

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

Гордієнко Ю.Г., Кочура Ю.П.

МЕТОДИ DEEP LEARNING

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

Методи Deep Learning

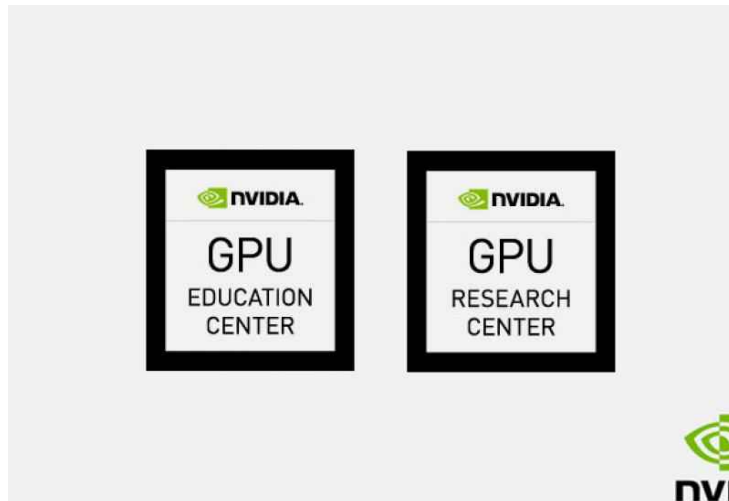
Лекція 1

Слайди лекцій і ДЕМО-ноутбуки для хмарного ресурсу Google Colaboratory:
<https://cloud.comsys.kpi.ua/f/439632>

Лекція 1 – Вступ до предмету

Курс містить матеріали, запропоновані NVIDIA Deep Learning Institute (DLI) в рамках спільних:

Дослідницький центр NVIDIA
і
Освітній центр NVIDIA.



<https://kpi.ua/nvidia-info>

МЕТОДИ ГЛИБОКОГО НАВЧАННЯ

ЛЕКЦІЯ 1: ВСТУП

Юрій Гордієнко, сертифікований інструктор DLI



DEEP
LEARNING
INSTITUTE



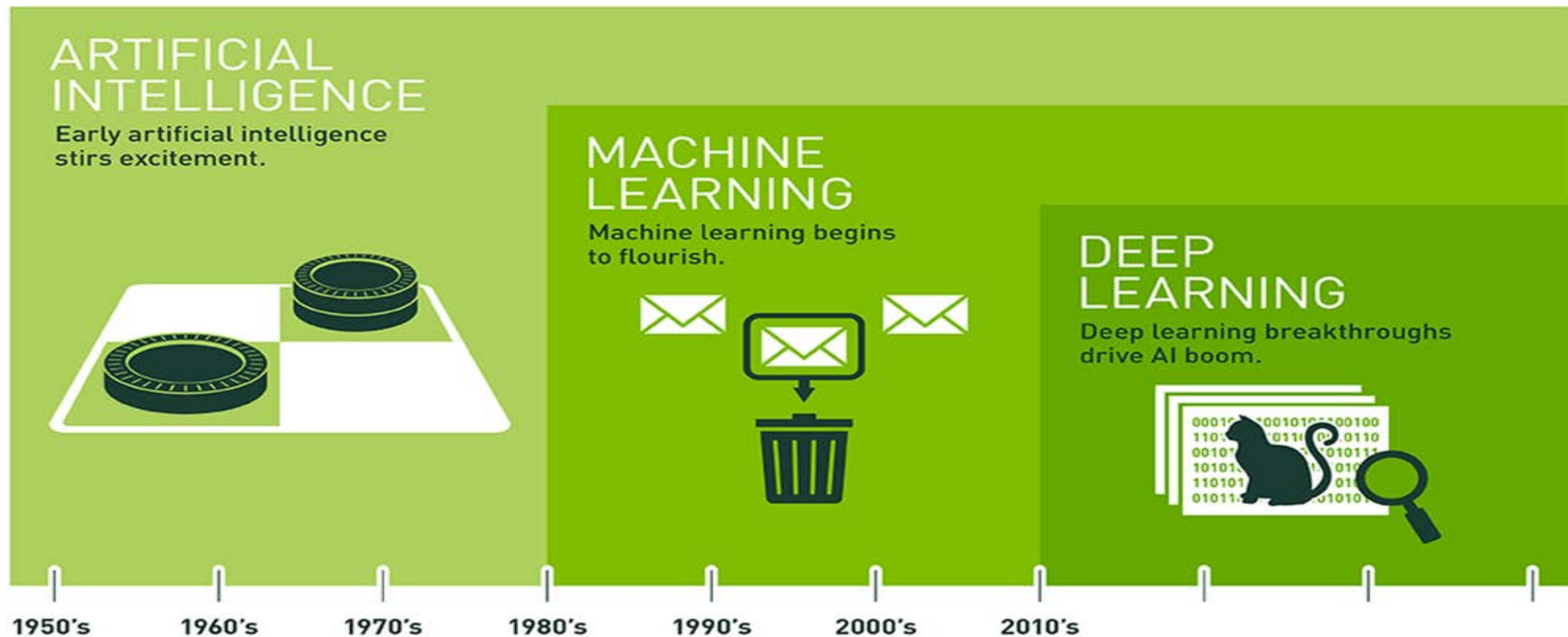
DEEP LEARNING INSTITUTE

Місія DLI

Навчить вас вирішувати найскладніші проблеми світу.

- Розробники, спеціалісти з обробки даних та інженери
- Безпілотні автомобілі, охорона здоров'я та робототехніка
- Навчання, оптимізація та розгортання глибоких нейронних мереж

ВИЗНАЧЕННЯ



ГЛИБКЕ НАВЧАННЯ ЦЕ МІЖ ГАЛУЗЯМИ

Інтернет-сервіси



- Класифікація зображення/відео
- Розпізнавання мови
- Обробка природної мови

Ліки



- Виявлення ракових клітин
- Діабетична оцінка
- Відкриття ліків

ЗМІ та розваги



- Субтитри до відео
- Пошук на основі вмісту
- Переклад у реальному часі

Безпека та оборона



- Розпізнавання обличчя
- Відеоспостереження
- Кібербезпека

Автономні машини

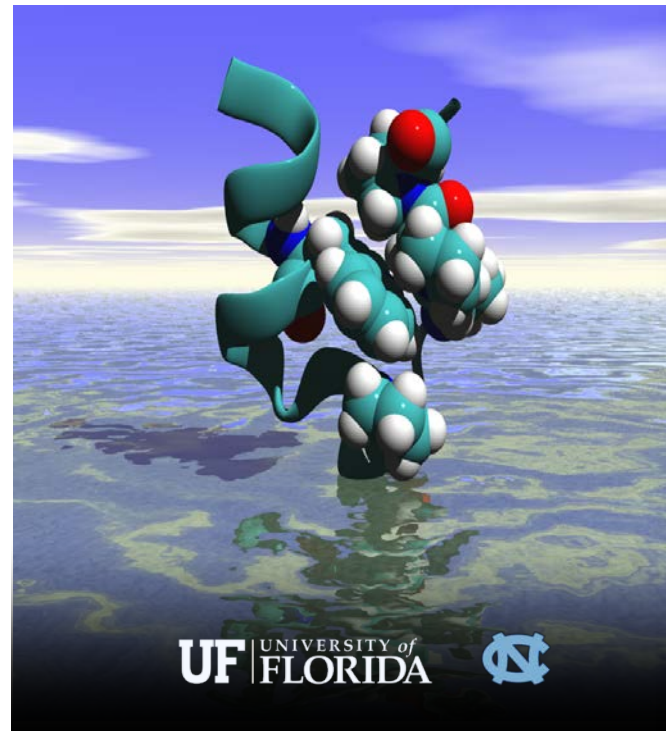


- Виявлення пішоходів
- Відстеження смуги руху
- Розпізнавати дорожні знаки

ГЛИБКЕ НАВЧАННЯ ТРАНСФОРМУЄ HPC



«Побачити» гравітацію в реальному часі



Прискорення відкриття ліків



92%

вірять, що ШІ вплине на їхню роботу

93%

використання глибокого навчання для досягнення позитивних результатів

inside
HPC

Опитування insideHPC.com
Листопад 2016 року

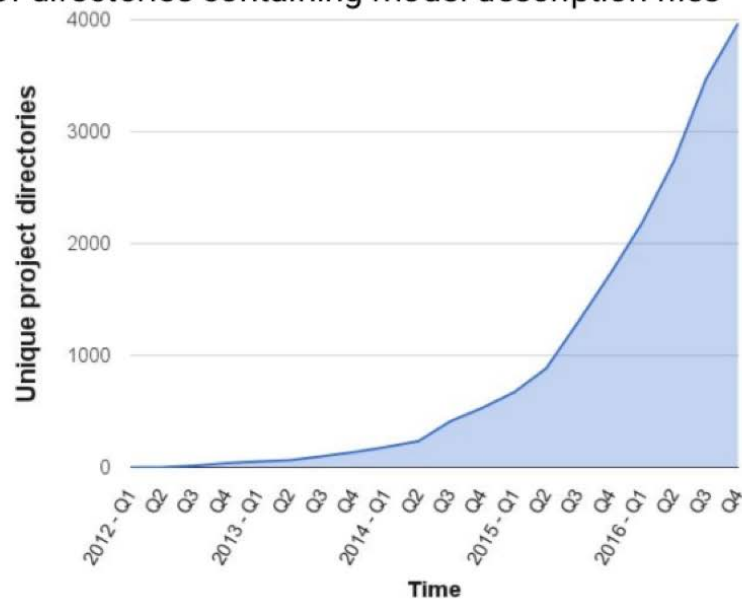


ШІ Є КРИТИЧНО ВАЖЛИВИМ ДЛЯ ІНТЕРНЕТ-ДОДАТКІВ

Користувачі очікують інтелектуальних послуг

Growing Use of Deep Learning at Google

of directories containing model description files



Across many products/areas:

- Android
- Apps
- drug discovery
- Gmail
- Image understanding
- Maps
- Natural language understanding
- Photos
- Robotics research
- Speech
- Translation
- YouTube
- ... many others ...



THE EXPANDING UNIVERSE OF MODERN AI

"THE BIG BANG"

Big Data
GPU
Algorithms



RESEARCH

CORE TECHNOLOGY / FRAMEWORKS



AI-as-a-PLATFORM



START-UPS



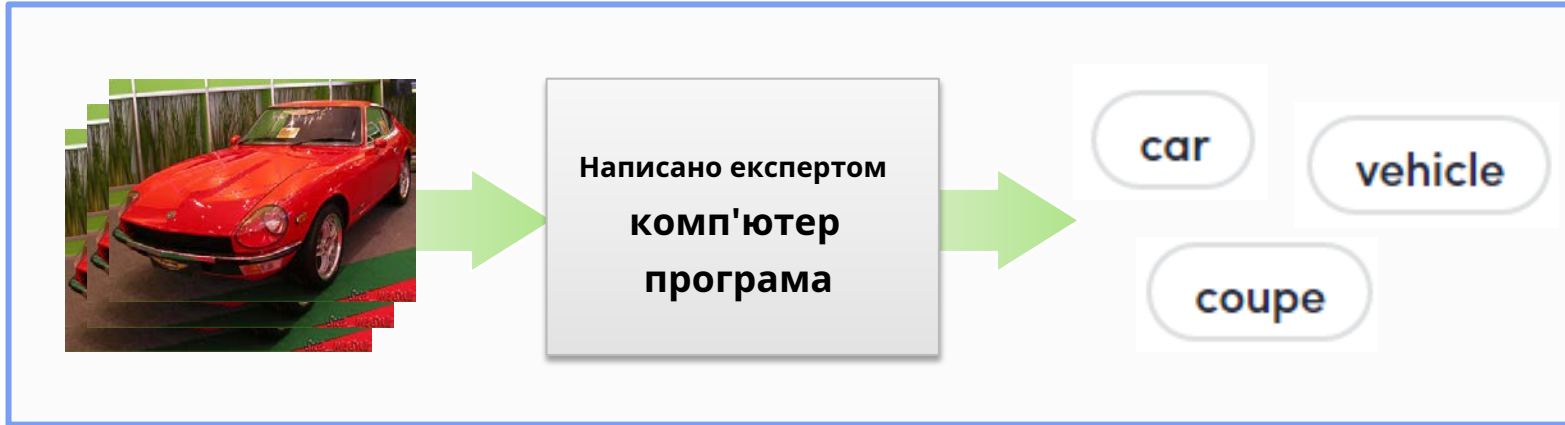
1,000+ AI START-UPS
\$5B IN FUNDING
 Source: Venture Scanner

INDUSTRY LEADERS



НОВА КОМП'ЮТЕРНА МОДЕЛЬ

Алгоритми, які навчаються на прикладах

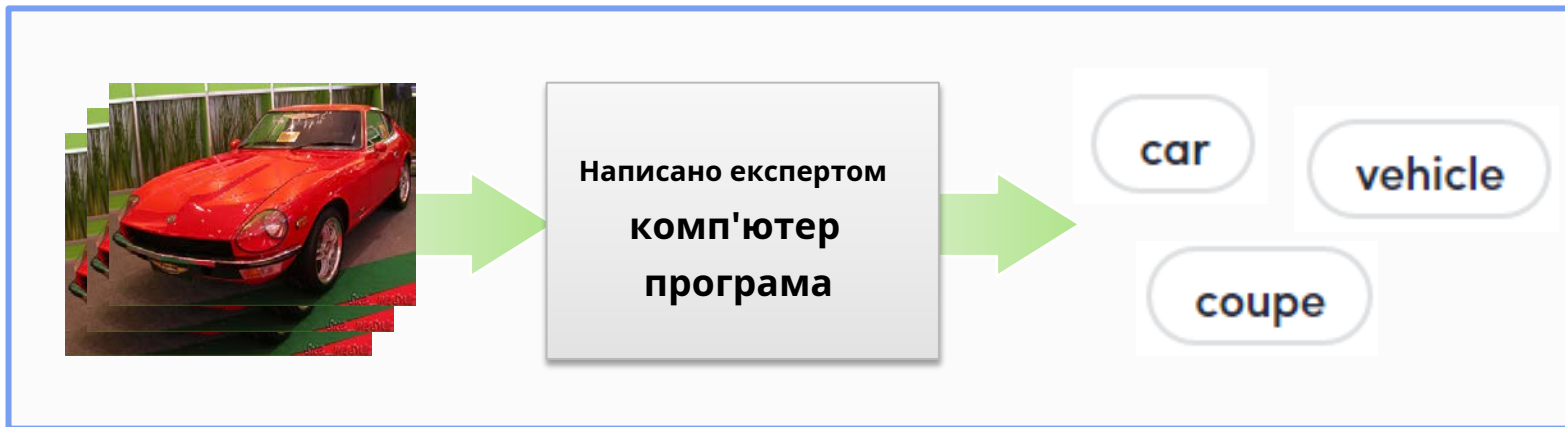


Традиційний підхід

- Потрібні експерти домену
- Забирає багато часу
- Схильний помилятися
- Не масштабується до нових проблем

НОВА КОМП'ЮТЕРНА МОДЕЛЬ

Алгоритми, які навчаються на прикладах



Традиційний підхід

- Потрібні експерти домену
- Забирає багато часу
- Схильний помилятися
- Не масштабується до нових проблем



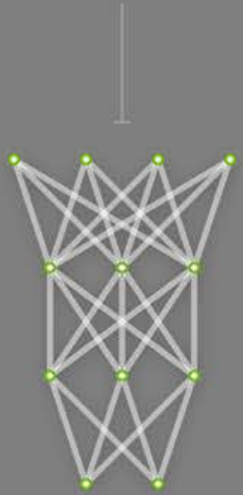
Підхід до глибокого навчання

- ✓ Вчиться на даних
- ✓ Легко розширюється
- ✓ Прискорення за допомогою графічних процесорів



DEEP LEARNING

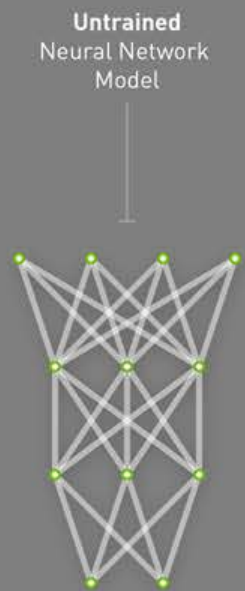
Untrained
Neural Network
Model



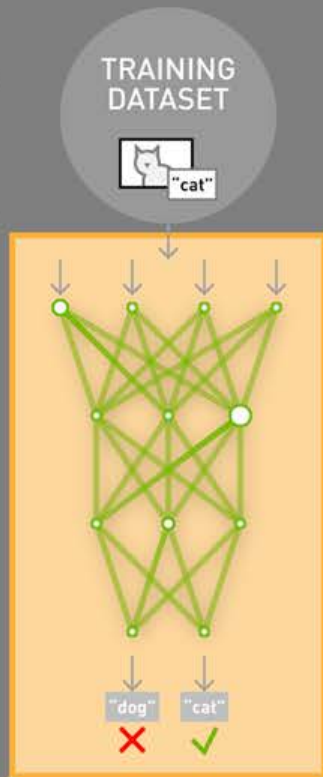
DEEP LEARNING

TRAINING

Learning a new capability
from existing data



Deep Learning
Framework



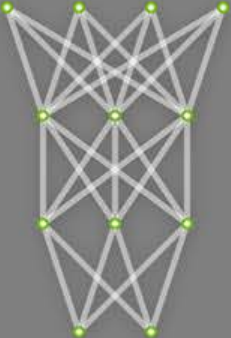
DEEP LEARNING

TRAINING

Learning a new capability
from existing data

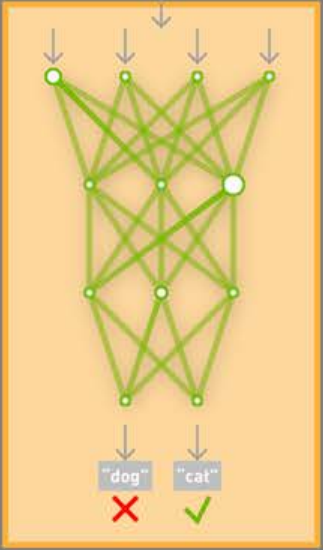


Untrained
Neural Network
Model

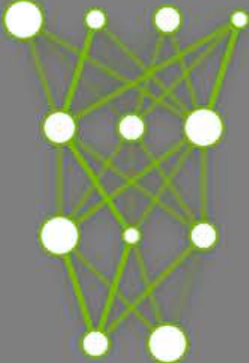


Deep Learning
Framework

TRAINING
DATASET



Trained Model
New Capability

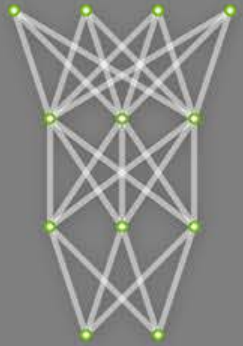


DEEP LEARNING

TRAINING

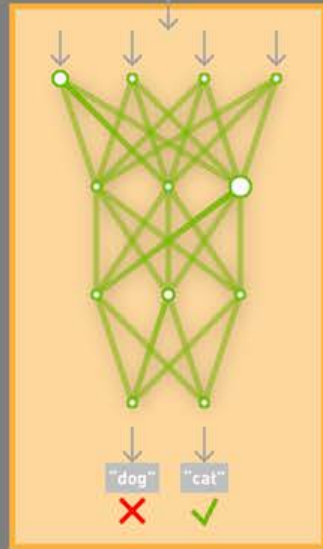
Learning a new capability
from existing data

Untrained
Neural Network
Model



Deep Learning
Framework

TRAINING
DATASET



Trained Model
New Capability



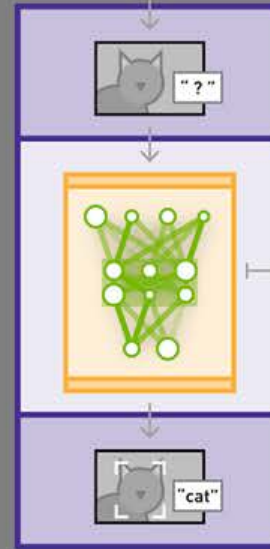
INFERENCE

Applying this capability
to new data

NEW
DATA



App or Service
Featuring Capability



Trained Model
Optimized for
Performance



ВИКЛИКИ

Потреби в глибокому навчанні	Чому
Науковці даних	Нова обчислювальна модель
Найновіші алгоритми	Швидко розвивається
Швидке навчання	Неможливо -> Практично
Платформи розгортання	Має бути доступним всюди

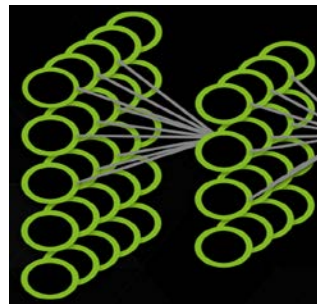
ГЛИБИННЕ НАВЧАННЯ NVIDIA ІНСТИТУТ

Онлайн-лабораторії для самостійного навчання та семінари під керівництвом інструкторів із глибокого навчання та прискорені обчислення

Пройдіть самостійні лабораторні роботи за адресою www.nvidia.com/dlilabs

Перегляньте майбутні семінари та замовте семінар на місці за адресою www.nvidia.com/dli

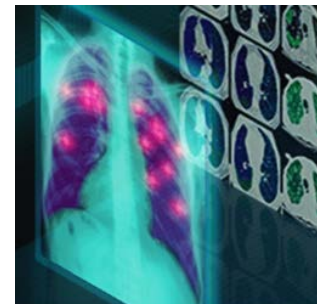
Викладачі можуть приєднатися до програми University Ambassador Program, щоб викладати курси DLI на кампусі та мати доступ до ресурсів. Дізнайтесь більше на www.nvidia.com/dli



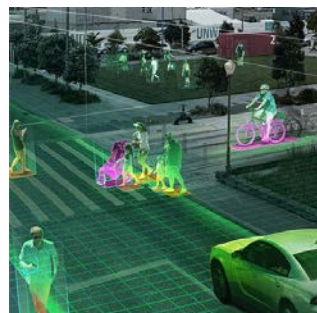
ОСНОВИ



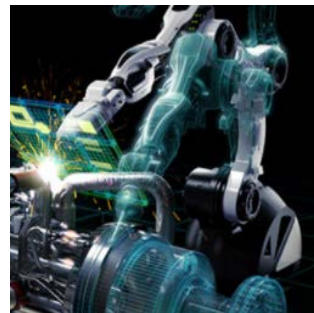
Автономні транспортні засоби



Охорона здоров'я



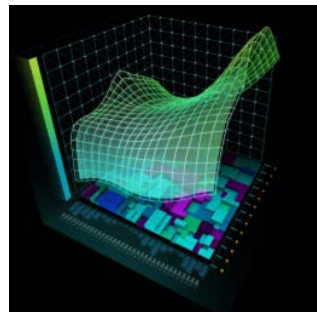
Інтелектуальне відео
Аналітика



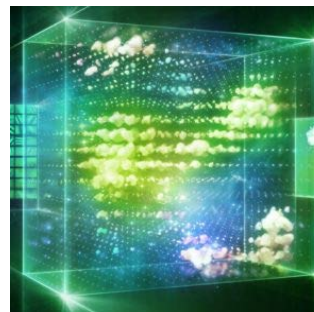
Робототехніка



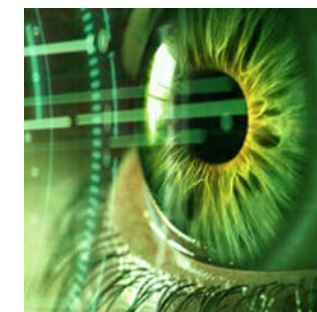
Розробка ігор та
Цифровий контент



Фінанси

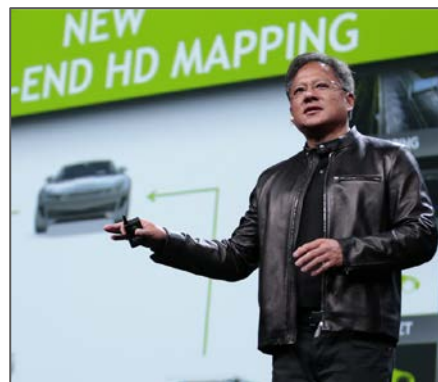
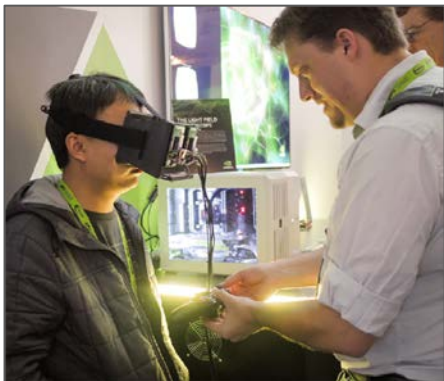


Прискорене обчислення



Віртуальна реальність

GPU TECHNOLOGY CONFERENCE



РОЗВИШІТЬ СВОЄ ПОГЛИБЛЕНЕ НАВЧАННЯ В GTC

Не пропустіть найважливішу у світі подію для розробників GPU

ПОДАВАТИ МАТЕРІАЛИ

Продемонструйте свою блискучу роботу **онлайн** на GTC 2021 - **дата уточнюється**

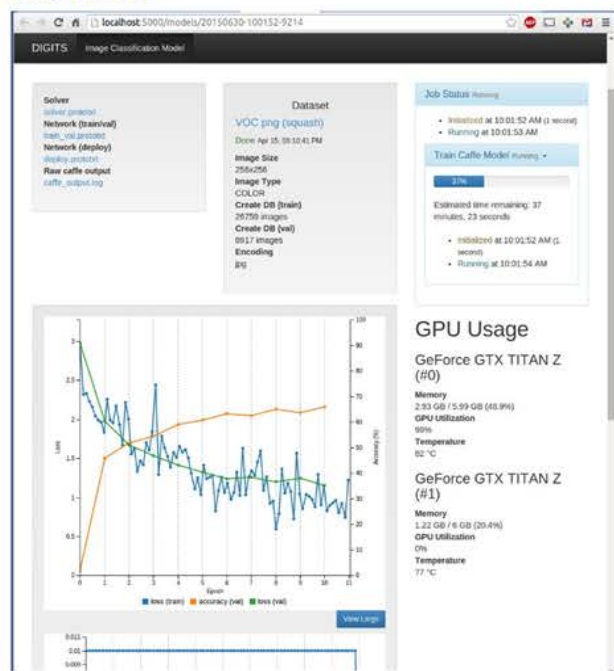
<https://www.nvidia.com/en-us/gtc/call-for-submissions/>

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ГЛИБОКОГО НАВЧАННЯ

NVIDIA DIGITS™

Interactively manage data and train deep learning models for image classification without the need to write code.

[Learn more](#)



Deep Learning Frameworks

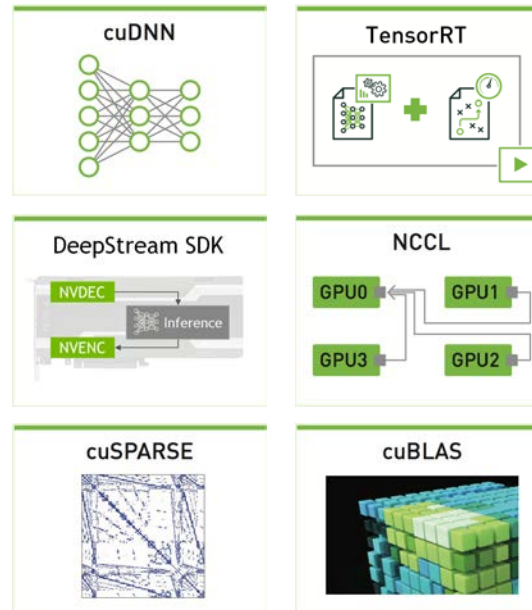
Design and train deep learning models using a high-level interface. Choose a deep learning framework that best suits your needs based on your choice of programming language, platform, and target application.

[Learn more](#)



NVIDIA Deep Learning SDK

This SDK delivers high-performance multi-GPU acceleration and industry-vetted deep learning algorithms, and is designed for easy drop-in acceleration for deep learning frameworks.



developer.nvidia.com/deep-learning

СІМЕЙСТВО НАСКРІЗНИХ ПРОДУКТІВ

НАВЧАННЯ

ПОВНІСТЮ ІНТЕГРОВАНІЙ СУПЕРКОМП'ЮТЕР DL



Станція DGX-1 і DGX

НАСТОЛЬНИЙ ПК



Титан X Паскаль

ЦЕНТР ОБРОБКИ ДАНИХ



Tesla P100

TTeeslalaVP110000

ВИСНОВОК

ЦЕНТР ОБРОБКИ ДАНИХ



Tesla P100/V100



Tesla P4

АВТОМОБІЛЬНА



Привід PX2

Вбудований



Jetson TX1



ВИКЛИКИ

Потреби в глибокому навчанні	Чому
Науковці даних	Нова обчислювальна модель
Найновіші алгоритми	Швидко розвивається
Швидке навчання	Неможливо -> Практично
Платформи розгортання	Має бути доступним всюди



ВИКЛИКИ

Потреби в глибокому навчанні	NVIDIA Доставляє
Науковці даних	DeepMind, FranRinegxCInesetditsustuep, pGlyTC, DIGITS
Найновіші алгоритми	DALP, CUDNN, eGvPoUlv-iAncgcelerated Frameworks
Швидке навчання	IDmGpXo,sVsi1b0le, P1P0r0a,ctiIcTaAIN X, A100(!)
Платформи розгортання	MunstsobreRTa,vaPi1la0b0l,ePe4v,eDrriiwehePrXe, Джетсон



ГОТОВІ ПОЧАТИ?

Контрольний список проекту

1. Яку проблему ви вирішуєте, які завдання DL?
2. Які дані ви маєте/потрібні та як вони позначені?
3. Яку структуру та інструменти глибокого навчання ви будете використовувати?
4. На якій(их) платформі(ях) ви будете навчатися та розгортатися?

ЯКУ ПРОБЛЕМУ ВИ ВИРІШУЄТЕ?

Визначення завдань AI/DL

ВХОДИ	ПИТАННЯ	ЗАВДАННЯ AI/DL	ПРИКЛАД ВИХОДИ
 Текстові дані  Зображення  відео  Аудіо	Є це <u>присутній</u> чи ні?	виявлення	Виявлення раку
	Що <u>типу</u> речі є це"?	Класифікація	пухлина Ідентифікація
	До того, що <u>ступінь</u> чи присутнє «це»?	Сегментація	Розмір/форма пухлини Аналіз
	Яка ймовірність <u>результат</u> ?	Прогнозування	Живучість Прогнозування
	Що ймовірно буде <u>задовольнити мету</u> ?	Рекомендація	Терапія Рекомендація

ВИБІР РАМКИ ГЛИБОКОГО НАВЧАННЯ

міркування

1. Тип проблеми
2. Платформи для навчання та розгортання
3. Доступні моделі DNN, підтримуються типи шарів
4. Останні алгоритми та прискорення GPU: cuDNN, NCCL тощо.
5. Модель/інтерфейси використання: GUI, командний рядок, мова програмування тощо.
6. Легко встановити та почати роботу: контейнери, документи, зразки коду, навчальні посібники, ...
7. Інтеграція підприємства, постачальники, екосистема

ПОЧИНАЙ ПРОСТО, ШВИДКО НАВЧАЙСЯ



Як один NVIDIA використовує глибоке навчання,
щоб не дати котам гадити на його газоні



ЩО ДАЛІ?

Вивчайте більше

Слухай [Подкаст NVIDIA AI](#) огляд
[прикладів ШІ в дії](#)

Пройдіть самостійну лабораторію

www.nvidia.com/dlilabs

Відвідайте семінар під керівництвом інструктора

Або замовте семінар на місці
www.nvidia.com/dli

Приєднайтеся до програми для розробників

<https://developer.nvidia.com/join>

Зв'яжіться з нами за адресою nvdli@nvidia.com



DEEP
LEARNING
INSTITUTE

www.nvidia.com/dli

Deep Learning Methods - Lecture 2 — Data, Datasets, Exploratory Data Analysis (EDA)

Yuri Gordienko, DLI Certified Instructor



DEEP
LEARNING
INSTITUTE



DEEP LEARNING INSTITUTE

DLI Mission

Training you to solve the world's most challenging problems.

- Developers, data scientists and engineers
- Self-driving cars, healthcare and robotics
- Training, optimizing, and deploying deep neural networks

This Lecture Overview

- Understand different types and formats of data
 - Be able to soundly select appropriate data
 - Have awareness of biases that exist
- Be able to refine questions to suite your true inquiry
- Understand how to parse text with regular expressions

Definitions

Definitions - Data

- Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation
 - Information in digital form that can be transmitted or processed
- Information output by a sensing device or organ that includes both useful and irrelevant or redundant information and must be processed to be meaningful

Definitions — Datum, Data, Dataset

- **Datum** - A single piece of information, which can be treated as an **observation**;
 - **Data** - The **plural** of datum; multiple observations;
- **Dataset** - A homogenous **collection** of data (each datum must have the same focus)

Definitions — Data Sources

- Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation
 - Information in digital form that can be transmitted or processed
- Information from sensors or organs that includes both useful and irrelevant or redundant information and must be processed to be meaningful

Definitions — Data Sources - Use Cases

- Measurements from a thermometer every hour for a year
- Counts from a person who tracks the days that a particular hummingbird visits his birdfeeder across an entire year
 - Tweets from Elon Musk
- Readouts from a mysterious sensor, for example, from wundergorund.com

Definitions — Data Sources - Use Cases

- Measurements from a thermometer every hour for a year
 - **Probably inaccurate data**
- Counts from a person who tracks the days that a particular hummingbird visits his birdfeeder across an entire year
 - Tweets from Elon Musk
- Readouts from a mysterious sensor, for example, from wundergorund.com

Definitions — Data Sources - Use Cases

- Measurements from a thermometer every hour for a year
 - **Probably inaccurate data**
- Counts from a person who tracks the days that a particular hummingbird visits his birdfeeder across an entire year
 - **Probably missing data**
 - Tweets from Elon Musk
- Readouts from a mysterious sensor, for example, from wundergorund.com

Definitions — Data Sources - Use Cases

- Measurements from a thermometer every hour for a year
 - **Probably inaccurate data**
- Counts from a person who tracks the days that a particular hummingbird visits his birdfeeder across an entire year
 - **Probably missing data**
 - Tweets from Trump
 - **Probably not 100% factually true**
- Readouts from a mysterious sensor, for example, from wundergorund.com

Definitions — Data Sources - Use Cases

- Measurements from a thermometer every hour for a year
 - **Probably inaccurate data**
- Counts from a person who tracks the days that a particular hummingbird visits his birdfeeder across an entire year
 - **Probably missing data**
 - Tweets from Trump
 - **Probably not 100% factually true**
- Readouts from a mysterious sensor, for example, from wundergorund.com

Data Processing

-

Workflow

-

Datasets

Recall -> Data Processing from Scientific Point of View

Ask an interesting question

Get the Data

Explore the Data

Model the Data

Visualize the Results

Dataset — Important Questions!

- What data is necessary to **answer** our question?
 - How **difficult** is it to analyze a dataset?
- Is the source **authoritative**? (.com, .net, .org, .gov, .name)
 - **Comprehensive** data vs sampled data?
 - **Biases**
- What is the allowed usage of data under its **license**?
 - **Who** collected the data?
 - **When** was the data collected?
 - **How** was the data collected?
 - **How** is the data **formatted**?
 - **Ethical** issues?

Dataset — Examples

- Open access data



29 billion words in
55 million articles in 309
languages

- Collected and digitized as part of generalized procedures of an institution



~610 million tweets
per day

<https://www.internetlivestats.com/twitter-statistics/>

Dataset — Important Questions!

- What data is necessary to **answer** our question?
 - How **difficult** is it to analyze a dataset?
- Is the source **authoritative**? (.com, .net, .org, .gov, .name)
 - **Comprehensive** data vs sampled data?
 - **Biases**
- What is the allowed usage of data under its **license**?
 - **Who** collected the data?
 - **When** was the data collected?
 - **How** was the data collected?
 - **How** is the data **formatted**?
 - **Ethical** issues?

Dataset — Common Problems!

- **Omission:** Using only arguments from one side
- **Source selection:** Include more sources or more authoritative sources for one side over the other
- **Story selection:** Regularly including stories that agree or reinforce the arguments of one side
- **Labelling:**
 - Using only arguments from one side
 - Labeling people on one side of the argument with labels and not the other
- **Spin:** Story provides only one interpretation of the events

Dataset —

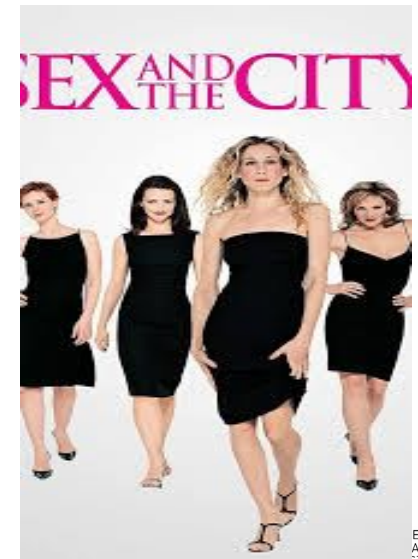
Common Problems — IMDb Example

- Registered users rate films 1-10 stars; they are an **overrepresented subpopulation** relative to the **general population**.
- Registered users who rate movies in their free time **over represents** a specific segment of the **general population**.

Example: “Men Are Sabotaging The Online Reviews Of TV Shows Aimed At Women”

60% who rated “*Sex in the City*” were women.

Women gave it a **8.1**, men gave it **5.8**.

