None
```

Part 2.Functions to visualize Gym-game-scenarios in Colab

Next we define functions used to show the video by adding it to the Colab notebook.

```
import gym
from gym.wrappers import Monitor
import glob
import io
import base64
from IPython.display import HTML
from pyvirtualdisplay import Display
from IPython import display as ipythondisplay

display = Display(visible=0, size=(1400, 900))
display.start()

"""
Utility functions to enable video recording of gym environment
and displaying it.
To enable video, just do "env = wrap_env(env)""
"""

def show_video():
  mp4list = glob.glob('video/*.mp4')
  if len(mp4list) > 0:
    mp4 = mp4list[0]
    video = io.open(mp4, 'r+b').read()
    encoded = base64.b64encode(video)
    ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
  else:
    print("Could not find video")


def wrap_env(env):
  env = Monitor(env, './video', force=True)
  return env
```

Now we are ready to play the game. We use a simple random agent.

MountainCar-v0

```
env = wrap_env(gym.make("MountainCar-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

MountainCarContinuous-v0

```
env = wrap_env(gym.make("MountainCarContinuous-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

CartPole-v1

```
env = wrap_env(gym.make("CartPole-v1"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
```

```
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

Breakout-v0

```
env = wrap_env(gym.make("Breakout-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;
```

```
env.close()
show_video()
```

Breakout-ram-v0

```
env = wrap_env(gym.make("Breakout-ram-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

Atlantis-v0

```
env = wrap_env(gym.make("Atlantis-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

0:00 / 1:11

Colab paid products - Cancel contracts here

Лекція 7 - Застосування EA для навчання з підкріпленням

на основі (C) роботи OpenAI, Heaton, Moore, Varoquaux, Grobler, Wirsansky

Короткий зміст:

- Встановлення DEAP (**кожного разу після запуску Colab VM!**),
- компоненти, необхідні для робочого процесу GA,
- *Проблеми навчання з підкріпленням (RL):*
    - MountainCar-v0,
    - MountainCarContinuous-v0,
    - CartPole-v1
    - ...
- порівняння продуктивності (точність і час роботи).

До кінця цієї лекції ви будете знати:

- знову ж таки, як використовувати вбудовані алгоритми структури DEAP для створення стислого коду
- як вирішити проблему *Reinforcement Learning* за допомогою рішень на основі GA для пошуку рішень,
- як експериментувати з різними налаштуваннями GA та інтерпретувати відмінності в результатах.

```
```

Get Figures for Text Description

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
! cp -r /content/drive/MyDrive/COLAB_EVO/EVO_Lecture07_CartPole/figures .
! ls figures
```

```
MLPRegressor.png
```

Installation and import of libraries

```
! pip install deap
```

```
Collecting deap
  Downloading https://files.pythonhosted.org/packages/99/d1/803c7a387d8a7e686
     |████████████████████████████████| 163kB 6.1MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
Installing collected packages: deap
Successfully installed deap-1.3.1
```

Library to support RL algorithms

Gym is a toolkit for developing and comparing reinforcement learning algorithms.

It supports teaching agents everything from walking to playing games like Pong or Pinball.

```
! pip install gym
```

```
Requirement already satisfied: gym in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cloudpickle<1.7.0,>=1.2.0 in /usr/local/lib/py
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.7/dist
Requirement already satisfied: pyglet<=1.5.0,>=1.4.0 in /usr/local/lib/python
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packag
```

Libraries to Render OpenAI Gym Environments in Colab

It is possible to visualize the activities performed in Gym (game your agent is playing), even on Colab. This section provides information on how to generate a video in Colab that shows you an episode of the game your agent is playing.

```
%%time
!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
```

```
CPU times: user 47 ms, sys: 13 ms, total: 59.9 ms
Wall time: 14.1 s
```

```
%%time
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
```

```
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
```

```
Collecting setuptools
  Downloading https://files.pythonhosted.org/packages/60/6a/dd9533ae9367a1f3
     |████████████████████████████████| 788kB 4.2MB/s
ERROR: datascience 0.10.6 has requirement folium==0.2.1, but you'll have foli
Installing collected packages: setuptools
  Found existing installation: setuptools 54.0.0
    Uninstalling setuptools-54.0.0:
      Successfully uninstalled setuptools-54.0.0
Successfully installed setuptools-54.1.1
CPU times: user 94.1 ms, sys: 46.3 ms, total: 140 ms
Wall time: 28.6 s
```

IMPORTANT: you should restart runtime!

Part 1.Introduction to the OpenAI Gym

Gym - Advanages and Limitations

OpenAI Gym aims to provide an easy-to-setup general-intelligence benchmark with a wide variety of differentt environments. The goal is to standardize how environments are defined in AI research publications so that published research becomes more easily reproducible. The project claims to provide the user with a simple interface.

OpenAI gym is pip-installed onto your local machine. There are a few significant limitations to be aware of:

- developers can only use Gym with Python (as of June 2017).
- OpenAI Gym can not directly render animated games in Google CoLab.

Because OpenAI Gym requires a graphics display, the only way to display Gym in Google CoLab is an embedded video. The presentation of OpenAI Gym game animations in Google CoLab is discussed later in this module.

Gym - Leaderboard

The OpenAI Gym does have a leaderboard, similar to Kaggle; however, the OpenAI Gym's leaderboard is much more informal compared to Kaggle. The user's local machine performs all

scoring. As a result, the OpenAI gym's leaderboard is strictly an "honor's system." The leaderboard is maintained the following GitHub repository:

- OpenAI Gym Leaderboard

If you submit a score, you are required to provide a writeup with sufficient instructions to

Gym - Environments

The centerpiece of Gym is the environment, which defines the "game" in which your reinforcement algorithm will compete. An environment does not need to be a game; however, it describes the following game-like features:

- **action space**: What actions can we take on the environment, at each step/episode, to alter the environment.
- **observation space**: What is the current state of the portion of the environment that we can observe. Usually, we can see the entire environment.

Gym - Basic Termonology

- **Agent** - The machine learning program or model that controls the actions. Step - One round of issuing actions that affect the observation space.
- **Episode** - A collection of steps that terminates when the agent fails to meet the environment's objective, or the episode reaches the maximum number of allowed steps.
- **Render** - Gym can render one frame for display after each episode.
- **Reward** - A positive reinforcement that can occur at the end of each episode, after the agent acts.
- **Nondeterministic** - For some environments, randomness is a factor in deciding what effects actions have on reward and changes to the observation space.

It is important to note that many of the gym environments specify that they are not nondeterministic even though they make use of random numbers to process actions. It is generally agreed upon (based on the gym GitHub issue tracker) that nondeterministic property means that a deterministic environment will still behave randomly even when given consistent seed value. The seed method of an environment can be used by the program to seed the random number generator for the environment.

Environment - Attributes

The Gym library allows us to query some of these attributes from environments. I created the following function to query gym environments.

```
import gym
```

```
def query_environment(name):
  env = gym.make(name)
  spec = gym.spec(name)
  print(f"Action Space: {env.action_space}")
  print(f"Observation Space: {env.observation_space}")
  print(f"Max Episode Steps: {spec.max_episode_steps}")
  print(f"Nondeterministic: {spec.nondeterministic}")
  print(f"Reward Range: {env.reward_range}")
  print(f"Reward Threshold: {spec.reward_threshold}")
```

Environment - Examples:

- MountainCar-v0,
- MountainCarContinuous-v0,
- CartPole-v1
- ...

MountainCar-v0

We will begin by looking at the MountainCar-v0 environment, which challenges an underpowered car to escape the valley between two mountains. The following code describes the Mountian Car environment.

```
query_environment("MountainCar-v0")
```

```
Action Space: Discrete(3)
Observation Space: Box(-1.2000000476837158, 0.6000000238418579, (2,), float32
Max Episode Steps: 200
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: -110.0
```

Actions

There are three distinct actions that can be taken:

- accelerate forward,
- decelerate,
- accelerate backwards.

Observation space

The observation space contains two continuous (floating point) values, as evident by the box object.

The observation space contains:

- the position and
- velocity of the car.

The car has 200 steps to escape for each episode.

Reward: The mountian car recieves NO incremental reward. The only reward for the car is given when it escapes the valley.

MountainCarContinuous-v0

There is also a continuous variant of the mountain car. This version does not simply have the motor on or off. For the continuous car the action space is a single floating point number that specifies how much forward or backward force is being applied.

```
query_environment("MountainCarContinuous-v0")
```

```
Action Space: Box(-1.0, 1.0, (1,), float32)
Observation Space: Box(-1.2000000476837158, 0.6000000238418579, (2,), float32
Max Episode Steps: 999
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 90.0
```

Note: ignore the warning above, it is a relativly inconsequential bug in OpenAI Gym.

CartPole-v1

The CartPole-v1 environment challenges the agent to move a cart while keeping a pole balanced.

Observation space

The environment has an observation space of 4 continuous numbers:

- Cart Position
- Cart Velocity
- Pole Angle
- Pole Velocity At Tip

Actions

To achieve this goal, the agent can take the following actions:

- Push cart to the left
- Push cart to the right

```
```

```
query_environment("CartPole-v1")
```

```
Action Space: Discrete(2)
Observation Space: Box(-3.4028234663852886e+38, 3.4028234663852886e+38, (4,),
Max Episode Steps: 500
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 475.0
```

Breakout-v0

Atari games, like breakout can use an observation space that is either equal to the size of the Atari screen (210x160) or even use the RAM memory of the Atari (128 bytes) to determine the state of the game. Yes thats bytes, not kilobytes!

```
query_environment("Breakout-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (210, 160, 3), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

Breakout-ram-v0

```
query_environment("Breakout-ram-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (128,), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

Atlantis-v0

```
query_environment("Atlantis-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (210, 160, 3), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

Functions to visualize game-scenarios in Colab

Next we define functions used to show the video by adding it to the Colab notebook.

```
import gym
from gym.wrappers import Monitor
import glob
import io
import base64
from IPython.display import HTML
from pyvirtualdisplay import Display
from IPython import display as ipythondisplay

display = Display(visible=0, size=(1400, 900))
display.start()

"""
Utility functions to enable video recording of gym environment
and displaying it.
To enable video, just do "env = wrap_env(env)""
"""

def show_video():
  mp4list = glob.glob('video/*.mp4')
  if len(mp4list) > 0:
    mp4 = mp4list[0]
    video = io.open(mp4, 'r+b').read()
    encoded = base64.b64encode(video)
    ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
  else:
    print("Could not find video")


def wrap_env(env):
  env = Monitor(env, './video', force=True)
  return env
```

Now we are ready to play the game. We use a simple random agent.

MountainCar-v0

```
env = wrap_env(gym.make("MountainCar-v0"))

observation = env.reset()

while True:

    env.render()
```

```
    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

MountainCarContinuous-v0

```
env = wrap_env(gym.make("MountainCarContinuous-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
```

```
      break;

env.close()
show_video()
```

CartPole-v1

```
env = wrap_env(gym.make("CartPole-v1"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

Breakout-v0

```
env = wrap_env(gym.make("Breakout-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

Breakout-ram-v0

```
env = wrap_env(gym.make("Breakout-ram-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

Atlantis-v0

```
env = wrap_env(gym.make("Atlantis-v0"))

observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)


    if done:
      break;

env.close()
show_video()
```

Part 3. GA Solution for RL problem - CartPole-v1

CartPole-v1 - Problem Description

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over.

Reward A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson:

AG Barto, RS Sutton and CW Anderson, *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem*, IEEE Transactions on Systems, Man, and Cybernetics, 1983.

Cited in 4063 sources.

Let's try it as a self-guided learning!

Use the following CartPole-v1 resources:

- description at Gym,
- Python-codes at github

Import Python libraries

In these and other lectures, we will use various Python packages:

- NumPy
- Matplotlib
- Seaborn

They are already pre-installed in Colab. Let's import them by the following code.