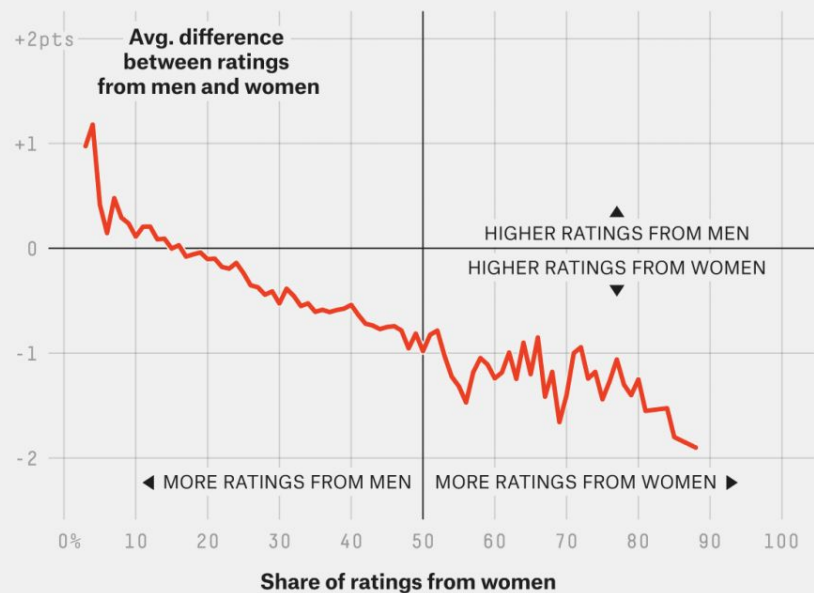


Dataset —

Common Problems — **IMDb Example**

Men tank the ratings of shows aimed at women

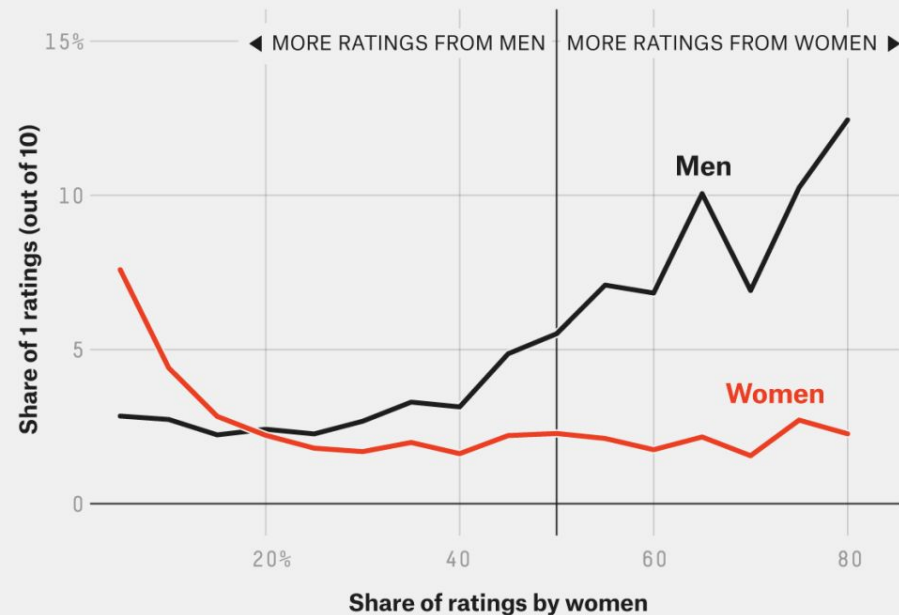
Average difference between IMDb ratings of TV shows from men and women by share of ratings from women



For English language shows with 1,000 or more ratings

Men are more likely to give the crappiest rating

Share of IMDb ratings of 1 (out of 10) for shows with at least 10,000 ratings by share of ratings from women*



*Rounded to nearest 5 percent

Dataset —

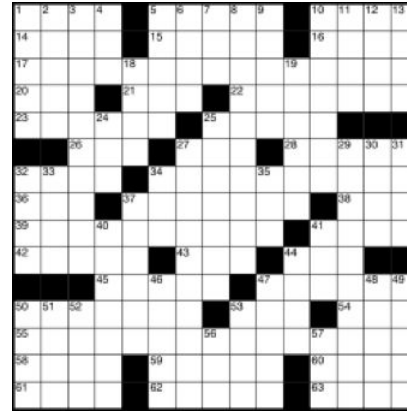
Common Problems — **Resume**

- Nearly all datasets **involve a human** in some way or another.
- This means that **nearly all** datasets probably has **bias**.
- Our goal: to **minimize the bias** as much as possible.
- For **models** (later), the **same** advice should be applied.

Datasets — Easy vs. Hard

hard for computers

easy for computers



	A	B	C
1	name	age	height
2	Michael	46	5'9"
3	Jim	31	6'0"
4	Pam	29	5'7"
5	Meredith	53	5'6"
6	Dwight	35	5'10"

Confusion at Palm Beach County polls

Some AI Gore supporters may have mistakenly voted for Pat Buchanan because of the ballot's design.

Although the Democrats are listed second in the column on the left, they are the third hole on the ballot.

Punching the second hole casts a vote for the Reform party.



easy for people



hard for people

Dataset — Easy vs. Hard

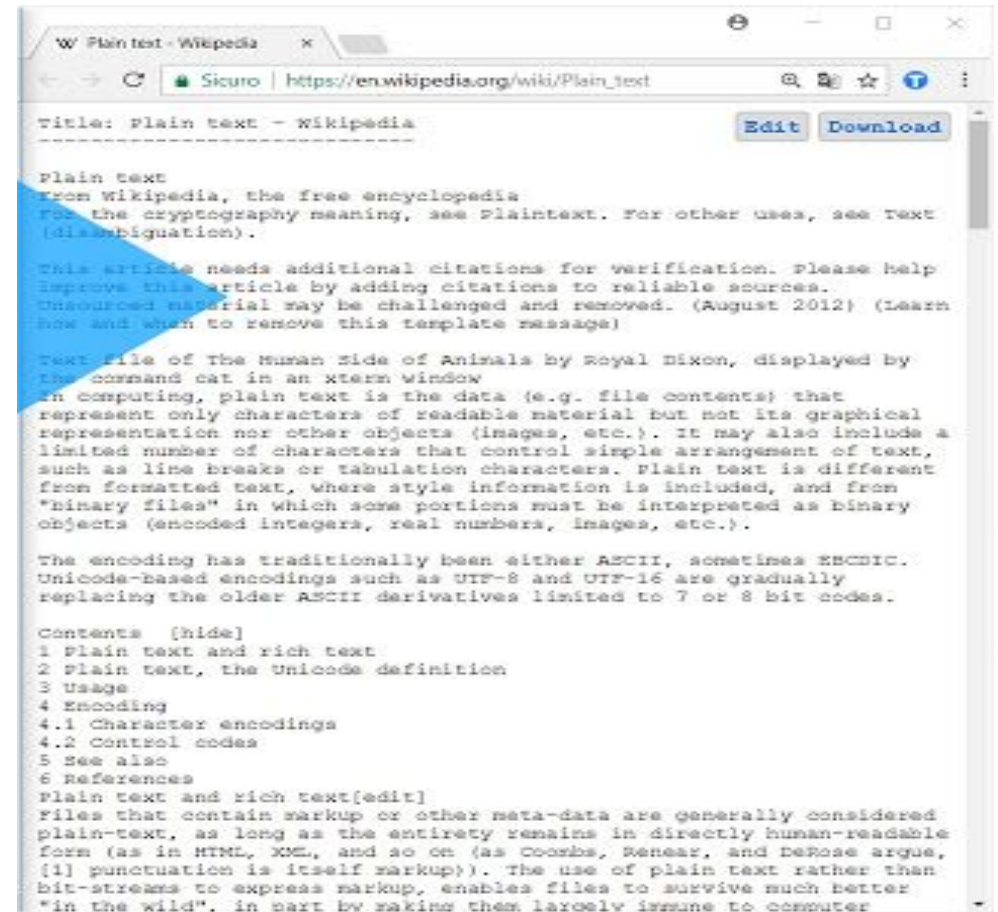
Computers are better at ‘understanding’ photos and videos, and text and numbers are much easier.

Why?

Structured data (e.g., spreadsheet formatted data)
is **much easier** than
unstructured data (e.g., free-flowing essays)

Dataset — Text Formats - Plain

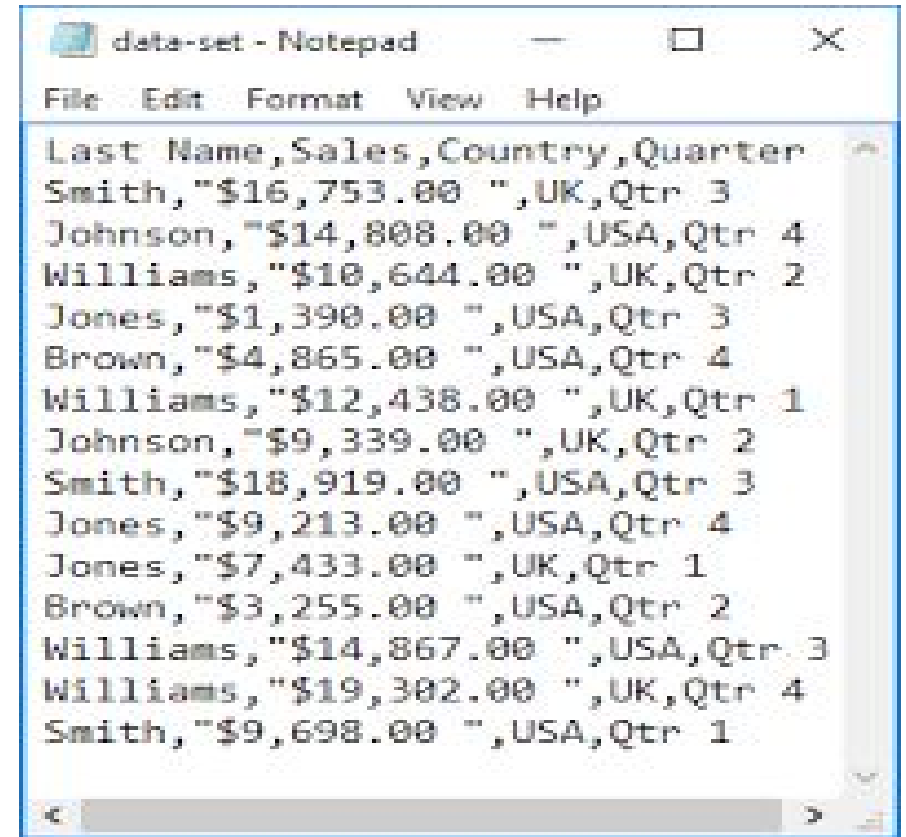
- File extension ends in .txt (generally)
- **No formatting:** font type, font size, color, etc.
- **Text is delimited** (position provided) by whitespace characters (space, tab, return)



Dataset — Text Formats - Plain

Delimiter: The character that separates each value

- Comma-separated (.csv)
- Tab-separated (.tsv)



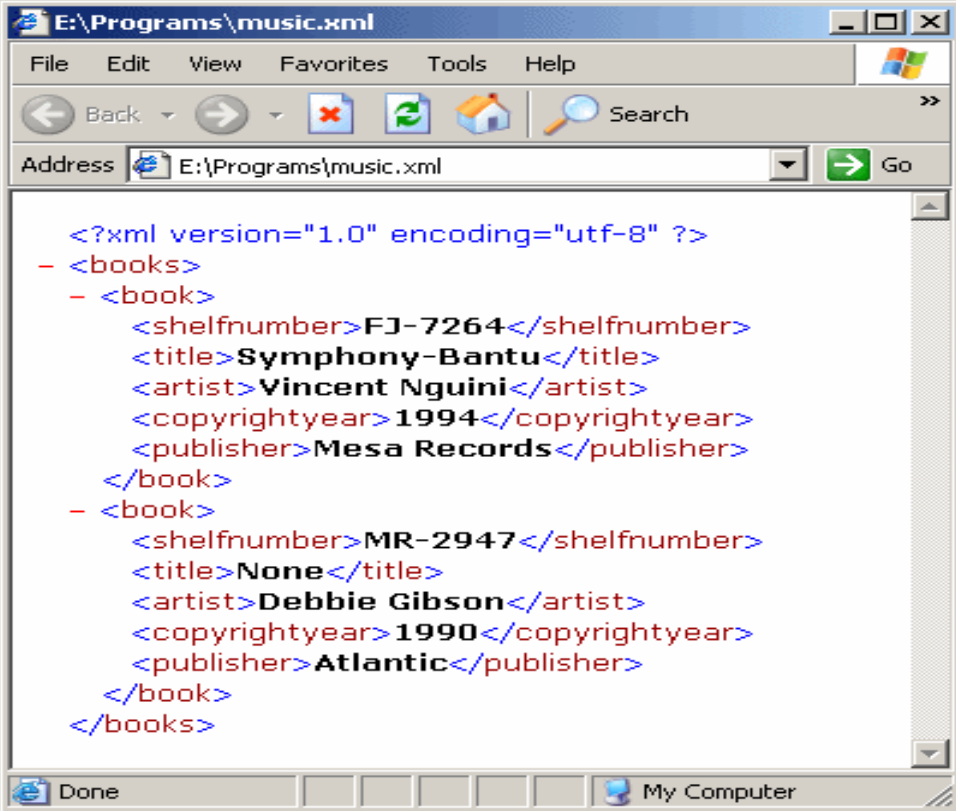
```
data-set - Notepad
File Edit Format View Help
Last Name,Sales,Country,Quarter
Smith,"$16,753.00 ",UK,Qtr 3
Johnson,"$14,808.00 ",USA,Qtr 4
Williams,"$10,644.00 ",UK,Qtr 2
Jones,"$1,390.00 ",USA,Qtr 3
Brown,"$4,865.00 ",USA,Qtr 4
Williams,"$12,438.00 ",UK,Qtr 1
Johnson,"$9,339.00 ",UK,Qtr 2
Smith,"$18,919.00 ",USA,Qtr 3
Jones,"$9,213.00 ",USA,Qtr 4
Jones,"$7,433.00 ",UK,Qtr 1
Brown,"$3,255.00 ",USA,Qtr 2
Williams,"$14,867.00 ",USA,Qtr 3
Williams,"$19,302.00 ",UK,Qtr 4
Smith,"$9,698.00 ",USA,Qtr 1
```

Dataset — Text Formats - XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

XML (.xml)

The colors aren't actually stored in the file, the editor just adds them on your screen to help make it look prettier.



```
<?xml version="1.0" encoding="utf-8" ?>
- <books>
- <book>
  <shelfnumber>FJ-7264</shelfnumber>
  <title>Symphony-Bantu</title>
  <artist>Vincent Nguini</artist>
  <copyrightyear>1994</copyrightyear>
  <publisher>Mesa Records</publisher>
</book>
- <book>
  <shelfnumber>MR-2947</shelfnumber>
  <title>None</title>
  <artist>Debbie Gibson</artist>
  <copyrightyear>1990</copyrightyear>
  <publisher>Atlantic</publisher>
</book>
</books>
```

Dataset — Text Formats - JSON

JSON (JavaScript Object Notation) is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types.

```
{
  hey: "guy",
  anumber: 243,
  - anobject: {
    whoa: "nuts",
    - anarray: [
      1,
      2,
      "thr<h1>ee"
    ],
    more: "stuff"
  },
  awesome: true,
  bogus: false,
  meaning: null,
  japanese: "明日がある。",
  link: http://jsonview.com,
  notLink: "http://jsonview.com is great"
}
```

- Like XML, data is annotated
- A nesting of key-value pairs
- Can be more space efficient than XML

Dataset — Text Formats - Resume

- They can all express the same content
- Plain Text doesn't have structure, but is universally robust
 - XML is the most verbose, harder to parse
 - JSON doesn't have `</stuff_here>` end tags
- JSON is more succinct than XML (easier to parse)

Data Processing

-

Workflow

-

Starting Question

Recall -> Data Processing from Scientific Point of View

Ask an interesting question

Get the Data

Explore the Data

Model the Data

Visualize the Results

Starting Question — Concrete

- It's crucial for your starting question (**assumption**) of data research to have **concrete defined terms that can be proven true or false.**
 - **Assumption:** “Voting turnout is high”
 - Where? Ukraine? World-wide?
 - What type of voting? Presidential races, local elections?
 - What is our metric? Number of total votes. Percentage of the population?
 - What's our actual time scale?

Starting Question — Resume

- The **more specific** your questions, the **more meaningful** your results can be.
- Aware of **biases** (both in your data and in your modelling) as much as you can. Doing so will ensure you are providing results that **accurately represent reality**, leading to more equitable interpretations and uses of your work.
- This is immensely important, ... for Data Science will **only continue to play** an increasingly powerful and **influential role** in our society and world at large.

Data Processing

-

Workflow

-

Parsing Data

Where do data come from?

- **Internal sources:** already collected by or is part of the overall data collection of your organization.
For example: business-centric data that is available in the organization data base to record day to day operations; scientific or experimental data.
- **Existing External Sources:** available in ready to read format from an outside source for free or for a fee.
For example: public government databases, stock market data, Yelp reviews, [your favorite sport]-reference.
- **External Sources Requiring Collection Efforts:** available from external source but acquisition requires special processing.
For example: data appearing only in print form, or data on websites.

Ways to gather online data

- How to get data generated, published or hosted online:
- **API (Application Programming Interface):** using a prebuilt set of functions developed by a company to access their services. Often pay to use. For example: Google Map API, Facebook API, Twitter API
- **RSS (Rich Site Summary):** summarizes frequently updated online content in standard format. Free to read if the site has one. For example: news-related sites, blogs
- **Web scraping:** using software, scripts or by-hand extracting data from what is displayed on a page or what is contained in the HTML file (often in tables).

Web scraping

- **Why do it?** Older government or smaller news sites might not have APIs for accessing data, or publish RSS feeds or have databases for download. Or, you don't want to pay to use the API or the database.
- You just want to explore: Are you violating their terms of service? Privacy concerns for website and their clients?
- You want to publish your analysis or product: Do they have an API or fee that you are bypassing? Are they willing to share this data? Are you violating their terms of service? Are there privacy concerns?

Types of data

- What kind of values are in your data (data types)?
-
- Simple or atomic:
- **Numeric:** integers, floats
- **Boolean:** binary or true false values
- **Strings:** sequence of symbols

Types of data - 2

- What kind of values are in your data (data types)? Compound, composed of a bunch of atomic types:
- **Date and time:** compound value with a specific structure
- **Lists:** a list is a sequence of values
- **Dictionaries:** A dictionary is a collection of key-value pairs, a pair of values $x : y$ where x is usually a string called the key representing the “name” of the entry, and y is a value of any type.
- Example: Student record: what are x and y ?
- First: Kevin
- Last: Rader
- Classes: [CS-109A, STAT139]

Data storage

- How is your data represented and stored (data format)?
- **Tabular Data:** a dataset that is a two-dimensional table, where each row typically represents a single data record, and each column represents one type of measurement (csv, dat, xlsx, etc.).
- **Structured Data:** each data record is presented in a form of a [possibly complex and multi-tiered] dictionary (json, xml, etc.)
- **Semistructured Data:** not all records are represented by the same set of keys or some data records are not represented using the key-value pair structure.

Tabular Data

- In tabular data, we expect each record or observation to represent a set of measurements of a single object or event.

First Look At The Data

```
In [27]: hubway_data = pd.read_csv('hubway_trips.csv', low_memory=False)
hubway_data.head()
```

Out[27]:

	seq_id	hubway_id	status	duration	start_date	strt_statn	end_date	end_statn	bike_nr	subsc_type	zip_code	birth_da
0	1	8	Closed	9	7/28/2011 10:12:00	23.0	7/28/2011 10:12:00	23.0	B00468	Registered	'97217	1976.0
1	2	9	Closed	220	7/28/2011 10:21:00	23.0	7/28/2011 10:25:00	23.0	B00554	Registered	'02215	1966.0
2	3	10	Closed	56	7/28/2011 10:33:00	23.0	7/28/2011 10:34:00	23.0	B00456	Registered	'02108	1943.0
3	4	11	Closed	64	7/28/2011 10:35:00	23.0	7/28/2011 10:36:00	23.0	B00554	Registered	'02116	1981.0
4	5	12	Closed	12	7/28/2011 10:37:00	23.0	7/28/2011 10:37:00	23.0	B00554	Registered	'97214	1983.0

Tabular Data

- Each type of measurement is called a **variable** or an **attribute** of the data (e.g. `seq_id`, `status` and `duration` are variables or attributes). The number of attributes is called the **dimension**. These are often called **features**.
- We expect each table to contain a set of **records** or **observations** of the same kind of object or event (e.g. our table above contains observations of rides/checkouts).

```
In [27]: hubway_data = pd.read_csv('hubway_trips.csv', low_memory=False)
hubway_data.head()
```

Out[27]:

	seq_id	hubway_id	status	duration	start_date	strt_statn	end_date	end_statn	bike_nr	subsc_type	zip_code	birth_da
0	1	8	Closed	9	7/28/2011 10:12:00	23.0	7/28/2011 10:12:00	23.0	B00468	Registered	'97217	1976.0
1	2	9	Closed	220	7/28/2011 10:21:00	23.0	7/28/2011 10:25:00	23.0	B00554	Registered	'02215	1966.0
2	3	10	Closed	56	7/28/2011	23.0	7/28/2011	23.0	B00456	Registered	'02108	1943.0

Data Types

- We'll see later that it's important to distinguish between classes of variables or attributes based on the type of values they can take on.
- **Quantitative variable:** is numerical and can be either:
- **discrete** - a finite number of values are possible in any bounded interval. For example: "Number of siblings" is a discrete variable
- **continuous** - an infinite number of values are possible in any bounded interval. For example: "Height" is a continuous variable
- **Categorical variable:** no inherent order among the values For example: "What kind of pet you have" is a categorical variable

Data Processing

-

Workflow

-

Exploration Data Analysis: Common Issues

Common Issues

- Common issues with data:
 - **Missing** values: how do we fill in?
 - **Wrong** values: how can we detect and correct?
 - **Messy** format
 - **Not usable**: the data cannot answer the question posed

Messy Data

The following is a table accounting for the number of produce deliveries over a weekend.

What are the variables in this dataset?

What object or event are we measuring?

	Friday	Saturday	Sunday
Morning	15	158	10
Afternoon	2	90	20
Evening	55	12	45

What's the issue?

How do we fix it?

Messy Data

We're measuring individual deliveries; the variables are Time, Day, Number of Produce.

	Friday	Saturday	Sunday
Morning	15	158	10
Afternoon	2	90	20
Evening	55	12	45

Problem: each column header represents a single value rather than a variable.

Row headers are “hiding” the Day variable. The values of the variable, “Number of Produce”, is not recorded in a single column.

Fixing Messy Data

We need to reorganize the information to make explicit the event we're observing and the variables associated to this event.

ID	Time	Day	Number
1	Morning	Friday	15
2	Morning	Saturday	158
3	Morning	Sunday	10
4	Afternoon	Friday	2
5	Afternoon	Saturday	9
6	Afternoon	Sunday	20
7	Evening	Friday	55
8	Evening	Saturday	12
9	Evening	Sunday	45

Common Causes of Messiness

- Column headers are values, not variable names
- Variables are stored in both rows and columns
- Multiple variables are stored in one column/entry
- Multiple types of experimental units stored in same table
- In general, we want each file to correspond to a dataset, each column to represent a single variable and each row to represent a single observation.
- We want to **tabularize** the data. This makes Python happy.

Data Processing

-

Workflow

-

Data Exploration: Descriptive Statistics

Basics of Sampling

- A **population** is the entire set of objects or events under study. Population can be hypothetical “all students” or all students in this class.
- A **sample** is a “representative” subset of the objects or events under study. Needed because it’s impossible or intractable to obtain or compute with population data.
 - Biases in samples:
- **Selection bias**: some subjects or records are more likely to be selected
- Volunteer/**nonresponse bias**: subjects or records who are not easily available are not represented

Sample - Mean

The **mean** of a set of n observations of a variable is denoted \bar{x} and is defined as:

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i$$

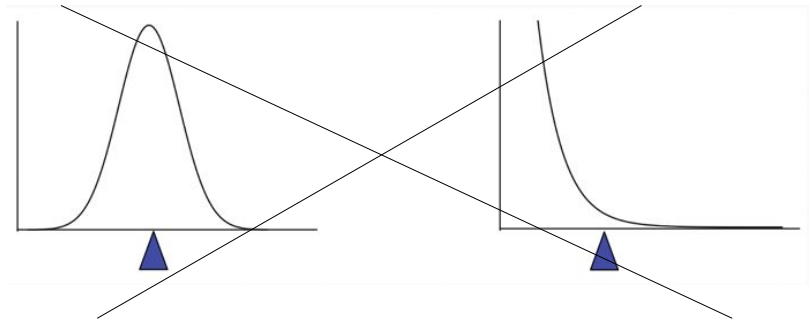


The **mean** describes what a “typical” sample value looks like, or where is the “center” of the distribution of the data. Note: there is always uncertainty involved when calculating a sample mean to estimate a population mean.

Sample - Median

The **median** of a set of n number of observations in a sample, ordered by value, of a variable is defined by

$$\text{Median} = \begin{cases} X_{(n+1)/2} & \text{if } n \text{ is odd} \\ \frac{X_{n/2} + X_{(n+1)/2}}{2} & \text{if } n \text{ is even} \end{cases}$$

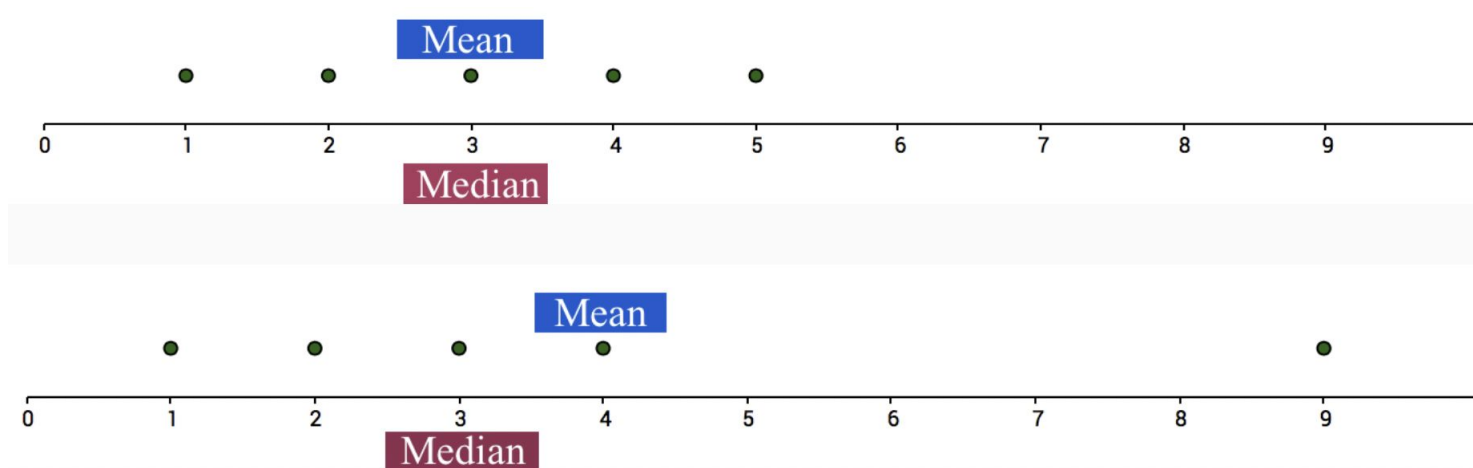


Example (already in order):
Ages: 17, 19, 21, 22, 23, 23, 23, 38
Median = $(22+23)/2 = 22.5$

The median also describes what a typical observation looks like, or where is the center of the distribution of the sample of observations.

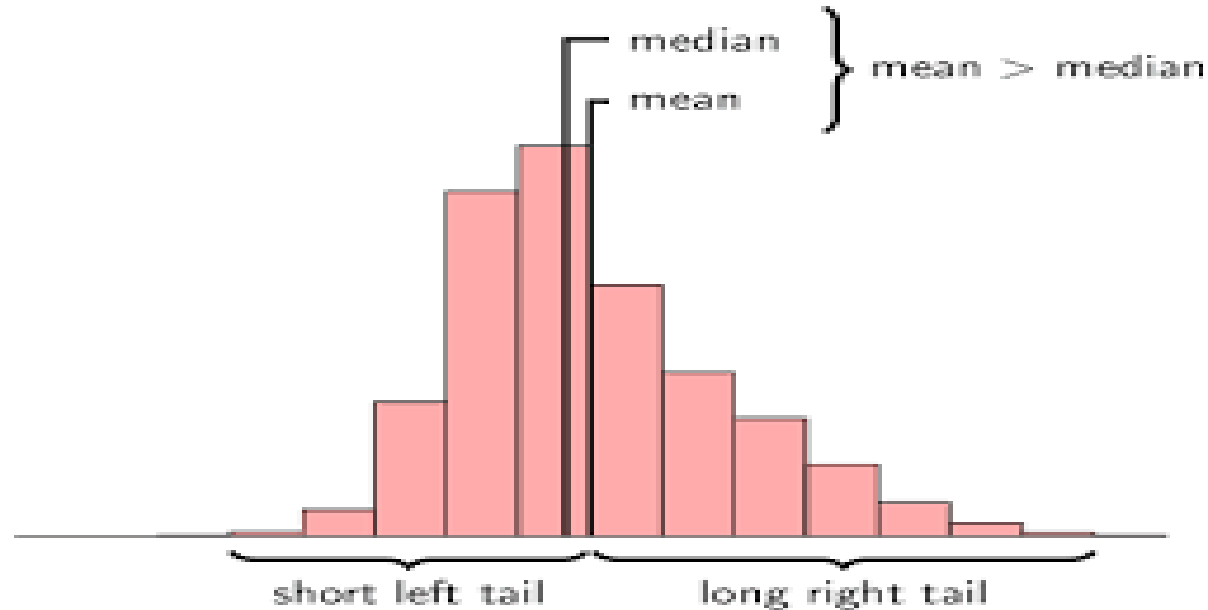
Sample - Mean vs. Median

The mean is sensitive to extreme values (**outliers**)



Mean, median, and skewness

The mean is sensitive to outliers:



The above distribution is called **right-skewed** since the mean is greater than the median. Note: **skewness** often

Computational time

How hard (in terms of algorithmic complexity) is it to calculate

the **mean**?

at most $O(n)$

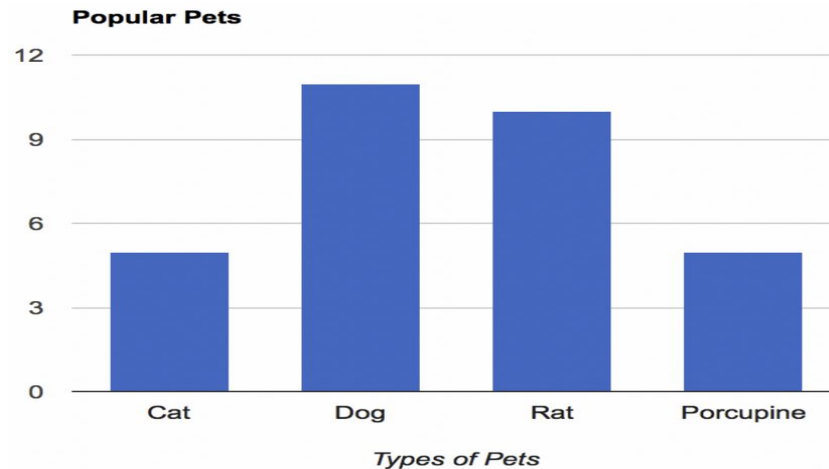
the **median**?

at most $O(n)$ or $O(n \log n)$

Note: Practicality of implementation should be considered!

Categorical Variables

For categorical variables, neither mean or median make sense. Why?



The mode might be a better way to find the most “representative” value.

Measures of Spread: Range

The spread of a sample of observations measures how well the mean or median describes the sample.

One way to measure spread of a sample of observations is via the **range**.

Range = Maximum Value - Minimum Value

Measures of Spread: Variance

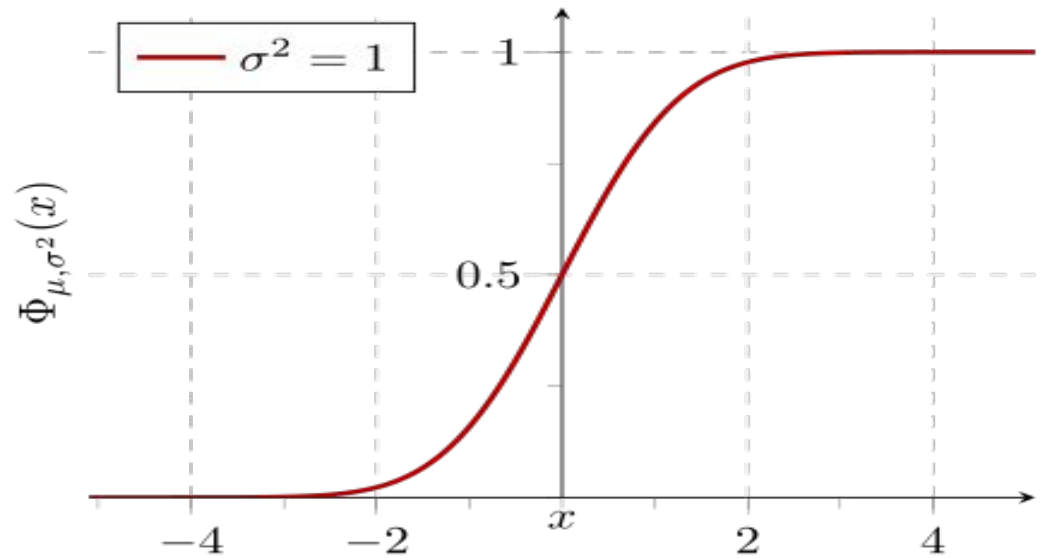
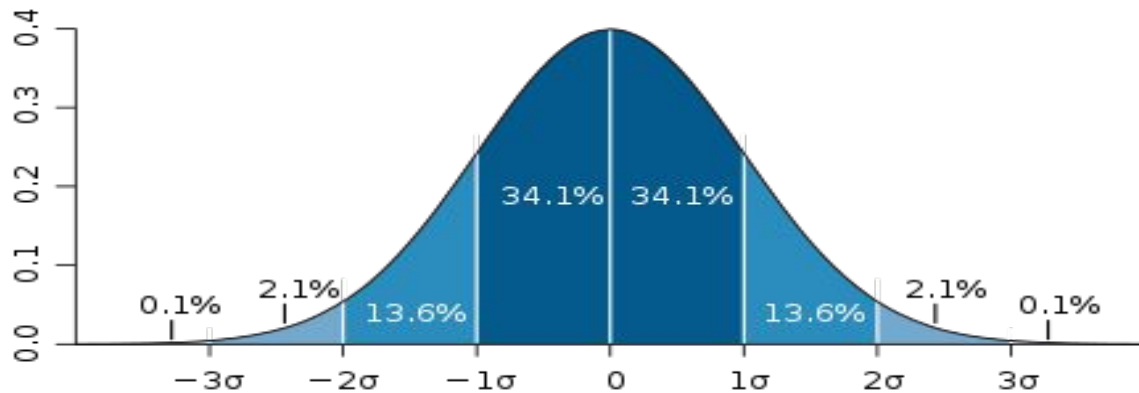
The (sample) **variance**, denoted σ^2 , measures how much on average the sample values deviate from the mean:

$$\sigma^2 \equiv \mathbf{D}(X) = \mathbf{E}[(X - \mu)^2] = \sum_x (x - \mu)^2 p(x)$$

Note: the difference measures the amount by which each x deviates from the mean.

Measures of Spread: Standard Deviation

The (sample) **standard deviation** is the square root of the variance



Data Processing

-

Workflow

-

Data Visualization (for EDA)

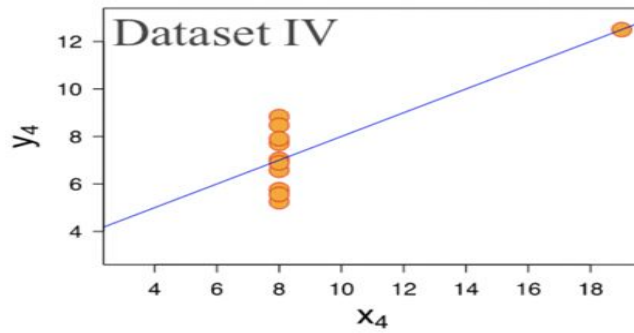
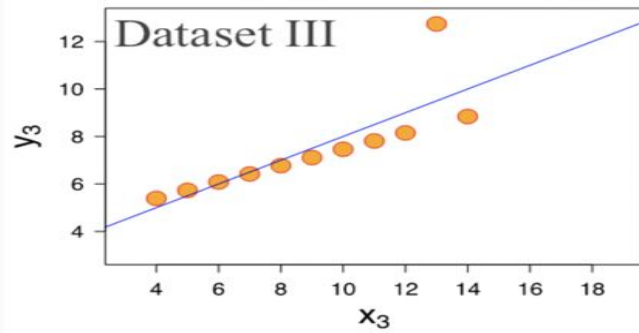
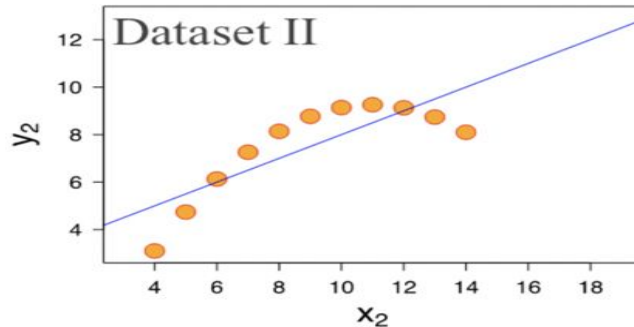
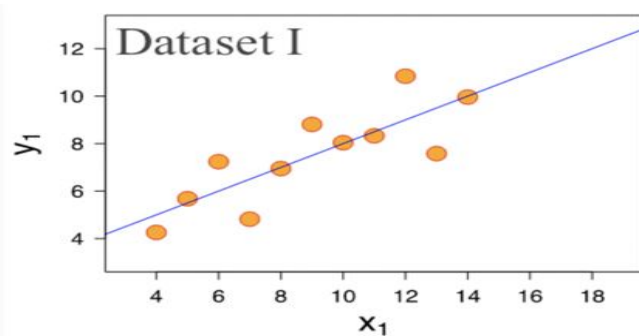
Anscombe's Quartet

- Anscombe's quartet comprises four data sets that have nearly identical simple descriptive statistics, yet have very different distributions and appear very different when graphed.

	Dataset I		Dataset II		Dataset III		Dataset IV	
	x	y	x	y	x	y	x	y
	10	8.04	10	9.14	10	7.46	8	6.58
	8	6.95	8	8.14	8	6.77	8	5.76
	13	7.58	13	8.74	13	12.74	8	7.71
	9	8.81	9	8.77	9	7.11	8	8.84
	11	8.33	11	9.26	11	7.81	8	8.47
	14	9.96	14	8.1	14	8.84	8	7.04
	6	7.24	6	6.13	6	6.08	8	5.25
	4	4.26	4	3.1	4	5.39	19	12.5
	12	10.84	12	9.13	12	8.15	8	5.56
	7	4.82	7	7.26	7	6.42	8	7.91
	5	5.68	5	4.74	5	5.73	8	6.89
Sum:	99.00	82.51	99.00	82.51	99.00	82.51	99.00	82.51
Avg:	9.00	7.50	9.00	7.50	9.00	7.50	9.00	7.50
Std:	3.32	2.03	3.32	2.03	3.32	2.03	3.32	2.03

Anscombe's Quartet

- They were constructed in 1973 by the statistician Francis Anscombe to demonstrate both the importance of graphing data before analyzing it and the effect of outliers and other influential observations on statistical properties.