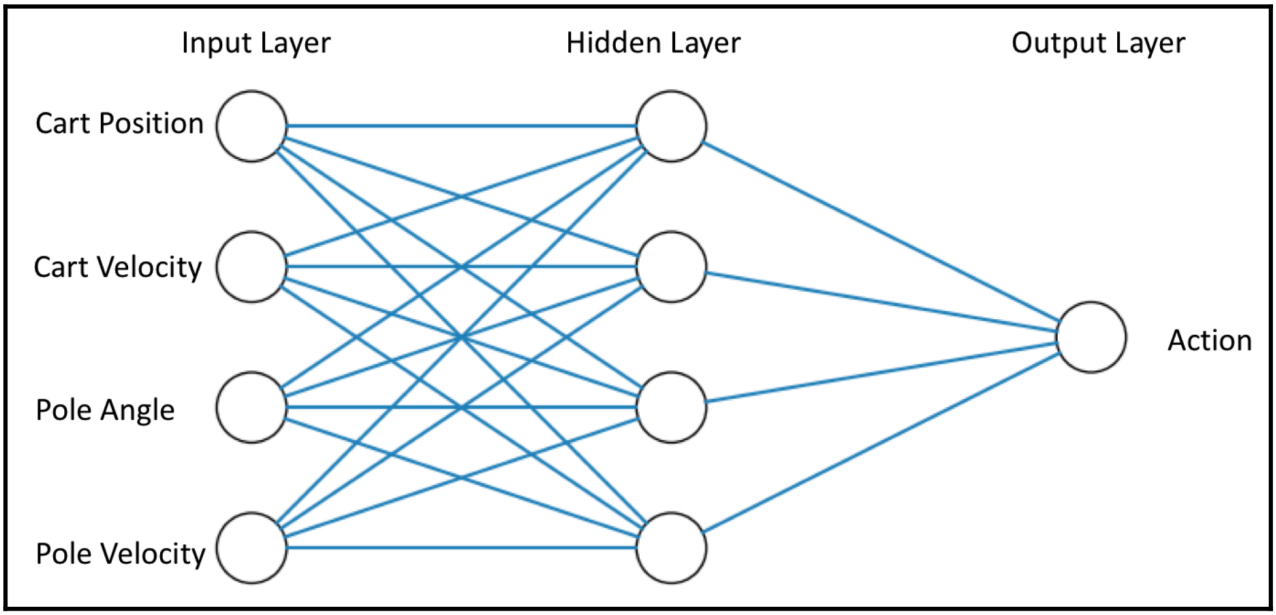
```
import gym
import time
import pickle
import random
import numpy

# for plotting
import matplotlib.pyplot as plt
import seaborn as sns
```

```
! rm -r ./video
! rm ./*pickle
```

```
rm: cannot remove './video': No such file or directory
rm: cannot remove './*pickle': No such file or directory
```

Actors - CartPole

```
from IPython.display import Image
Image('./figures/MLPRegressor.png')
```




```
import gym
import time

import numpy as np
import pickle

from sklearn.neural_network import MLPRegressor

from sklearn.exceptions import ConvergenceWarning
from sklearn.utils.testing import ignore_warnings

INPUTS = 4
HIDDEN_LAYER = 4
OUTPUTS = 1

class CartPole:

    def __init__(self, randomSeed=None):
```

```
        #self.env = gym.make('CartPole-v1')
        self.env = wrap_env(gym.make('CartPole-v1'))
        self.env.seed(randomSeed)

        if randomSeed is not None:
            self.env.seed(randomSeed)

    def __len__(self):
        return INPUTS * HIDDEN_LAYER + HIDDEN_LAYER * OUTPUTS + HIDDEN_LAYER + OUT

    @ignore_warnings(category=ConvergenceWarning)
    def initMlp(self, netParams):
        """
        initializes a MultiLayer Perceptron (MLP) Regressor with the desired netwo
        and network parameters (weights and biases).
        :param netParams: a list of floats representing the network parameters (we
        :return: initialized MLP Regressor
        """

        # create the initial MLP:
        mlp = MLPRegressor(hidden_layer_sizes=(HIDDEN_LAYER,), max_iter=1)

        # This will initialize input and output layers, and nodes weights and bias
        # we are not otherwise interested in training the MLP here, hence the sett
        mlp.fit(np.random.uniform(low=-1, high=1, size=INPUTS).reshape(1, -1), np.

        # weights are represented as a list of 2 ndarrays:
        # - hidden layer weights: INPUTS x HIDDEN_LAYER
        # - output layer weights: HIDDEN_LAYER x OUTPUTS
        numWeights = INPUTS * HIDDEN_LAYER + HIDDEN_LAYER * OUTPUTS
        weights = np.array(netParams[:numWeights])
        mlp.coefs_ = [
            weights[0:INPUTS * HIDDEN_LAYER].reshape((INPUTS, HIDDEN_LAYER)),
            weights[INPUTS * HIDDEN_LAYER:].reshape((HIDDEN_LAYER, OUTPUTS))
        ]

        # biases are represented as a list of 2 ndarrays:
        # - hidden layer biases: HIDDEN_LAYER x 1
        # - output layer biases: OUTPUTS x 1
        biases = np.array(netParams[numWeights:])
        mlp.intercepts_ = [biases[:HIDDEN_LAYER], biases[HIDDEN_LAYER:]]

        return mlp

    def getScore(self, netParams):
        """
        calculates the score of a given solution, represented by the list of float
        by creating a corresponding MLP Regressor, initiating an episode of the Ca
        running it with the MLP controlling the actions, while using the observati
        Higher score is better.
        :param netParams: a list of floats representing the network parameters (we
        :return: the calculated score value
        """
```

```
        mlp = self.initMlp(netParams)

        self.env.reset()

        actionCounter = 0
        totalReward = 0
        observation = self.env.reset()
        action = int(mlp.predict(observation.reshape(1, -1)) > 0)

        while True:
            actionCounter += 1
            observation, reward, done, info = self.env.step(action)
            totalReward += reward

            if done:
                break
            else:
                action = int(mlp.predict(observation.reshape(1, -1)) > 0)
                #print(action)

        return totalReward

    def saveParams(self, netParams):
        """
        serializes and saves a list of network parameters using pickle
        :param netParams: a list of floats representing the network parameters (we
        """
        savedParams = []
        for param in netParams:
            savedParams.append(param)

        pickle.dump(savedParams, open("cart-pole-data.pickle", "wb"))

    def replayWithSavedParams(self):
        """
        deserializes a saved list of network parameters and uses it to replay an e
        """
        savedParams = pickle.load(open("cart-pole-data.pickle", "rb"))
        self.replay(savedParams)

    def replay(self, netParams):
        """
        renders the environment and uses the given network parameters to replay an
        :param netParams: a list of floats representing the network parameters (we
        """
        mlp = self.initMlp(netParams)

        self.env.render()

        actionCounter = 0
        totalReward = 0
        observation = self.env.reset()
        action = int(mlp.predict(observation.reshape(1, -1)) > 0)

        while True:
```

```
            actionCounter += 1
            self.env.render()
            observation, reward, done, info = self.env.step(action)
            totalReward += reward

            print(actionCounter, ": -------------------------")
            print("action = ", action)
            print("observation = ", observation)
            print("reward = ", reward)
            print("totalReward = ", totalReward)
            print("done = ", done)
            print()

            if done:
                break
            else:
                time.sleep(0.03)
                action = int(mlp.predict(observation.reshape(1, -1)) > 0)

        self.env.close()

    def replayVideo(self):
        #self.env.close()
        show_video()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: Futu
  warnings.warn(message, FutureWarning)
```

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

# Create the instance of the MountainCar class:
cartPole = CartPole(RANDOM_SEED)

NUM_OF_PARAMS = len(cartPole)
# boundaries for layer size parameters:

# weight and bias values are bound between -1 and 1:
BOUNDS_LOW, BOUNDS_HIGH = -1.0, 1.0  # boundaries for all dimensions
```

GA Solution

```
from deap import base
from deap import creator
from deap import tools
from deap import algorithms
```

```
# Genetic Algorithm constants:
```

```
POPULATION_SIZE = 100
P_CROSSOVER = 0.9  # probability for crossover
P_MUTATION = 0.5   # probability for mutating an individual
MAX_GENERATIONS = 40
HALL_OF_FAME_SIZE = 3
CROWDING_FACTOR = 10.0  # crowding factor for crossover and mutation
```

Genetic Tools

```
toolbox = base.Toolbox()

# define a single objective, maximizing fitness strategy:
creator.create("FitnessMax", base.Fitness, weights=(1.0,))

# create the Individual class based on list:
creator.create("Individual", list, fitness=creator.FitnessMax)

# helper function for creating random real numbers uniformly distributed within a
# it assumes that the range is the same for every dimension
def randomFloat(low, up):
    return [random.uniform(l, u) for l, u in zip([low] * NUM_OF_PARAMS, [up] * NUM

# create an operator that randomly returns a float in the desired range:
toolbox.register("attrFloat", randomFloat, BOUNDS_LOW, BOUNDS_HIGH)

# create an operator that fills up an Individual instance:
toolbox.register("individualCreator",
                 tools.initIterate,
                 creator.Individual,
                 toolbox.attrFloat)

# create an operator that generates a list of individuals:
toolbox.register("populationCreator",
                 tools.initRepeat,
                 list,
                 toolbox.individualCreator)


# fitness calculation using the CrtPole class:
def score(individual):
    return cartPole.getScore(individual),


toolbox.register("evaluate", score)


# genetic operators:
toolbox.register("select", tools.selTournament, tournsize=2)

toolbox.register("mate",
                 tools.cxSimulatedBinaryBounded,
                 low=BOUNDS_LOW,
```

```
                 up=BOUNDS_HIGH,
                 eta=CROWDING_FACTOR)

toolbox.register("mutate",
                 tools.mutPolynomialBounded,
                 low=BOUNDS_LOW,
                 up=BOUNDS_HIGH,
                 eta=CROWDING_FACTOR,
                 indpb=1.0/NUM_OF_PARAMS)
```

```
/usr/local/lib/python3.7/dist-packages/deap/creator.py:141: RuntimeWarning: A
  RuntimeWarning)
/usr/local/lib/python3.7/dist-packages/deap/creator.py:141: RuntimeWarning: A
  RuntimeWarning)
```

Elitism Tools

```
def eaSimpleWithElitism(population, toolbox, cxpb, mutpb, ngen, stats=None,
             halloffame=None, verbose=__debug__):
    """This algorithm is similar to DEAP eaSimple() algorithm, with the modificati
    halloffame is used to implement an elitism mechanism. The individuals containe
    halloffame are directly injected into the next generation and are not subject
    genetic operators of selection, crossover and mutation.
    """
    logbook = tools.Logbook()
    logbook.header = ['gen', 'nevals'] + (stats.fields if stats else [])

    # Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in population if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    if halloffame is None:
        raise ValueError("halloffame parameter must not be empty!")

    halloffame.update(population)
    hof_size = len(halloffame.items) if halloffame.items else 0

    record = stats.compile(population) if stats else {}
    logbook.record(gen=0, nevals=len(invalid_ind), **record)
    if verbose:
        print(logbook.stream)

    # Begin the generational process
    for gen in range(1, ngen + 1):

        # Select the next generation individuals
        offspring = toolbox.select(population, len(population) - hof_size)

        # Vary the pool of individuals
        offspring = algorithms.varAnd(offspring, toolbox, cxpb, mutpb)
```

```
        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        # add the best back to population:
        offspring.extend(halloffame.items)

        # Update the hall of fame with the generated individuals
        halloffame.update(offspring)

        # Replace the current population by the offspring
        population[:] = offspring

        # Append the current generation statistics to the logbook
        record = stats.compile(population) if stats else {}
        logbook.record(gen=gen, nevals=len(invalid_ind), **record)
        if verbose:
            print(logbook.stream)

    return population, logbook
```

GA Workflow

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('***************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
end = time.time()
time_NNA = end - start
print("Time Elapsed = ", time_NNA)
```

```
**************************
gen     nevals  max     avg
0       100     500     18.35
1       94      500     20.6
2       93      500     23.76
3       94      500     38.88
4       93      500     48.58
5       93      500     50.68
6       92      500     62.93
7       93      500     47.7
8       92      500     59.98
9       94      500     75.68
10      95      500     55.81
11      85      500     76.89
12      93      500     79.83
13      92      500     73.48
14      85      500     66.83
15      96      500     69.02
16      94      500     98.28
17      95      500     91.89
18      93      500     92.16
19      93      500     100.52
20      90      500     112.75
21      89      500     112.59
22      92      500     149.5
23      95      500     177.98
24      96      500     228.95
25      89      500     239.97
26      91      500     298.21
27      91      500     277.71
28      95      500     356.53
29      93      500     426.83
30      94      500     426.92
31      93      500     371.19
32      92      500     415.99
33      91      500     447.65
34      90      500     431.27
35      94      500     445.97
36      91      500     448.41
37      91      500     449.5
38      88      500     470.01
39      93      500     473.17
40      90      500     460.75
Time Elapsed =  102.10462594032288
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMax = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
minFitnessValues_GA, meanFitnessValues_GA = logbook.select("max", "avg")
print('History of maxFitnessValues_GA =',minFitnessValues_GA)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA)

# save best solution for a replay:
```

```
#car.saveActions(best)
cartPole.saveParams(best)
```

```
Best solution:  [0.9039409992284728, 0.06655498740733878, -0.6699136774532918
Best FitnessMax = 500.00000
History of maxFitnessValues_GA = [500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 5
History of meanFitnessValues_GA = [18.35, 20.6, 23.76, 38.88, 48.58, 50.68, 6
```

```
# Replay the best solution - TEXT version
#car.replaySavedActions()
cartPole.replayWithSavedParams()
```

```
1 : -------------------------
action =  1
observation =  [ 0.00669915  0.22729017  0.02071759 -0.2462488 ]
reward =  1.0
totalReward =  1.0
done =  False

2 : -------------------------
action =  0
observation =  [0.01124495 0.03187854 0.01579262 0.05289627]
reward =  1.0
totalReward =  2.0
done =  False

3 : -------------------------
action =  1
observation =  [ 0.01188253  0.22677053  0.01685054 -0.23476242]
reward =  1.0
totalReward =  3.0
done =  False

4 : -------------------------
action =  0
observation =  [0.01641794 0.03141193 0.01215529 0.06318771]
reward =  1.0
totalReward =  4.0
done =  False

5 : -------------------------
action =  1
observation =  [ 0.01704617  0.22635751  0.01341905 -0.2256355 ]
reward =  1.0
totalReward =  5.0
done =  False

6 : -------------------------
action =  0
observation =  [0.02157332 0.03104637 0.00890634 0.07124991]
reward =  1.0
totalReward =  6.0
done =  False

7 : -------------------------
action =  1
observation =  [ 0.02219425  0.22603951  0.01033134 -0.21860977]
reward =  1.0
```

```
totalReward =  7.0
done =  False

8 : --------------------------
action =  0
observation =  [0.02671504 0.03077141 0.00595914 0.07731411]
reward =  1.0
totalReward =  8.0
done =  False

9 : --------------------------
```

```
cartPole.replayVideo()
```

```

```

```
# Replay the best solution - VIDEO version
#env.close()
show_video()
```

```
# find average score of 100 episodes using the best solution found:
print("Running 100 episodes using the best solution...")
scores = []
for test in range(100):
  scores.append(cartPole.getScore(best))
  print("scores = ", scores)
  print("Avg. score = ", sum(scores) / len(scores))
```

Part 4.What about the solution dependence on GA conditions?

... with various RANDOM_SEED ...

Results for various RANDOM_SEEDs

RANDOM_SEED = 42

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)
```

```
# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('***************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
end = time.time()
time_42 = end - start
print("Time Elapsed = ", time_42)
```

```
***************************
gen     nevals  max     avg
0       100     500     18.35
1       94      500     20.6
2       93      500     23.76
3       94      500     38.88
4       93      500     48.58
5       93      500     50.68
6       92      500     62.93
7       93      500     47.7
8       92      500     59.98
9       94      500     75.68
10      95      500     55.81
11      85      500     76.89
12      93      500     79.83
13      92      500     73.48
14      85      500     66.83
15      96      500     69.02
16      94      500     98.28
17      95      500     91.89
18      93      500     92.16
19      93      500     100.52
20      90      500     112.75
21      89      500     112.59
22      92      500     149.5
23      95      500     177.98
24      96      500     228.95
25      89      500     239.97
26      91      500     298.21
27      91      500     277.71
28      95      500     356.53
29      93      500     426.83
30      94      500     426.92
31      93      500     371.19
32      92      500     415.99
33      91      500     447.65
34      90      500     431.27
35      94      500     445.97
36      91      500     448.41
37      91      500     449.5
```

```
38      88      500     470.01
39      93      500     473.17
40      90      500     460.75
Time Elapsed =  100.9089777469635
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMax = %1.5f" % best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_42, meanFitnessValues_GA_42 = logbook.select("max", "avg")
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_42)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_42)
```

```
Best solution:  [0.9039409992284728, 0.06655498740733878, -0.6699136774532918
Best FitnessMax = 500.00000
History of maxFitnessValues_GA = [500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 5
History of meanFitnessValues_GA = [18.35, 20.6, 23.76, 38.88, 48.58, 50.68, 6
```

RANDOM_SEED = 666

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 666
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('**************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                                    toolbox,
                                                    cxpb=P_CROSSOVER,
                                                    mutpb=P_MUTATION,
                                                    ngen=MAX_GENERATIONS,
```

```
                                                    stats=stats,
                                                    halloffame=hof,
                                                    verbose=True)
end = time.time()
time_666 = end - start
print("Time Elapsed = ", time_666)
```

```
    ************************
    gen     nevals  max     avg
    0       100     107     13.16
    1       94      107     15.3
    2       90      195     21.15
    3       92      195     22.12
    4       90      309     28.75
    5       93      500     40.27
    6       93      500     51.37
    7       92      500     55.92
    8       89      500     60.71
    9       91      500     52.74
    10      90      500     46.15
    11      89      500     48.56
    12      94      500     48.64
    13      96      500     50.98
    14      92      500     72.16
    15      91      500     63.82
    16      92      500     77.75
    17      90      500     82.13
    18      96      500     101.53
    19      93      500     109.37
    20      88      500     127.58
    21      93      500     177.51
    22      94      500     218.88
    23      94      500     238.59
    24      95      500     248.42
    25      95      500     243.61
    26      93      500     254.23
    27      90      500     286.46
    28      94      500     338.73
    29      94      500     376.32
    30      94      500     411.44
    31      89      500     434.94
    32      92      500     437.33
    33      91      500     434.47
    34      92      500     440.55
    35      92      500     461.34
    36      88      500     460.73
    37      91      500     445.49
    38      87      500     451.93
    39      95      500     433.93
    40      93      500     446.73
    Time Elapsed =  100.75136065483093
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMax = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])
```

```
# extract statistics:
maxFitnessValues_GA_666, meanFitnessValues_GA_666 = logbook.select("max", "avg")
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_666)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_666)
```

```
Best solution:  [-0.2582487265171046, 0.24179682019576307, -0.150886079529518
Best FitnessMax = 500.00000
History of minFitnessValues_GA = [107.0, 107.0, 195.0, 195.0, 309.0, 500.0, 5
History of meanFitnessValues_GA = [13.16, 15.3, 21.15, 22.12, 28.75, 40.27, 5
```

RANDOM_SEED = 1042

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 1042
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('**************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
end = time.time()
time_1042 = end - start
print("Time Elapsed = ", time_1042)
```

```
**************************
gen     nevals  max     avg
0       100     98      14.05
1       93      369     20.27
2       90      369     24.69
```

```
3       95      500     37.39
4       90      500     34
5       94      500     36.67
6       90      500     47.13
7       91      500     58.5
8       95      500     85.11
9       92      500     81.75
10      94      500     83.59
11      92      500     98.02
12      94      500     120.35
13      89      500     136.98
14      87      500     136.72
15      91      500     134.35
16      93      500     128.28
17      95      500     140.19
18      93      500     149.64
19      90      500     161.43
20      95      500     204.21
21      93      500     208.19
22      91      500     234.49
23      90      500     283.42
24      93      500     336.07
25      92      500     339.78
26      89      500     353.78
27      85      500     360.43
28      88      500     376.7
29      95      500     374.63
30      88      500     394.9
31      94      500     357.41
32      90      500     370.26
33      92      500     378.91
34      89      500     362.35
35      83      500     402.14
36      89      500     372.56
37      93      500     393.47
38      96      500     431.22
39      93      500     402.09
40      85      500     411.7
Time Elapsed =  108.02887892723083
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMin = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_1042, meanFitnessValues_GA_1042 = logbook.select("max", "avg")
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_1042)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_1042)
```

```
Best solution:  [0.5889028101863627, -0.7057154551790351, 0.42332444618582854
Best FitnessMin = 500.00000
History of maxFitnessValues_GA = [98.0, 369.0, 369.0, 500.0, 500.0, 500.0, 50
History of meanFitnessValues_GA = [14.05, 20.27, 24.69, 37.39, 34.0, 36.67, 4
```

RESUME

For various RANDOM_SEED we can obtain **different**:

- **solutions** :) ... of course,
- **performance** (fitness function value),
- **history**.

The reason is the stochastic manner of parameter change during evolution.

... with various GA parameters ... like Crossover Probability

It takes a small change in *P_CROSSOVER* variable.

P_CROSSOVER = 0.1

```
P_CROSSOVER = 0.1  # probability for crossover
```

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 1042
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('***************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
```

```
end = time.time()
time_1042_CR0p1 = end - start
print("Time Elapsed = ", time_1042_CR0p1)
```

```
**************************
gen     nevals  max     avg
0       100     98      14.05
1       93      369     20.27
2       90      369     24.69
3       95      500     37.39
4       90      500     34
5       94      500     36.67
6       90      500     47.13
7       91      500     58.5
8       95      500     85.11
9       92      500     81.75
10      94      500     83.59
11      92      500     98.02
12      94      500     120.35
13      89      500     136.98
14      87      500     136.72
15      91      500     134.35
16      93      500     128.28
17      95      500     140.19
18      93      500     149.64
19      90      500     161.43
20      95      500     204.21
21      93      500     208.19
22      91      500     234.49
23      90      500     283.42
24      93      500     336.07
25      92      500     339.78
26      89      500     353.78
27      85      500     360.43
28      88      500     376.7
29      95      500     374.63
30      88      500     394.9
31      94      500     357.41
32      90      500     370.26
33      92      500     378.91
34      89      500     362.35
35      83      500     402.14
36      89      500     372.56
37      93      500     393.47
38      96      500     431.22
39      93      500     402.09
40      85      500     411.7
Time Elapsed =  106.09052872657776
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMin = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_1042_CR0p1, meanFitnessValues_GA_1042_CR0p1 = logbook.select("
```

```
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_1042_CR0p1)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_1042_CR0p1)
```

```
Best solution:  [0.5889028101863627, -0.7057154551790351, 0.42332444618582854
Best FitnessMin = 500.00000
History of minFitnessValues_GA = [98.0, 369.0, 369.0, 500.0, 500.0, 500.0, 50
History of meanFitnessValues_GA = [14.05, 20.27, 24.69, 37.39, 34.0, 36.67, 4
```

P_CROSSOVER = 0.2

```
P_CROSSOVER = 0.2  # probability for crossover
```

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 1042
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('**************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
end = time.time()
time_1042_CR0p2 = end - start
print("Time Elapsed = ", time_1042_CR0p2)
```