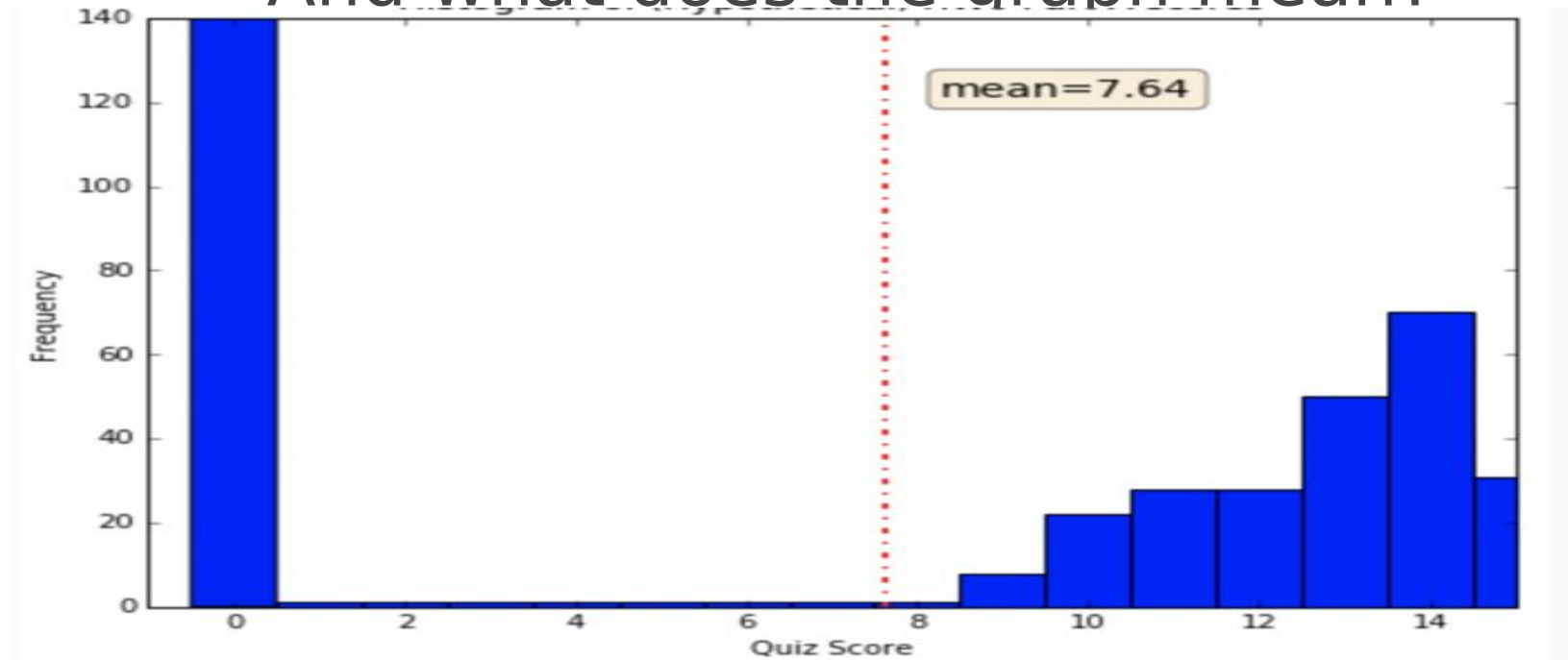


More Visualization Motivation

The average score for school class: 7.64

What does that mean?

And what does the graph mean?



More Visualization Motivation

Visualizations help us to analyze and explore the data:

- Identify hidden patterns and trends
 - Formulate/test hypotheses
- Communicate any modeling results
- Present information and ideas succinctly
 - Provide evidence and support
 - Influence and persuade
- Determine the next step in analysis/modeling
 -

Other Types of Visualizations

What do you want your visualization to show about your data?

Distribution: how a variable or variables in the dataset distribute over a range of possible values.

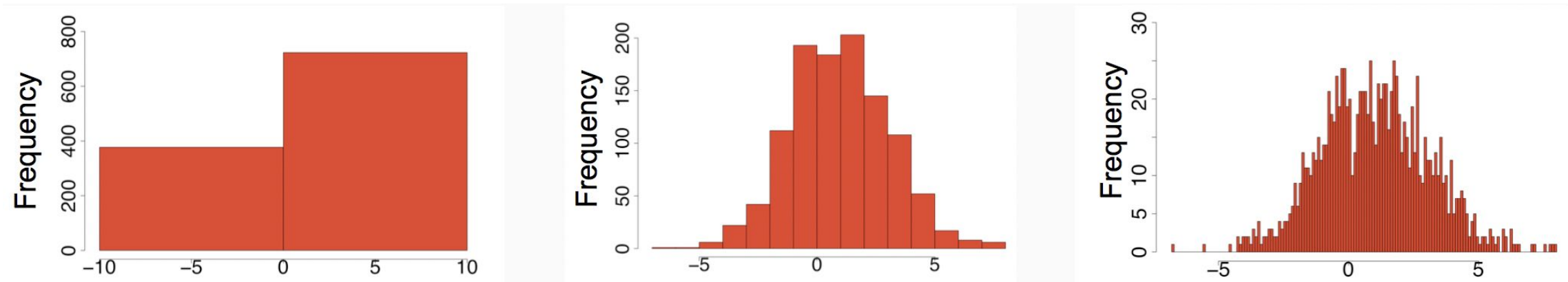
Relationship: how the values of multiple variables in the dataset relate

Composition: how the dataset breaks down into subgroups

Comparison: how trends in multiple variable or datasets compare

Histogram

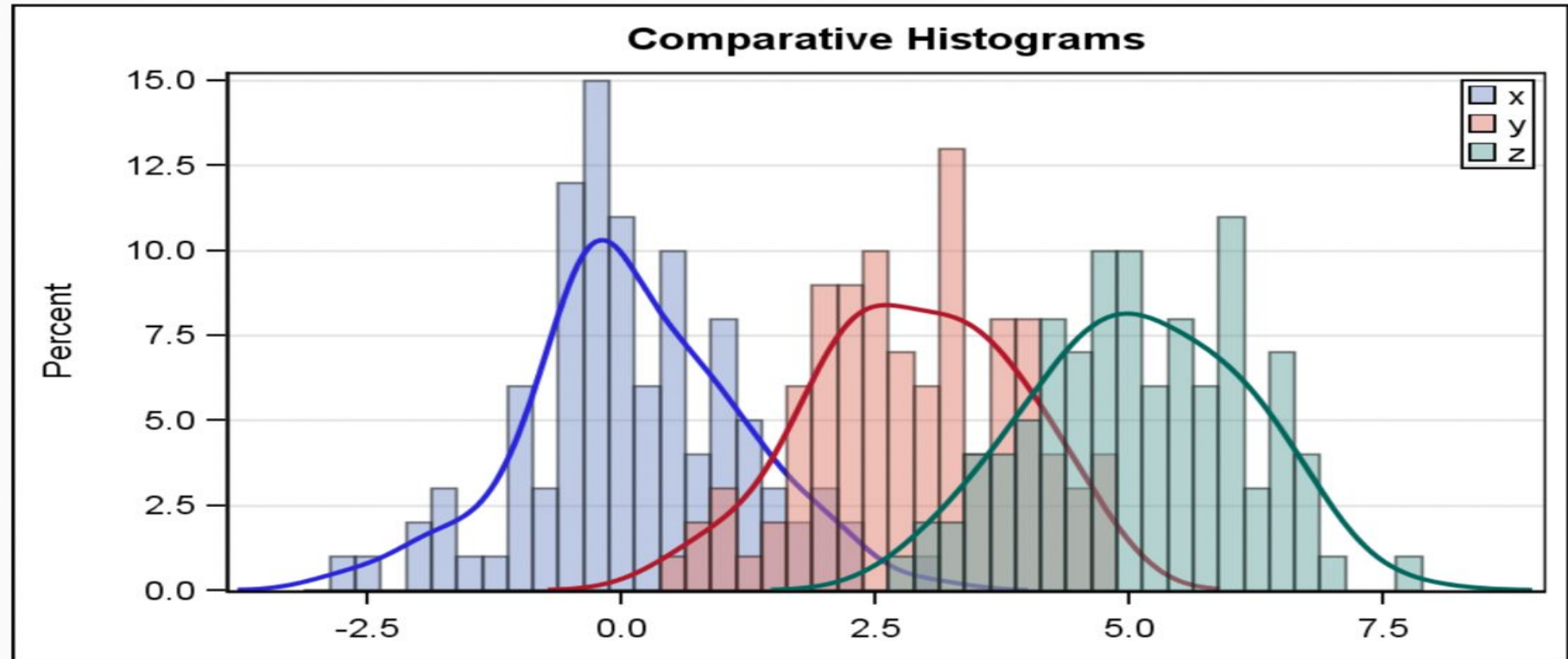
A **histogram** is a way to visualize how 1-dimensional data is distributed across certain values.



Note: Trends in histograms are sensitive to number of bins.

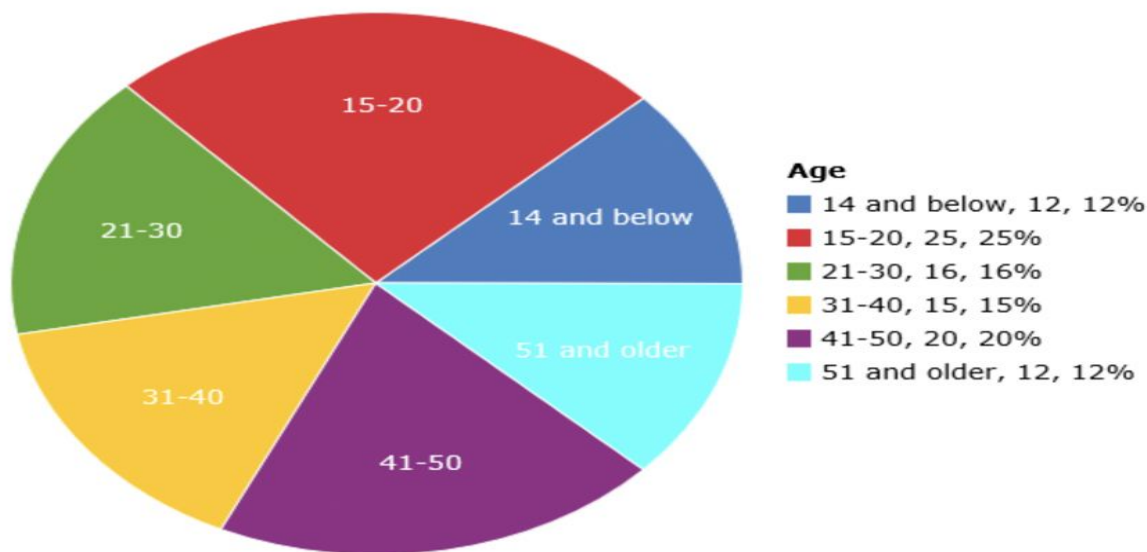
Multiple Histogram

Plotting **multiple histograms** (and **kernel density estimates** of the distribution, here) on the same axes is a way to visualize how different variables compare (or how a variable differs over specific groups).



Pie Chart

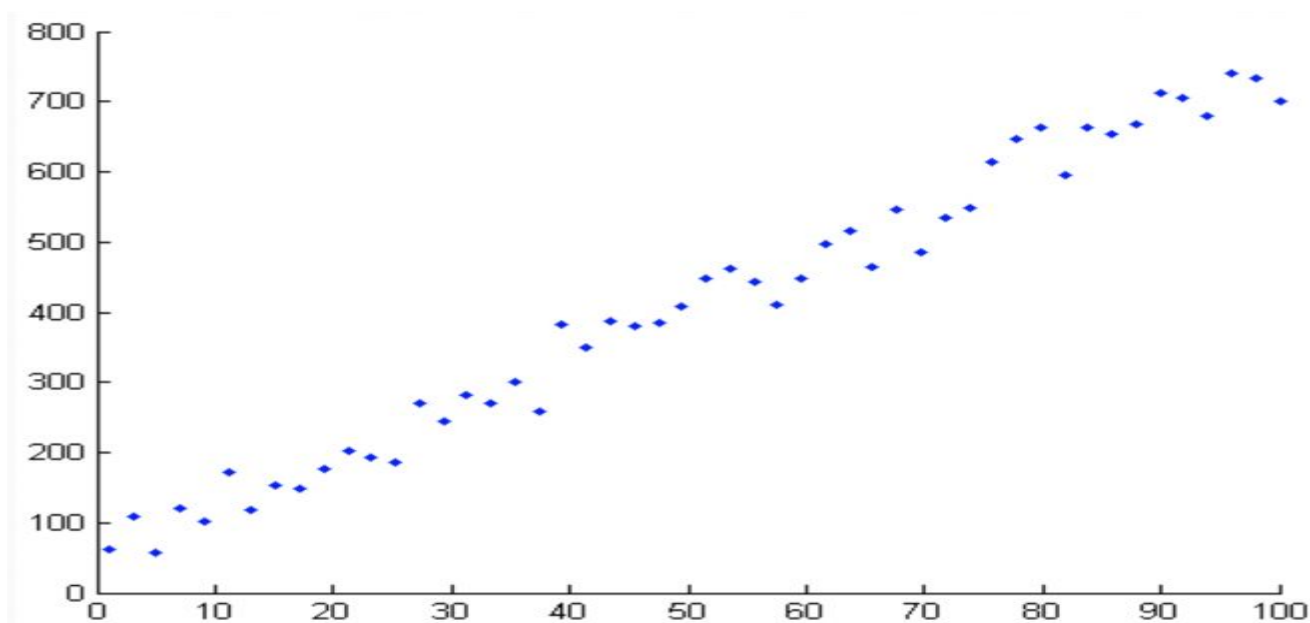
A **pie chart** is a way to visualize the static composition (aka, distribution) of a variable (or single group).



Pie charts are often frowned upon (and bar charts are used instead). Why?

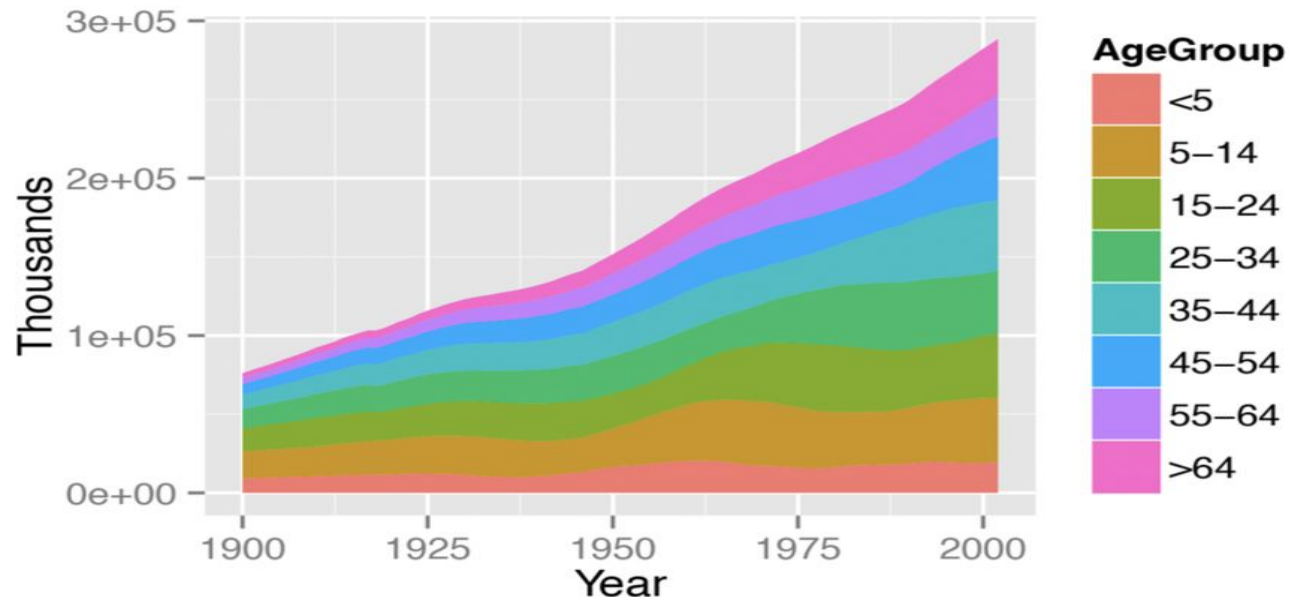
Scatter Plot

A **scatter plot** is a way to visualize the relationship between two different attributes of multi-dimensional data.



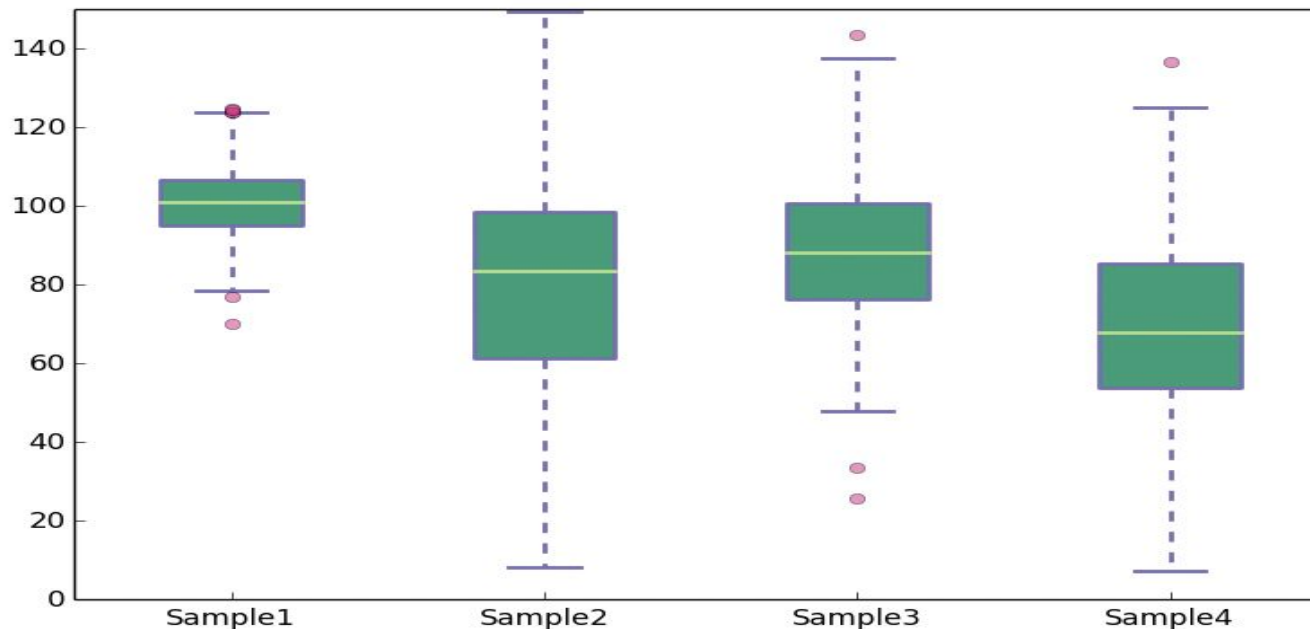
Stacked Area Plot

A **stacked area graph** is a way to visualize the composition of a group as it changes over time (or some other quantitative variable). This shows the relationship of a categorical variable (AgeGroup) to a quantitative variable (year).



Boxplot

A **boxplot** is a simplified visualization to compare a quantitative variable across groups. It highlights the range, quartiles, median and any outliers present in a data set.



Some Complex Cases

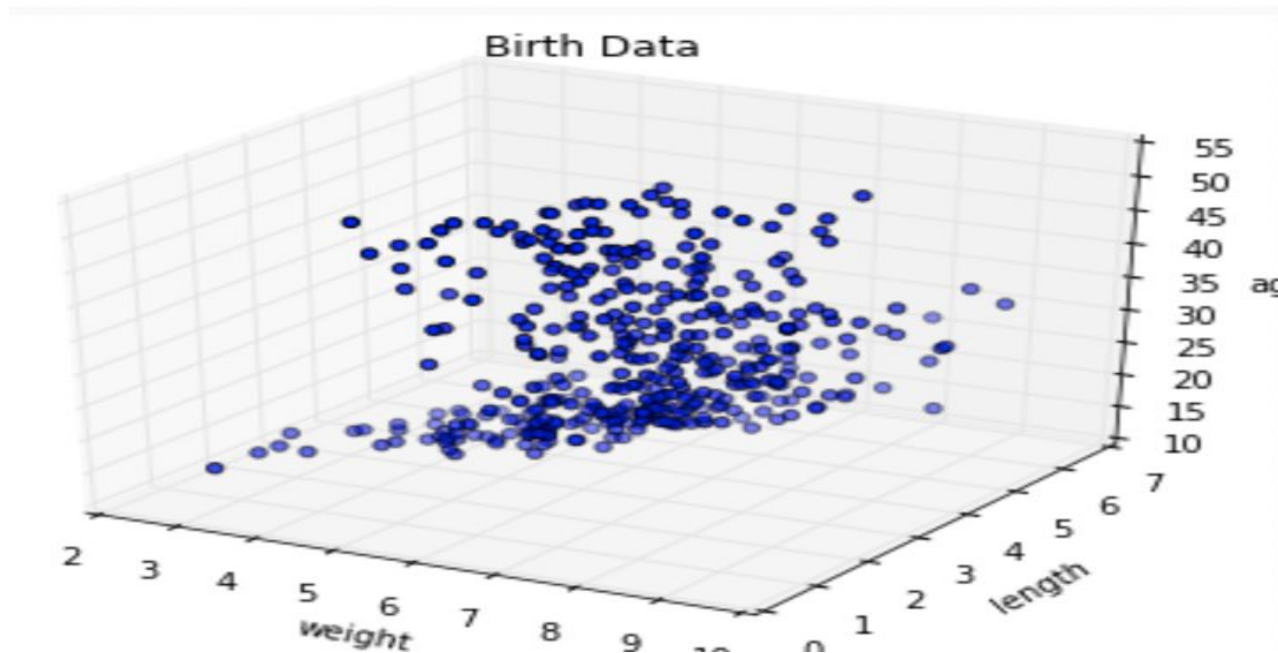
Often your dataset seem too complex to visualize:

Data is too **high dimensional** (how do you plot 100 variables on the same set of axes?)

Some variables are **categorical** (how do you plot values like Cat or No?)

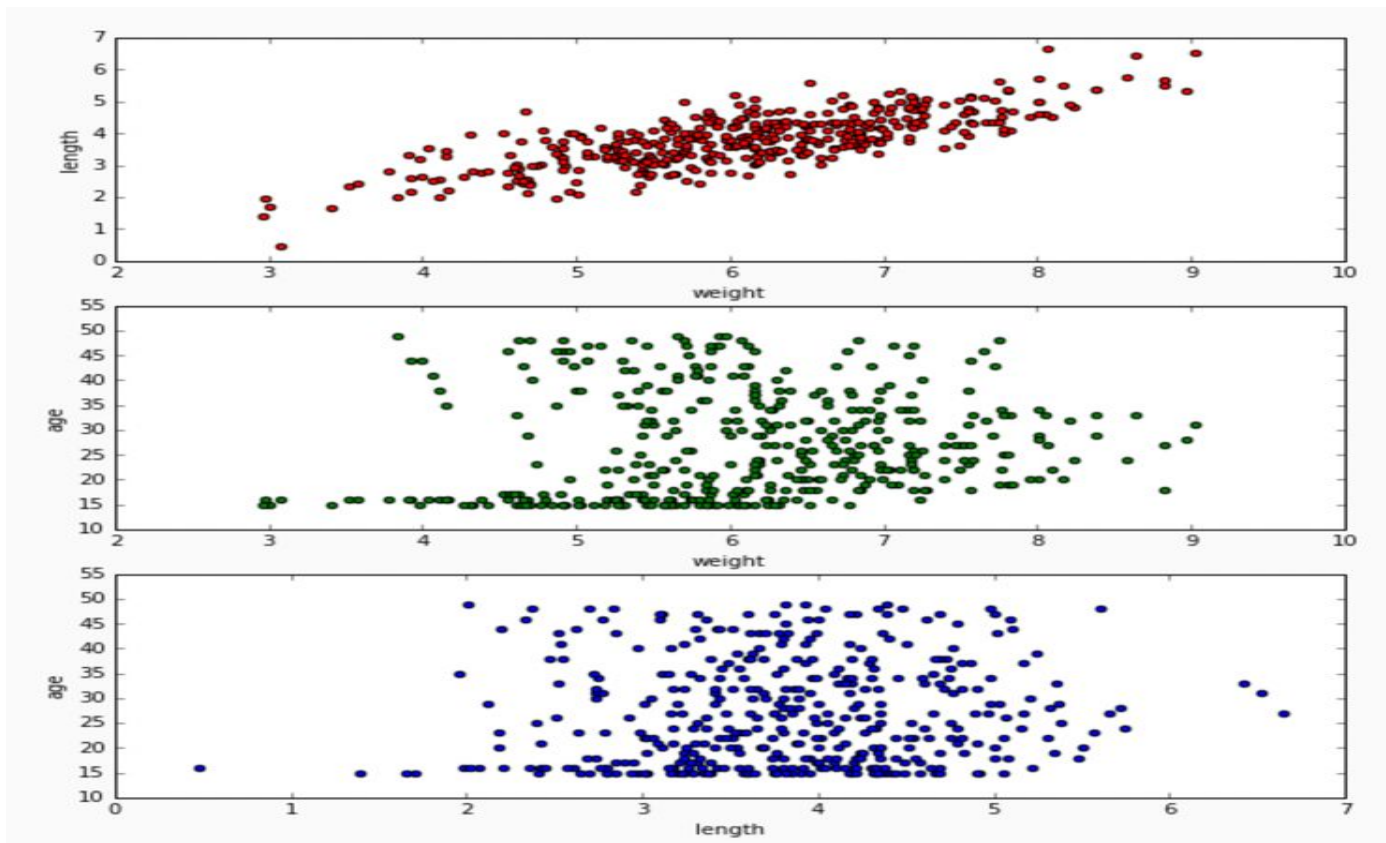
Many Dimensions

When the data is high dimensional, a scatter plot of all data attributes can be impossible or unhelpful



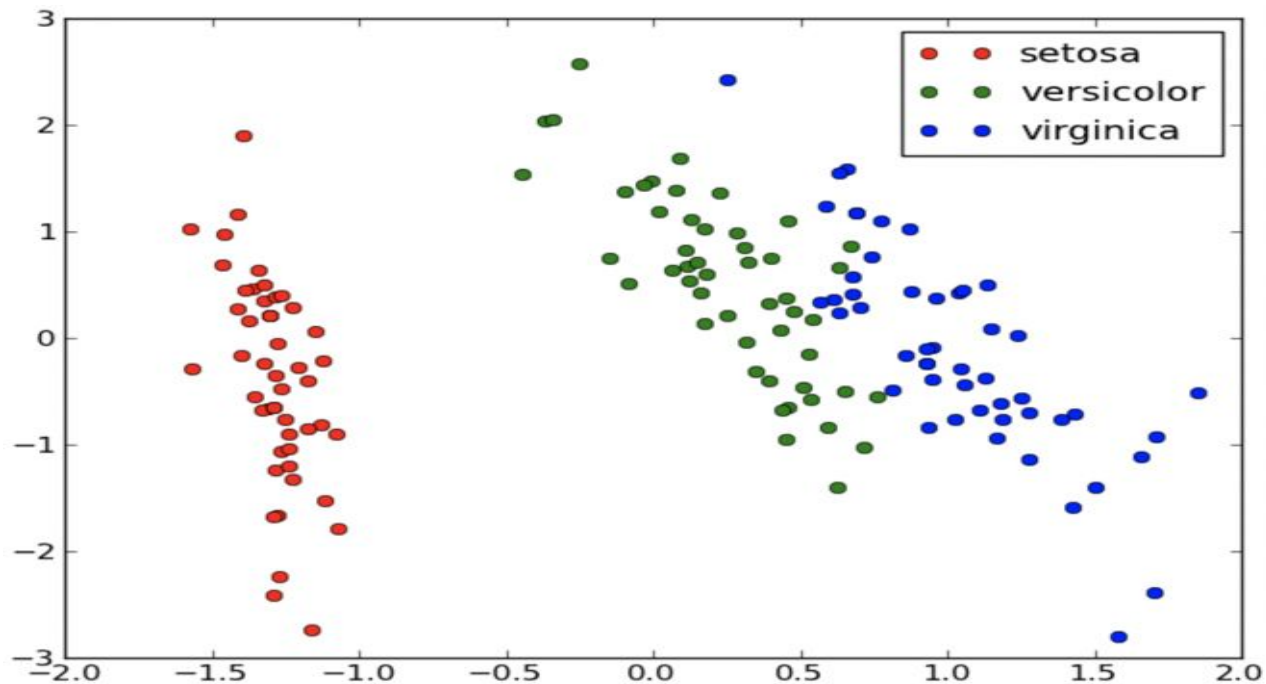
Reducing complexity

Relationships may be easier to spot by producing **multiple** plots of **lower** dimensionality.



Reducing complexity

For 3D data, color coding a categorical attribute can be “effective”

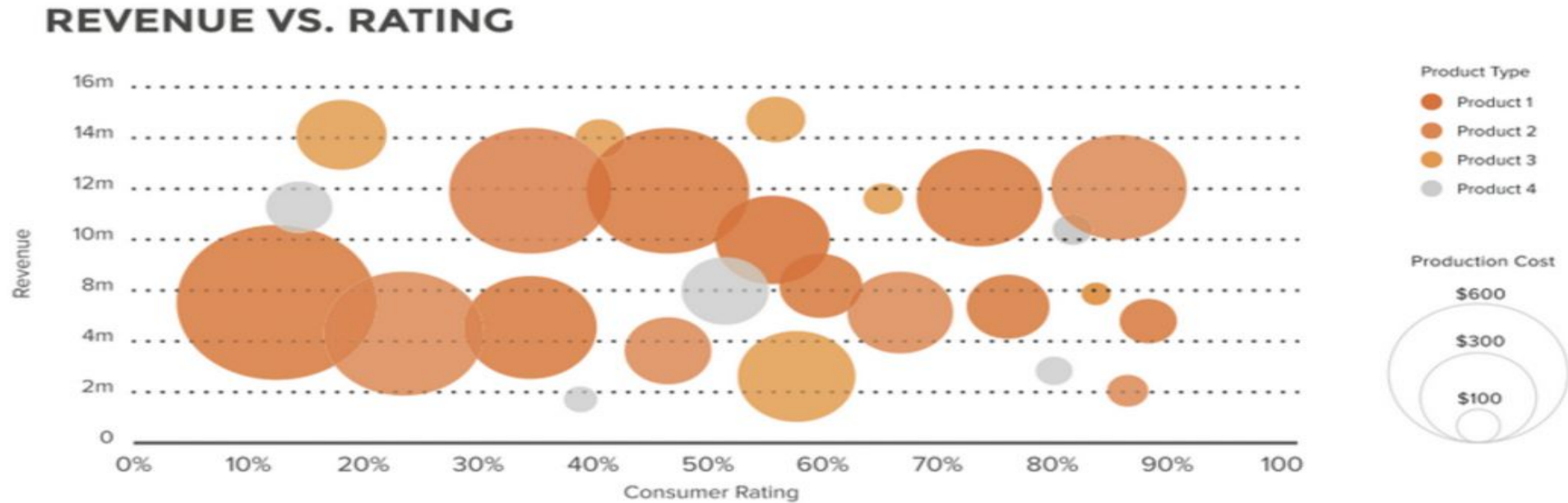


This visualizes a set of Iris measurements.
The variables are:
petal length,
sepal length,
Iris type (setosa, versicolor,
virginica).

Except when it's not effective.

Reducing complexity — 3D Bubble Plot

For 3D data, a quantitative attribute can be encoded by size in a bubble chart.



The above visualizes a set of consumer products. The variables are: revenue, consumer rating, product type and product cost.

Data Processing

-

Workflow

-

EDA Example

Recall -> Data Processing -> **EDA**

Ask an interesting question

Get the Data — **EDA!**

Explore the Data

Model the Data

Visualize the Results

EDA for Hubway Data

Introduction: Hubway is metro-Boston's public bike share program, with more than 1600 bikes at 160+ stations across the Greater Boston area. Hubway is owned by four municipalities in the area.

By 2016, Hubway operated 185 stations and 1750 bicycles, with 5 million ride since launching in 2011.

The Data: In April 2017, Hubway held a Data Visualization Challenge at the Microsoft NERD Center in Cambridge, releasing 5 years of trip data.

The Question: What does the data tell us about the ride share

Customer Question -> Data Science Question

Original customer question:

‘What does the data tell us about the ride share program?’
is not good for scientific investigation. Before we can improve the question, we should look at the data - **EDA!**

seq_id	hubway_id	status	duration	start_date	strt_statn	end_date	end_statn	bike_nr	subsc_type	zip_code	birth_date	gender	
0	1	8	Closed	9	7/28/2011 10:12:00	23.0	7/28/2011 10:12:00	23.0	B00468	Registered	'97217	1976.0	Male
1	2	9	Closed	220	7/28/2011 10:21:00	23.0	7/28/2011 10:25:00	23.0	B00554	Registered	'02215	1966.0	Male
2	3	10	Closed	56	7/28/2011 10:33:00	23.0	7/28/2011 10:34:00	23.0	B00456	Registered	'02108	1943.0	Male
3	4	11	Closed	64	7/28/2011 10:35:00	23.0	7/28/2011 10:36:00	23.0	B00554	Registered	'02116	1981.0	Female
4	5	12	Closed	12	7/28/2011 10:37:00	23.0	7/28/2011 10:37:00	23.0	B00554	Registered	'97214	1983.0	Female

Based on the data, what kind of **concrete questions** can we ask?

The Data Exploration/Question Refinement Cycle - 1

Who? Who's using the bikes?

Refine into specific hypotheses:

- More men or more women?
- Older or younger people?
- Subscribers or one time users?

The Data Exploration/Question Refinement Cycle - 2

Where? Where are bikes being checked out?

Refine into specific hypotheses:

- More in Boston than Cambridge?
- More in commercial or residential?
- More around tourist attractions?

Sometimes the data is given to you in pieces and must be merged!

The Data Exploration/Question Refinement Cycle - 3

When? When are the bikes being checked out?

Refine into specific hypotheses:

- More during the weekend than on the weekdays?
- More during rush hour?
- More during the summer than the fall?

Sometimes the feature you want to explore doesn't exist in the data, and must be engineered!

The Data Exploration/Question Refinement Cycle - 4

Why? For what reasons/activities are people checking out bikes?

Refine into specific hypotheses:

- More bikes are used for recreation than commute?
- More bikes are used for touristic purposes?
- Bikes are use to bypass traffic?

Do we have the data to answer these questions with reasonable certainty?

What data do we need to collect in order to answer these

The Data Exploration/Question Refinement Cycle - 5

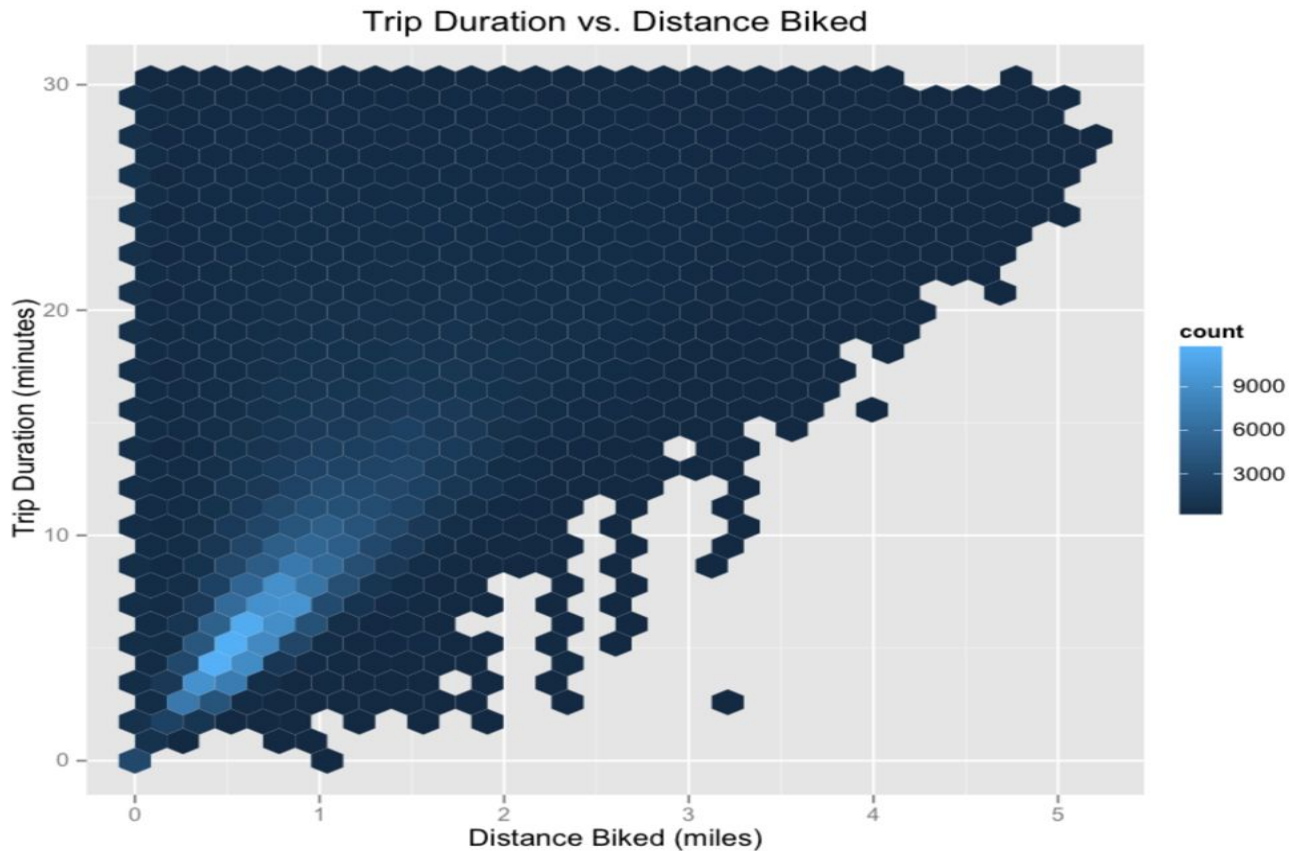
How? Questions that combine variables.

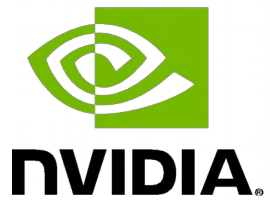
- How does user demographics impact the duration the bikes are being used? Or where they are being checked out?
- How does weather or traffic conditions impact bike usage?
- How do the characteristics of the station location affect the number of bikes being checked out?

How questions are about modeling relationships between different variables.

Reducing complexity

So how well did we do in formulating creative hypotheses and manipulating the data for answers?





DEEP
LEARNING
INSTITUTE

www.nvidia.com/dli



Lecture 1: Demo - Lab Work

Lecture 1: Demo - Lab Work

Download the data from

<https://cloud.comsys.kpi.ua/s/7oW5GRWHpkKmAmC>

unzip and put it in COLAB_DS directory (before create this directory!) at your Google Drive.

```
# Mount your Google drive for data input-output
from google.colab import drive
drive.mount('/content/drive')
```

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

```
# Check the paths
! pwd
! ls /content/drive
! ls /content/drive/MyDrive
```

```
# Check directory for availability of your HUBWAY-dataset
! mkdir /content/drive/MyDrive/COLAB_DS
```

```
! ls /content/drive/MyDrive/COLAB_DS
```

```
hubway_data  hubway_network_analysis.ipynb  Lab01_EDA_hubway_datasets.ipynb
```

```
import sys
import datetime
import numpy as np
import scipy as sp
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from math import radians, cos, sin, asin, sqrt
from sklearn.linear_model import LinearRegression

sns.set(style="ticks")
%matplotlib inline
```

```
import os
```

```
DATA_HOME = '/content/drive/MyDrive/COLAB_DS/hubway_data'
```



```
HUBWAY_STATIONS_FILE = os.path.join(DATA_HOME, 'hubway_stations.csv')
HUBWAY_TRIPS_FILE = os.path.join(DATA_HOME, 'hubway_trips.csv')
```

```
hubway_data = pd.read_csv(HUBWAY_TRIPS_FILE, index_col=0, low_memory=False)
hubway_data.head()
```

```
/usr/local/lib/python3.6/dist-packages/numpy/lib/arraysetops.py:580
mask |= (ar1 == a)
```

seq_id	hubway_id	status	duration	start_date	strt_statn	end_dat
1	8	Closed	9	7/28/2011 10:12:00	23.0	7/28/201 10:12:0
2	9	Closed	220	7/28/2011 10:21:00	23.0	7/28/201 10:25:0
3	10	Closed	56	7/28/2011 10:22:00	23.0	7/28/201 10:24:0

Who? Who's using the bikes?