```
**************************
```

```
gen     nevals  max     avg
0       100     61      12.76
1       84      139     18.63
2       87      139     20.28
3       86      139     20.38
4       88      262     27.71
5       87      500     36.11
6       89      500     46.12
7       76      500     50.12
8       83      500     69.22
9       83      500     102.08
10      90      500     112.98
11      90      500     135.09
12      92      500     151.42
13      79      500     196.46
14      89      500     206.88
15      85      500     254.88
16      87      500     307.1
17      92      500     343.59
18      87      500     368.36
19      89      500     361.83
20      90      500     386.21
21      84      500     401.56
22      87      500     450.47
23      88      500     428.24
24      89      500     446.1
25      86      500     427.84
26      90      500     465.95
27      87      500     464.36
28      89      500     468.08
29      83      500     461.77
30      88      500     457.34
31      83      500     487.66
32      83      500     485.6
33      83      500     457.34
34      80      500     478.28
35      88      500     470.93
36      87      500     464.49
37      89      500     479.88
38      87      500     462.48
39      90      500     492.46
40      85      500     472.49
Time Elapsed =  137.8567316532135
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMin = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_1042_CR0p2, meanFitnessValues_GA_1042_CR0p2 = logbook.select("
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_1042_CR0p2)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_1042_CR0p2)
```

```
Best solution:  [0.8956039608596099, 0.013569898457665541, 0.8330109873780233
Best FitnessMin = 500.00000
History of maxFitnessValues_GA = [61.0, 139.0, 139.0, 139.0, 262.0, 500.0, 50
History of meanFitnessValues_GA = [12.76, 18.63, 20.28, 20.38, 27.71, 36.11,
```

P_CROSSOVER = 0.4

```
P_CROSSOVER = 0.4  # probability for crossover
```

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 1042
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('**************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
end = time.time()
time_1042_CR0p4 = end - start
print("Time Elapsed = ", time_1042_CR0p4)
```

```
**************************
gen     nevals  max     avg
0       100     98      14.05
1       57      98      15.38
2       76      413     23.66
3       66      413     32.9
4       65      413     40.32
5       71      413     54.39
6       65      413     76.21
```

```
7       57      500     101.52
8       65      500     127.46
9       62      500     165.06
10      74      500     164.39
11      66      500     185.03
12      74      500     168.55
13      59      500     244.16
14      63      500     310.33
15      70      500     359.5
16      61      500     422.41
17      73      500     444.25
18      63      500     481.18
19      66      500     467.12
20      60      500     470.3
21      74      500     484.17
22      69      500     481.37
23      64      500     479.4
24      71      500     484.64
25      65      500     470.95
26      64      500     468.52
27      74      500     472.32
28      67      500     476.02
29      66      500     477.52
30      61      500     492.61
31      72      500     469.78
32      59      500     469.91
33      68      500     464.12
34      68      500     478.64
35      64      500     464.44
36      72      500     481.83
37      65      500     474.61
38      67      500     470.35
39      69      500     453.34
40      64      500     455.02
Time Elapsed =  116.45055270195007
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMin = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_1042_CR0p4, meanFitnessValues_GA_1042_CR0p4 = logbook.select("
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_1042_CR0p4)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_1042_CR0p4)
```

```
Best solution:  [0.6272811775924462, 0.4841432610995863, 0.6791695047075679,
Best FitnessMin = 500.00000
History of maxFitnessValues_GA = [98.0, 98.0, 413.0, 413.0, 413.0, 413.0, 413
History of meanFitnessValues_GA = [14.05, 15.38, 23.66, 32.9, 40.32, 54.39, 7
```

P_CROSSOVER = 0.8

```
P_CROSSOVER = 0.8  # probability for crossover
```

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 1042
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('**************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                          toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          halloffame=hof,
                                          verbose=True)
end = time.time()
time_1042_CR0p8 = end - start
print("Time Elapsed = ", time_1042_CR0p8)
```

```
**************************
gen     nevals  max     avg
0       100     98      14.05
1       91      228     17.72
2       86      228     24.53
3       85      228     31.98
4       84      228     30.83
5       90      310     34.15
6       86      500     43.3
7       87      500     45.03
8       82      500     44.2
9       88      500     50.58
10      90      500     72.82
11      84      500     69.19
12      86      500     84.47
```

```
13      88      500     103.33
14      90      500     119.05
15      91      500     156.08
16      86      500     210.4
17      90      500     244.17
18      85      500     282.28
19      84      500     347.16
20      90      500     369.82
21      86      500     421.07
22      86      500     390.78
23      91      500     405.6
24      86      500     412.67
25      81      500     429.95
26      89      500     431.46
27      84      500     423.54
28      93      500     437.7
29      88      500     448.48
30      84      500     452.05
31      84      500     470.85
32      86      500     456.84
33      84      500     475.83
34      92      500     457.2
35      89      500     461.42
36      81      500     459.49
37      92      500     470.96
38      91      500     460.97
39      88      500     457.45
40      88      500     464.49
Time Elapsed =  126.49325156211853
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMin = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_1042_CR0p8, meanFitnessValues_GA_1042_CR0p8 = logbook.select("
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_1042_CR0p8)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_1042_CR0p8)
```

```
Best solution:  [-0.3228837696778911, -0.053312503649015824, 0.29497242129595
Best FitnessMin = 500.00000
History of maxFitnessValues_GA = [98.0, 228.0, 228.0, 228.0, 228.0, 310.0, 50
History of meanFitnessValues_GA = [14.05, 17.72, 24.53, 31.98, 30.83, 34.15,
```

P_CROSSOVER = 0.9

```
P_CROSSOVER = 0.9  # probability for crossover
```

```
# Set the random seed
# for reprodicibility of results:
RANDOM_SEED = 1042
```

```
random.seed(RANDOM_SEED)

# Create the instance of the MountainCartPole class:
#car = MountainCar(RANDOM_SEED)
cartPole = CartPole(RANDOM_SEED)
```

```
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)

# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", numpy.max)
stats.register("avg", numpy.mean)

# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

print('**************************')
start = time.time()
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = eaSimpleWithElitism(population,
                                                        toolbox,
                                                        cxpb=P_CROSSOVER,
                                                        mutpb=P_MUTATION,
                                                        ngen=MAX_GENERATIONS,
                                                        stats=stats,
                                                        halloffame=hof,
                                                        verbose=True)
end = time.time()
time_1042_CR0p9 = end - start
print("Time Elapsed = ", time_1042_CR0p9)
```

```
**************************
gen     nevals  max     avg
0       100     98      14.05
1       93      369     20.27
2       90      369     24.69
3       95      500     37.39
4       90      500     34
5       94      500     36.67
6       90      500     47.13
7       91      500     58.5
8       95      500     85.11
9       92      500     81.75
10      94      500     83.59
11      92      500     98.02
12      94      500     120.35
13      89      500     136.98
14      87      500     136.72
15      91      500     134.35
16      93      500     128.28
17      95      500     140.19
18      93      500     149.64
19      90      500     161.43
20      95      500     204.21
21      93      500     208.19
22      91      500     234.49
```

```
23      90      500     283.42
24      93      500     336.07
25      92      500     339.78
26      89      500     353.78
27      85      500     360.43
28      88      500     376.7
29      95      500     374.63
30      88      500     394.9
31      94      500     357.41
32      90      500     370.26
33      92      500     378.91
34      89      500     362.35
35      83      500     402.14
36      89      500     372.56
37      93      500     393.47
38      96      500     431.22
39      93      500     402.09
40      85      500     411.7
Time Elapsed =  107.74227976799011
```

```
# print best solution found:
best = hof.items[0]
print("Best solution: ", best)
print("Best FitnessMin = %1.5f" % best.fitness.values[0])
#print("Best Fitness = ", best.fitness.values[0])

# extract statistics:
maxFitnessValues_GA_1042_CR0p9, meanFitnessValues_GA_1042_CR0p9 = logbook.select("
print('History of maxFitnessValues_GA =',maxFitnessValues_GA_1042_CR0p9)
print('History of meanFitnessValues_GA =',meanFitnessValues_GA_1042_CR0p9)
```

```
Best solution:  [0.5889028101863627, -0.7057154551790351, 0.42332444618582854
Best FitnessMin = 500.00000
History of maxFitnessValues_GA = [98.0, 369.0, 369.0, 500.0, 500.0, 500.0, 50
History of meanFitnessValues_GA = [14.05, 20.27, 24.69, 37.39, 34.0, 36.67, 4
```

RESUME

Again ... for various P_CROSSOVER we can obtain **different**:

- **solutions** :) ... of course,
- **performance** (fitness function value),
- **history**.

The reasons are

- the stochastic manner of parameter change during evolution,
- BUT ... more important ... different levels of gene exchange.

... with various GA parameters ... like Mutation Probability

(let's try it as a self-guided learning!)

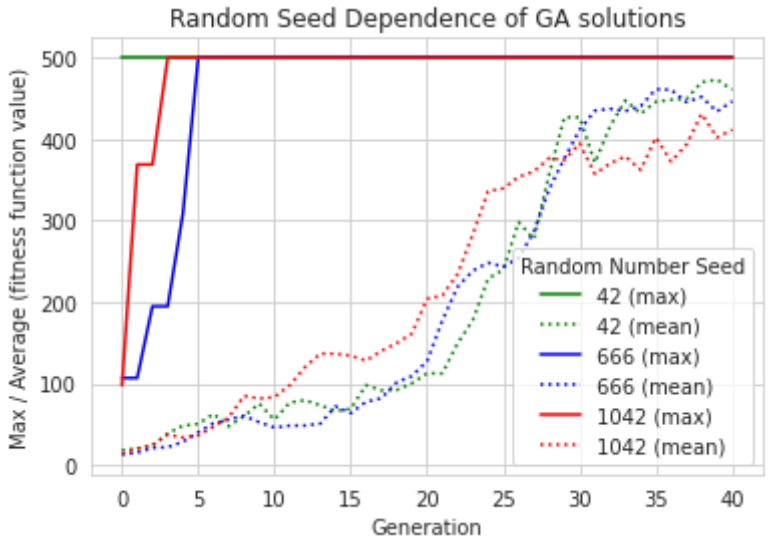
It takes a small change in *P_MUTATION* variable.

Comparison Plots

Random Seed Dependence

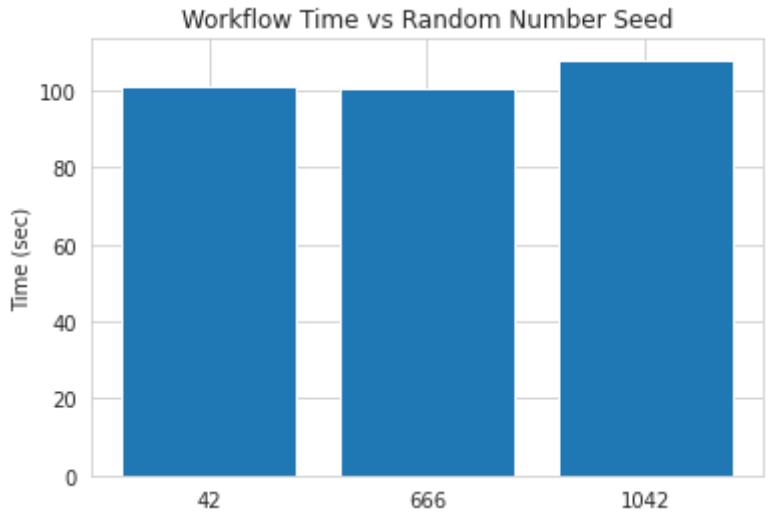
Fitness Function

```
sns.set_style("whitegrid")

# Classic grid search solution
#plt.hlines(accuracy_classic_solution, 0, 5, linestyle = 'solid', label='Classic g

# NN architecture
plt.plot(maxFitnessValues_GA_42, color='green', label='42 (max)')
plt.plot(meanFitnessValues_GA_42, color='green', linestyle = 'dotted', label='42 (

# NN hyperparameter
plt.plot(maxFitnessValues_GA_666, color='blue', label='666 (max)')
plt.plot(meanFitnessValues_GA_666, color='blue', linestyle = 'dotted', label='666

# NN architecture + hyperparameter
plt.plot(maxFitnessValues_GA_1042, color='red', label='1042 (max)')
plt.plot(meanFitnessValues_GA_1042, color='red', linestyle = 'dotted', label='1042

plt.xlabel('Generation')
plt.ylabel('Max / Average (fitness function value)')
plt.title('Random Seed Dependence of GA solutions')
plt.legend(title='Random Number Seed')
plt.show()
```



Time

```
import matplotlib.pyplot as plt

x = ['42','666','1042']
y = [time_42,time_666,time_1042]
plt.bar(x,y)
plt.ylabel('Time (sec)')
plt.title('Workflow Time vs Random Number Seed')
plt.show()
```

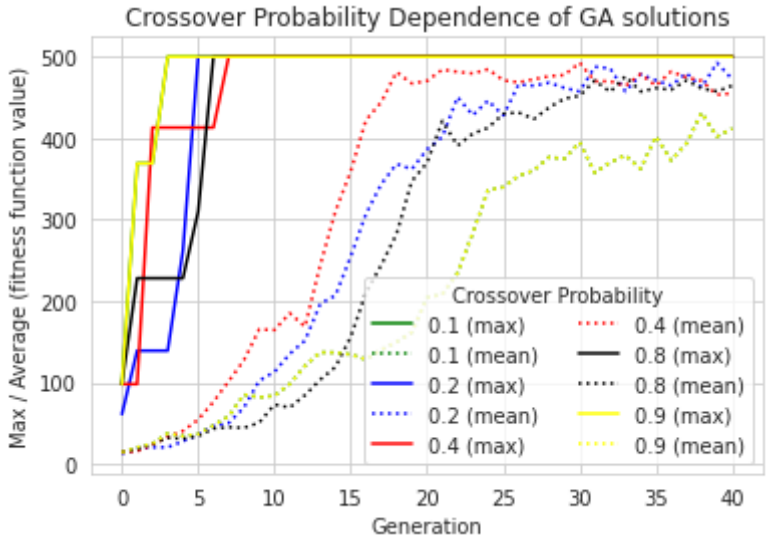


Crossover Probability Dependence

Fitness Function

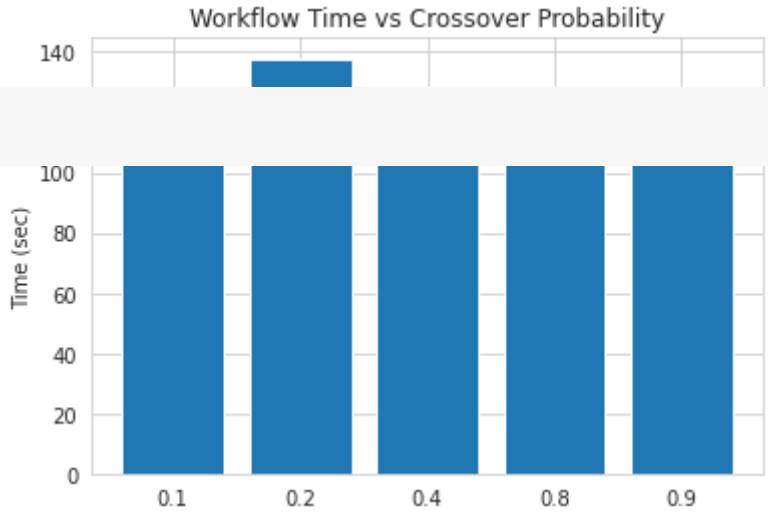
```
sns.set_style("whitegrid")

# Classic grid search solution
#plt.hlines(accuracy_classic_solution, 0, 5, linestyle = 'solid', label='Classic g

# NN architecture
plt.plot(maxFitnessValues_GA_1042_CR0p1, color='green', label='0.1 (max)')
plt.plot(meanFitnessValues_GA_1042_CR0p1, color='green', linestyle = 'dotted', lab

# NN hyperparameter
plt.plot(maxFitnessValues_GA_1042_CR0p2, color='blue', label='0.2 (max)')
plt.plot(meanFitnessValues_GA_1042_CR0p2, color='blue', linestyle = 'dotted', labe

# NN architecture + hyperparameter
plt.plot(maxFitnessValues_GA_1042_CR0p4, color='red', label='0.4 (max)')
plt.plot(meanFitnessValues_GA_1042_CR0p4, color='red', linestyle = 'dotted', label
```

```
# NN architecture + hyperparameter
plt.plot(maxFitnessValues_GA_1042_CR0p8, color='black', label='0.8 (max)')
plt.plot(meanFitnessValues_GA_1042_CR0p8, color='black', linestyle = 'dotted', lab

# NN architecture + hyperparameter
plt.plot(maxFitnessValues_GA_1042_CR0p9, color='yellow', label='0.9 (max)')
plt.plot(meanFitnessValues_GA_1042_CR0p9, color='yellow', linestyle = 'dotted', la

plt.xlabel('Generation')
plt.ylabel('Max / Average (fitness function value)')
plt.title('Crossover Probability Dependence of GA solutions')
plt.legend(title='Crossover Probability', ncol=2)
plt.show()
```



Time

```
import matplotlib.pyplot as plt

x = ['0.1','0.2','0.4','0.8','0.9']
y = [time_1042_CR0p1, time_1042_CR0p2, time_1042_CR0p4, time_1042_CR0p8, time_1042
plt.bar(x,y)
plt.ylabel('Time (sec)')
plt.title('Workflow Time vs Crossover Probability')
plt.show()
```



Лекція 08 - Нейроеволюція - EvoJAX

на основі (C) роботи Google Brain, Yujin Tang, Yingtao Tian, David Ha

Короткий зміст:

- Інсталяція EvoJAX (**кожного разу після запуску Colab VM!**),
- компоненти, необхідні для робочого процесу EA,
- *Проблема навчання з підкріпленням (RL):*
    - CartPole-v1
- градієнти політики з дослідженням на основі параметрів,
- і інші.

До кінця цієї лекції ви будете знати:

- знову ж таки, як використовувати вбудовані алгоритми структури DEAP для створення стислого коду
- як вирішити проблему *Reinforcement Learning* за допомогою рішень на основі EA для пошуку рішень,
- як використовувати градієнти політики з дослідженням на основі параметрів,
- як експериментувати з різними налаштуваннями GA та інтерпретувати відмінності в результатах.

Попередня умова

Перш ніж почати, нам потрібно встановити EvoJAX з EvoJAX-github та імпортувати деякі бібліотеки. **Примітка** У нашій статті ми проводили експерименти на графічних процесорах NVIDIA V100. Ваші результати можуть відрізнятися від наших.

```
from IPython.display import clear_output, Image

!pip install evojax

clear_output()
```

```
import os
import numpy as np
import jax
import jax.numpy as jnp

from evojax.task.cartpole import CartPoleSwingUp
```

```
from evojax.task.cartpole import CartPoleSwingUp
from evojax.policy.mlp import MLPPolicy
from evojax.algo import PGPE
from evojax import Trainer
from evojax.util import create_logger
```

```
# Let's create a directory to save logs and models.
log_dir = './log'
logger = create_logger(name='EvoJAX', log_dir=log_dir)
logger.info('Welcome to the tutorial on Neuroevolution algorithm creation!')

logger.info('Jax backend: {}'.format(jax.local_devices()))
!nvidia-smi --query-gpu=name --format=csv,noheader
```

```
EvoJAX: 2022-02-11 02:06:40,128 [INFO] Welcome to the tutorial on Neuroevolut
absl: 2022-02-11 02:06:40,137 [INFO] Unable to initialize backend 'tpu_driver
absl: 2022-02-11 02:06:40,322 [INFO] Unable to initialize backend 'tpu': INVA
EvoJAX: 2022-02-11 02:06:40,324 [INFO] Jax backend: [GpuDevice(id=0, process_
Tesla K80
```

Introduction

EvoJAX has three major components: the *task*, the *policy network* and the *neuroevolution algorithm*. Once these components are implemented and instantiated, we can use a trainer to start the training process. The following code snippet provides an example of how we use EvoJAX.

```
seed = 42  # Wish me luck!

# We use the classic cart-pole swing up as our tasks, see
# https://github.com/google/evojax/tree/main/evojax/task for more example tasks.
# The test flag provides the opportunity for a user to
# 1. Return different signals as rewards. For example, in our MNIST example,
#    we use negative cross-entropy loss as the reward in training tasks, and the
#    classification accuracy as the reward in test tasks.
# 2. Perform reward shaping. It is common for RL practitioners to modify the
#    rewards during training so that the agent learns more efficiently. But this
#    modification should not be allowed in tests for fair evaluations.
hard = False
train_task = CartPoleSwingUp(harder=hard, test=False)
test_task = CartPoleSwingUp(harder=hard, test=True)

# We use a feedforward network as our policy.
# By default, MLPPolicy uses "tanh" as its activation function for the output.
policy = MLPPolicy(
    input_dim=train_task.obs_shape[0],
    hidden_dims=[64, 64],
    output_dim=train_task.act_shape[0],
    logger=logger,
```

```
)

# We use PGPE as our evolution algorithm.
# If you want to know more about the algorithm, please take a look at the paper:
# https://people.idsia.ch/~juergen/nn2010.pdf
solver = PGPE(
    pop_size=64,
    param_size=policy.num_params,
    optimizer='adam',
    center_learning_rate=0.05,
    seed=seed,
)

# Now that we have all the three components instantiated, we can create a
# trainer and start the training process.
trainer = Trainer(
    policy=policy,
    solver=solver,
    train_task=train_task,
    test_task=test_task,
    max_iter=600,
    log_interval=100,
    test_interval=200,
    n_repeats=5,
    n_evaluations=128,
    seed=seed,
    log_dir=log_dir,
    logger=logger,
)
_ = trainer.run()
```

```
EvoJAX: 2022-02-11 02:06:43,518 [INFO] MLPPolicy.num_params = 4609
EvoJAX: 2022-02-11 02:06:43,687 [INFO] Start to train for 600 iterations.
EvoJAX: 2022-02-11 02:07:10,038 [INFO] Iter=100, size=64, max=712.8441, avg=6
EvoJAX: 2022-02-11 02:07:29,392 [INFO] Iter=200, size=64, max=782.5107, avg=7
EvoJAX: 2022-02-11 02:07:31,972 [INFO] [TEST] Iter=200, #tests=128, max=816.3
EvoJAX: 2022-02-11 02:07:51,419 [INFO] Iter=300, size=64, max=920.6417, avg=8
EvoJAX: 2022-02-11 02:08:10,756 [INFO] Iter=400, size=64, max=921.8397, avg=8
EvoJAX: 2022-02-11 02:08:10,907 [INFO] [TEST] Iter=400, #tests=128, max=934.5
EvoJAX: 2022-02-11 02:08:30,258 [INFO] Iter=500, size=64, max=932.7117, avg=8
EvoJAX: 2022-02-11 02:08:49,644 [INFO] [TEST] Iter=600, #tests=128, max=955.2
EvoJAX: 2022-02-11 02:08:49,652 [INFO] Training done, best_score=935.1467
```

```
# Let's visualize the learned policy.

def render(task, algo, policy):
    """Render the learned policy."""

    task_reset_fn = jax.jit(test_task.reset)
    policy_reset_fn = jax.jit(policy.reset)
    step_fn = jax.jit(test_task.step)
    act_fn = jax.jit(policy.get_actions)

    params = algo.best_params[None, :]
```

```
    task_s = task_reset_fn(jax.random.PRNGKey(seed=seed)[None, :])
    policy_s = policy_reset_fn(task_s)

    images = [CartPoleSwingUp.render(task_s, 0)]
    done = False
    step = 0
    reward = 0
    while not done:
        act, policy_s = act_fn(task_s, params, policy_s)
        task_s, r, d = step_fn(task_s, act)
        step += 1
        reward = reward + r
        done = bool(d[0])
        if step % 3 == 0:
            images.append(CartPoleSwingUp.render(task_s, 0))
    print('reward={}'.format(reward))
    return images


imgs = render(test_task, solver, policy)
gif_file = os.path.join(log_dir, 'cartpole.gif')
imgs[0].save(
    gif_file, save_all=True, append_images=imgs[1:], duration=40, loop=0)
Image(open(os.path.join(log_dir, 'cartpole.gif'),'rb').read())
```