Refine into specific hypotheses:

- More men or more women?
- Older or younger people?
- Subscribers or one time users?

```
# Let's do some cleaning first by removing empty cells or replacing them with NaN.
# Pandas can do this.
# we will learn a lot about pandas
hubway_data['gender'] = hubway_data['gender'].replace(np.nan, 'NaN', regex=True).va
```

```
# we drop
hubway_data['birth_date'].dropna()
age_col = 2020.0 - hubway_data['birth_date'].values
```

```
# matplotlib can create a plot with two sub-plots.
# we will learn a lot about matplotlib
fig, ax = plt.subplots(1, 2, figsize=(15, 6))
```

```
# find all the unique value of the column gender
# numpy can do this
# we will learn a lot about numpy
gender_counts = np.unique(hubway_data['gender'].values, return_counts=True)
```



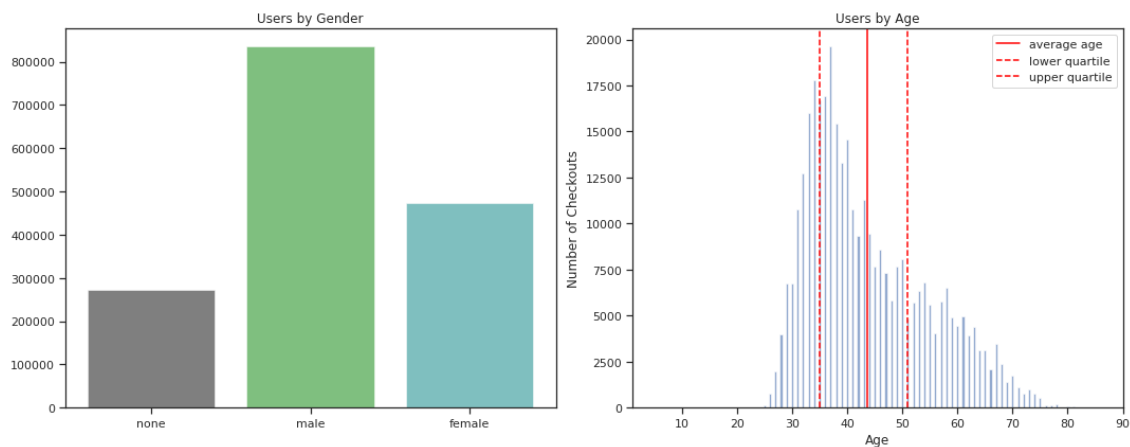
```

ax[0].bar(range(3), gender_counts[1], align='center', color=['black', 'green', 'teal'])
ax[0].set_xticks([0, 1, 2])
ax[0].set_xticklabels(['none', 'male', 'female'])
ax[0].set_title('Users by Gender')

age_col = 2020.0 - hubway_data['birth_date'].dropna().values
age_counts = np.unique(age_col, return_counts=True)
ax[1].bar(age_counts[0], age_counts[1], align='center', width=0.4, alpha=0.6)
ax[1].axvline(x=np.mean(age_col), color='red', label='average age')
ax[1].axvline(x=np.percentile(age_col, 25), color='red', linestyle='--', label='lower quartile')
ax[1].axvline(x=np.percentile(age_col, 75), color='red', linestyle='--', label='upper quartile')
ax[1].set_xlim([1, 90])
ax[1].set_xlabel('Age')
ax[1].set_ylabel('Number of Checkouts')
ax[1].legend()
ax[1].set_title('Users by Age')

plt.tight_layout()
plt.savefig('who.png', dpi=300)

```



▼ Challenge

There is actually a mistake in the code above. Can you find it?

Soon you will be skillful enough to answers many "who" questions

▼ Where? Where are bikes being checked out?

Refine into specific hypotheses:

1. More in Boston than Cambridge?
2. More in commercial or residential?
3. More around tourist attractions?

```
# using pandas again to read the station locations
station_data = pd.read_csv(HUBWAY_STATIONS_FILE, low_memory=False)[['id', 'lat', 'lng']]
station_data.head()
```

	id	lat	lng
0	3	42.340021	-71.100812
1	4	42.345392	-71.069616
2	5	42.341814	-71.090179
3	6	42.361285	-71.065140
4	7	42.353412	-71.044624

```
# Sometimes the data is given to you in pieces and must be merged!
# we want to combine the trips data with the station locations. pandas to the rescue!
hubway_data_with_gps = hubway_data.join(station_data.set_index('id'), on='strt_statn')
hubway_data_with_gps.head()
```

	hubway_id	status	duration	start_date	strt_statn	end_date
seq_id						
1	8	Closed	9	7/28/2011 10:12:00	23.0	7/28/2011 10:12:00
2	9	Closed	220	7/28/2011 10:21:00	23.0	7/28/2011 10:25:00
3	10	Closed	56	7/28/2011 10:22:00	23.0	7/28/2011 10:24:00

```
len(hubway_data_with_gps)
```

1579025

```
hubway_data_with_gps.head(-3)
```

seq_id	hubway_id	status	duration	start_date	strt_statn	end_da
1	8	Closed	9	7/28/2011 10:12:00	23.0	7/28/20 10:12:00
2	9	Closed	220	7/28/2011 10:21:00	23.0	7/28/20 10:25:00
3	10	Closed	56	7/28/2011 10:33:00	23.0	7/28/20 10:34:00
4	11	Closed	64	7/28/2011 10:35:00	23.0	7/28/20 10:36:00
5	12	Closed	12	7/28/2011 10:37:00	23.0	7/28/20 10:37:00
...
1579018	1748015	Closed	900	11/30/2013 23:17:00	76.0	11/30/20 23:32:00

```
import pandas as pd
import folium
from folium.plugins import HeatMap
```

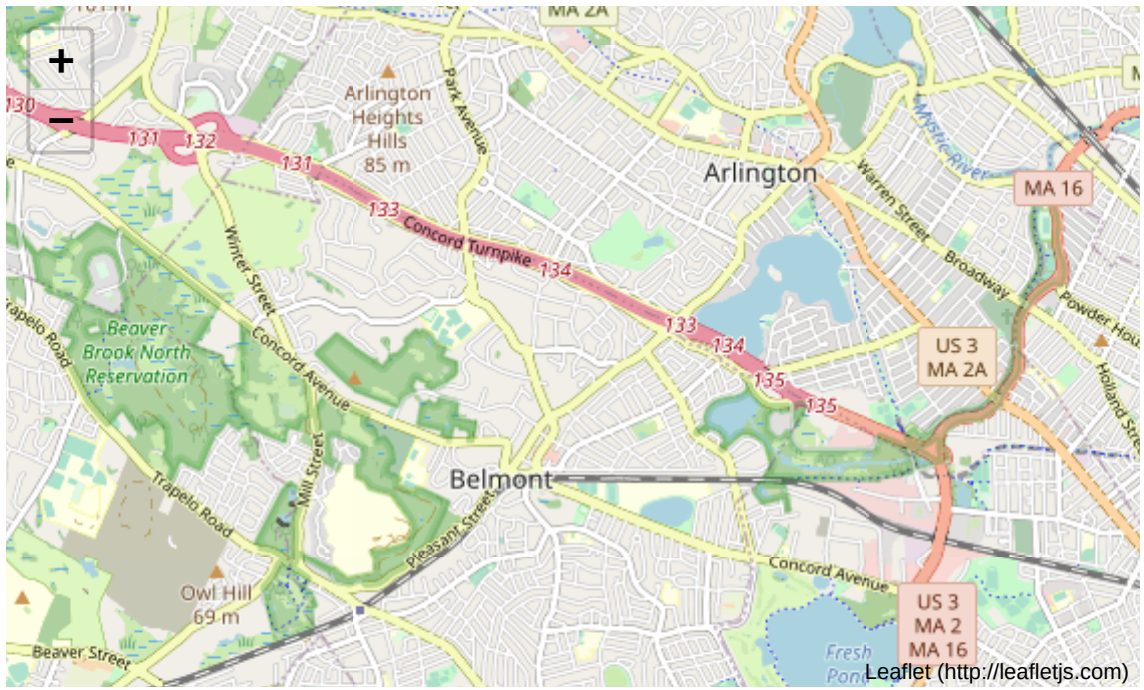
```
#for_map = pd.read_csv('campaign_contributions_for_map.tsv', sep='\t')
#for_map = hubway_data_with_gps.head(1000)
for_map = hubway_data_with_gps.head(200000)

#max_amount = float(for_map['Amount'].max())
max_amount = float(for_map['duration'].max())

hmap = folium.Map(location=[42.35, -71.05], zoom_start=13, )

hm_wide = HeatMap( list(zip(for_map.lat.values, for_map.lng.values, for_map.duration.values)),
                    min_opacity=0.2,
                    max_val=max_amount,
                    radius=27, blur=15,
                    max_zoom=1,
                    )

hmap.add_child(hm_wide)
```



You should obtain something similar to the next image ...



▼ When? When are the bikes being checked out?

Refine into specific hypotheses:

1. More during the weekend than on the weekdays?
2. More during rush hour?
3. More during the summer than the fall?

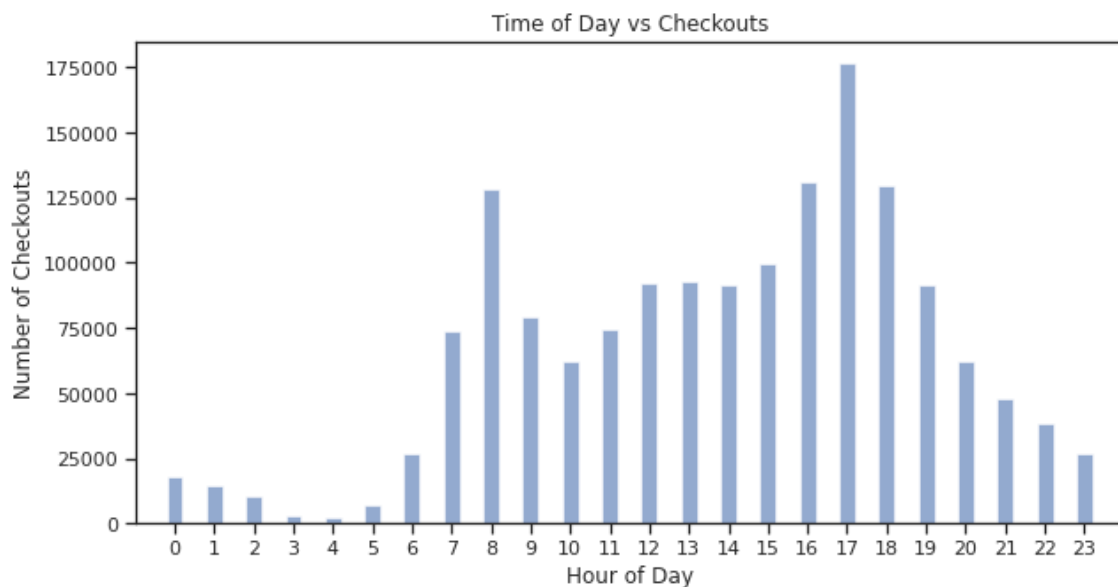
```
# Sometimes the feature you want to explore doesn't exist in the data, and must be
```

```
# to find the time of the day we will use the start_date column and extract the hour  
# we use list comprehension
```

```
# we will be doing a lot of those
check_out_hours = hihway_data['start date'].apply(lambda s: int(s[-8:-6]))
fig, ax = plt.subplots(1, 1, figsize=(10, 5))

check_out_counts = np.unique(check_out_hours, return_counts=True)
ax.bar(check_out_counts[0], check_out_counts[1], align='center', width=0.4, alpha=0.5)
ax.set_xlim([-1, 24])
ax.set_xticks(range(24))
ax.set_xlabel('Hour of Day')
ax.set_ylabel('Number of Checkouts')
ax.set_title('Time of Day vs Checkouts')

plt.show()
```



Why? For what reasons/activities are people checking out bikes?

Refine into specific hypotheses:

1. More bikes are used for recreation than commute?
2. More bikes are used for touristic purposes?
3. Bikes are use to bypass traffic?

Do we have the data to answer these questions with reasonable certainty? What data do we need to collect in order to answer these questions?

▼ How? Questions that combine variables.

1. How does user demographics impact the duration the bikes are being used? Or where they are being checked out?
2. How does weather or traffic conditions impact bike usage?
3. How do the characteristics of the station location affect the number of bikes being checked out?

How questions are about modeling relationships between different variables.

```
# Here we define the distance from a point as a python function.
# We set Boston city center long and lat to be the default value.
# you will become experts in building functions and using functions just like this

def haversine(pt, lat2=42.355589, lon2=-71.060175):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    lon1 = pt[0]
    lat1 = pt[1]

    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 3956 # Radius of earth in miles
    return c * r
```

```
# use only the checkouts that we have gps location
station_counts = np.unique(hubway_data_with_gps['strt_statn'].dropna(), return_cou
counts_df = pd.DataFrame({'id':station_counts[0], 'checkouts':station_counts[1]})
counts_df = counts_df.join(station_data.set_index('id'), on='id')
counts_df.head()
```

	id	checkouts	lat	lng
0	3.0	9734	42.340021	-71.100812
1	4.0	18058	42.345392	-71.069616
2	5.0	10630	42.341814	-71.090179
3	6.0	23322	42.361285	-71.065140
4	7.0	9163	42.353412	-71.044624

```
# add to the pandas dataframe the distance using the function we defined above and
counts_df.loc[:, 'dist_to_center'] = list(map(haversine, counts_df[['lng', 'lat']])
counts_df.head()
```

	id	checkouts	lat	lng	dist_to_center
0	3.0	9734	42.340021	-71.100812	2.335706
1	4.0	18058	42.345392	-71.069616	0.853095
2	5.0	10630	42.341814	-71.090179	1.802423
3	6.0	23322	42.361285	-71.065140	0.467803
4	7.0	9163	42.353412	-71.044624	0.807582

```
# we will use sklearn to fit a linear regression model
# we will learn a lot about modeling and using sklearn
reg_line = LinearRegression()
reg_line.fit(counts_df['dist_to_center'].values.reshape((len(counts_df['dist_to_center']), 1)))

# use the fitted model to predict
distances = np.linspace(counts_df['dist_to_center'].min(), counts_df['dist_to_center'].max(), 100)
```

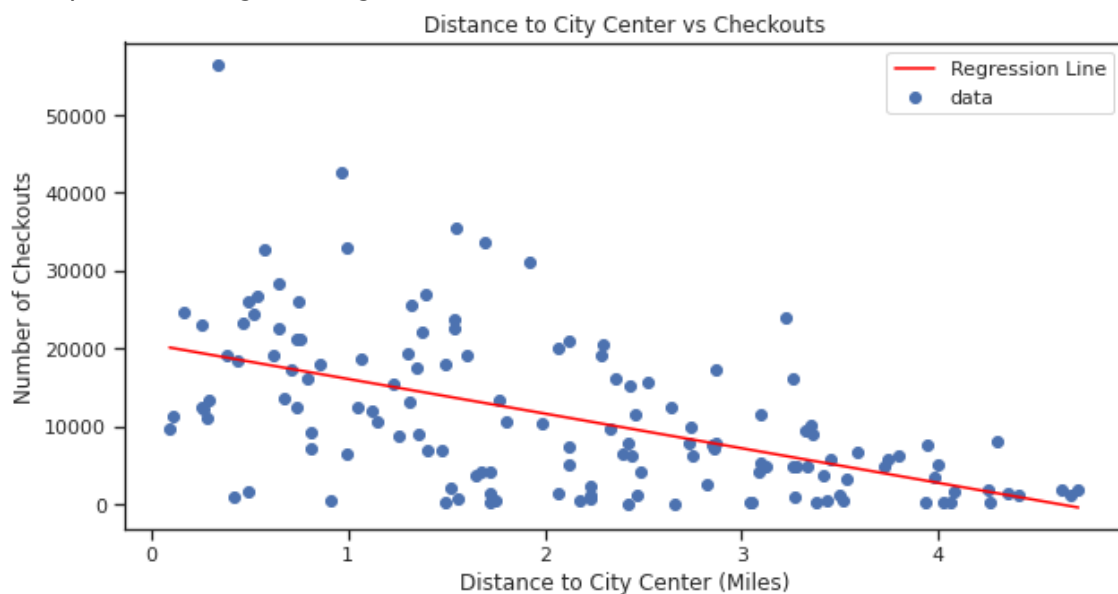
```
fig, ax = plt.subplots(1, 1, figsize=(10, 5))

ax.scatter(counts_df['dist_to_center'].values, counts_df['checkouts'].values, label='data')

ax.plot(distances, reg_line.predict(distances.reshape((len(distances), 1))), color='red')

ax.set_xlabel('Distance to City Center (Miles)')
ax.set_ylabel('Number of Checkouts')
ax.set_title('Distance to City Center vs Checkouts')
ax.legend()
```

<matplotlib.legend.Legend at 0x7f01be5a0e48>



[Colab paid products - Cancel contracts here](#)



Deep Learning - Lecture 3 – Deep Learning Main Principles

Yuri Gordienko, DLI Certified Instructor



DEEP
LEARNING
INSTITUTE

DEEP LEARNING INSTITUTE

DLI Mission



Training you to solve the world's most challenging problems.

- Developers, data scientists and engineers
- Self-driving cars, healthcare and robotics
- Training, optimizing, and deploying deep neural networks



The GPU Teaching Kit is licensed by NVIDIA and New York University under the Creative Commons Attribution-NonCommercial 4.0 International License.

Deck credit: Y. LeCun
MA Ranzato

Who is Y. LeCun?

Who is Yann LeCun?

He is a founding father of convolutional neural nets (CNNs).

He is also one of the main creators of the DjVu image compression technology (together with Léon Bottou and Patrick Haffner).

Chief AI Scientist at Facebook.

LeCun received the 2018 Turing Award, together with Yoshua Bengio and Geoffrey Hinton, for their work on deep learning.

LeCun - together with Geoffrey Hinton and Yoshua Bengio - are referred to by some as the "Godfathers of AI" and "Godfathers of Deep Learning" ... **BUT ... !!!**



2016 IEEE CIS Neural Networks Pioneer Award goes to Jürgen Schmidhuber



Jürgen Schmidhuber is recipient of the 2016 IEEE CIS Neural Networks Pioneer Award, for "pioneering contributions to deep learning and neural networks."

<http://cis.ieee.org/award-recipient.html>

Who is Schmidhuber?

Juergen Schmidhuber: Godel Machines, Meta-Learning, and LSTMs

<https://www.youtube.com/watch?v=3Flo6evmweo>

Who is Schmidhuber?

With his students Sepp Hochreiter, Felix Gers, Fred Cummins, Alex Graves, and others, Schmidhuber published increasingly sophisticated versions of a type of **recurrent neural network** called the **long short-term memory (LSTM)**.

First results were already reported in Hochreiter's **diploma thesis** (1991) which analyzed and overcame the famous vanishing gradient problem.

The name LSTM was introduced in a tech report (1995) leading to **the most cited LSTM publication** (1997).

Deep Learning since ...

<https://people.idsia.ch/~juergen/deeplearning.html>



2016 IEEE CIS Neural Networks Pioneer Award goes to Jürgen Schmidhuber

The award ceremony took place on the 27th of July 2016 in Vancouver at the Award Banquet of [IJCNN 2016](#).

Transcript of the 3 min acceptance speech:

Dear IEEE,

it is a great honor to be listed among previous awardees such as K. Fukushima, who is present at this conference, the father of the deep convolutional neural architecture everybody is using now for computer vision.

The only thing that makes me a bit sad at this otherwise happy moment is that the Ukrainian mathematician A. G. Ivakhnenko, the father of deep learning himself, never got this award. His team had deep multilayer perceptrons with 8 layers or so back in the 1960s when I was a baby, at a time when others still focused on the limitations of shallow nets with a single layer. Apparently he was so far ahead of his time that even the not so young members of the award committees failed to appreciate the depth of his work.

Who is A. G. Ivakhnenko?

...

the father of deep learning himself

and

перший виконуючий обов'язки декана ФІОТ
- перший “неофіційний” декан нашого факультету

Who is A. G. Ivakhnenko?

Ukrainian mathematician most famous for developing the Group Method of Data Handling (GMDH), a method of inductive statistical learning, for which he is sometimes referred to as the **"Father of Deep Learning"**.

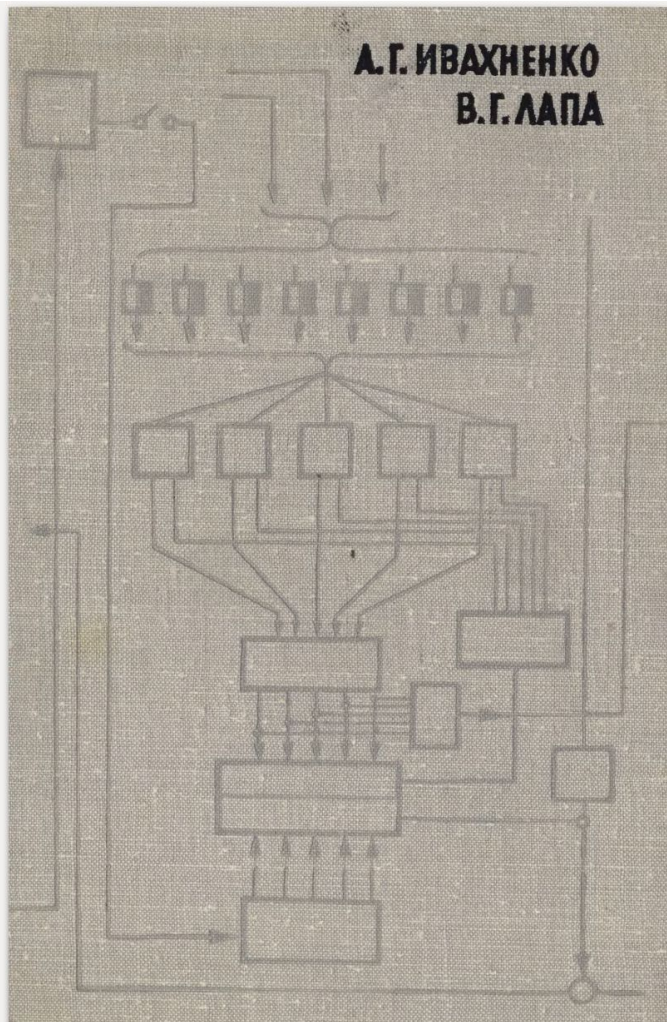
Main results in the context of DL:

- Principle of construction of self-organizing deep learning networks,
- Design of multilayered neural networks with active neurons, where each neuron is an algorithm,
- Principle of self-learning pattern recognition. It was demonstrated at first in the cognitive system "Alpha", created under his leadership.



Alexey Ivakhnenko

(Олексій Григорович Івахненко)
(30 March 1913 – 16 October 2007)



АКАДЕМИЯ НАУК УКРАИНСКОЙ ССР

А. Г. ИВАХНЕНКО
В. Г. ЛАПА

**КИБЕРНЕТИЧЕСКИЕ
ПРЕДСКАЗЫВАЮЩИЕ
УСТРОЙСТВА**



Киев — 1965

Перша книжка А.Г.Івахненка про його систему "Альфа" - першу 8-шарову глибинну мережу.

Система "Альфа" - перша 8-шарова глибинна мережа

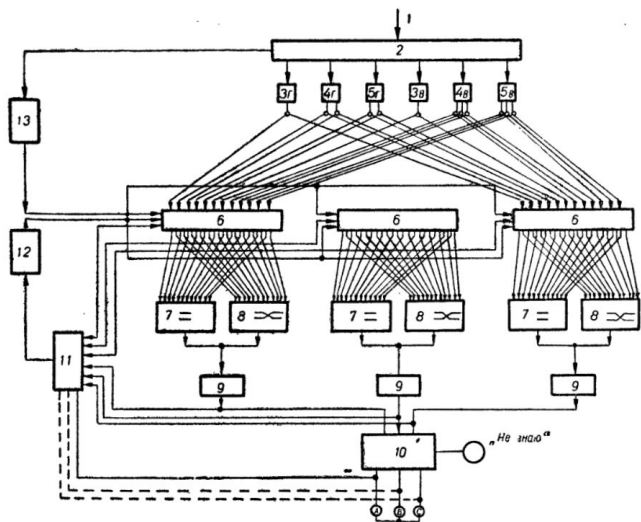


Рис. 49. Структурная схема распознающей системы с положительной обратной связью.

1 — различаемый образ; 2 — входное устройство; 3-5 — датчики; 6 — группы ассоциирующих элементов; 7 — прямые усилители; 8 — реверсирующие усилители; 9 — сумматоры; 10 — индикатор большого напряжения; 11 — управление порядком самообучения; 12 — положительная связь самообучения; 13 — разомкнутая связь обучения.



О.Г. Івахненко (справа) та Норберт Вінер (зліва) під час конференції ІФАК у Києві (1960 р.)

Насправді вона була розроблена ще раніше, не у 1965-1971, як пише Шмідхубер, а у 1962 році.

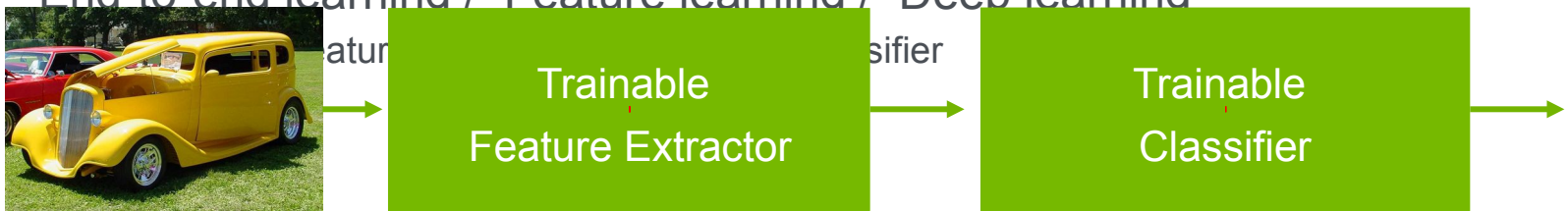
21. Івахненко О.Г., Системи, що саморганізуються, з додатними зворотними зв'язками. "Автоматика", №3, 1962.

Deep learning = Learning representations/features

- The traditional model of pattern recognition (since the late 50's)
 - Fixed/engineered features (or fixed kernel) + trainable classifier

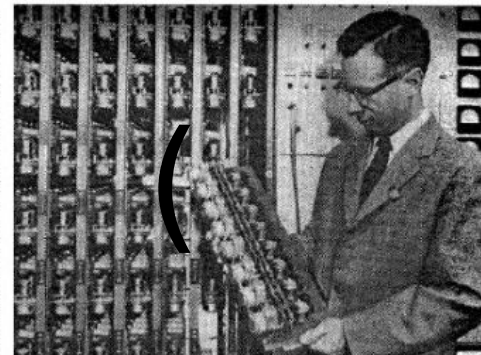
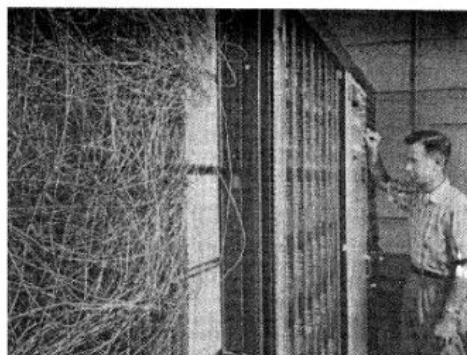
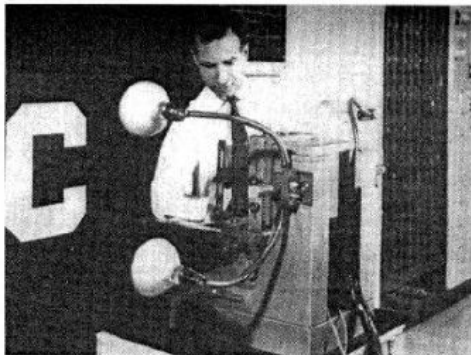
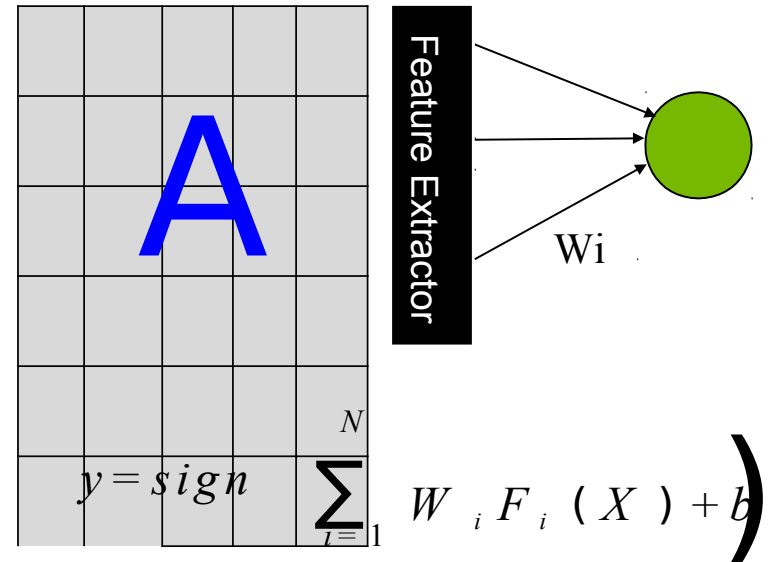


- End-to-end learning / Feature learning / Deep learning



This basic model has not evolved much since the 50's

- The first learning machine: the **Perceptron**
 - Built at Cornell in 1960
- The Perceptron was a **linear classifier** on top of a simple **feature extractor**
- The vast majority of practical applications of ML today use glorified **linear classifiers** or glorified template matching.
- **Designing a feature extractor requires considerable efforts by experts.**



Linear machines and their limitations

Linear machines: regression with mean square

Linear regression, mean square loss:

- Decision rule:
- Loss function:
- Gradient of loss:
- Update rule:
- Direct solution: solve linear system

Linear machines

Perception

- Decision rule: $(F$ is the threshold function)
- Loss function:
- Gradient of loss:
- Update rule:
- Direct solution: find W such that

Linear machines: logistic regression

Logistic regression, negative log-likelihood loss function:

- Decision rule:
- Loss function:
- Gradient of loss:
- Update rule:

General gradient-based supervised learning machine

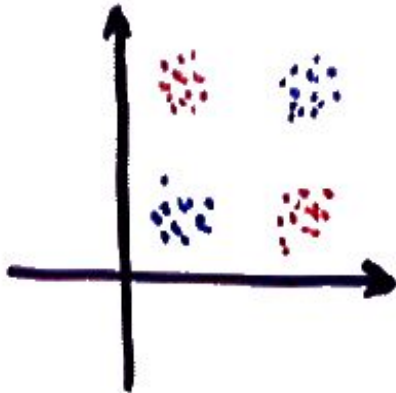
Neural nets, and many other models:

- Decision rule: $y = F(W, X)$, where F is some function, and W some parameter vector.
- Loss function: $L(y, f)$ where $D(y, f)$ measures the “discrepancy” between A and B .
- Gradient loss:
- Update rule:

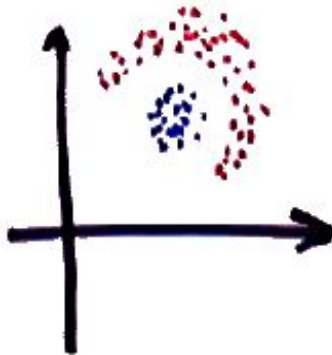
Three questions:

- What architecture $F(W, X)$.
- What loss function $L(W, y^i, X^i)$.
- What optimization method.

Limitations of Linear Machines

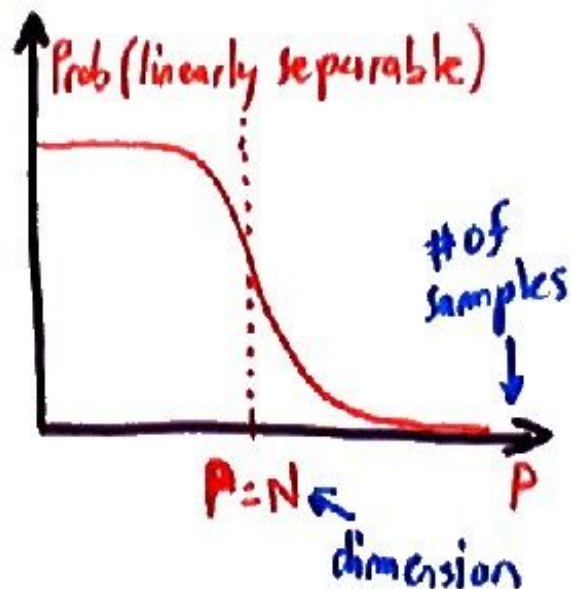


The *Linearly separable* dichotomies are the partitions that are realizable by a linear classifier (the boundary between the classes is a hyperplane).



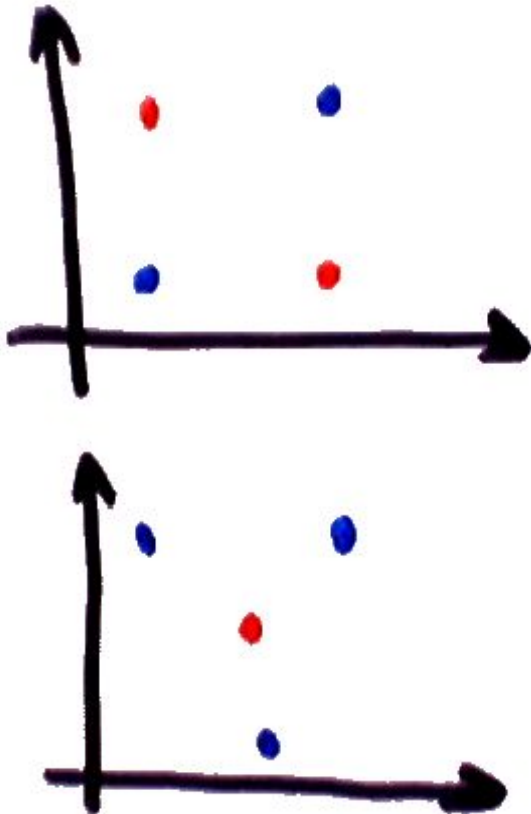
Number of Linearly Separable Dichotomies

The probability that P samples of dimension N are linearly separable goes to zero very quickly as P grows larger than N (Cover's theorem, 1966).



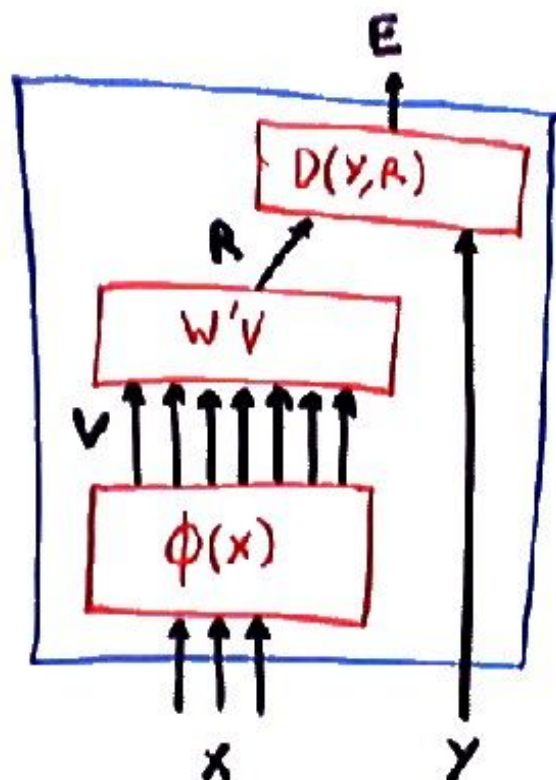
- Problem: there are 2^P possible dichotomies of P points.
- Only about N are linearly separable.
- If P is larger than N , the probability that a random dichotomy is linearly separable is very, very small.

Example of Non-Linearly Separable Dichotomies



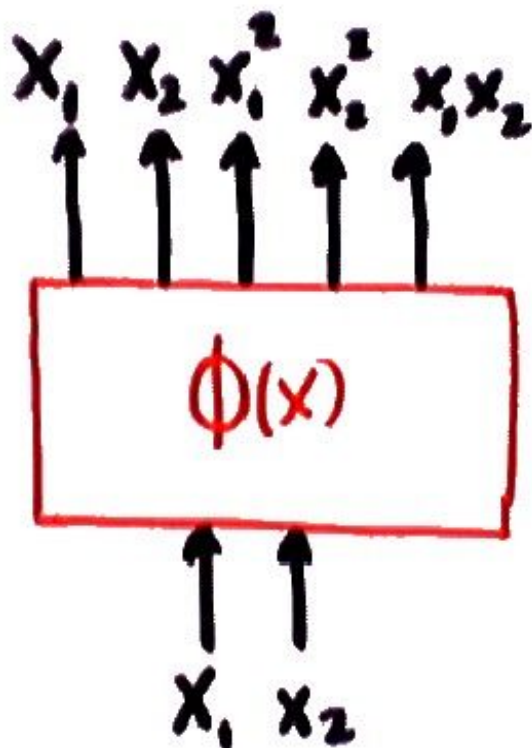
- Some seemingly simple dichotomies are not linearly separable
- **Question:** How do we make a given problem linearly separable?

Making N Larger: Preprocessing



- **Answer 1:** we make N larger by augmenting the input variables with new “features”.
- we map/project X from its original N -dimensional space into a higher dimensional space where things are more likely to be linearly separable, using a vector function $\Phi(X)$.
- $E(Y, X, W) = D(Y, R)$
- $R = f(W'V)$
- $V = \Phi(X)$

Adding Cross-Product Terms



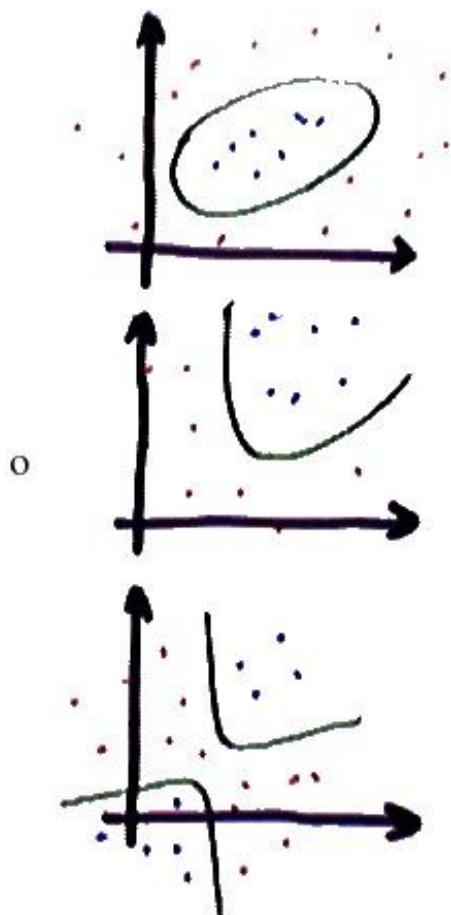
- Polynomial Expansion.
- If our original input variables are $(1, x_1, x_2)$, we construct a new *feature vector* with the following components:

$$\Phi(1, x_1, x_2) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

i.e. we add all the cross-products of the original variables.

- we map/project X from its original N -dimensional space into a higher dimensional space with $N(N+1)/2$ dimensions.

Polynomial Mapping



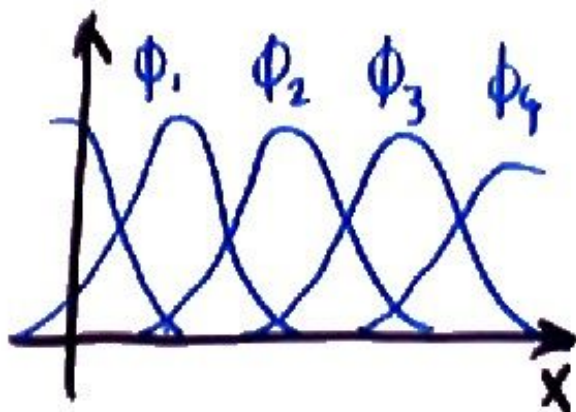
- Many new functions are now separable with the new architecture.
- With cross-product features, the family of class boundaries in the original space is the conic sections (ellipse, parabola, hyperbola).
- to each possible boundary in the original space corresponds a linear boundary in the transformed space.
- Because this is essentially a linear classifier with a preprocessing, we can use standard linear learning algorithms (perceptron, linear regression, logistic regression...).

Problems with polynomial mapping

- We can generalize this idea to higher degree polynomials, adding cross-product terms with 3, 4 or more variables
- Unfortunately, the number of terms is the number of combinations d choose N , which grows like N^d , where d is the degree, and N the number of original variables
- In particular, the number of free parameters that must be learned is also of order N^d .
- This is impractical for large N and for $d > 2$
- Example: handwritten digit recognition (16x16 pixel images). Number of variables: 256. degree 2: 32,896 variables. Degree 3: 2,796,160. degree 4: 247,460,160...

Next Idea: Tile the Space

place a number of equally-spaced “bumps” that cover the entire input space.



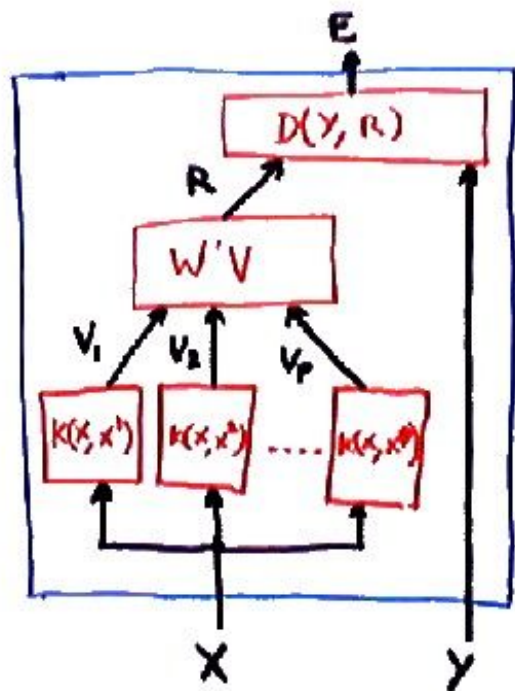
- For classification, the bumps can be Gaussians
- For regression, the basis functions can be wavelets, sine/cosine, splines (pieces of polynomials)....
- **problem:** this does not work with more than a few dimensions.
- The number of bumps necessary to cover an N dimensional space grows exponentially with N .

Sample-Centered Basis Functions (Kernels)

Place the center of a basis function around each training sample. That way, we only spend resources on regions of the space where we actually have training samples.

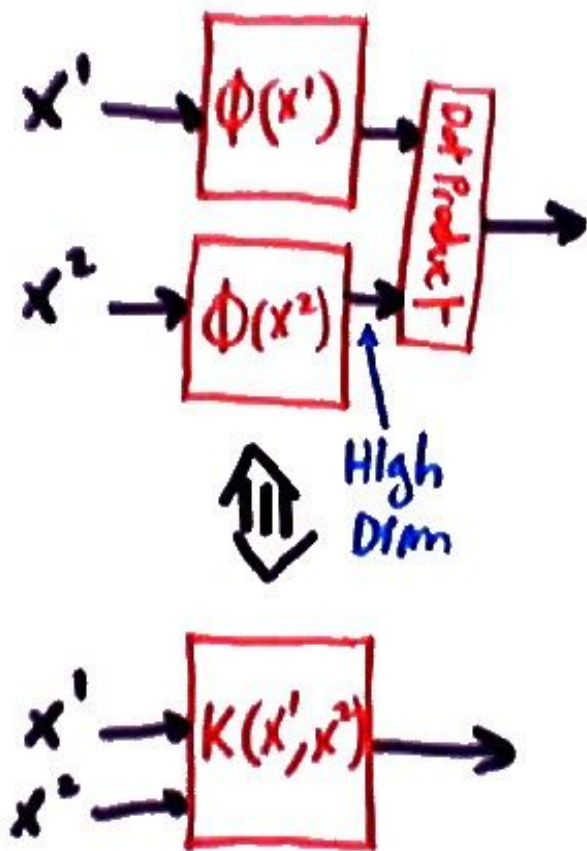
- Discriminant function:

$$f(X, W) = \sum_{k=1}^{k=P} W_k K(X, X^k)$$



- $K(X, X')$ often takes the form of a *radial basis function*:
 $K(X, X') = \exp(b\|X - X'\|^2)$ or a polynomial $K(X, X') = (X \cdot X' + 1)^m$
- This is a very common architecture, which can be used with a number of energy functions.
- In particular, this is the architecture of the so-called **Support Vector Machine** (SVM), but the energy function of the SVM is a bit special. We will study it later in the course.

The Kernel Trick



- If the kernel function $K(X, X')$ verifies the *Mercer conditions*, then there exist a mapping Φ , such that $\Phi(X) \cdot \Phi(X') = K(X, X')$.
- The Mercer conditions are that K must be symmetric, and must be positive definite (i.e $K(X, X)$ must be positive for all X).
- In other words, if we want to map our X into a high-dimensional space (so as to make them linearly separable), and all we have to do in that space is compute dot products, we can take a shortcut and simply compute $K(X^1, X^2)$ without going through the high-dimensional space.
- This is called the “**kernel trick**”. It is used in many so-called Kernel-based methods, including Support Vector Machines.

Examples of Kernels

- **Quadratic kernel:** $\Phi(X) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$ then

$$K(X, X') = \Phi(X) \cdot \Phi(X') = (X \cdot X' + 1)^2$$

- **Polynomial kernel:** this generalizes to any degree d . The kernel that corresponds to $\Phi(X)$ being a polynomial of degree d is

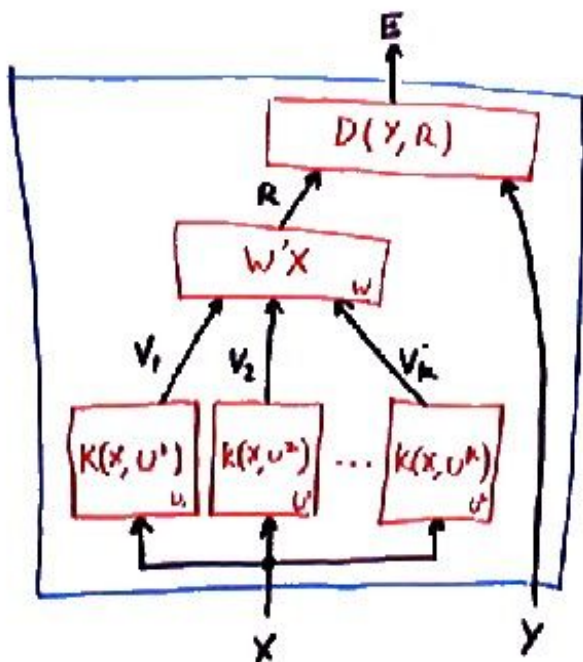
$$K(X, X') = \Phi(X) \cdot \Phi(X') = (X \cdot X' + 1)^d.$$

- **Gaussian Kernel:**

$$K(X, X') = \exp(-b\|X - X'\|^2)$$

This kernel, sometimes called the Gaussian Radial Basis Function, is very commonly used.

Sparse Basis Functions

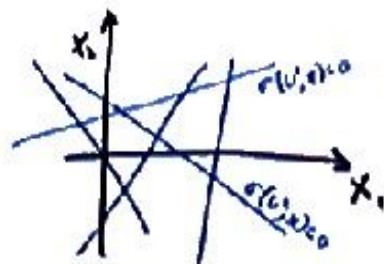
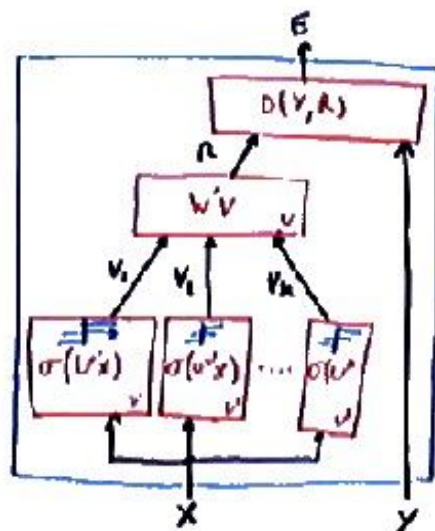


- Place the center of a basis function around areas containing training samples.
- Idea 1: use an unsupervised clustering algorithm (such as K-means or mixture of Gaussians) to place the centers of the basis functions in areas of high sample density.
- Idea 2: adjust the basis function centers through gradient descent in the loss function.

The discriminant function F is:

$$F(X, W, U^1, \dots, U^K) = \sum_{k=1}^{k=K} W_k K(X, U^k)$$

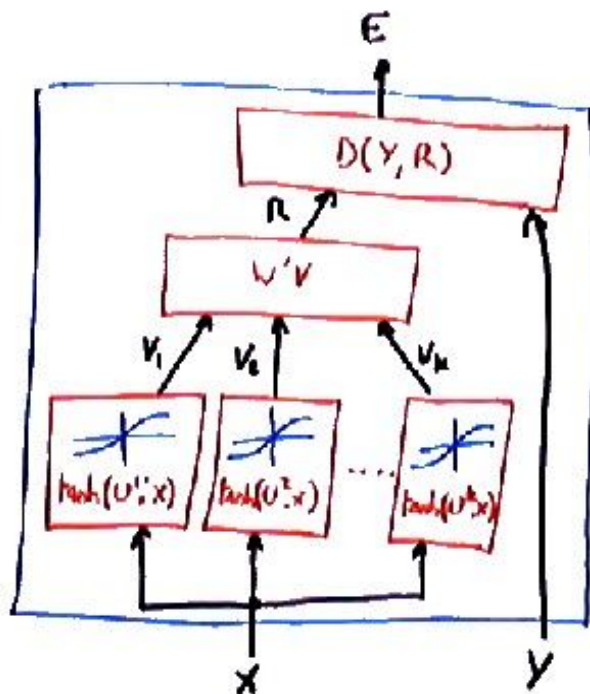
Other Idea: Random Directions



- Partition the space in lots of little domains by randomly placing lots of hyperplanes.
- Use many variables of the type $q(W^k X)$, where q is the threshold function (or some other squashing function) and W_k is a randomly picked vector.
- This is the original Perceptron.
- Without the non-linearity, the whole system would be linear (product of linear operations), and therefore would be no more powerful than a linear classifier.
- **problem:** a bit of a wishful thinking, but it works occasionally.

Neural Net with a Single Hidden Layer

A particularly interesting type of basis function is the sigmoid unit: $V_k = \tanh(U'^k X)$



- a network using these basis functions, whose output is $R = \sum_{k=1}^{K} W_k V_k$ is called a *single hidden-layer neural network*.
- Similarly to the RBF network, we can compute the gradient of the loss function with respect to the U^k :

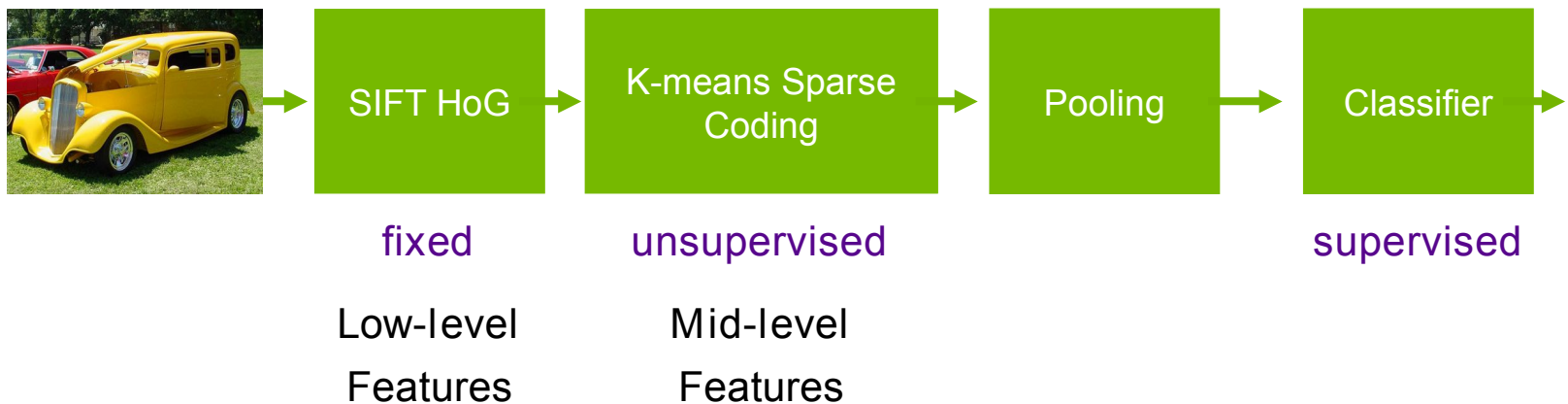
$$\begin{aligned}\frac{\partial L(W)}{\partial U^j} &= \frac{\partial L(W)}{\partial R} W_j \frac{\partial \tanh(U_j' X)}{\partial U_j} \\ &= \frac{\partial L(W)}{\partial R} W_j \tanh'(U_j' X) X'\end{aligned}$$

Any well-behaved function can be approximated as close as we wish by such networks (but K might be very large).

Architecture of “mainstream” pattern recognition systems

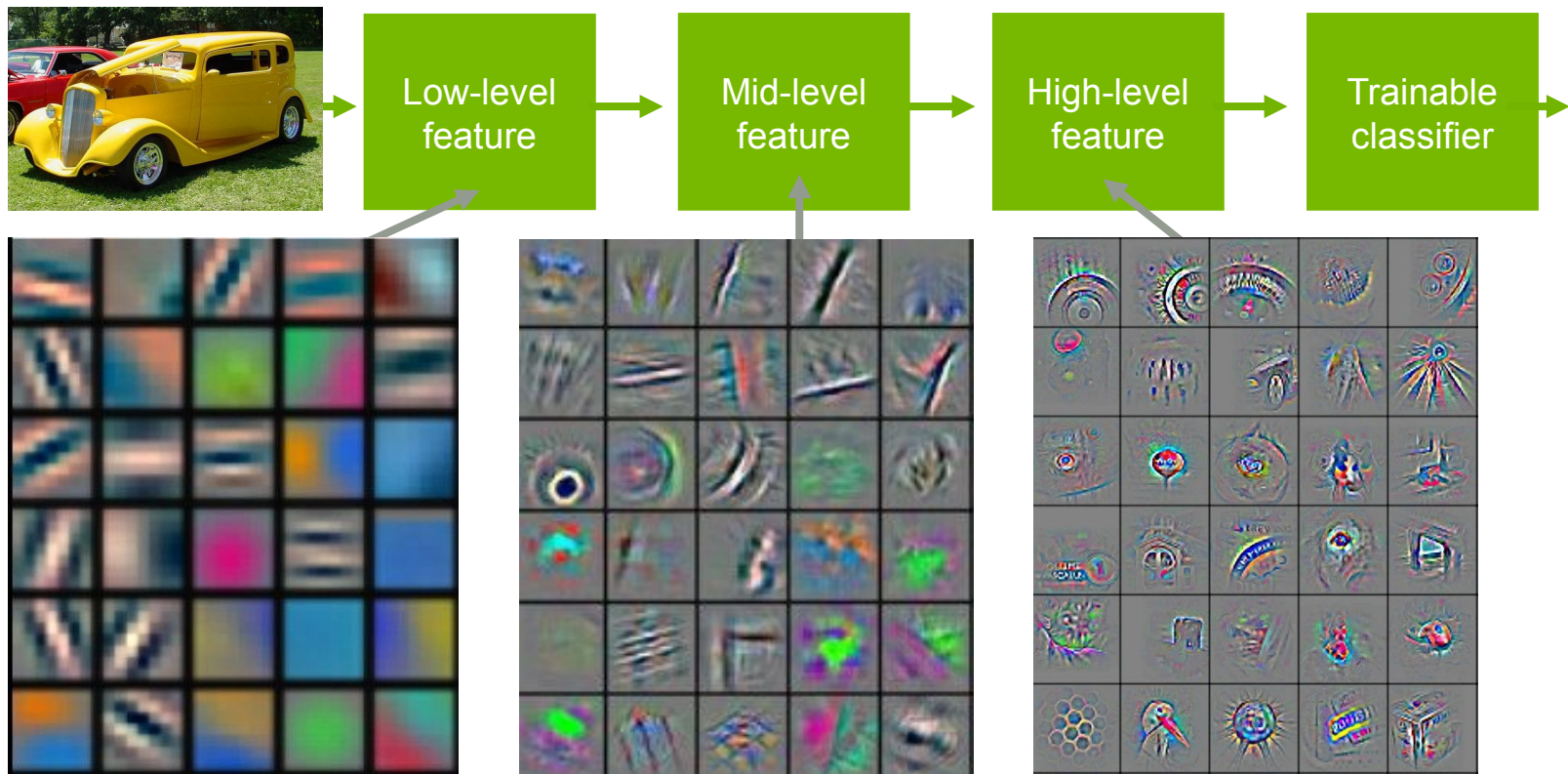
– Modern architecture for pattern recognition

– Speech recognition: early 90's – 2011



Deep learning = learning hierarchical representations

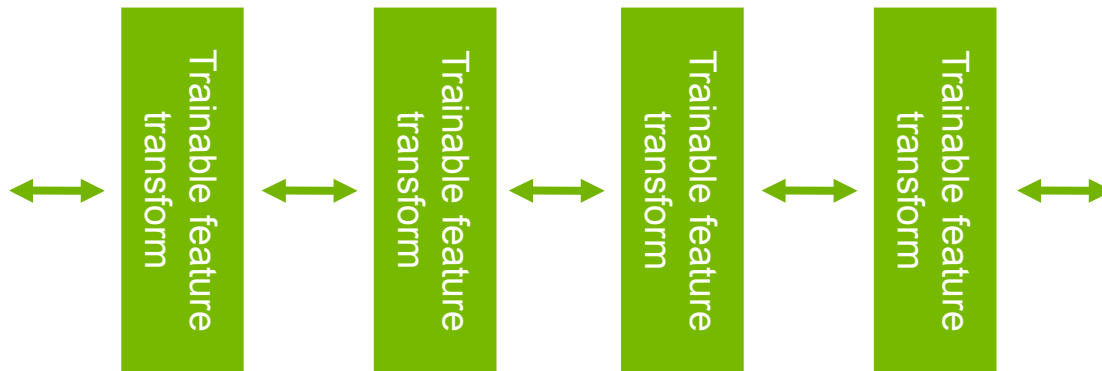
It's **deep** if it has **more than one stage** of non-linear feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

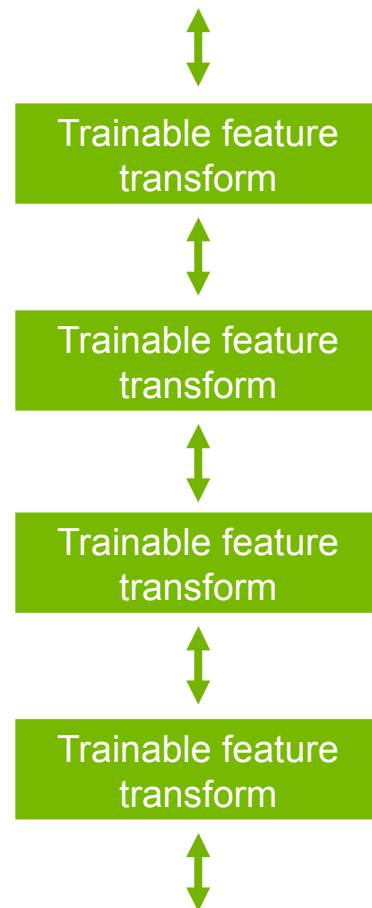
Trainable feature hierarchy

- Hierarchy of representations with increasing level of abstraction Each stage is a kind of trainable feature transform
- Image recognition
 - Pixel → edge → texton → motif → part → object
- Text
 - Character → word → word group → clause → sentence → story
- Speech
 - Sample → spectral band → sound → ... → phone → phoneme → word



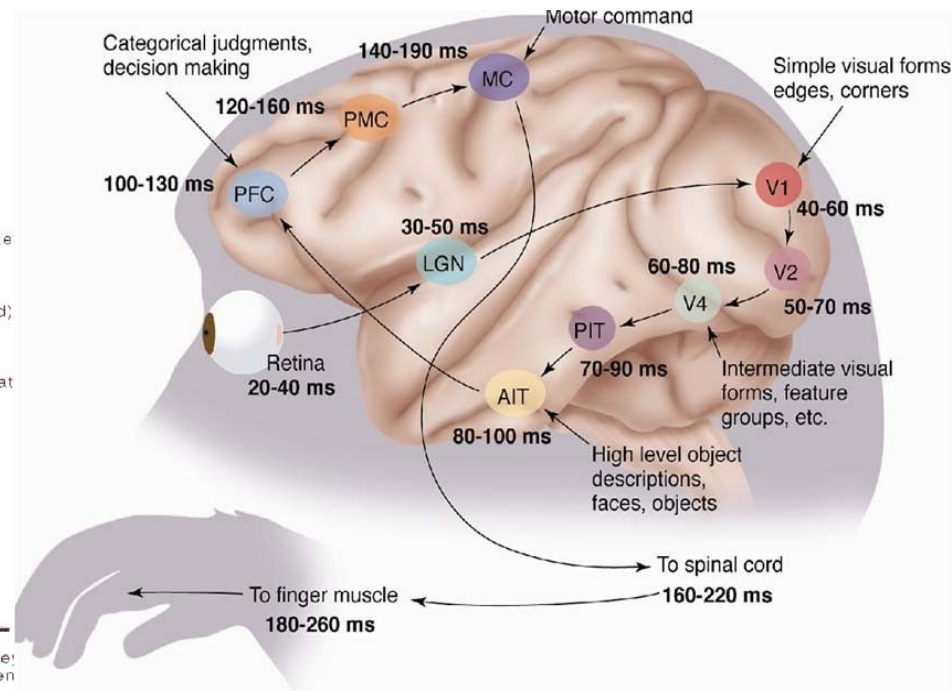
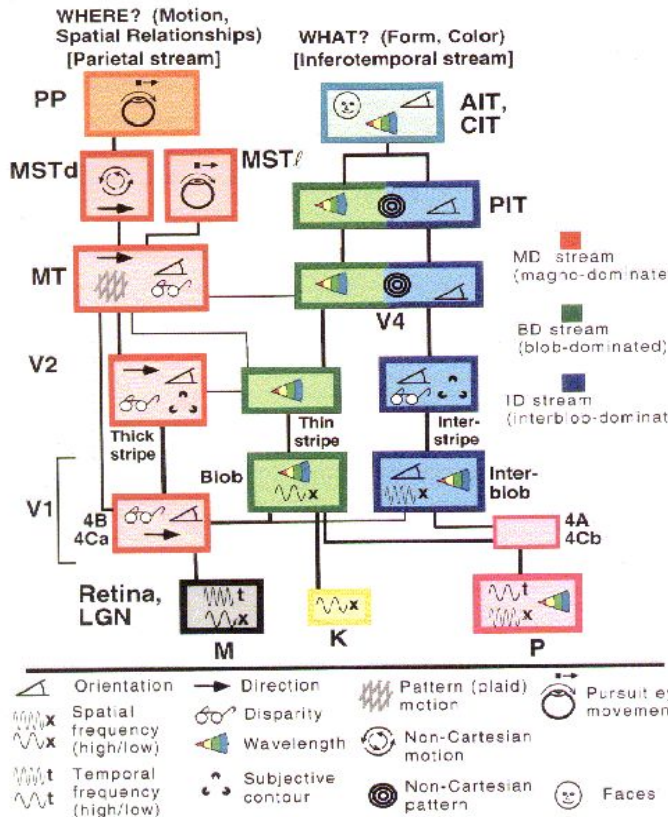
Learning representations: a challenge for ML, CV, AI, neuroscience, cognitive science...

- **How do we learn representations** of the perceptual world?
 - How can a perceptual system build itself by looking at the world?
 - How much prior structure is necessary
- **ML/AI**: how do we learn features or feature hierarchies?
 - What is the fundamental principle? What is the learning algorithm? What is the architecture?
- **Neuroscience**: how does the cortex learn perception?
 - Does the cortex “run” a single, general learning algorithm? (or a small number of them)
- **CogSci**: how does the mind learn abstract concepts on top of less abstract ones?
- **Deep Learning addresses the problem of learning hierarchical representations with a single algorithm**
 - Or perhaps with a few algorithms



The mammalian visual cortex is hierarchical

- The ventral (recognition) pathway in the visual cortex has multiple stages Retina - LGN - V1 - V2 - V4 - PIT - AIT
- **Lots of intermediate representations**



[picture from Simon Thorpe]

[Gallant & Van Essen]

Let's be inspired by nature, but not too much

- It's nice imitate Nature,
- But we also need to **understand**
 - How do we know which details are important?
 - Which details are merely the result of evolution, and the constraints of biochemistry?
- For airplanes, we developed aerodynamics and compressible fluid dynamics.
 - We figured that feathers and wing flapping weren't crucial
- **QUESTION: What is the equivalent of aerodynamics for understanding intelligence?**



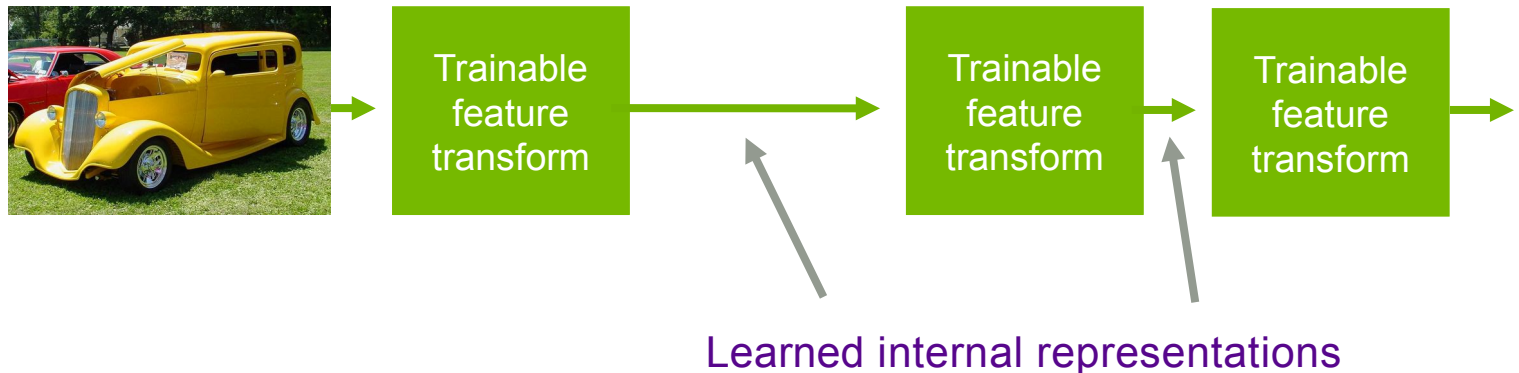
L'Avion III de Clément Ader, 1897

(Musée du CNAM, Paris)

His Eole took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it.

Trainable feature hierarchies: end-to-end learning

- A hierarchy of trainable feature transforms
 - Each module transforms its input representation into a higher-level one.
 - High-level features are more global and more invariant
 - Low-level features are shared among categories



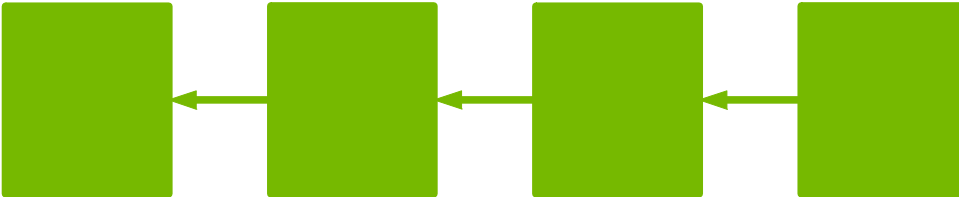
- How can we make all the modules trainable and get them to learn appropriate representations?

Three types of deep architectures

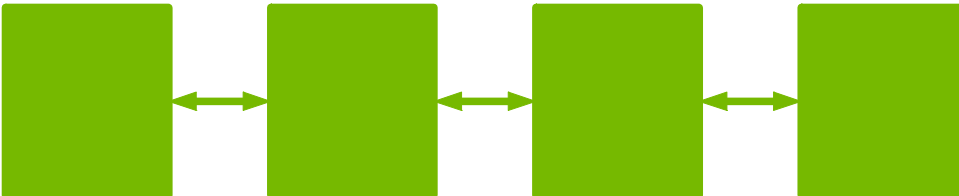
- Feed-forward: multilayer neural nets, convolutional nets



- Feed-back: stacked sparse coding, deconvolutional nets



- Bi-directional: Deep Boltzmann Machines, stacked auto-encoders



Three types of training protocols

- Purely Supervised
 - Initialize parameters randomly Train in supervised mode
 - Typically with SGD, using backprop to compute gradients
 - Used in most practical systems for speech and image recognition
- Unsupervised, layerwise + supervised classifier on top
 - Train each layer unsupervised, one after the other
 - Train a supervised classifier on top, keeping the other layers fixed
 - Good when very few labeled samples are available
- Unsupervised, layerwise + global supervised fine-tuning
 - Train each layer unsupervised, one after the other
 - Add a classifier layer, and retrain the whole thing supervised
 - Good when label set is poor (e.g. pedestrian detection)
- Unsupervised pre-training often uses regularized auto-encoders

Do we really need deep architectures?

- **Theoretician's dilemma**: “We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?”

$$y = \sum_{i=1}^P \alpha_i K(X, X^i) \quad y = F(W^1 . F(W^0 . X))$$

- kernel machines (and 2-layer neural nets) are “universal”.

- Deep learning machines

$$y = F(W^K . F(W^{K-1} . F(\dots F(W^0 . X) \dots)))$$

- **Deep machines are more efficient for representing certain classes of functions**, particularly those involved in visual recognition

- They can represent more complex functions with less “hardware”

- We need an efficient parameterization of the class of functions that are useful for “AI” tasks (vision, audition, NLP...)

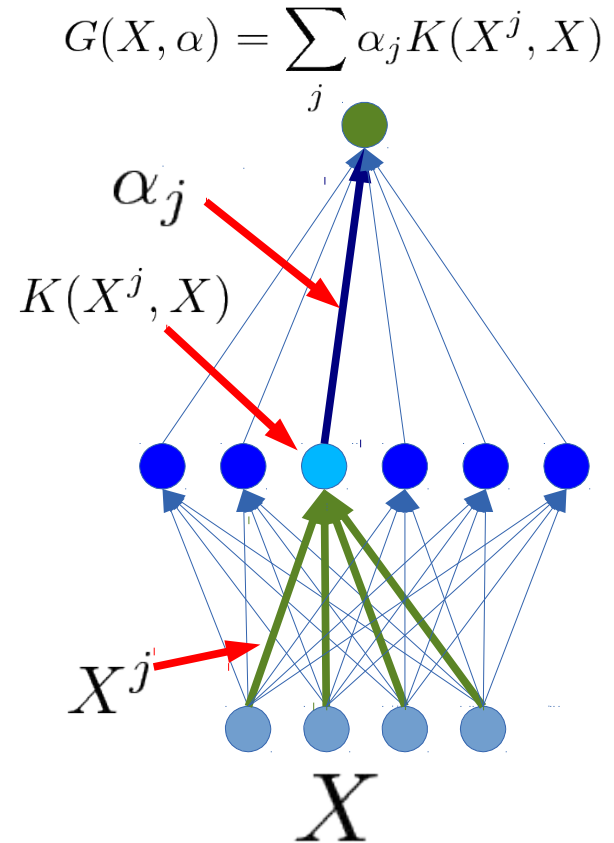
Why would deep architectures be more efficient?

[Bengio & LeCun 2007 “Scaling Learning Algorithms Towards AI”]

- A deep architecture trades space for time (or breadth for depth)
 - More layers (more sequential computation),
 - But less hardware (less parallel computation).
- Example1: N-bit parity
 - requires $N-1$ XOR gates in a tree of depth $\log(N)$.
 - Even easier if we use threshold gates
 - requires an exponential number of gates if we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).
- Example2: circuit for addition of 2 N-bit binary numbers
 - Requires $O(N)$ gates, and $O(N)$ layers using N one-bit adders with ripple carry propagation.
 - Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
 - Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms $O(2^N)$

Which models are deep?

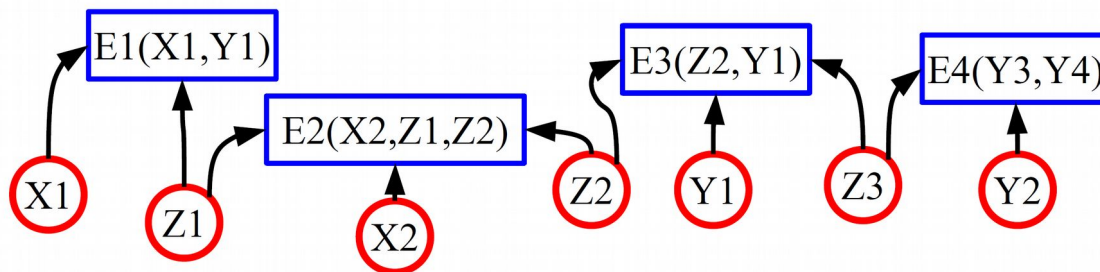
- 2-layer models are not deep (even if you train the first layer)
 - Because there is no feature hierarchy
- Neural nets with 1 hidden layer are not deep
- SVMs and Kernel methods are not deep
 - Layer1: kernels; layer2: linear
 - The first layer is “trained” in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- Classification trees are not deep
 - No hierarchy of features. All decisions are made in the input space



Are graphical models deep?

- There is no opposition between graphical models and deep learning.
 - Many deep learning models are formulated as factor graphs
 - Some graphical models use deep architectures inside their factors
- Graphical models can be deep (but most are not). Factor graph: sum of energy functions
 - Over inputs X , outputs Y and latent variables Z . Trainable parameters: W

$$-\log P(X, Y, Z | W) \propto E(X, Y, Z, W) = \sum_i E_i(X, Y, Z, W_i)$$



- Each energy function can contain a deep network
- The whole factor graph can be seen as a deep network

Deep learning: A theoretician's nightmare?

- Deep Learning involves non-convex loss functions
 - With non-convex losses, all bets are off
 - Then again, every speech recognition system ever deployed has used non-convex optimization (GMMs are non convex).
- But to some of us all “interesting” learning is non convex
 - Convex learning is invariant to the order in which sample are presented (only depends on asymptotic sample frequencies).
 - Human learning isn't like that: we learn simple concepts before complex ones. The order in which we learn things matter.

Deep learning: A theoretician's nightmare?

– No generalization bounds?

- Actually, the usual VC bounds apply: most deep learning systems have a finite VC dimension
- We don't have tighter bounds than that.
- But then again, how many bounds are tight enough to be useful for model selection?

– It's hard to prove anything about deep learning systems

- Then again, if we only study models for which we can prove things, we wouldn't have speech, handwriting, and visual object recognition systems today.

Deep learning: A theoretician's paradise?

- Deep learning is about representing high-dimensional data
 - There has to be interesting theoretical questions there what is the geometry of natural signals?
 - Is there an equivalent of statistical learning theory for unsupervised learning?
 - What are good criteria on which to base unsupervised learning?
- Deep learning systems are a form of latent variable factor graph
 - Internal representations can be viewed as latent variables to be inferred, and deep belief networks are a particular type of latent variable models.
 - The most interesting deep belief nets have intractable loss functions: how do we get around that problem?
- Lots of theory at the 2012 IPAM summer school on deep learning
 - Wright's parallel SGD methods, Mallat's “scattering transform”, Osher's “split Bregman” methods for sparse modeling, Morton's “algebraic geometry of DBN”,....

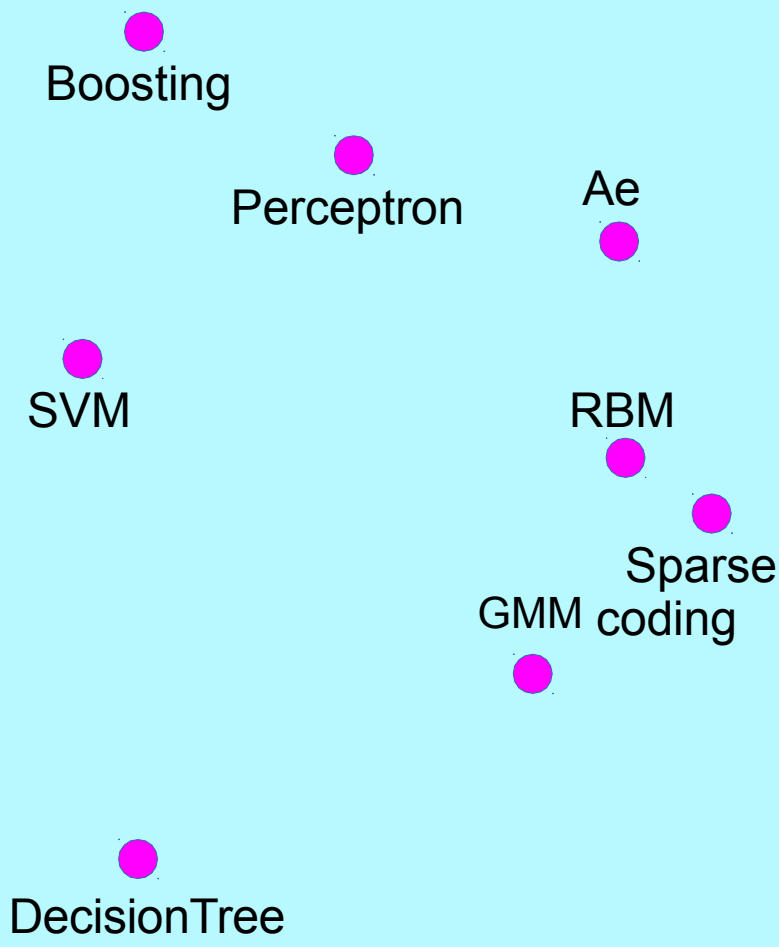
Deep learning and feature learning today

- Deep learning has been the hottest topic in speech recognition in the last 2 years
 - A few long-standing performance records were broken with deep learning methods
 - Microsoft and google have both deployed dl-based speech recognition system in their products
 - Microsoft, google, IBM, nuance, AT&T, and all the major academic and industrial players in speech recognition have projects on deep learning
- Deep learning is the hottest topic in computer vision
 - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
 - But the record holders on ImageNet and semantic segmentation are convolutional nets
- Deep learning is becoming hot in natural language processing
- Deep learning/feature learning in applied mathematics
 - The connection with applied math is through sparse coding, non-convex optimization, stochastic gradient algorithms, etc...