```
    reward=[940.53296]
```

This tutorial walks you through the process of creating a new neuroevolution algoritm.

To contribute an algorithm implementation to EvoJAX, all you need to do is to implement the `NEAlgorithm` interface.
The interface is defined as the following and you can see the related Python file here:

```
class NEAlgorithm(ABC):
    """Interface of all Neuro-evolution algorithms in EvoJAX."""

    pop_size: int

    @abstractmethod
    def ask(self) -> jnp.ndarray:
        """Ask the algorithm for a population of parameters.
        Returns
            A Jax array of shape (population_size, param_size).
        """
        raise NotImplementedError()

    @abstractmethod
    def tell(self, fitness: Union[np.ndarray, jnp.ndarray]) -> None:
        """Report the fitness of the population to the algorithm.
        Args:
            fitness - The fitness scores array.
        """
        raise NotImplementedError()

    @property
    def best_params(self) -> jnp.ndarray:
        raise NotImplementedError()

    @best_params.setter
    def best_params(self, params: Union[np.ndarray, jnp.ndarray]) -> None:
        raise NotImplementedError()
```

Wrap an existing implementation

`NEAlgorithm` adopts the well-known “ask” and “tell” interfaces, where the former requests the algorithm to generate a population of parameters and the latter reports the parameters' fitness scores so that the algorithm can update its internal states. We think the conventional interface

for the neuroevolution algorithms brings familiarity to the developers and thus reduces the required learning effort. Moreover, the interface is also used by many existing algorithms, it is therefore possible for the practitioners to quickly plug in existing algorithms for sanity checks. In the first part of this tutorial, we will create an implementation that wraps CMA-ES. Please take a look at this wonderful tutorial for more information about CMA-ES.

```
import cma
from evojax.algo.base import NEAlgorithm


class CMAWrapper(NEAlgorithm):
    """This is a wrapper of CMA-ES."""

    def __init__(self, param_size, pop_size, init_stdev=0.1, seed=0):

        self.pop_size = pop_size
        self.params = None
        self._best_params = None

        # We create CMA-ES in a simplest form.
        self.cma = cma.CMAEvolutionStrategy(
            x0=np.zeros(param_size),
            sigma0=init_stdev,
            inopts={
                'popsize': pop_size,
                'seed': seed if seed > 0 else 42,
                'randn': np.random.randn,
            },
        )

        # We jit-compile some utility functions.
        self.jnp_array = jax.jit(jnp.array)
        self.jnp_stack = jax.jit(jnp.stack)

    def ask(self):
        self.params = self.cma.ask()
        return self.jnp_stack(self.params)

    def tell(self, fitness):
        # CMA-ES minimizes, so we negate the fitness.
        self.cma.tell(self.params, -np.array(fitness))
        self._best_params = np.array(self.cma.result.xfavorite)

    @property
    def best_params(self):
        return self.jnp_array(self._best_params)

    @best_params.setter
    def best_params(self, params):
        self._best_params = np.array(params)
```

Notice that our implementation above is extremely simple, we haven't used many options or functions provided by CMA-ES.
But let's plug in this implementation to our cart-pole earlier example and see how it works.
Alert Depending on your CPUs, running the following cell may take some time.

```
# Instead of PGPE, we use our CMAWrapper now.
solver = CMAWrapper(
    pop_size=64,
    param_size=policy.num_params,
    seed=seed,
)
trainer = Trainer(
    policy=policy,
    solver=solver,
    train_task=train_task,
    test_task=test_task,
    max_iter=600,
    log_interval=100,
    test_interval=200,
    n_repeats=5,
    n_evaluations=128,
    seed=seed,
    log_dir=log_dir,
    logger=logger,
)
_ = trainer.run()
```

```
EvoJAX: 2022-02-11 02:08:59,845 [INFO] Start to train for 600 iterations.
(32_w,64)-aCMA-ES (mu_w=17.6,w_1=11%) in dimension 4609 (seed=42, Fri Feb 11
EvoJAX: 2022-02-11 02:14:31,792 [INFO] Iter=100, size=64, max=643.9105, avg=4
EvoJAX: 2022-02-11 02:20:00,840 [INFO] Iter=200, size=64, max=692.3575, avg=5
EvoJAX: 2022-02-11 02:20:01,894 [INFO] [TEST] Iter=200, #tests=128, max=751.8
EvoJAX: 2022-02-11 02:25:27,575 [INFO] Iter=300, size=64, max=718.8022, avg=5
EvoJAX: 2022-02-11 02:31:06,983 [INFO] Iter=400, size=64, max=747.0325, avg=5
EvoJAX: 2022-02-11 02:31:07,139 [INFO] [TEST] Iter=400, #tests=128, max=706.6
EvoJAX: 2022-02-11 02:36:32,660 [INFO] Iter=500, size=64, max=725.8452, avg=6
EvoJAX: 2022-02-11 02:41:55,297 [INFO] [TEST] Iter=600, #tests=128, max=764.3
EvoJAX: 2022-02-11 02:41:55,305 [INFO] Training done, best_score=743.6188
```

The simple CMA-ES wrapper worked! However, we also notice that the training time increased significantly.
Although the task and the policy networks are accelerated by GPUs, the `cma.CMAEvolutionStrategy` implementation we used in the code above relies on CPUs, and that is why we see the drop in training speed.

Nevertheless, being able to wrapper an existing algorithm and plug that in EvoJAX's training pipeline serves as sanity checks and helps debugging when you migrate algorithms to EvoJAX.
Next, we will show you how to implement an algorithm in JAX from scratch.

Simple PGPE in JAX

We are going to implement a very simple version of PGPE, users interested in the algorithm can take a look at the paper and also check out some popular implementations (example1, example2).

In a nutshell, PGPE samples the policy network parameters θ from Gaussian distributions. It maintains the means μ and the standard deviations σ of the Gaussian distributions, and then estimates the gradients of these parameters using the following formulae:

$$\Delta\mu_i = \alpha(r-b)(\theta_i - \mu_i)\,, \Delta\sigma_i = \alpha(r-b)\frac{(\theta_i-\mu_i)^2-{\sigma_i}^2}{\sigma_i}$$

where α is the learning rate and b is a baseline from the reward r.

The following code snippet provides a sample implementation of PGPE.
Note This simplified version ignores popular tricks such as converting the rewards to ranks, using modern optimizers for parameter update, etc.

```
from evojax.algo.base import NEAlgorithm


class SimplePGPE(NEAlgorithm):
    """A simplified version of PGPE."""

    def __init__(self, param_size, pop_size,
                 lr_mu=0.05, lr_sigma=0.1, init_stdev=0.1, seed=0):

        self.pop_size = pop_size
        assert pop_size % 2 == 0, "pop_size must be a multpile of 2."
        n_directs = pop_size // 2
        self.noises = jnp.zeros(param_size)
        self.params = jnp.zeros(param_size)
        self.mu = jnp.zeros(param_size)
        self.sigma = jnp.ones(param_size) * init_stdev
        self.rand_key = jax.random.PRNGKey(seed=seed)

        def ask_fn(key, mu, sigma):
            next_key, sample_key = jax.random.split(key=key, num=2)
            perturbations = jax.random.normal(
                key=sample_key, shape=(n_directs, param_size)) * sigma[None, :]
            params = jnp.vstack([perturbations, -perturbations]) + mu[None, :]
            return params, perturbations, next_key

        self.ask_fn = jax.jit(ask_fn)

        def tell_fn(rewards, mu, sigma, perturbations):
            fitness = jnp.array(rewards).reshape([2, n_directs])

            # To map to the formulae above:
```

```
            # (r - b) = (avg_fitness - b) and (theta - mu) = perturbations
            avg_fitness = fitness.mean(axis=0)
            b = jnp.mean(fitness)

            # Update the means.
            grad_mu = (
                (avg_fitness - b)[:, None] * perturbations
            ).mean(axis=0)
            new_mu = mu + lr_mu * grad_mu

            # Update the sigmas.
            # We constrain the change of sigma to prevent numerical errors.
            grad_sigma = (
                (avg_fitness - b)[:, None] *
                (perturbations ** 2 - (sigma ** 2)[None, :]) / sigma[None, :]
            ).mean(axis=0)
            new_sigma = jnp.clip(
                sigma + lr_sigma * grad_sigma, 0.8 * sigma, 1.2 * sigma)

            return new_mu, new_sigma

        self.tell_fn = jax.jit(tell_fn)

    def ask(self):
        self.params, self.noises, self.rand_key = self.ask_fn(
            self.rand_key, self.mu, self.sigma)
        return self.params

    def tell(self, fitness):
        self.mu, self.sigma = self.tell_fn(
            fitness, self.mu, self.sigma, self.noises)

    @property
    def best_params(self):
        return self.mu

    @best_params.setter
    def best_params(self, params):
        self.mu = jnp.array(params)
```

```
# Let's test our simple PGPE.
solver = SimplePGPE(
    pop_size=64,
    param_size=policy.num_params,
    seed=seed,
)
trainer = Trainer(
    policy=policy,
    solver=solver,
    train_task=train_task,
    test_task=test_task,
    max_iter=1000,
    log_interval=100,
    test_interval=200,
```

```
    n_repeats=5,
    n_evaluations=128,
    seed=seed,
    log_dir=log_dir,
    logger=logger,
)
_ = trainer.run()
```

```
EvoJAX: 2022-02-11 02:41:55,585 [INFO] Start to train for 1000 iterations.
EvoJAX: 2022-02-11 02:42:16,350 [INFO] Iter=100, size=64, max=413.9725, avg=1
EvoJAX: 2022-02-11 02:42:35,561 [INFO] Iter=200, size=64, max=512.9973, avg=3
EvoJAX: 2022-02-11 02:42:36,577 [INFO] [TEST] Iter=200, #tests=128, max=556.6
EvoJAX: 2022-02-11 02:42:55,783 [INFO] Iter=300, size=64, max=523.7679, avg=4
EvoJAX: 2022-02-11 02:43:14,984 [INFO] Iter=400, size=64, max=567.3138, avg=5
EvoJAX: 2022-02-11 02:43:15,137 [INFO] [TEST] Iter=400, #tests=128, max=585.5
EvoJAX: 2022-02-11 02:43:34,341 [INFO] Iter=500, size=64, max=586.1516, avg=5
EvoJAX: 2022-02-11 02:43:53,552 [INFO] Iter=600, size=64, max=567.7144, avg=5
EvoJAX: 2022-02-11 02:43:53,705 [INFO] [TEST] Iter=600, #tests=128, max=636.5
EvoJAX: 2022-02-11 02:44:13,604 [INFO] Iter=700, size=64, max=592.1466, avg=4
EvoJAX: 2022-02-11 02:44:32,814 [INFO] Iter=800, size=64, max=603.3476, avg=5
EvoJAX: 2022-02-11 02:44:32,966 [INFO] [TEST] Iter=800, #tests=128, max=665.1
EvoJAX: 2022-02-11 02:44:52,172 [INFO] Iter=900, size=64, max=632.6639, avg=5
EvoJAX: 2022-02-11 02:45:11,339 [INFO] [TEST] Iter=1000, #tests=128, max=643.
EvoJAX: 2022-02-11 02:45:11,346 [INFO] Training done, best_score=592.5771
```

Despite its simplicity, the training and test scores rise steadily. You can see our complete implementation of PGPE here.

We hope this tutorial helps. Please let us (evojax-dev@google.com) know if you have any problems or suggestions, thanks!

```

```

Лекція 08 - Нейроеволюція – EvoJAX –Додаткові матеріали

****
Нейроеволюція
https://en.wikipedia.org/wiki/Neuroevolution
****

EvoJAX

****
Стаття:
https://arxiv.org/abs/2202.05008
https://arxiv.org/pdf/2202.05008.pdf

****
Коди + зошити:
EvoJAX: апаратно-прискорена нейроеволюція
https://github.com/google/evojax

****
Блоги:
EvoJAX: чудовий фреймворк для найглибших завдань
https://rezayazdanfar.medium.com/evojax-a-great-framework-for-most-deep-tasks-10adf685c152
6 хвилин читання

Апаратно-прискорений набір інструментів EvoJAX від Google Brain значно покращує нейроеволюційні обчислення
https://medium.com/syncedreview/google-brains-evojax-hardware-accelerated-toolkit-significantly-improves-neuroevolutionary-7943f92adb
4 хвилини читання

****
Презентація співавтора з рецензентами:
EvoJAX: апаратно-прискорена нейроеволюція
https://www.youtube.com/watch?v=vfz0XfZ_AbM
1:22:08

EvoJAX: Hardware-Accelerated Neuroevolution

Yujin Tang
yujintang@google.com
Google Brain

Yingtao Tian
alantian@google.com
Google Brain

David Ha
hadavid@google.com
Google Brain

ABSTRACT

Evolutionary computation has been shown to be a highly effective method for training neural networks, particularly when employed at scale on CPU clusters. Recent work have also showcased their effectiveness on hardware accelerators, such as GPUs, but so far such demonstrations are tailored for very specific tasks, limiting applicability to other domains. We present EvoJAX, a scalable, general purpose, hardware-accelerated neuroevolution toolkit. Building on top of the JAX library, our toolkit enables neuroevolution algorithms to work with neural networks running in parallel across multiple TPU/GPUs. EvoJAX achieves very high performance by implementing the evolution algorithm, neural network and task all in NumPy, which is compiled just-in-time to run on accelerators. We provide extensible examples of EvoJAX for a wide range of tasks, including supervised learning, reinforcement learning and generative art. Since EvoJAX can find solutions to most of these tasks within minutes on a single accelerator, compared to hours or days when using CPUs, our toolkit can significantly shorten the iteration cycle of evolutionary computation experiments.
EvoJAX is available at **https://github.com/google/evojax**

ACM Reference Format:
Yujin Tang, Yingtao Tian, and David Ha. 2022. EvoJAX: Hardware-Accelerated Neuroevolution. In *2022 Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3528770

1 INTRODUCTION

Hardware accelerators have played an important role in advancing the state-of-the-art for deep learning (DL), enabling rapid training of neural networks and shorter research iteration cycles for their development [12]. But much of this progress is restricted to systems that rely on gradient descent, a highly effective optimization method when we provide it with a well-defined objective function. But in areas such as artificial life, complex systems, computational biology, and even classical physics [18], much of the interesting behaviors we observe take place near the chaotic states, where a system is constantly transitioning between order and disorder. It can be argued that intelligent life and even civilization are all complex systems operating at the *edge of chaos* [3, 16]. If we wish to study these systems, we need efficient methods to simulate and find solutions in complex systems.

GECCO '22, July 9–13, 2022, Boston, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3528770

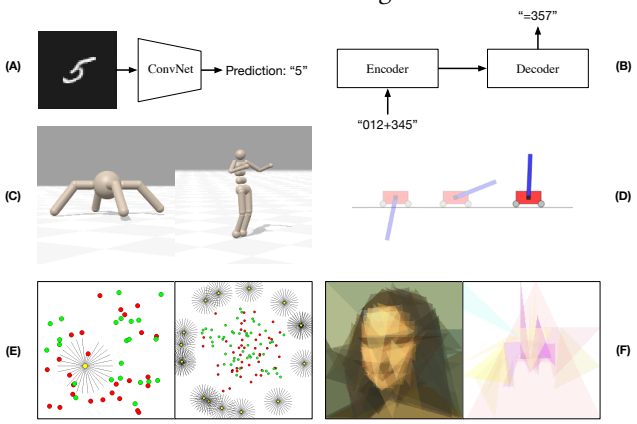


Figure 1: EvoJAX Examples. (A) MNIST classification. (B) Seq2Seq learning. (C) Robotic control. (D) Cart-pole swing up. (E) Left: WaterWorld wherein the agent (yellow) tries to get food (green) while avoiding poison (red). Right: A version of WaterWorld with multiple agents. (F) Abstract painting with only triangles. Left: Painting a concrete image. Right: Painting the concept "Walt Disney World".

Neural networks are a promising approach for modeling complex systems [9, 19], and neuroevolution has made great progress in developing methods for evolving neural networks to solve a wide range of problems. Evolution-based methods have been shown to find state-of-the-art solutions for reinforcement learning (RL) [8, 13, 22, 25, 29]. A policy with non-differentiable operations can solve many more tasks than one that is fully differentiable [20, 27, 28, 33]. More importantly, the removal of the requirement of a differentiable policy also liberates the researchers' mind, enabling higher levels of creativity for looking at problems and directions differently from the mainstream. In a sense, enabling researchers to use neural networks beyond gradient-based methods also enables the broader machine learning (ML) research community to explore in a way that is also less "grad student descent" [7]-based.

However, the progress of hardware-accelerated computational methods for evolution has not kept pace with ML, or even RL. Much of computational evolution is still conducted using CPU clusters, largely ignoring the recent breakthroughs in hardware accelerators such as GPUs/TPUs. Recent work started to demonstrate effectiveness of GPUs for neuroevolution [25], but so far such demonstrations are tailored for specific tasks [24], limiting their applicability to other domains. To enable greater access to hardware accelerators for neuroevolution researchers, we developed EvoJAX, a scalable, general purpose, neuroevolution toolkit. Building on the JAX library [1], our toolkit enables neuroevolution algorithms to work with neural networks running in parallel across multiple TPU/GPUs. EvoJAX achieves very high performance by implementing the evolution algorithm, neural network and task all in NumPy, which is compiled just-in-time to run on accelerators.

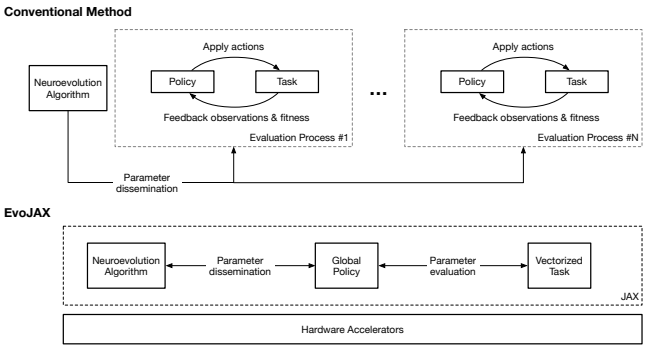


Figure 2: Architectural Overview of EvoJAX.

In this paper, we describe the design of EvoJAX and show how one can use and extend EvoJAX for neuroevolution research. We showcase several extensible examples of EvoJAX for a wide range of tasks, including supervised learning (image classification, seq-to-seq), RL (cart-pole swing-up [6], Brax locomotion [5], multi-agent water world), and generative art (image approximation with shapes, CLIP-guided abstract art [30]). We show that EvoJAX can find solutions to most of these tasks within minutes on GPU/TPUs, compared to hours or days when using CPUs. We believe our toolkit can significantly shorten the experimental iteration cycle for researchers working with evolutionary computation. We have also created several tutorials and notebooks as part of this open-source project to make adapting EvoJAX for novel use cases straightforward.

2 SYSTEM DESIGN

EvoJAX aims to improve the neuroevolution training efficiency by implementing the entire pipeline in modern ML frameworks that support hardware acceleration. We choose JAX[1] in our current implementation due to its wide variety of hardware support and its matured features of auto-vectorization, device-parallelism, just-in-time compilation, etc. As we will see in Section 4, as long as the component interfaces are properly implemented, EvoJAX also allows user extensions with other frameworks.

Figure 2 gives an overview of how EvoJAX works. There are three major components – the neuroevolution algorithm, the policy and the task. Although these components are common in conventional neuroevolution implementations, we highlight the key differences that make EvoJAX much more efficient:

Modern ML Optimizers Researchers and practitioners in the field of DL have been focusing on inventing optimization algorithms [21] and techniques [15, 32, 34] that are both fast and effective. Although these techniques were tailored for gradient-based optimizations, they can be directly applied to gradient estimation-based evolutionary algorithms [17, 23] too. By leveraging JAX-based libraries [1, 10, 11], EvoJAX not only achieves significant speed-up but also provides the users with the tools and the interfaces to develop their own implementations in a mature framework.

Global Policy In conventional neuroevolution implementations, it is a common practice to spawn multiple processes for parameters evaluation. To achieve hardware acceleration, the implementation adopts one of the DL frameworks and then each of the evaluation processes maintains a separate computational graph for the same policy. Unfortunately, most DL frameworks are not designed for multi-process training scenarios and often cause difficulties. Moreover, when these processes are run on the same accelerator, maintaining identical copies of the computational graph is a waste of resource. Conforming to the "Single-Program, Multiple-Data" (SPMD) model [4], EvoJAX solves this by building a global policy and treat both the task observations and the policy parameters as data for the computational graph. This global policy design is easy to implement as it is consistent with DL frameworks, and in the experiments we observe high data-throughput.

Vectorized Tasks Same as the policies, conventional methods also create copies of the tasks in the spawned processes for independent parameters evaluations. To be compliant with EvoJAX's global policy design, we propose to group these tasks in a vectorized form. In terms of implementation, this can be achieved by either creating the task in auto-vectorizaton supported frameworks or by creating a task observations collector on top of all the evaluation processes. EvoJAX adopts the first method.

Device Parallelism Thanks to the device-parallelism support in JAX, EvoJAX is capable of scaling its training procedure almost linearly to the available hardware accelerators. Utilizing EvoJAX's training pipeline, this device parallelism is automatically managed and is transparent to the users. As we will see in Section 3, together with the previously mentioned features, EvoJAX significantly shortens the training time for novel and non-trivial tasks.

EvoJAX defines simple yet functionally complete interfaces for the three components, any implementations that are compliant with the interfaces can be seamlessly integrated (see Section 4).

Finally, in addition to the mentioned major components, EvoJAX also comes with a trainer and a simulation manager that help orchestrate and manage the training process. They contain detailed implementations of task roll-out seeds generation, efficient training loops, time profiling and logistics operations such as logging, testing and periodic model saving. Convenient as they are, we point out that EvoJAX is a flexible toolkit, where it is possible to use any component independently (e.g., using a custom training loop).

3 EVOJAX EXAMPLES

We provide a total of six examples (see Figure 1) to showcase the capacity, efficiency and the usage of EvoJAX online in the format of Python scripts and notebooks. The examples are designed to feature different aspects of EvoJAX and are in three categories: Supervised Learning Tasks, Control Tasks and Novel Tasks. As the experimental setups, "Robotic Control" was trained with TPUs, "Concrete and Abstract Painting" was trained with 8 NVIDIA V100 GPUs, and the rest were trained with 1 NVIDIA V100 GPU.

Supervised Learning Tasks They provide both the data and the ground-truth labels to train the policy. In EvoJAX, supervised learning tasks are modelled as single-step tasks, the examples in this category are thus isolated from other factors to prove the correctness and efficiency of our algorithms' implementation.

- *MNIST Classification*. Here, we train a convolutional neural network (ConvNet) with 10K parameters with EvoJAX. Although MNIST is a solved problem in DL, it is non-trivial for neuroevolution in terms of achieving high test accuracy within a short time (e.g., in minutes). We show that EvoJAX can train the ConvNet to reach $> 98\%$ test accuracy within 5 minutes.

- *Seq2Seq Learning*. It has recently been shown that genetic algorithms (GA) can train large models [20]. Here, we show that EvoJAX is also capable of training a large network with hundreds of thousands of parameters. We adopt a seq-to-seq task where the policy is required to output a sequence after observing a query sequence. Concretely, the query is a sequence that represents the addition of two randomly generated integers (e.g., "012+345="), we pad the numbers with leading 0's so that they have equal lengths) and the result is a sequence representing the answer. Using an LSTM-based seq2seq [26] model, EvoJAX achieves $> 99\%$ test accuracy within tens of minutes.

While one would obviously use gradient-descent for such tasks in practice, the point is to show that neuroevolution can also solve them to some degree of accuracy within a short amount of time, which will be useful when these models are adapted within a more complicated task where gradient-based approaches may not work.

Control Tasks The purpose of including control tasks are twofold: 1) Unlike supervised learning tasks, control tasks in EvoJAX have undetermined number of steps, we thus use these examples to demonstrate the efficiency of our task roll-out loops. 2) We wish to show the speed-up benefit of implementing tasks in JAX and illustrate how to implement one from scratch.

- *Robotic Control*. Brax [5] is a differentiable physics engine implemented in JAX that simulates environments made up of rigid bodies, joints, and actuators. We show that it is easy to wrap Brax tasks in EvoJAX, and it takes EvoJAX tens of minutes to solve a robotic locomotion task on Colab TPUs.
- *Cart-Pole Swing Up*. Through this classic control task, we illustrate how a task is implemented from scratch in JAX and integrated into EvoJAX's training pipeline. In our implementation, a user can command the initial states to be randomly sampled from a narrow (easy version) or a wide (hard version) range of possible settings, with the latter being much harder to solve. EvoJAX solves both versions within minutes.

Novel Tasks In this last category, we go beyond simple illustrations and show examples of novel tasks that are more practical and attractive to researchers in the genetic and evolutionary computation area, with the goal of helping them try out ideas in EvoJAX.

- *WaterWorld*. In this task [14], an agent tries to get as much food as possible while avoiding poisons. EvoJAX is able to train the agent in tens of minutes. Furthermore, we demonstrate that multi-agents training in EvoJAX is possible. Here, we spawn the entire population in the same task roll-out and directly measure each agent's performance in a multi-agent world. This training scheme automatically generates task complexity beyond human design, and is beneficial for learning policies that can deal with interactions between agents and environmental uncertainties.
- *Concrete and Abstract Painting*. We reproduce the results from a computational creativity work [30]. The original work, whose implementation requires multiple CPUs and GPUs, could be accelerated on a single GPU efficiently using EvoJAX, which was not possible before. Moreover, with multiple GPUs/TPUs, EvoJAX can further speed up the mentioned work almost linearly. We also show that the modular design of EvoJAX allows its components be used independently – in this case it is possible to use only the neuroevolution algorithms from EvoJAX while leveraging one's own training loops and environment implantation.

Table 1: Time Comparisons. We report the training time for both methods to achieve widely accepted test scores.

	Baseline	EvoJAX
MNIST	36 min	3 min
Cart-Pole Swing Up (Hard Version)	37 min	2 min
Locomotion (Ant)[1]	201 min	9 min

We summarize EvoJAX's benefit via these examples. First of all, EvoJAX brings significant training speed up. In Table 1 we show the time costs of training some popular tasks with both a conventional setup and EvoJAX.[1] On modest hardware accelerators, EvoJAX trains $10 \sim 20$ times faster which leads to quicker idea iterations. Secondly, the capability of training multi-agents in a complex setting that is beyond human design supplies training environmental richness. And finally, EvoJAX puts the entire pipeline on unified hardware setups and that allows the practitioners to simplify complex hardware arrangements. As an example, for the substantial load of computation in our Abstract Painting example, the baseline needs to use both GPUs and CPUs, while EvoJAX only uses GPUs.

4 EXTENDING EVOJAX

A goal of EvoJAX is to provide researchers with an infrastructure that allows fast idea iterations. With EvoJAX it is possible to devise more effective neuroevolution algorithms, to explore novel policy architectures, and to experiment with new tasks. EvoJAX has carefully defined interfaces, as long as these interfaces are properly implemented, a user extended module can be integrated into the pipeline seamlessly.