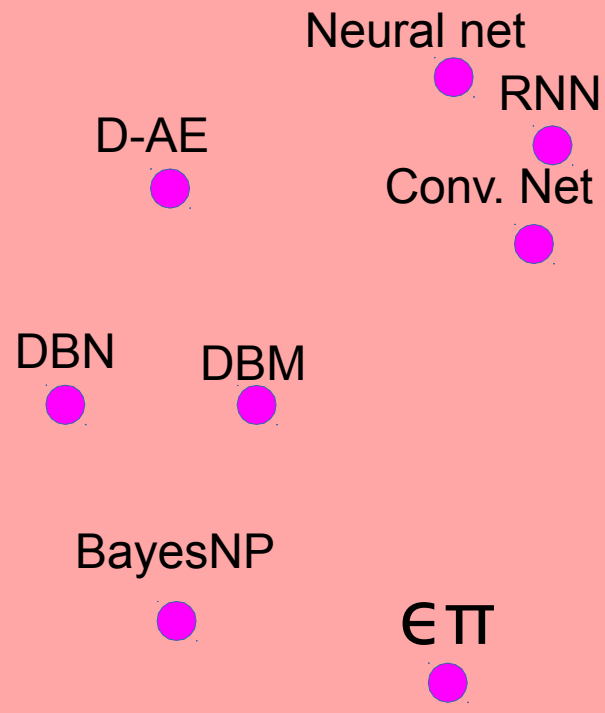
In many fields, feature learning has caused a revolution (methods used in commercially deployed systems)

- Speech Recognition I (late 1980s)
 - Trained mid-level features with Gaussian mixtures (2-layer classifier)
- Handwriting Recognition and OCR (late 1980s to mid 1990s)
 - Supervised convolutional nets operating on pixels
- Face & People Detection (early 1990s to mid 2000s)
 - Supervised convolutional nets operating on pixels (YLC 1994, 2004, Garcia 2004)
 - Haar features generation/selection (Viola-Jones 2001)
- Object Recognition I (mid-to-late 2000s: Ponce, Schmid, Yu, YLC....)
 - Trainable mid-level features (K-means or sparse coding)
- Low-Res Object Recognition: road signs, house numbers (early 2010's)
 - Supervised convolutional net operating on pixels
- Speech Recognition II (circa 2011)
 - Deep neural nets for acoustic modeling
- Object Recognition III, Semantic Labeling (2012, Hinton, YLC,...)
 - Supervised convolutional nets operating on pixels

Shallow

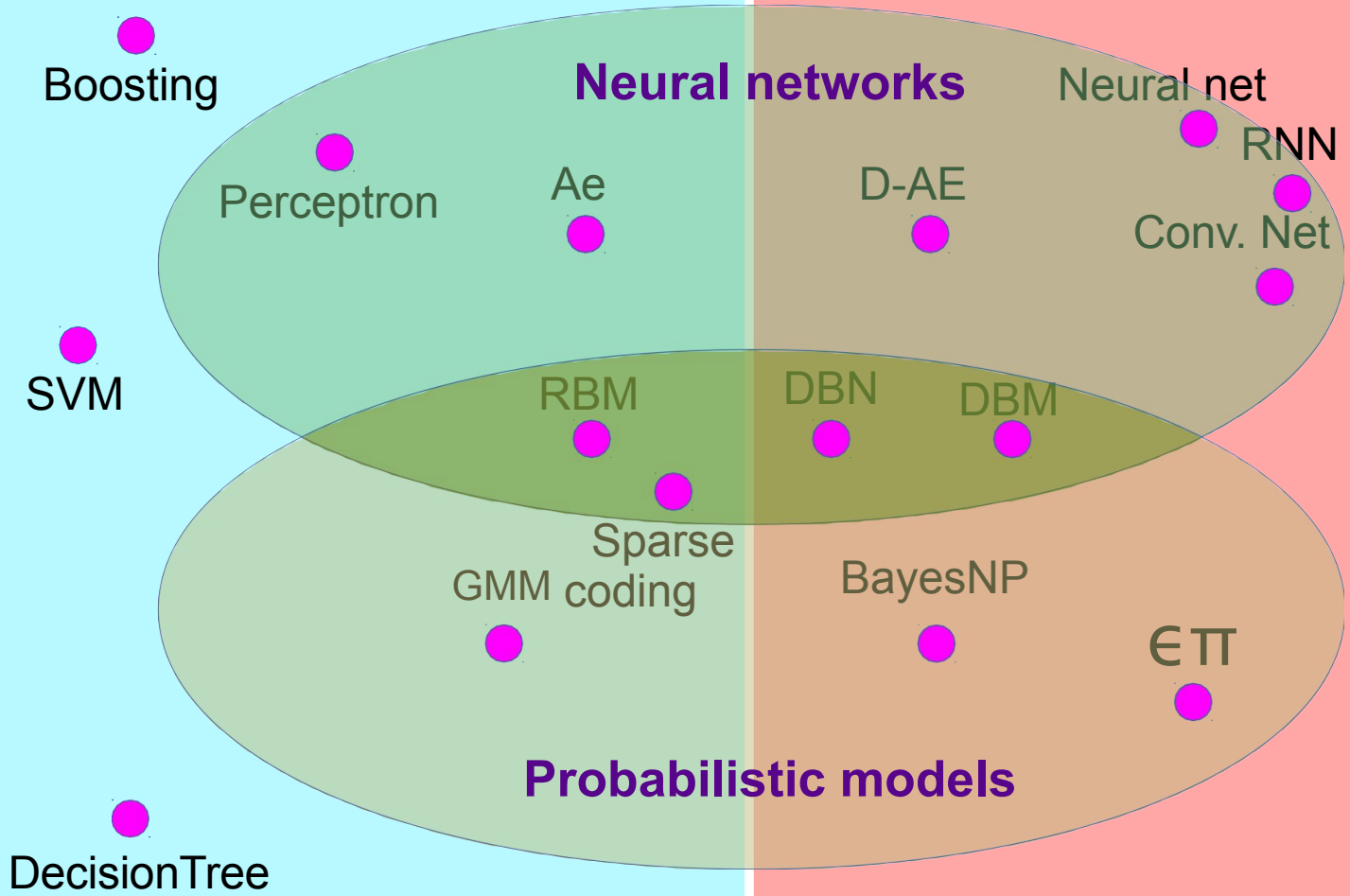


Deep



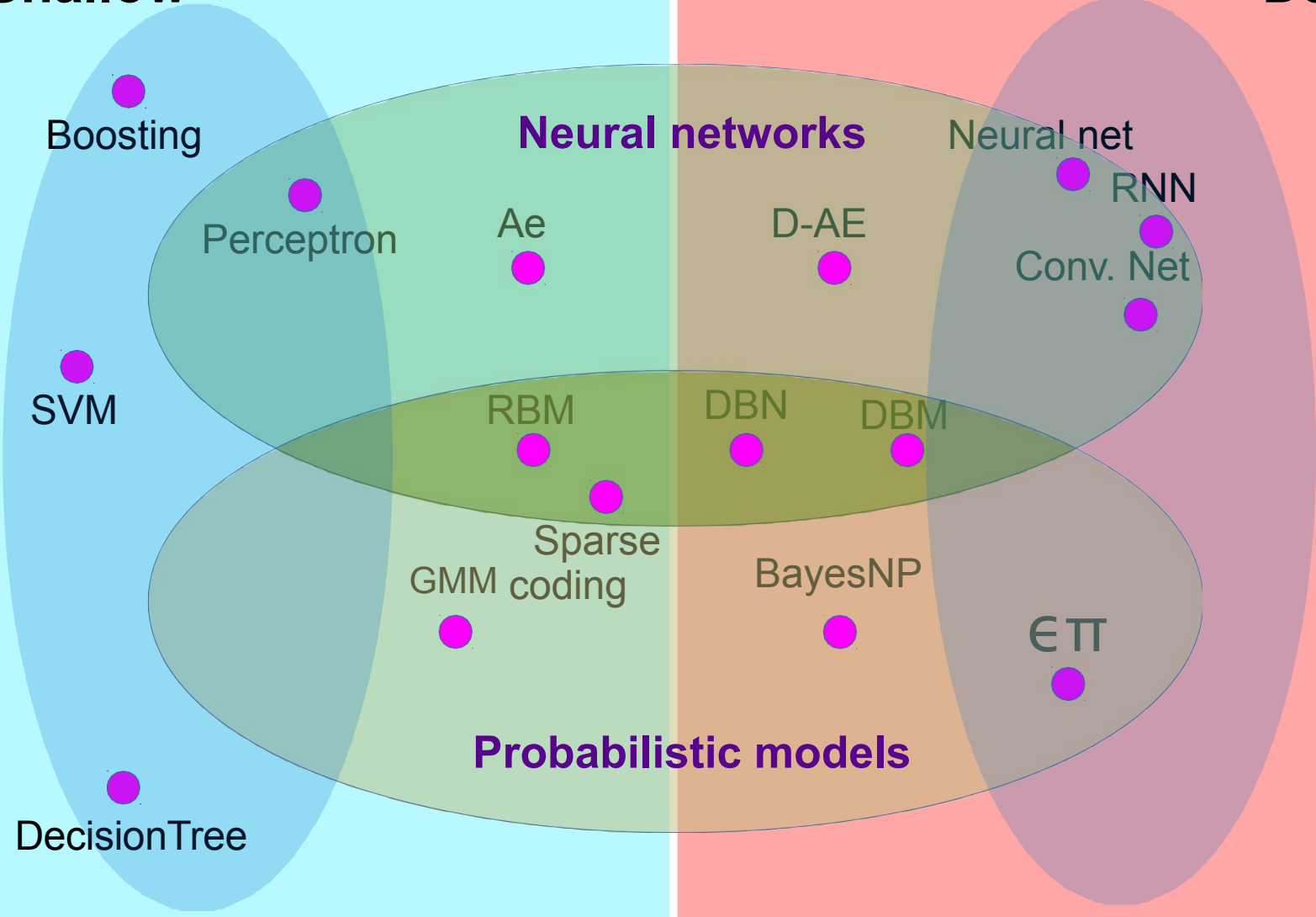
Shallow

Deep



Shallow

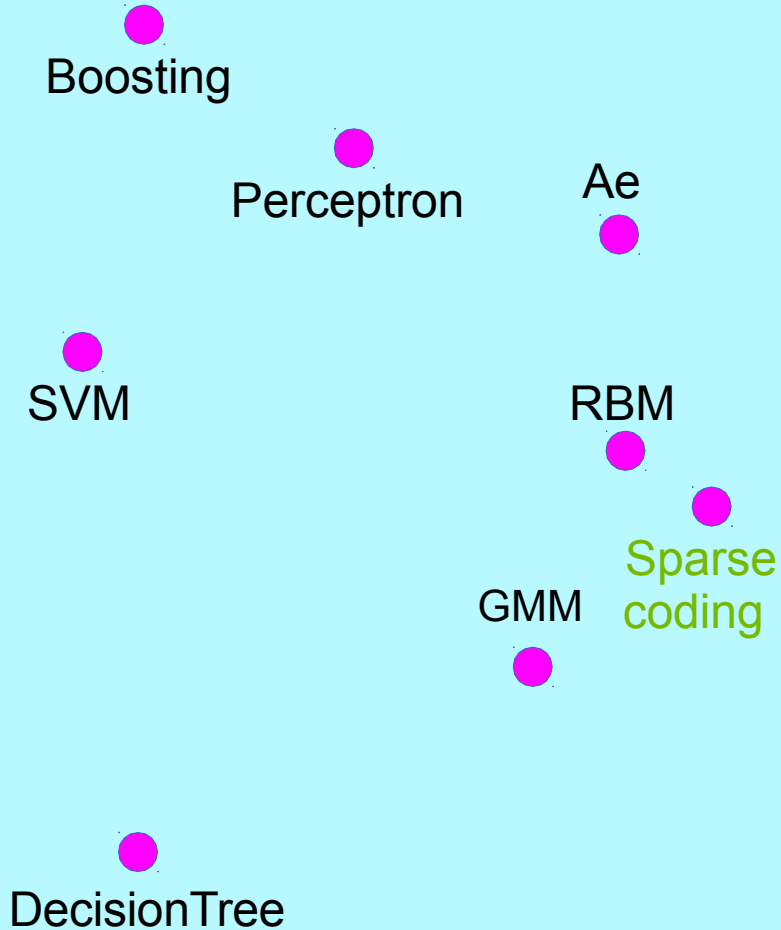
Deep



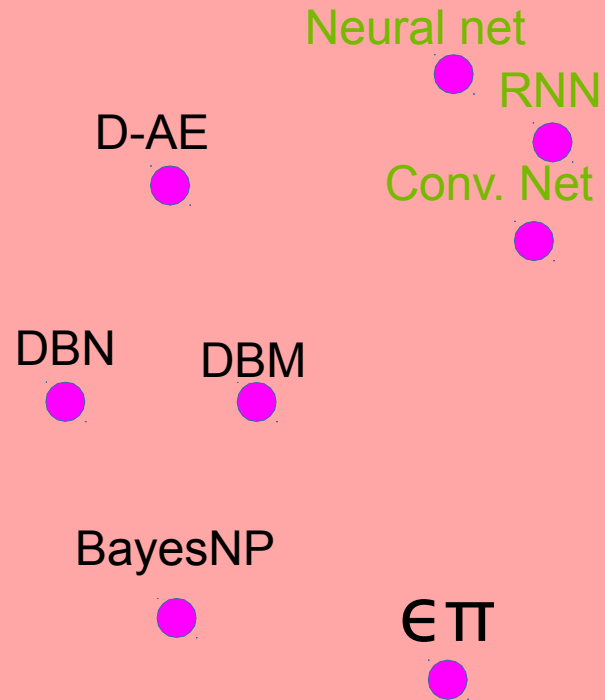
Supervised

Supervised

Shallow



Deep

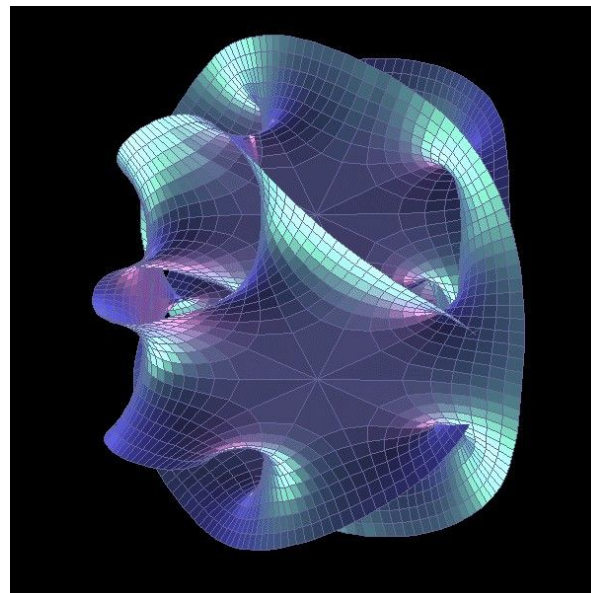


In this talk, we'll focus on **the simplest and typically most effective methods**

What are good features?

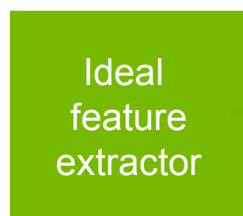
Discovering the hidden structure in high-dimensional data the manifold hypothesis

- Learning representations of data:
 - Discovering & disentangling the independent explanatory factors
- The manifold hypothesis:
 - Natural data lives in a low-dimensional (non-linear) manifold
 - Because variables in natural data are mutually dependent



Discovering the hidden structure in high-dimensional data

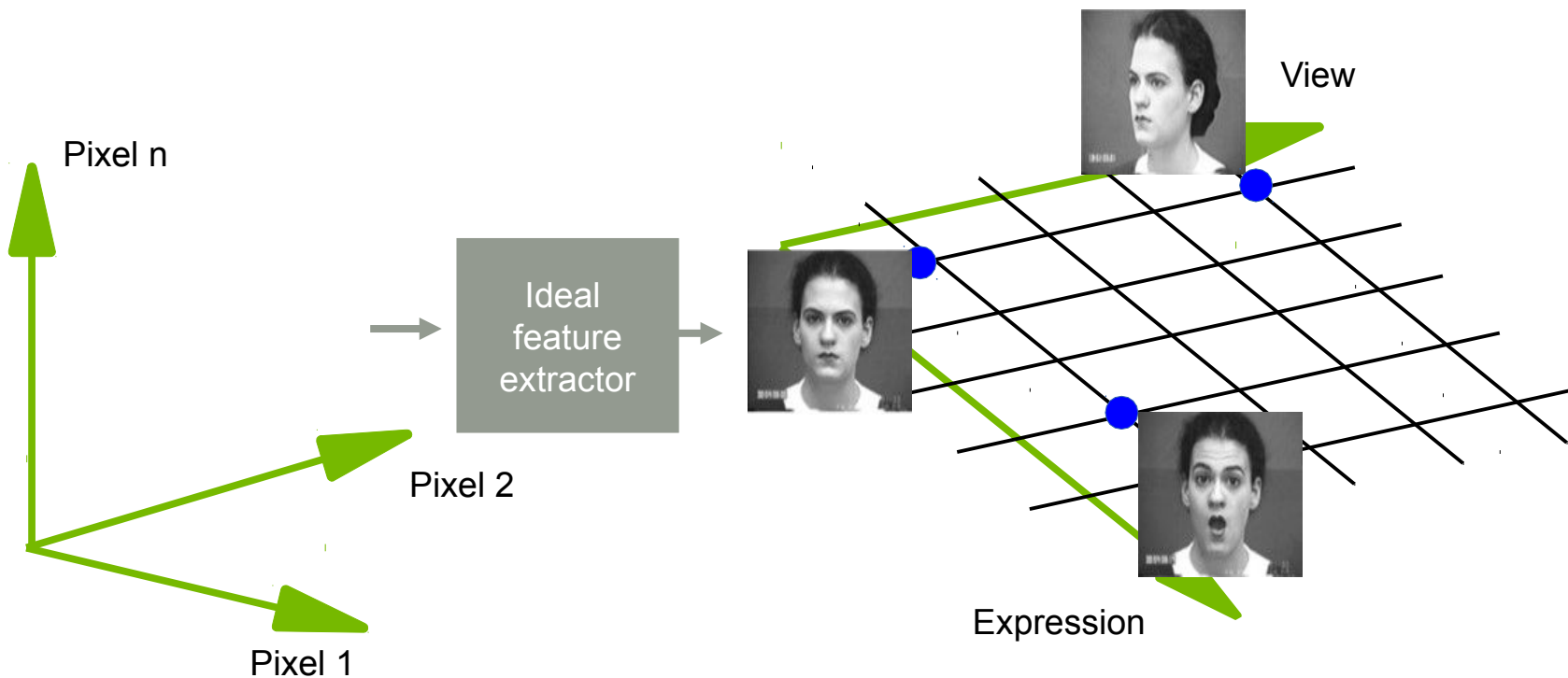
- Example: all face images of a person
 - 1000x1000 pixels = 1,000,000 dimensions
 - But the face has 3 Cartesian coordinates and 3 Euler angles and humans have less than about 50 muscles in the face
 - Hence the manifold of face images for a person has <56 dimensions
- The perfect representations of a face image:
 - Its coordinates on the face manifold
 - Its coordinates away from the manifold
- We do not have good and general methods to learn functions that turns an image into this kind of representation



1.2	Face/not face
-3	Pose
0.2	Lighting
-2...	Expression

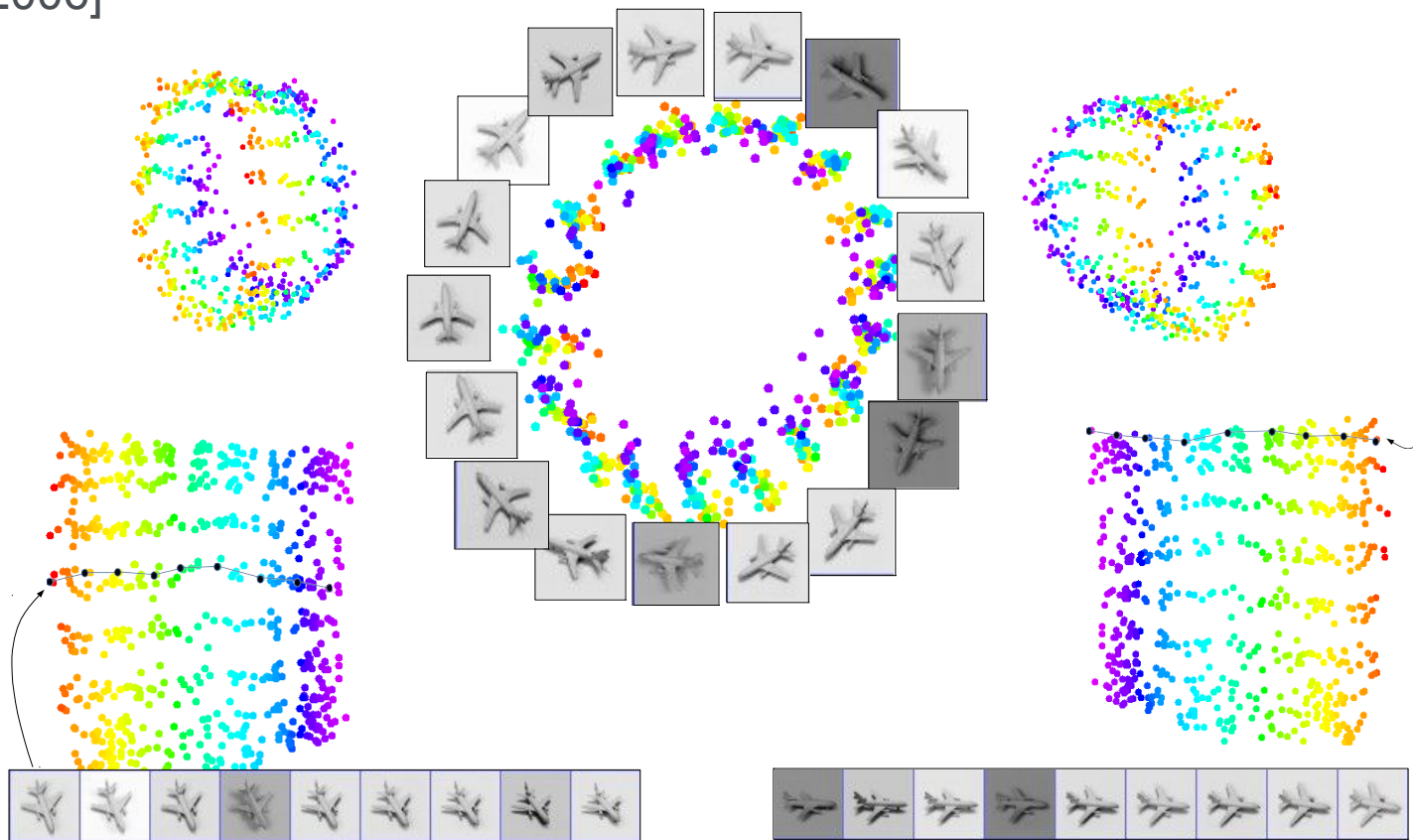
Disentangling factors of variation

The ideal disentangling feature extractor



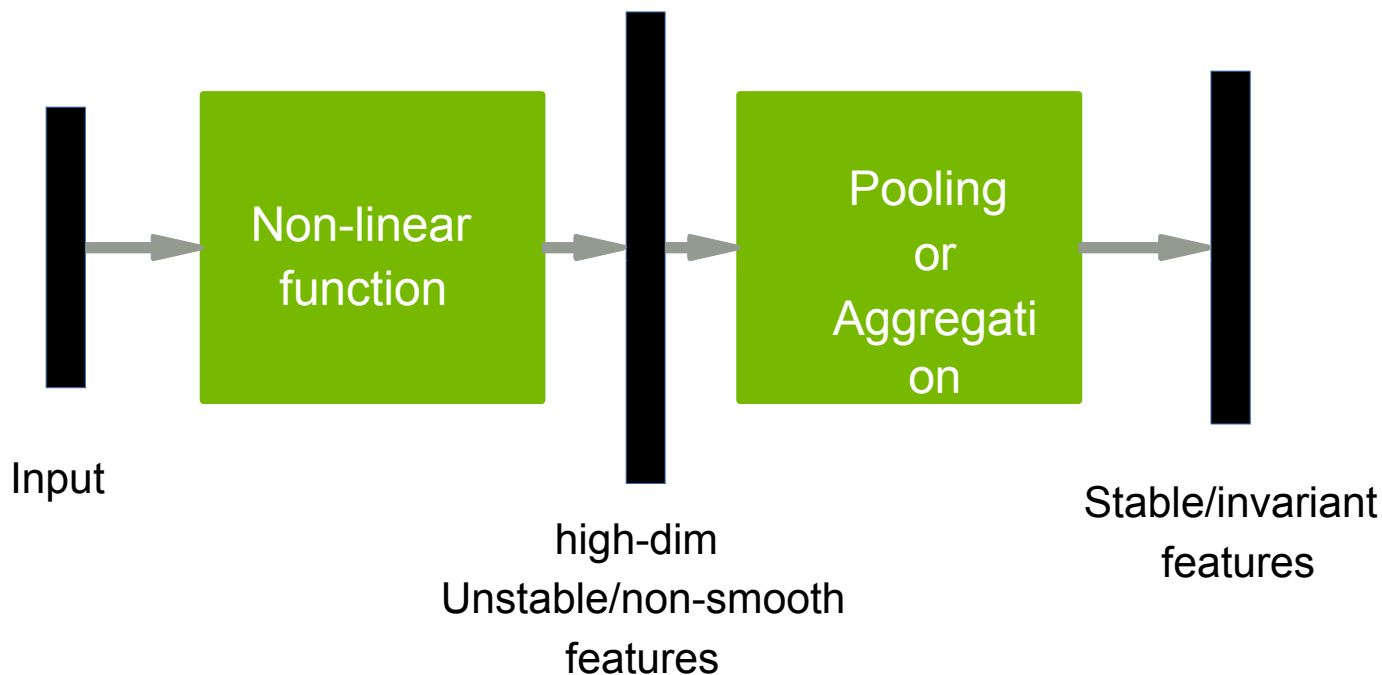
Data manifold & invariance: Some variations must be eliminated

- Azimuth-Elevation manifold. Ignores lighting. [Hadsell et al. CVPR 2006]



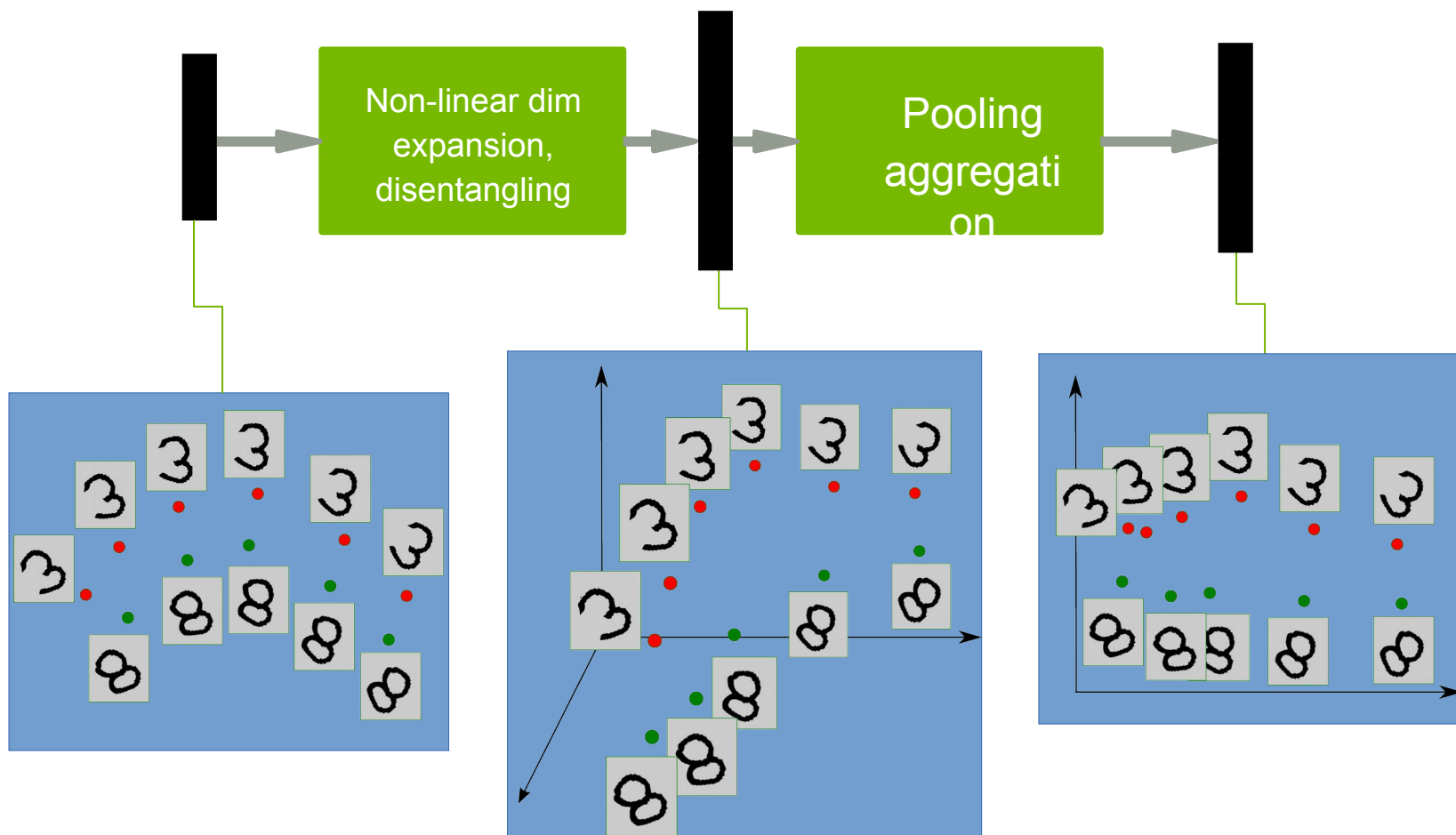
Basic idea for invariant feature learning

- Embed the input **non-linearly** into a high(er) dimensional space
 - In the new space, things that were non separable may become separable
- Pool regions of the new space together
 - Bringing together things that are semantically similar. Like pooling.



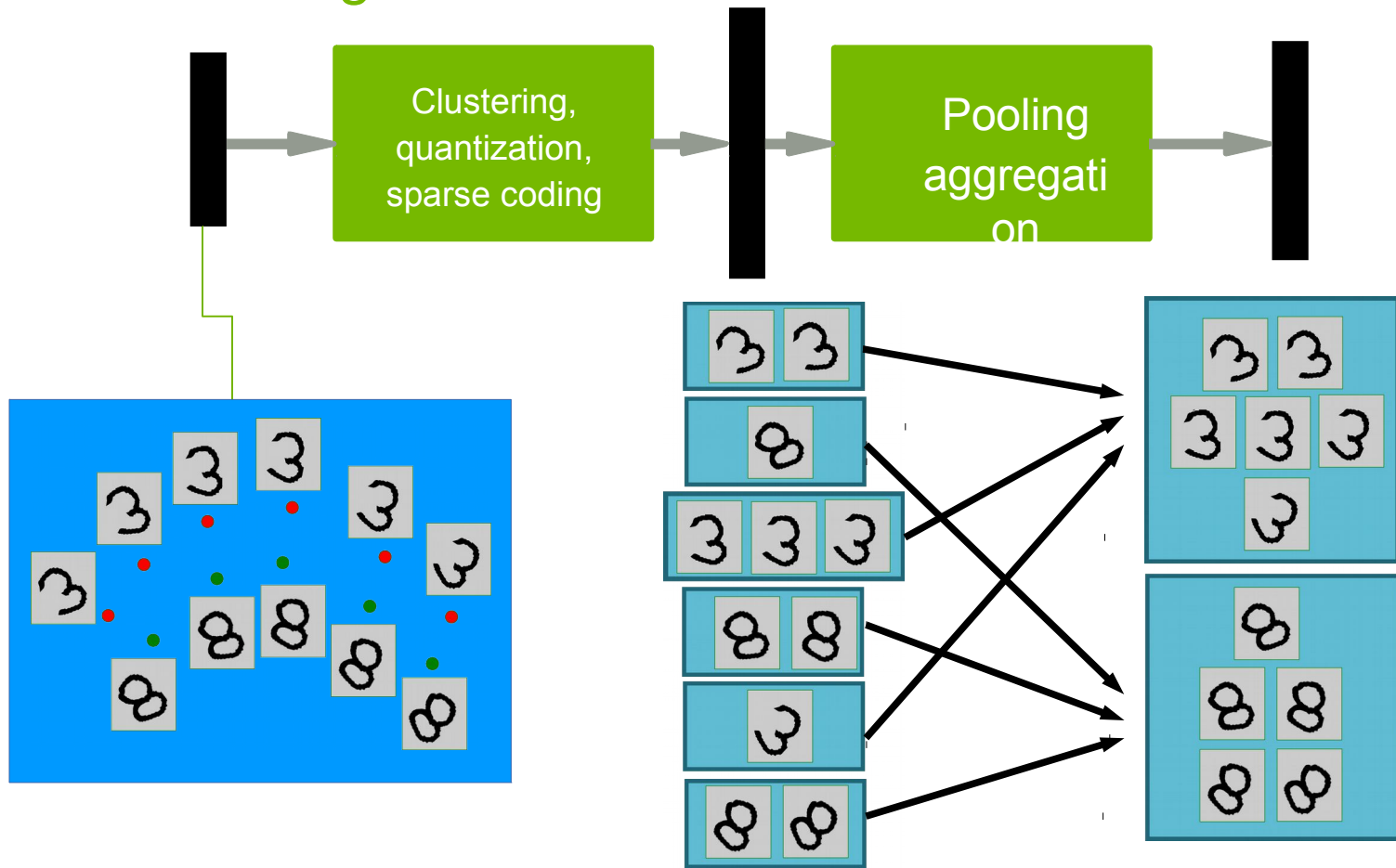
Non-linear expansion → pooling

Entangled data manifolds



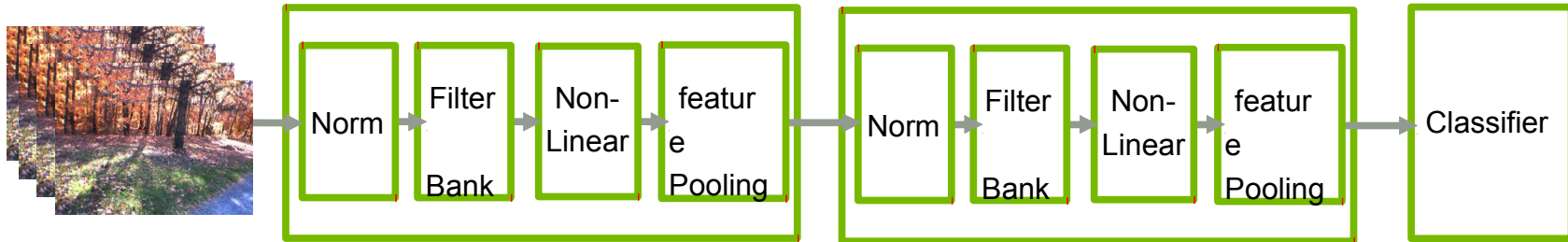
Sparse non-linear expansion → pooling

Use clustering to break things apart, pool together similar things



Overall architecture:

Normalization → filter bank → non-linearity → pooling



- Stacking multiple stages of
 - [Normalization → filter bank → non-linearity → pooling].
- **Normalization**: variations on whitening
 - Subtractive: average removal, high pass filtering
 - Divisive: local contrast normalization, variance normalization
- **Filter bank**: dimension expansion, projection on overcomplete basis
- **Non-linearity**: sparsification, saturation, lateral inhibition....
 - Rectification (relu), component-wise shrinkage, tanh, winner-takes-all
- **Pooling**: aggregation over space or feature type

$$- X_i; L_p: \sqrt[p]{X_i^p}; PROB: \frac{1}{b} \log \left(\sum_i e^{bX_i} \right)$$

Software

- Torch7: learning library that supports neural net training
 - <http://www.torch.ch>
 - <http://code.cogbits.com/wiki/doku.php> (tutorial with demos by C. Farabet)
 - <http://elearn.sf.net> (C++ Library with convent support by P. Sermanet)
- Python-based learning library (U. Montreal)
 - <http://deeplearning.net/software/theano/> (does automatic differentiation)
- RNN
 - www.fit.vutbr.cz/~imikolov/rnnlm (language modeling)
 - <http://sourceforge.net/apps/mediawiki/index.php> (LSTM)
- CUDAMat & GNumPy
 - code.google.com/p/cudamat
 - www.cs.toronto.edu/~tijmen/gnumpy.htm
- Misc
 - www.deeplearning.net//software_links

References

Convolutional nets

- LeCun, Bottou, Bengio and Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998
- Krizhevsky, Sutskever, Hinton “ImageNet Classification with deep convolutional neural networks” NIPS 2012
- Jarrett, Kavukcuoglu, Ranzato, LeCun: What is the Best Multi-Stage Architecture for
Object Recognition?, Proc. International Conference on Computer Vision (ICCV'09), IEEE, 2009
- Kavukcuoglu, Sermanet, Boureau, Gregor, Mathieu, LeCun: Learning Convolutional Feature Hierarchies for Visual Recognition, Advances in Neural Information Processing Systems (NIPS 2010), 23, 2010
- see yann.lecun.com/exdb/publis for references on many different kinds of convnets.
- see <http://www.cmap.polytechnique.fr/scattering/> for scattering networks (similar to convnets but with less learning and stronger mathematical foundations)

References

Applications of RNNs

- Mikolov “Statistical language models based on neural networks” PhD thesis 2012
- Boden “A guide to RNNs and backpropagation” Tech Report 2002
- Hochreiter, Schmidhuber “Long short term memory” Neural Computation 1997
- Graves “Offline arabic handwriting recognition with multidimensional neural networks” Springer 2012
- Graves “Speech recognition with deep recurrent neural networks” ICASSP 2013

References

Applications of convolutional nets

- Farabet, Couprie, Najman, LeCun, “Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers”, ICML 2012
- Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala and Yann LeCun: Pedestrian Detection with Unsupervised Multi-Stage Feature Learning, CVPR 2013
- D. Cirestan, A. Giusti, L. Gambardella, J. Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. NIPS 2012
- Raia Hadsell, Pierre Sermanet, Marco Scoffier, Ayse Erkan, Koray Kavackuoglu, Urs Muller and Yann LeCun: Learning Long-Range Vision for Autonomous Off-Road Driving, Journal of Field Robotics, 26(2):120-144, February 2009
- Burger, Schuler, Harmeling: Image Denoising: Can Plain Neural Networks Compete with BM3D?, Computer Vision and Pattern Recognition, CVPR 2012,

References

Deep learning & energy-based models

– Deep learning & energy-based models

- Y. Bengio, Learning Deep Architectures for AI, Foundations and Trends in Machine Learning, 2(1), pp.1-127, 2009.
- LeCun, Chopra, Hadsell, Ranzato, Huang: A Tutorial on Energy-Based Learning, in Bakir, G. and Hofman, T. and Schölkopf, B. and Smola, A. and Taskar, B. (Eds), Predicting Structured Data, MIT Press, 2006
- M. Ranzato Ph.D. Thesis “Unsupervised Learning of Feature Hierarchies” NYU 2009

– Practical guide

- Y. LeCun et al. Efficient BackProp, Neural Networks: Tricks of the Trade, 1998
- L. Bottou, Stochastic gradient descent tricks, Neural Networks, Tricks of the Trade Reloaded, LNCS 2012.
- Y. Bengio, Practical recommendations for gradient-based training of deep architectures, ArXiv 2012



NEW YORK UNIVERSITY

Deep Learning Teaching Kit

Thank you

DEEP LEARNING METHODS

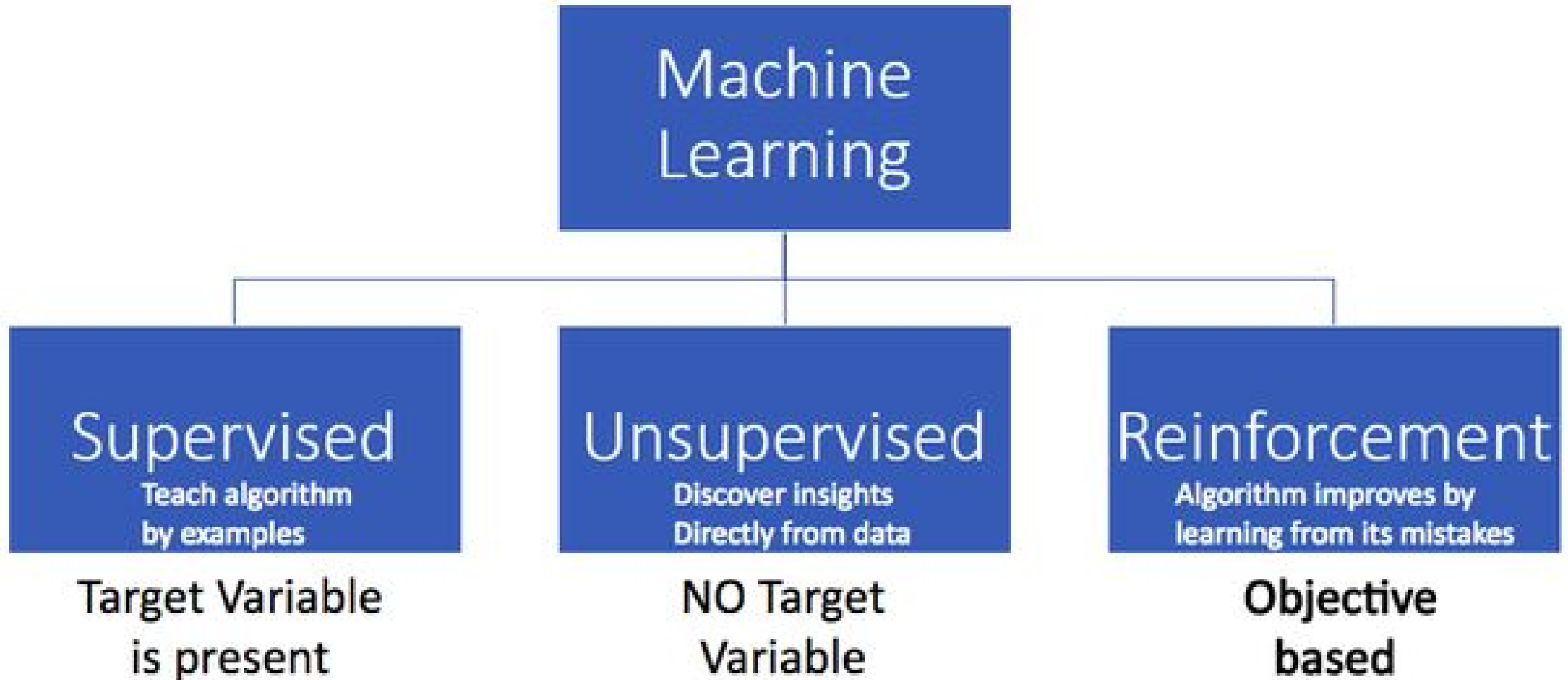
LECTURE 4: CATEGORIES, TYPES, ORIGIN, DEVELOPMENT

Yuri Gordienko, DLI Certified Instructor



TYPES OF LEARNING ALGORITHMS

Types of Learning Algorithms



Types of Learning Algorithms

Supervised

Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

Unsupervised

Discover Insights
Directly from data

NO Target
Variable

Unsupervised Learning — Neural Networks (NN):

- Autoencoder
- Generative Adversarial Networks

Reinforcement

Algorithm improves by
learning from its mistakes

Objective
based

Reinforcement Learning

- Networks for Learning Actions, Values, and Policies

Types of Learning Algorithms

Supervised

Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

Unsupervised

Discover Insights
Directly from data

NO Target
Variable

Unsupervised Learning — Neural Networks (NN):

- Autoencoder
- Generative Adversarial Networks

Reinforcement

Algorithm improves by
learning from its mistakes

Objective
based

Reinforcement Learning

- Networks for Learning Actions, Values, and Policies

Types of Learning Algorithms

Supervised

Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

Unsupervised

Discover Insights
Directly from data

NO Target
Variable

Unsupervised Learning — Neural Networks (NN):

- Autoencoder
- Generative Adversarial Networks

Reinforcement

Algorithm improves by
learning from its mistakes

Objective
based

Reinforcement Learning

- Networks for Learning Actions, Values, and Policies

Types of Learning Algorithms

Supervised

Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

Unsupervised

Discover Insights
Directly from data

NO Target
Variable

Unsupervised Learning — Neural Networks (NN):

- Autoencoder
- Generative Adversarial Networks

Reinforcement

Algorithm improves by
learning from its mistakes

Objective
based

Reinforcement Learning

- Networks for Learning Actions, Values, and Policies

TYPES OF LEARNING ALGORITHMS
-
NEURAL NETWORK ARCHITECTURES
-
SUPERVISED

Types of Learning Algorithms

Supervised
Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

1. Feed Forward Neural Networks

Input:

A few
numbers

Network:

Dense
Encoder

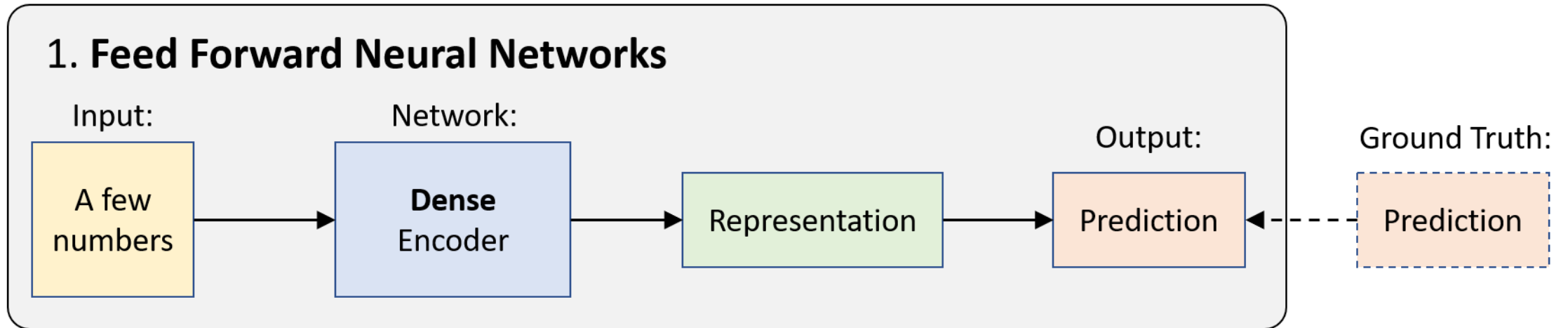
Representation

Output:

Prediction

Ground Truth:

Prediction



Types of Learning Algorithms

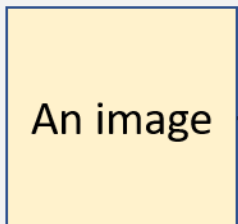
Supervised
Teach algorithm
by examples
Target Variable
is present

Supervised Learning — Neural Networks (NN):

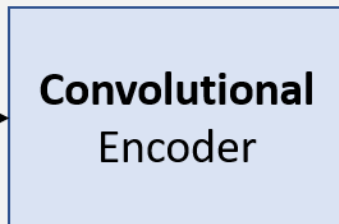
- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
 - Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

2. Convolutional Neural Networks

Input:



Network:



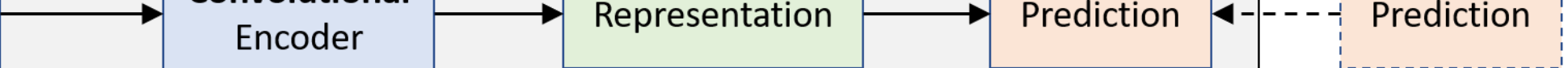
Representation

Output:

Prediction

Ground Truth:

Prediction



Types of Learning Algorithms

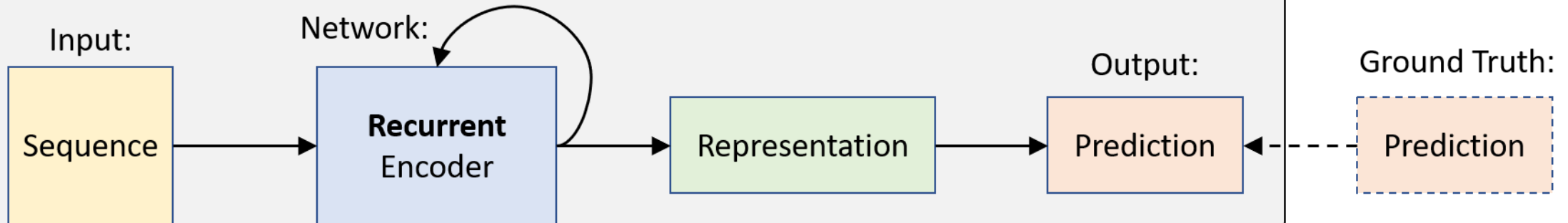
Supervised
Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

3. Recurrent Neural Networks



Types of Learning Algorithms

Supervised
Teach algorithm
by examples

Target Variable
is present

Supervised Learning — Neural Networks (NN):

- Feed-Forward NN (FNN)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Encoder-Decoder Architectures (EDA)

4. Encoder-Decoder Architectures

Input:

Image,
Text,
etc.

Network:

Any
Encoder

Representation

Network:

Any
Decoder

Output:

Image,
Text,
etc.

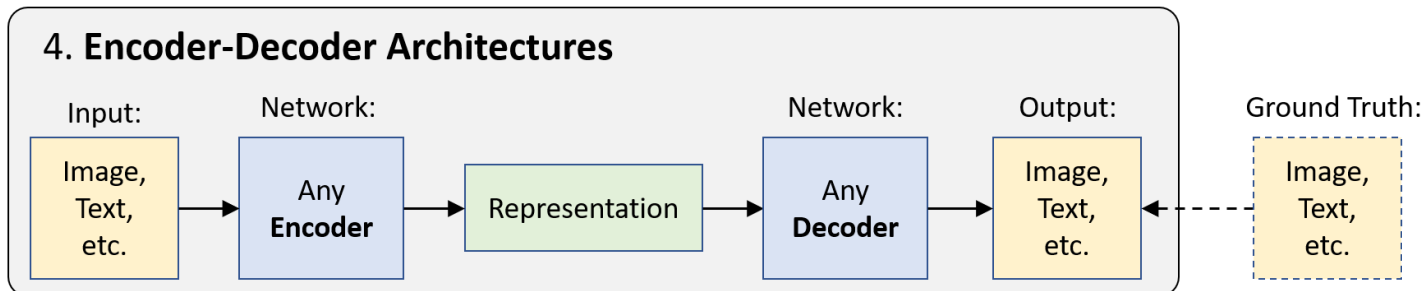
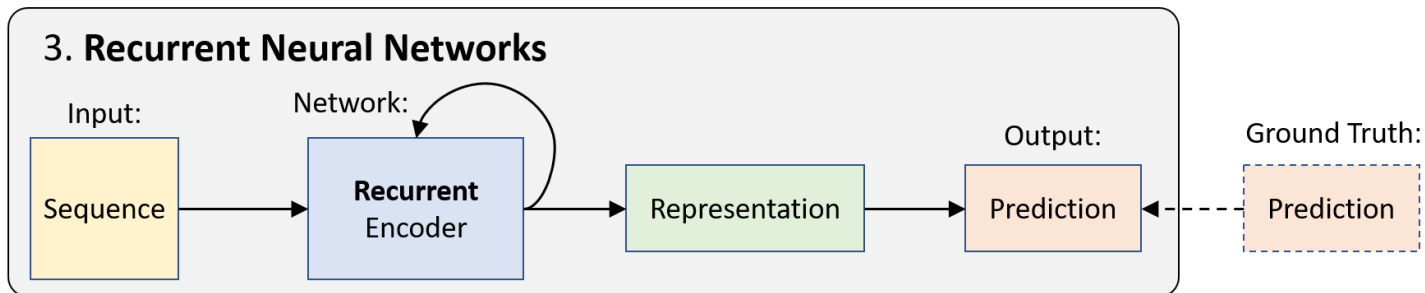
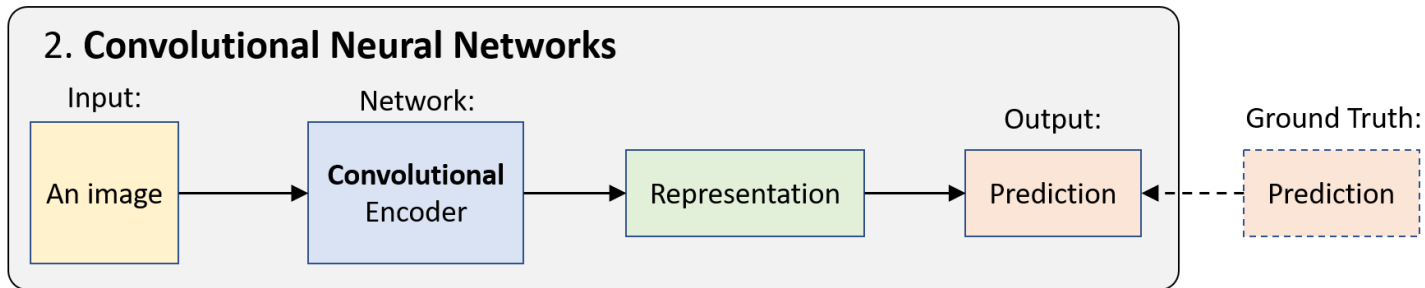
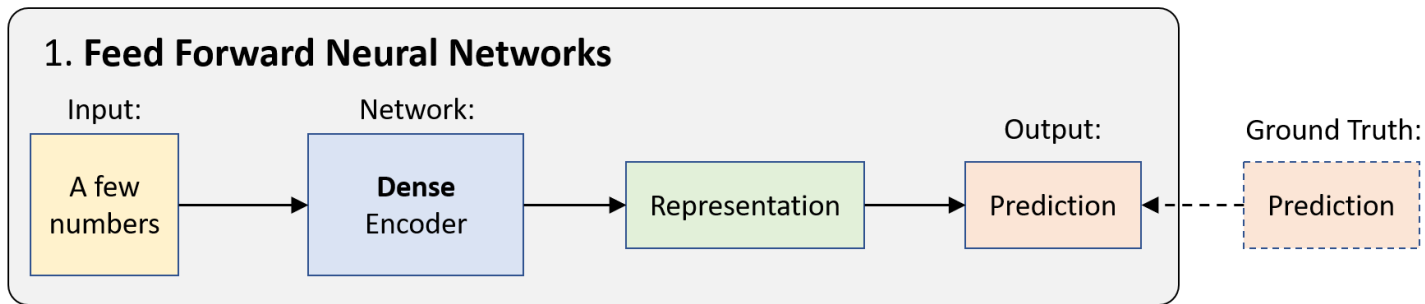
Ground Truth:

Image,
Text,
etc.



Supervised
Teach algorithm
by examples

Target Variable
is present



TYPES OF LEARNING ALGORITHMS
-
NEURAL NETWORK ARCHITECTURES
-
UNSUPERVISED

Types of Learning Algorithms

Unsupervised

Discover Insights
Directly from data

NO Target
Variable

Unsupervised Learning — Neural Networks (NN):

- Autoencoder
- Generative Adversarial Networks

5. Autoencoder

Input:

Image,
Text,
etc.

Network:

Any
Encoder

Representation

Throw away after training

Network:

Any
Decoder

Ground Truth:

Exact copy
of input

Types of Learning Algorithms

Unsupervised

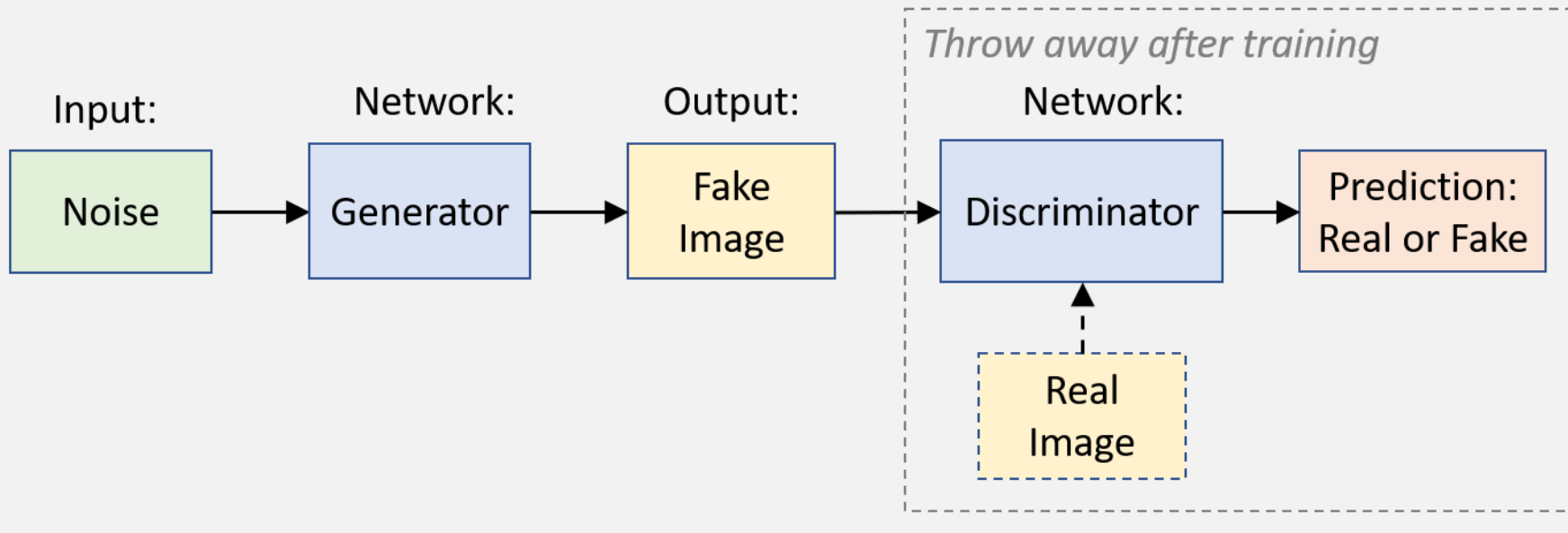
Discover Insights
Directly from data

NO Target
Variable

Unsupervised Learning — Neural Networks (NN):

- Autoencoder
- Generative Adversarial Networks

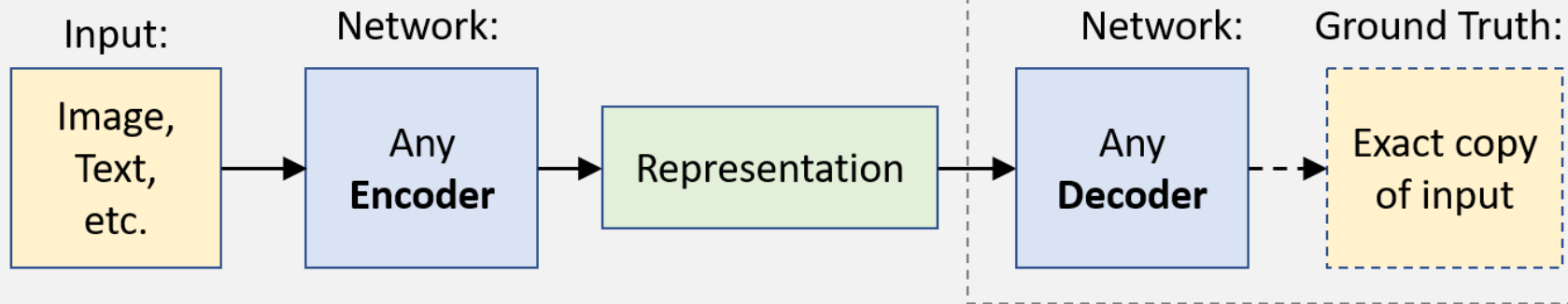
6. Generative Adversarial Networks



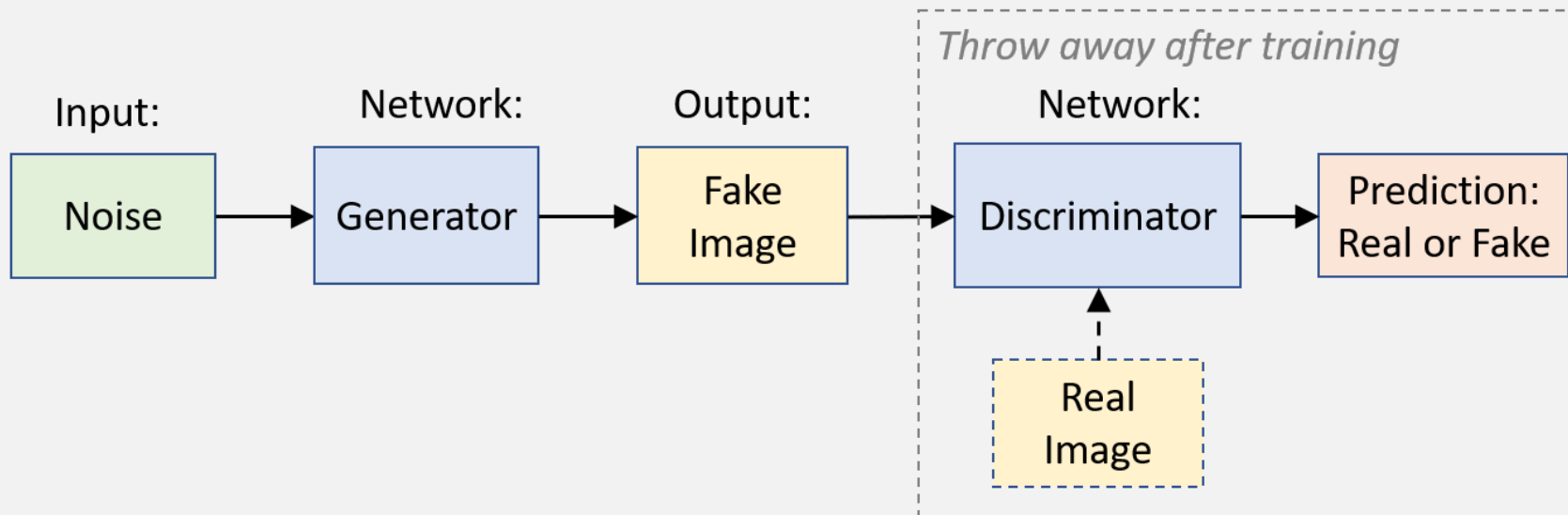
Unsupervised
Discover insights
Directly from data

NO Target
Variable

5. Autoencoder



6. Generative Adversarial Networks



TYPES OF LEARNING ALGORITHMS
-
NEURAL NETWORK ARCHITECTURES
-
REINFORCEMENT

Types of Learning Algorithms

Reinforcement Learning

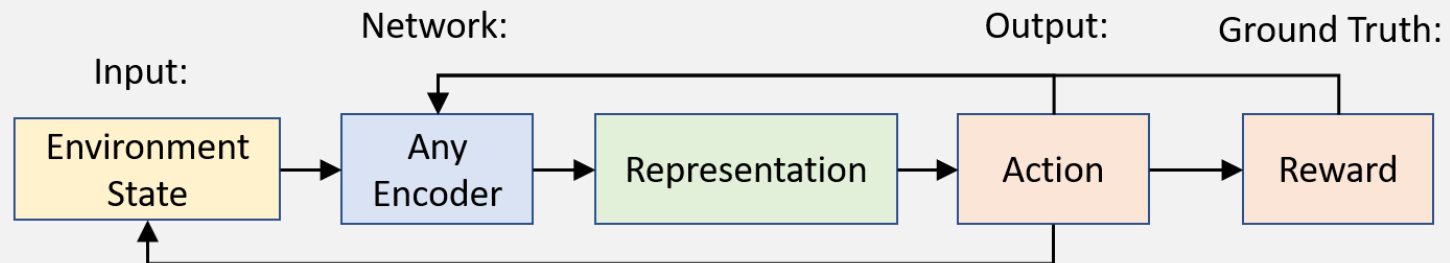
Reinforcement

Algorithm improves by
learning from its mistakes

Objective
based

- Networks for Learning Actions, Values, and Policies

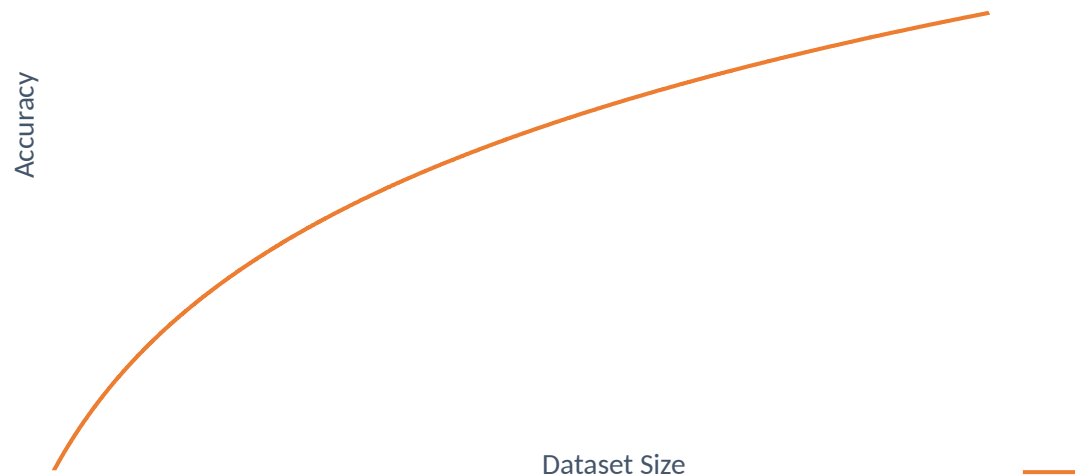
7. Networks for Learning Actions, Values, and Policies



HOW THEY APPEARED
-
MOTIVATION

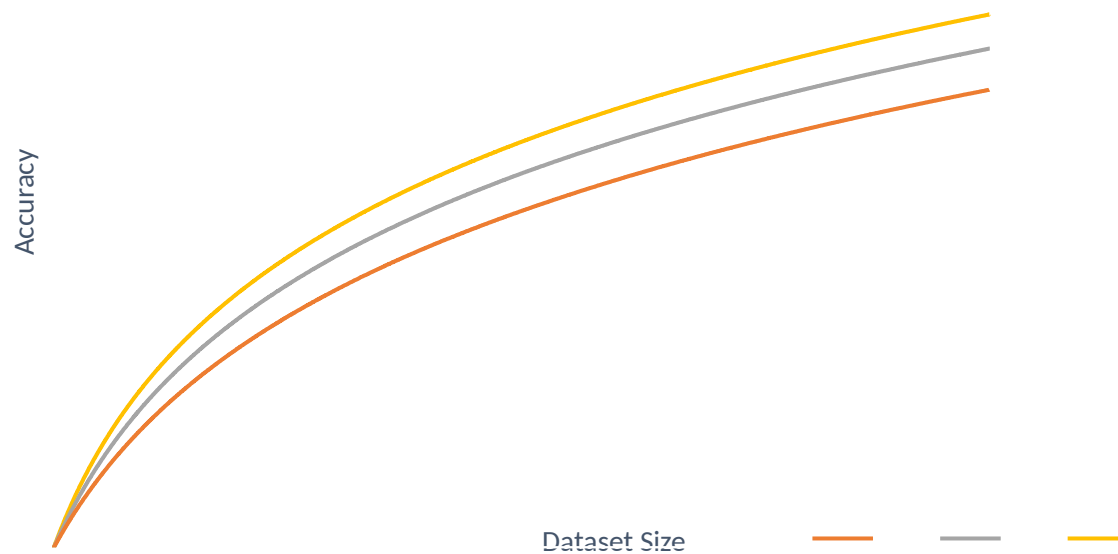
NEURAL NETWORKS ARE NOT NEW

They just historically never worked well



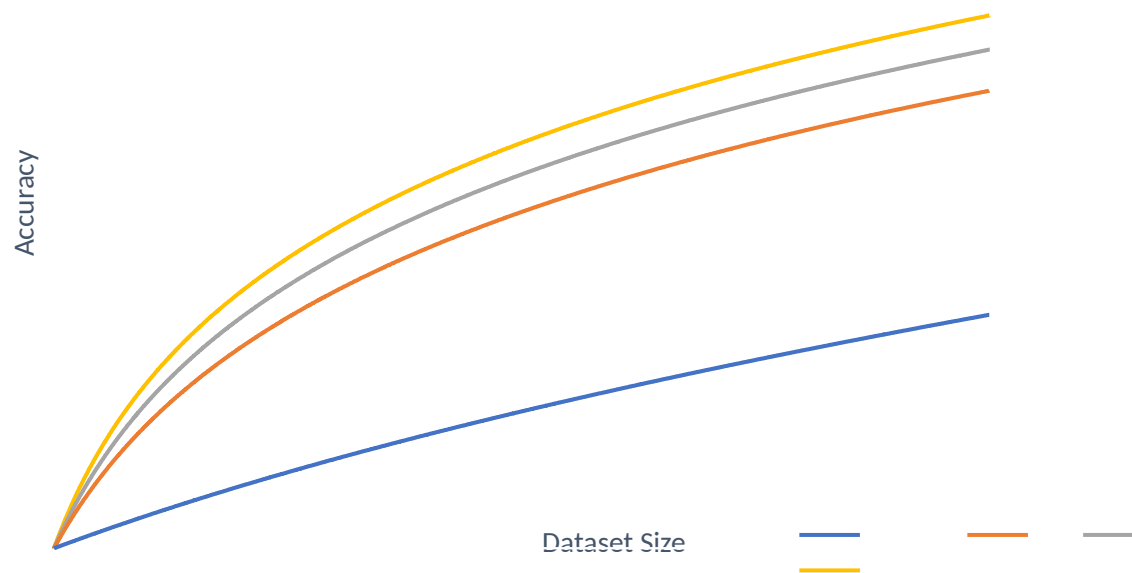
NEURAL NETWORKS ARE NOT NEW

They just historically never worked well



NEURAL NETWORKS ARE NOT NEW

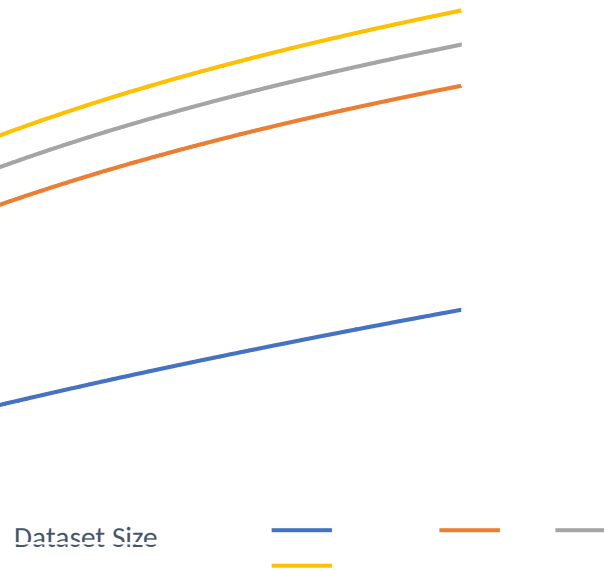
They just historically never worked well



NEURAL NETWORKS ARE NOT NEW

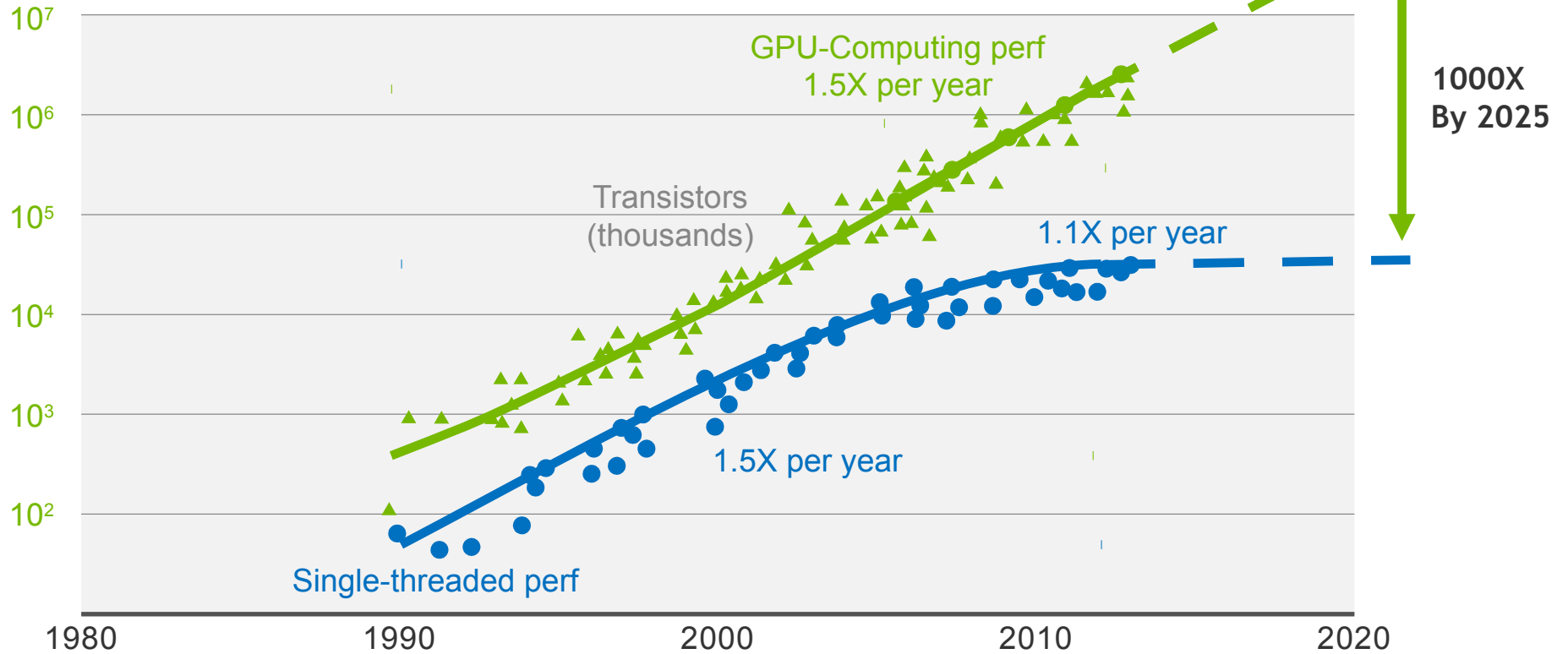
Historically we never had large datasets or computers

The MNIST (1999) database contains 60,000 training images and 10,000 testing images.



COMPUTE

Historically we never had large datasets or compute



AI BIG BANG PILLARS:

- HIGH-PERFORMANCE COMPUTING**
- BIG DATA**
- DEEP MODELS**

AI BIG BANG PILLARS:

- HIGH-PERFORMANCE COMPUTING
- BIG DATA
- DEEP MODELS

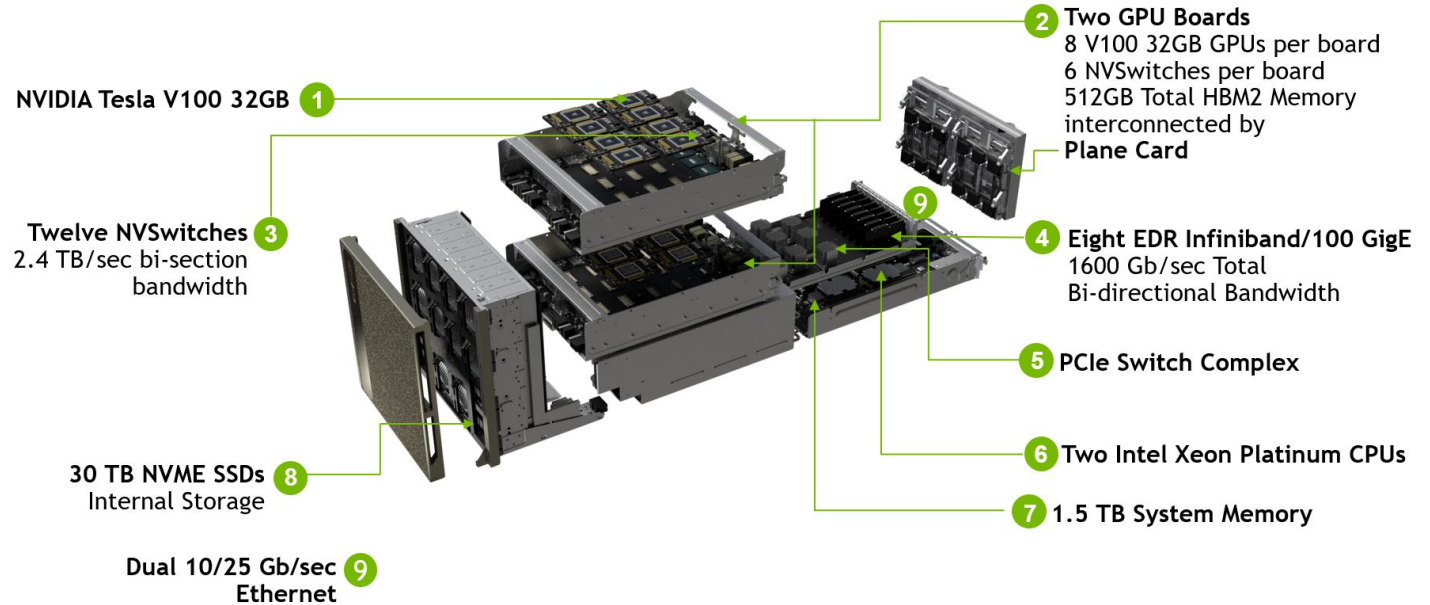
CONTEXT

1.759 petaFLOPs in November 2009



CONTEXT

2 petaFLOPs - today



100 EXAFLOPS

=

2 YEARS ON A DUAL CPU SERVER

AI BIG BANG PILLARS:

- HIGH-PERFORMANCE COMPUTING
 - **BIG DATA**
 - DEEP MODELS

EXPLODING DATASETS

Logarithmic relationship between the dataset size and accuracy

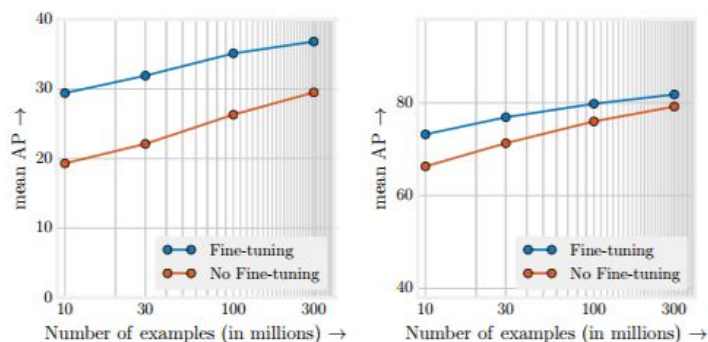


Figure 4. Object detection performance when initial checkpoints are pre-trained on different subsets of JFT-300M from scratch. x-axis is the data size in log-scale, y-axis is the detection performance in mAP@[.5,.95] on COCO minival* (left), and in mAP@.5 on PASCAL VOC 2007 test (right).