```
import jax.numpy as jnp

class TaskState: obs: jnp.ndarray
class PolicyState: keys: jnp.ndarray
class NEAlgorithm:
    def ask(self) -> jnp.ndarray: pass
    def tell(self, fitness: jnp.ndarray) -> None: pass
class PolicyNetwork:
    def reset(self, states: TaskState) -> PolicyState: pass
    def get_actions(self, t_states: TaskState, params: jnp.ndarray,
                    p_states: PolicyState) \
                        -> Tuple[jnp.ndarray, PolicyState]: pass
class VectorizedTask:
      def reset(self, key: jnp.ndarray) -> TaskState: pass
      def step(self, state: TaskState, action: jnp.ndarray) \
                    -> Tuple[TaskState, jnp.ndarray, jnp.ndarray]: pass
```

Figure 3: Major Component Interfaces in EvoJAX.

Devising New Algorithms Users interested in inventing new neuroevolution algorithms should implement *NEAlgorithm* in Figure 3, which serves as the base class for all neuroevolution algorithms in EvoJAX. Being consistent with most conventional implementations, *NEAlgorithm* adopts the "ask" and "tell" interfaces, where the former requests the algorithm to generate a population of parameters and the latter reports the parameters evaluation results back to the algorithm for internal states update. Taking on the conventional interfaces for the neuroevolution algorithms not only brings familiarity to the developers and thus reducing the required

[1]We use the code from [27] as the baseline. For the Locomotion task, we use PyBullet Ant in the baseline and Brax Ant in EvoJAX. The baseline is trained with 96 CPUs.

learning effort, but also allows the practitioners to quickly plug in existing algorithms for sanity checks by writing a simple wrapper.

Exploring Novel Policy Architectures *PolicyNetwork* in Figure 3 defines the policy interface, all policies in EvoJAX implement the *get_actions* method. The method puts no restrictions on what the policy network should be or how it should behave, giving full freedom for neural architecture search (NAS). Because EvoJAX conforms to the SPMD model, *get_actions* accepts three parameters: the vectorized task states, the population parameters and the policy's internal states. At the beginning of a roll-out, each individual in the population sees identical observations, they will then diverge due to the population's different behaviors. Because JAX requires pure functions, the policy's states (e.g., random seeds, LSTM cell states, etc) are passed to *get_actions* via a Flax [10] dataclass *p_states*, which is initialized by *PolicyNetwork.reset*. The method returns the actions and the updated policy states. At runtime, calling *get_actions* is equivalent to passing a batch of data through the model.

Experimenting with More Tasks In Figure 3, *VectorizedTask* forms the base for all EvoJAX tasks. Similar to OpenAI's Gym environments [2], the interface defines the *reset* and the *step* methods. Following the pure-function principle of JAX, one major difference between EvoJAX tasks and Gym environments is that EvoJAX's tasks do not keep internal states. Instead, these states are encapsulated in a *TaskState* instance and carried over the roll-out steps. Similar to *PolicyState*, users can inherit *TaskState* and create one's own task specific state to encapsulate arbitrary information besides the environment observations. In most tasks, the initial states are generated via a procedure of randomness. The *reset* method thus accepts *key*'s that act as seeds for the random process.

5 LIMITATIONS AND FUTURE WORKS

EvoJAX is based on the JAX framework, which is based on the familiar NumPy and is thus friendly to researchers accustomed to such tools. However, practitioners may have to take effort to understand the subtleties of JAX in order to maximize its performance. The time spent on learning the JAX framework may translate to a delayed adoption of EvoJAX, hence much of our focus so far has been on creating examples and tutorials that others can use as templates to build upon. Another limitation of EvoJAX is the compatibility with existing non-parallelizable tasks. Although it is possible to create an observation collector on top of the evaluation processes to mimic the behavior of *VectorizedTask*, the operation involves inter-process communications that becomes a bottleneck, preventing such tasks from the benefit of hardware-acceleration.

In the future, we plan to release more neuroevolution algorithm implementations to EvoJAX in addition to PGPE [23, 31] in the current release. We will add more policies and tasks to both demonstrate a wider variety of examples in order to encourage greater adoption of EvoJAX, with the goal of further enhancing the computation tools available in evolutionary computation research.

REFERENCES

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/google/jax

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. 2016. Openai gym. *arXiv:1606.01540* (2016).

[3] Leon Chua, Valery Sbitnev, and Hyongsuk Kim. 2012. Neurons are poised near the edge of chaos. *International Journal of Bifurcation and Chaos* 22, 04 (2012).

[4] Frederica Darema. 2001. The spmd model: Past, present and future. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer.

[5] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. 2021. *Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation.* http://github.com/google/brax

[6] Daniel Freeman, David Ha, and Luke Metz. 2019. Learning to Predict Without Looking Ahead: World Models Without Forward Prediction. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc.

[7] Oguzhan Gencoglu, Mark van Gils, Esin Guldogan, Chamin Morikawa, Mehmet Süzen, Mathias Gruber, Jussi Leinonen, and Heikki Huttunen. 2019. HARK Side of Deep Learning–From Grad Student Descent to Automated Machine Learning. *arXiv:1904.07633* (2019).

[8] David Ha. 2020. Slime Volleyball Gym Environment.

[9] David Ha and Yujin Tang. 2021. Collective Intelligence for Deep Learning: A Survey of Recent Developments. *arXiv:2111.14377* (2021).

[10] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. 2020. *Flax: A neural network library and ecosystem for JAX.* http://github.com/google/flax

[11] Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. 2020. *Optax: composable gradient transformation and optimisation, in JAX!* http://github.com/deepmind/optax

[12] Sara Hooker. 2021. The hardware lottery. *Commun. ACM* 64, 12 (2021), 58–65.

[13] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv:1711.09846* (2017).

[14] Andrej Karpathy. 2015. *REINFORCEjs.* https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html

[15] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv:1609.04836* (2016).

[16] Roger Lewin. 1999. *Complexity: Life at the edge of chaos.* University of Chicago.

[17] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search of static linear policies is competitive for reinforcement learning. In *The 32nd Conference on Neural Information Processing Systems*. 1805–1814.

[18] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. 2021. Gradients are Not All You Need. *arXiv:2111.05803* (2021).

[19] Sebastian Risi. 2021. The Future of Artificial Intelligence is Self-Organizing and Self-Assembling. https://sebastianrisi.com/self_assembling_ai.

[20] Sebastian Risi and Kenneth O Stanley. 2019. Deep neuroevolution of recurrent and discrete world models. In *Proceedings of GECCO*. 456–462.

[21] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv:1609.04747* (2016).

[22] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864* (2017).

[23] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. 2010. Parameter-exploring policy gradients. *Neural Networks* 23, 4 (2010), 551–559.

[24] Felipe Such. 2018. *Accelerating Deep Neuroevolution: Train Atari in Hours on a Single Personal Computer.* https://eng.uber.com/accelerated-neuroevolution/

[25] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv:1712.06567* (2017).

[26] I. Sutskever, O. Vinyals, and Q. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in NIPS*. 3104–3112.

[27] Yujin Tang and David Ha. 2021. The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning. In *The 35th Conference on Neural Information Processing Systems*.

[28] Yujin Tang, Duong Nguyen, and David Ha. 2020. Neuroevolution of Self-Interpretable Agents. In *Genetic and Evolutionary Computation Conference*.

[29] Yujin Tang, Jie Tan, and Tatsuya Harada. 2020. Learning agile locomotion via adversarial training. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6098–6105.

[30] Yingtao Tian and David Ha. 2021. Modern Evolution Strategies for Creativity: Fitting Concrete Images and Abstract Concepts. *arXiv:2109.08857* (2021).

[31] Nihat Engin Toklu, Paweł Liskowski, and Rupesh Kumar Srivastava. 2020. ClipUp: A Simple and Powerful Optimizer for Distribution-Based Policy Evolution. In *International Conference on Parallel Problem Solving from Nature*. 515–527.

[32] Twan Van Laarhoven. 2017. L2 regularization versus batch and weight normalization. *arXiv:1706.05350* (2017).

[33] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. 2019. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv:1901.01753* (2019).

[34] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. 2019. How does learning rate decay help modern neural networks? *arXiv:1908.01878* (2019).