Initialization	mIOU
ImageNet	73.6
300M	75.3
ImageNet+300M	76.5

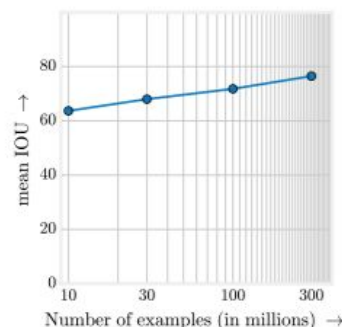
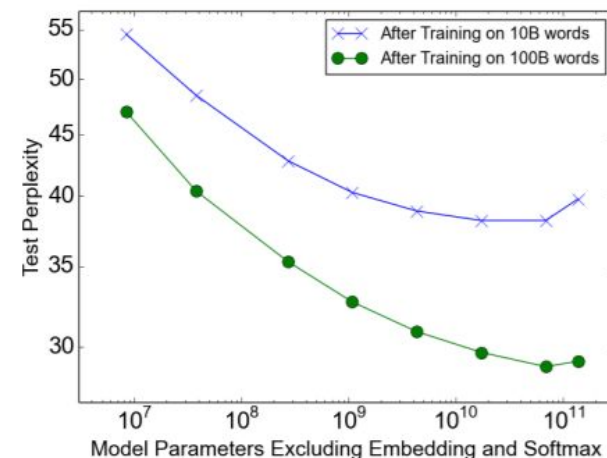
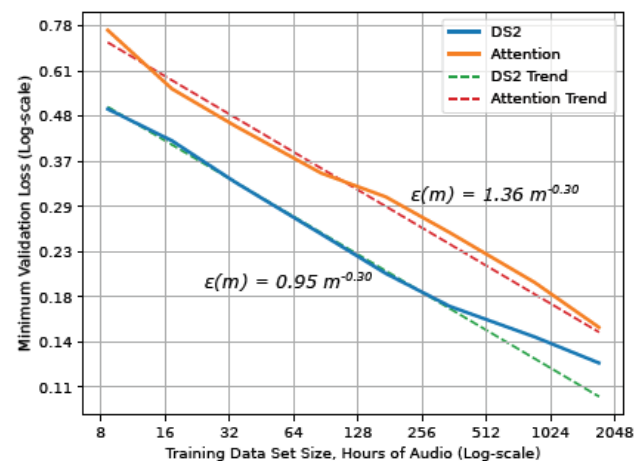
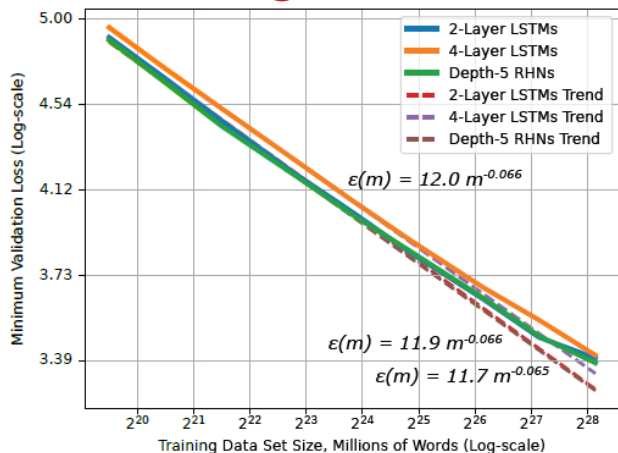


Figure 6. Semantic segmentation performance on Pascal VOC 2012 val set. (left) Quantitative performance of different initializations; (right) Impact of data size on performance.

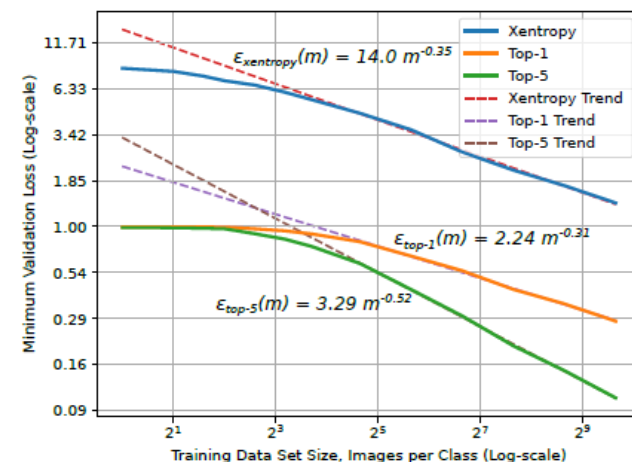
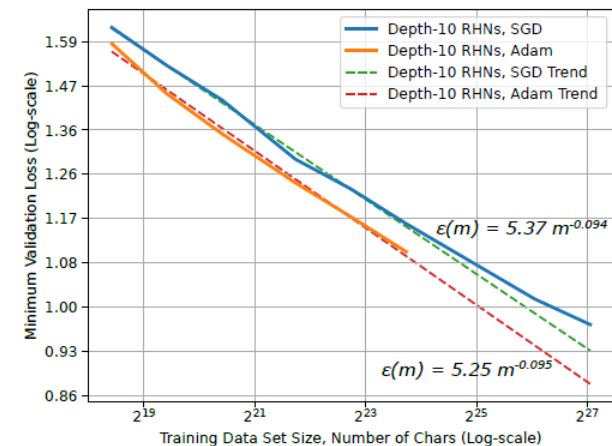
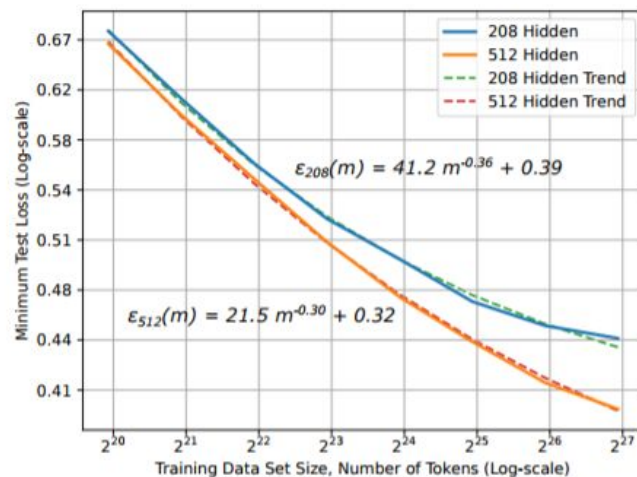


EXPLODING DATASETS

Logarithmic relationship between the dataset size and accuracy

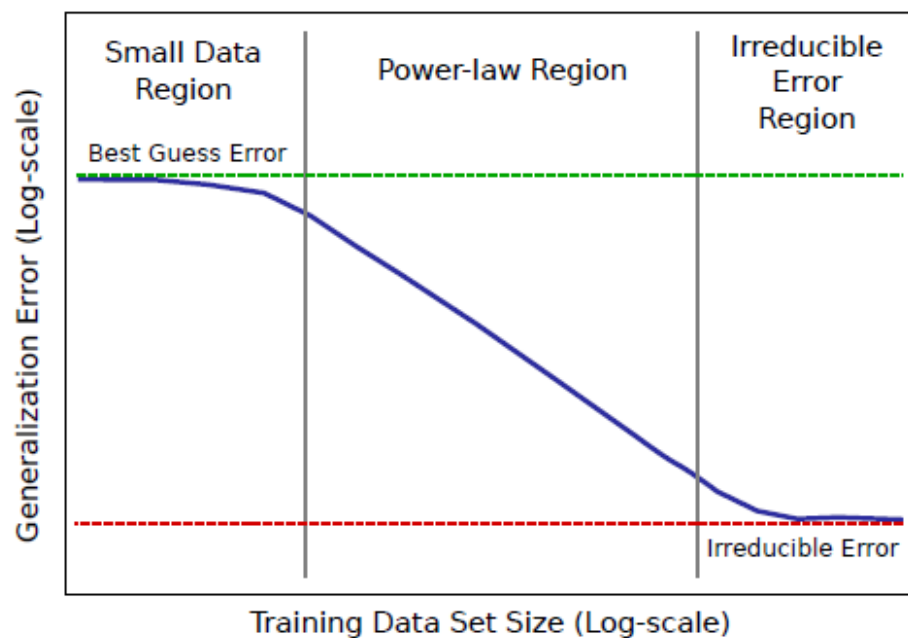


- Translation
- Language Models
- Character Language Models
- Image Classification
- Attention Speech Models



EXPLODING DATASETS

Logarithmic relationship between the dataset size and accuracy



AI BIG BANG PILLARS:

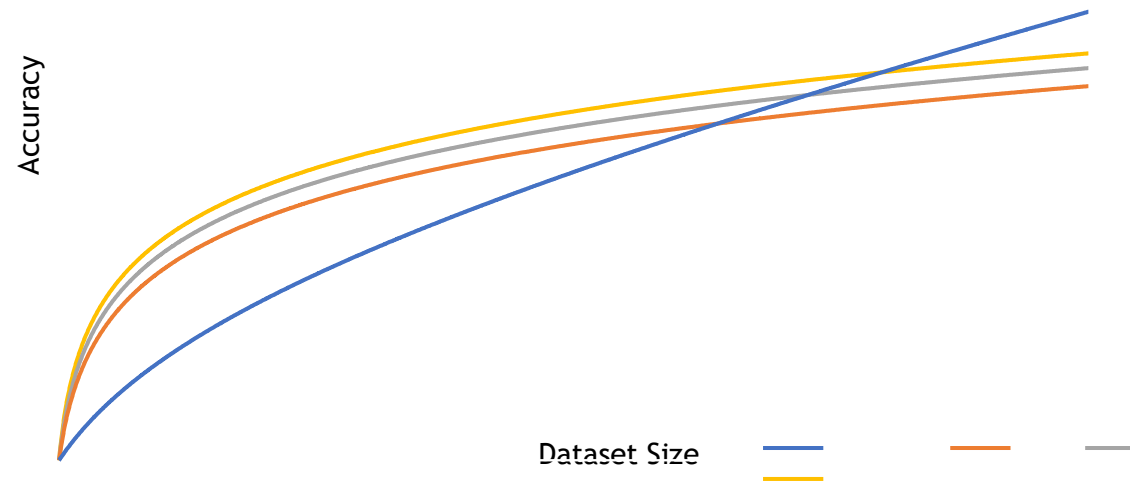
- HIGH-PERFORMANCE COMPUTING**
- BIG DATA**
- DEEP MODELS**

To Tackle Increasingly Complex Challenges



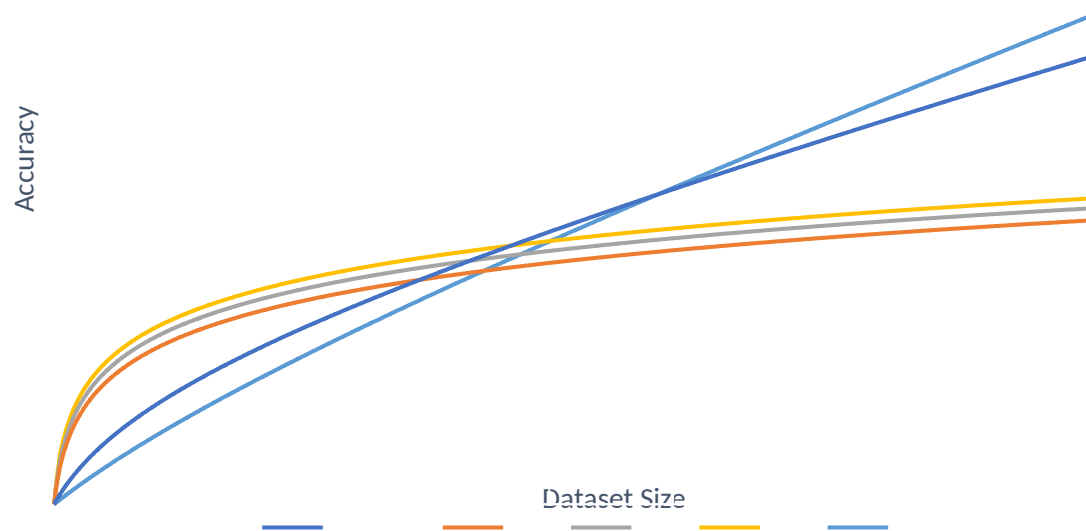
NEURAL NETWORKS ARE NOT NEW

But that changed and transformed the way we do machine learning



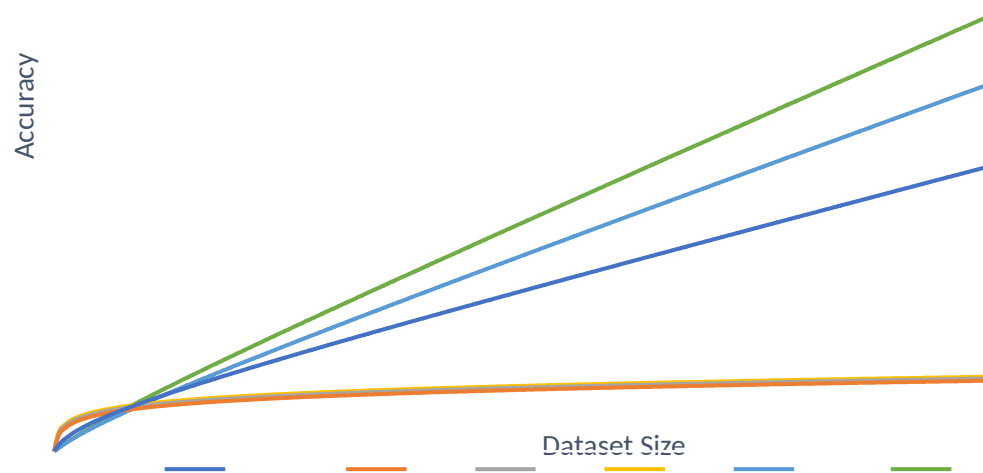
NEURAL NETWORKS ARE NOT NEW

Data and model size the key to accuracy

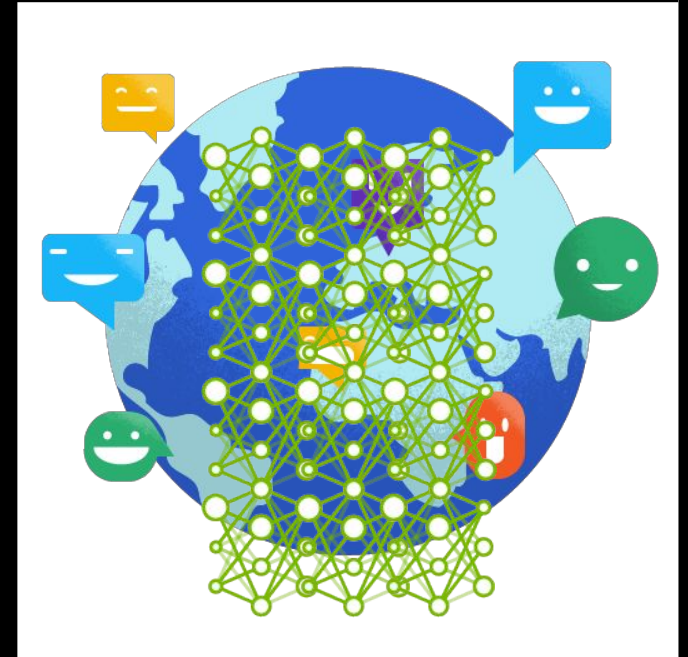


NEURAL NETWORKS ARE NOT NEW

Exceeding human level performance

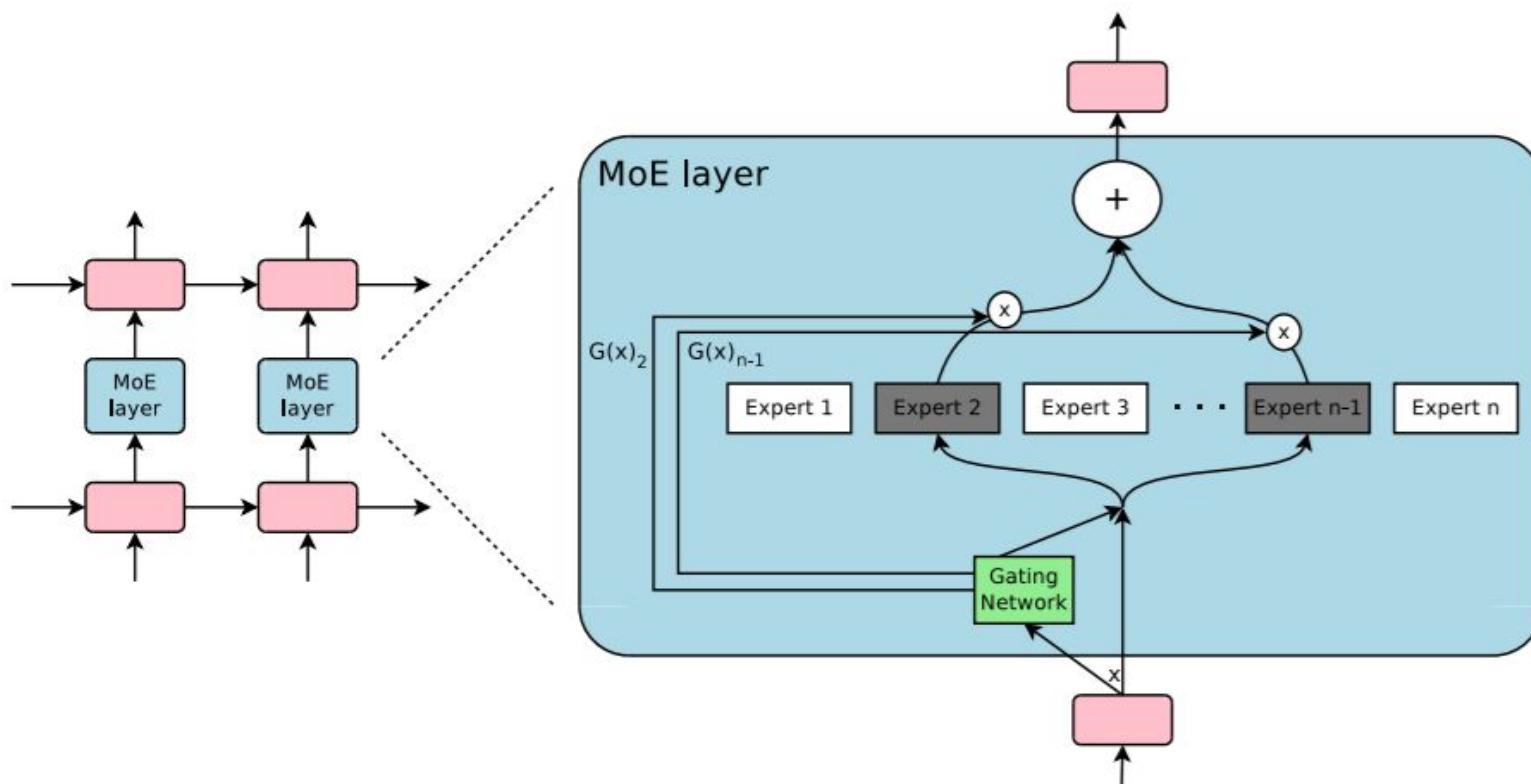


To Tackle Increasingly Complex Challenges



EXPLODING MODEL COMPLEXITY

Larger models are made possible



Shazeer, Noam, et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer." arXiv preprint arXiv:1701.06538 (2017).

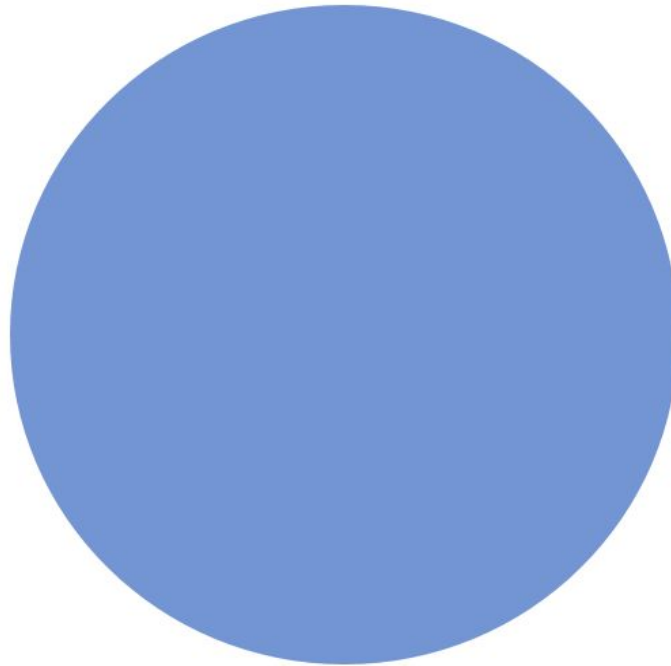
Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

Nuts and Bolts of Applying Deep Learning, Andrew Ng, 2016 - <https://youtu.be/F1ka6a13S9I>

EXPLODING MODEL COMPLEXITY

„Outrageously large neural networks“ - size does matter

VGG 19 vs Google LSTM using Sparsely-Gated Mixture-of-Experts layer



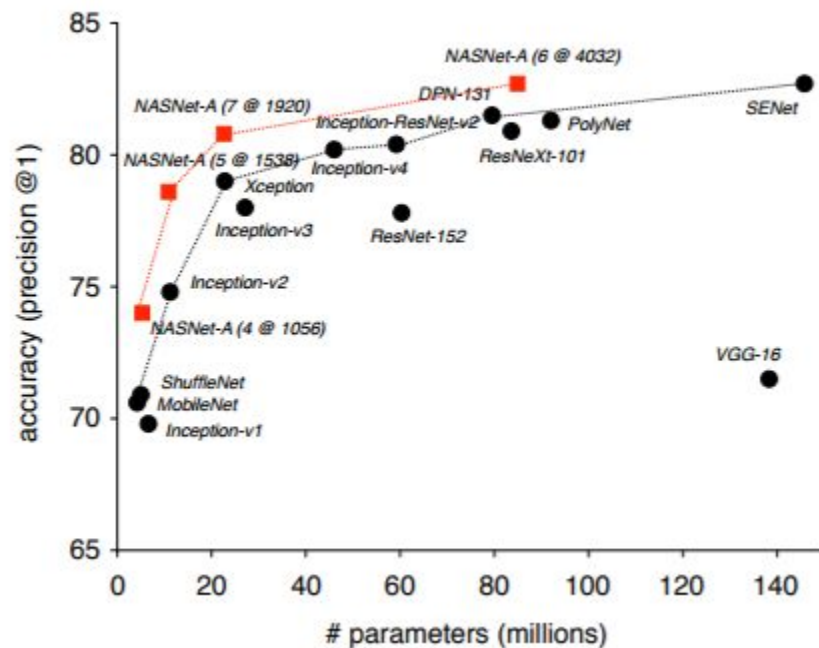
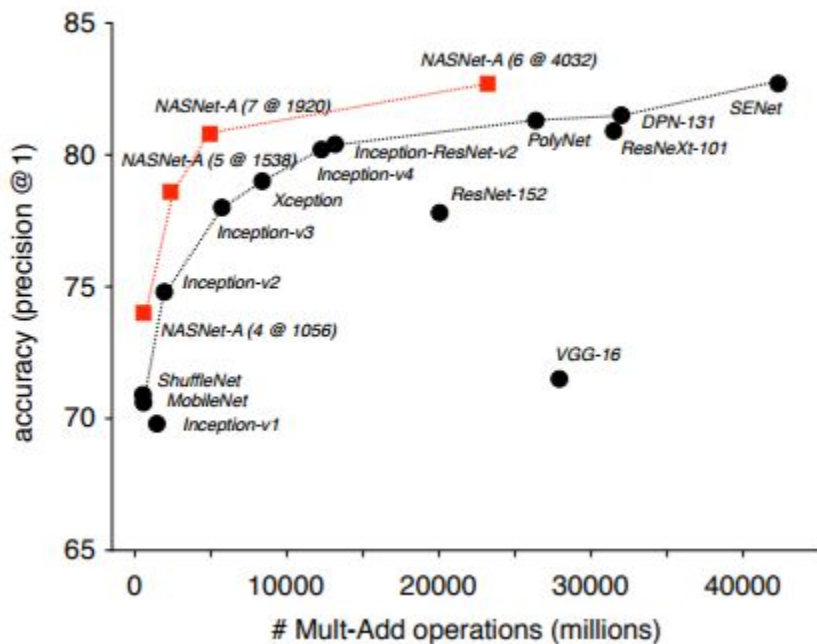
EXPLODING MODEL COMPLEXITY

Good news - model size scales sublinearly



EVIDENCE FROM IMAGE PROCESSING

Good news - model size scales sublinearly



IMPLICATIONS

Making complex problems easy

Making unsolvable problems
expensive

“For any size of the data it’s a good idea to always make the data look small by using a huge model.”

Geoffrey Hinton

FUNDAMENTAL CHANGE TO THE ECONOMY

Impact

BBC

Menu

NEWS | PIDGIN

Home | Nigeria | Africa | World | Video | Audio | Sport | Entertainment

UAE: First minister of artificial intelligence don land

19 October 2017



EUROPEAN COMMISSION

Press Release Database

European Commission > Press releases database > Press Release details

European Commission - Press release

Commission proposes to invest EUR 1 billion in world-class European supercomputers

Brussels, 11 January 2018

Andrus Anspiv, European Commission Vice-President for the Digital Single Market, said: "Supercomputers are the engine to power the digital economy. It is a tough race and today the EU is lagging behind: we do not have any supercomputers in the world's top-ten. With the EuroHPC initiative we want to give European researchers and companies world-leading supercomputer capacity by 2020 - to develop technologies such as artificial intelligence and build the future's everyday applications in areas like health, security or engineering."

China's Got a Huge Artificial Intelligence Plan

Bloomberg News

July 21, 2017, 4:04 AM GMT+1 Updated on July 21, 2017, 8:12 AM GMT+1

- Priorities are intelligent robotics, vehicles, virtual reality
- AI seen contributing up to \$15.7 trillion worldwide by 2030

Microsoft just officially listed AI as one of its top priorities, replacing mobile

- Satya Nadella's "mobile-first and cloud-first world" line is out.
- The change comes after Microsoft formed the Artificial Intelligence and Research group.

Jordan Novet | @jordannovet

Published 5:48 PM ET Wed, 2 Aug 2017 | Updated 7:00 PM ET Fri, 4 Aug 2017



INNOVATION

The 10 tech companies that have invested the most money in AI

Of the tech giants, Google is the biggest investor in AI by billions.

By Olivia Krauth | January 12, 2018, 1:12 PM PST

- | | |
|------------------------------|---------------------------------|
| 1. Google - \$3.9 billion | 6. Uber - \$680 million |
| 2. Amazon - \$871 million | 7. Twitter - \$629 million |
| 3. Apple - \$786 million | 8. AOL - \$191.7 million |
| 4. Intel - \$776 million | 9. Facebook - \$60 million |
| 5. Microsoft - \$690 million | 10. Salesforce - \$32.8 million |

France to spend \$1.8 billion on AI to compete with U.S., China

Mathieu Rosemain, Michel Rose

SCIENCE NEWS MARCH 29, 2018 / 2:36 PM / 10 DAYS AGO

REUTERS World Business Markets Politics TV

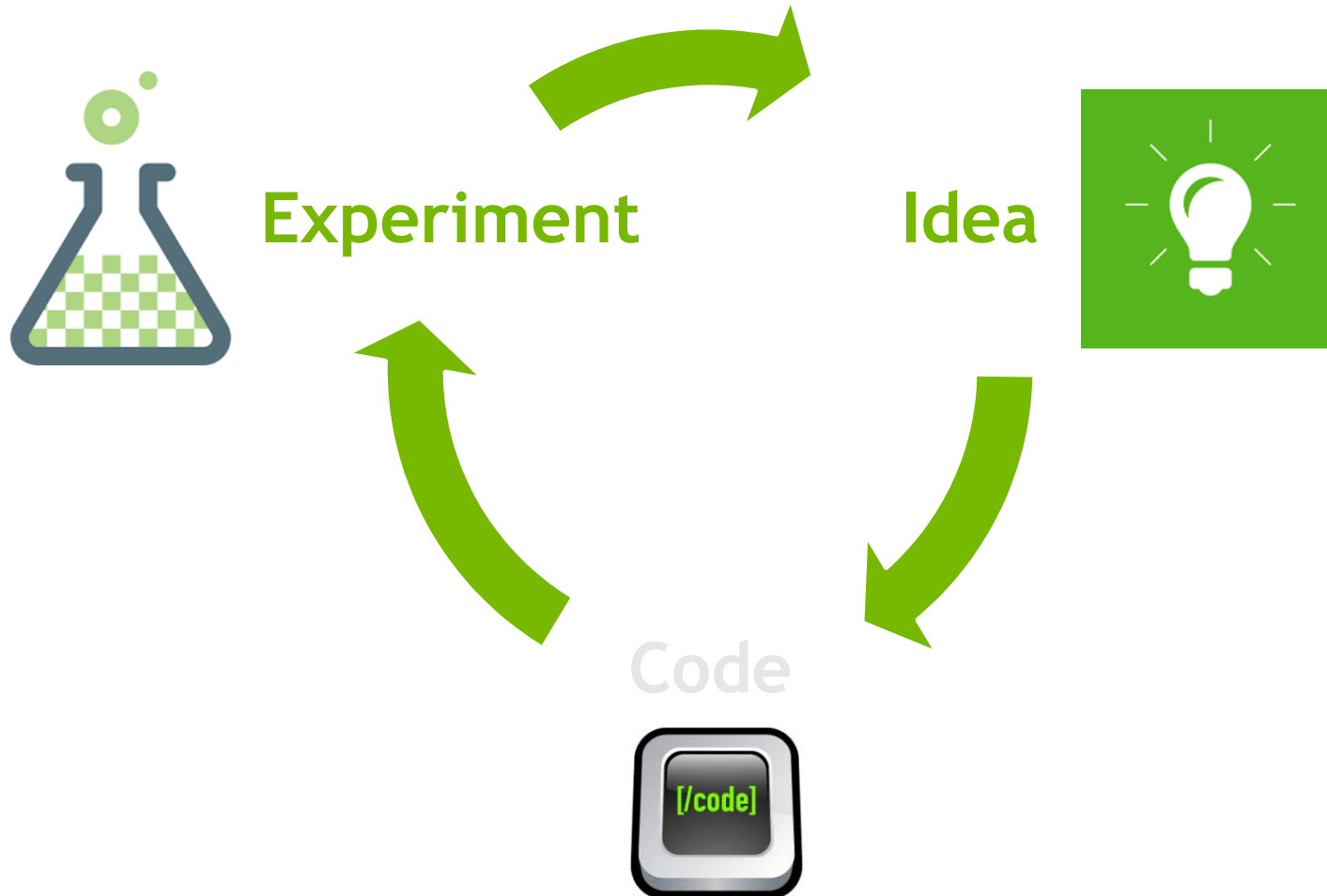
IMPLICATIONS

Good and bad news

- ▶ The good news: Requirements are predictable.
 - ▶ We can predict how much data we will need
 - ▶ We can predict how much computing power we will need
- ▶ The **bad** news: The values can be significant.

IMPLICATIONS

Experimental Nature of Deep Learning - Unacceptable training time



IMPLICATIONS

Automotive example

Majority of useful problems are too complex for a single GPU training

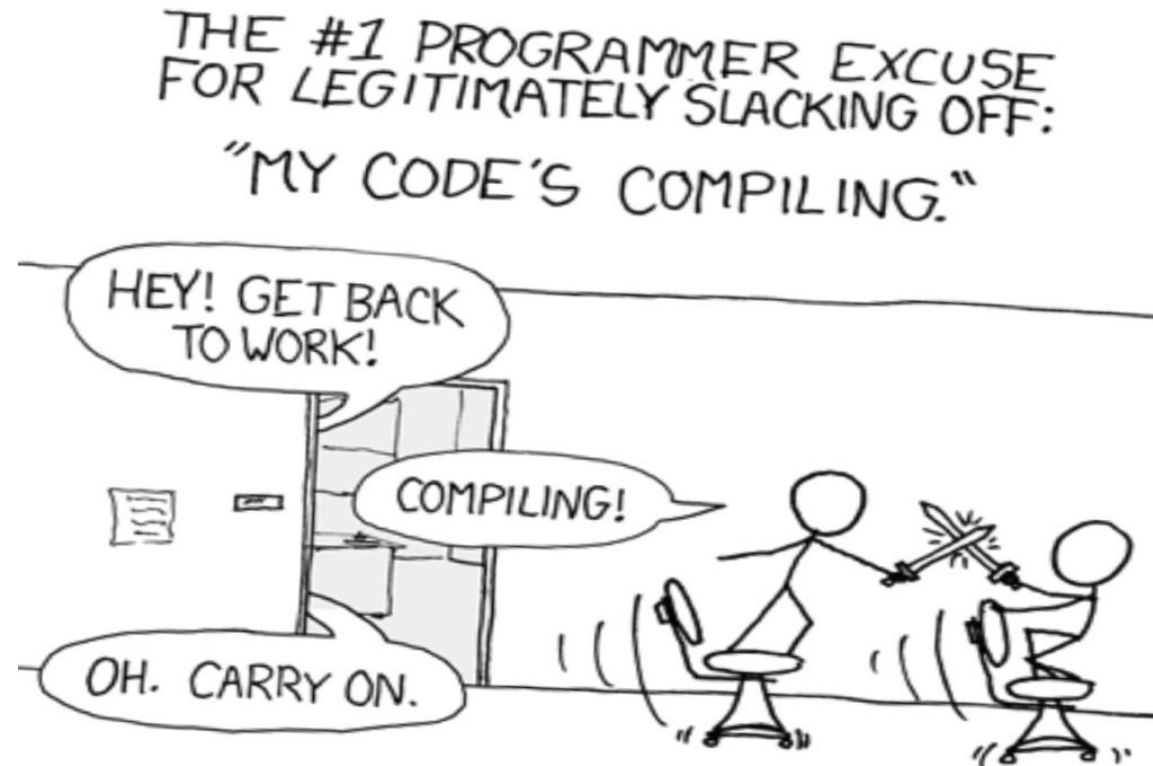
	VERY CONSERVATIVE	CONSERVATIVE
Fleet size (data capture per hour)	100 cars / 1TB/hour	125 cars / 1.5TB/hour
Duration of data collection	260 days * 8 hours	325 days * 10 hours
Data Compression factor	0.0005	0.0008
Total training set	104 TB	437.5 TB
InceptionV3 training time (with 1 Pascal GPU)	9.1 years	42.6 years
AlexNet training time (with 1 Pascal GPU)	1.1 years	5.4 years

100 TERABYTES EQUALS
600 MILLION BOOKS
OR
18 TIMES
THE PRINTED COLLECTION OF
THE LIBRARY OF CONGRESS



CONCLUSIONS

What does your team do in the mean time



CONCLUSIONS

What does your team do in the mean time

THE #1 PROGRAMMER EXCUSE
FOR LEGITIMATELY SLACKING OFF:
"MY DNN IS TRAINING"



CONCLUSIONS

Need to scale the training process for a single job



1 NVIDIA DGX-1

	VERY CONSERVATIVE	CONSERVATIVE
Total training set	104 TB	487.5 TB
InceptionV3 (one DGX-1V)	166 days (5+ months)	778 days (2+ years)
AlexNet (one DGX-1V)	21 days (3 weeks)	98 days (3 months)
InceptionV3 (10 DGX-1V's)	16 days (2+ weeks)	77 days (11 weeks)
AlexNet (10 DGX-1V's)	2.1 days	9.8 days

Training From Months or Years



10 NVIDIA DGX-1's

To Weeks or Days

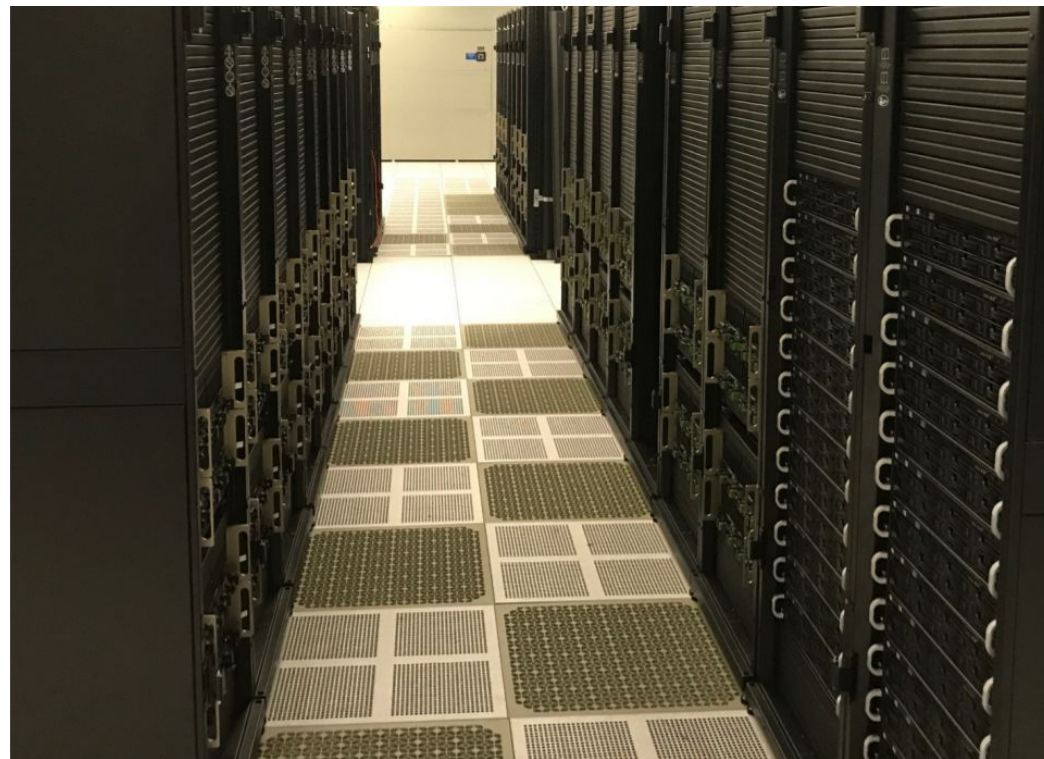
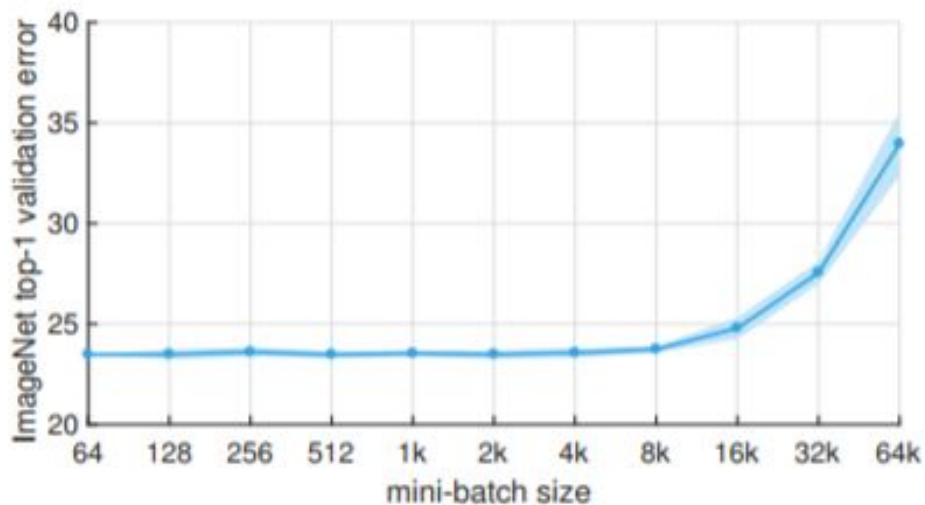


PRACTICAL EXAMPLES OF LARGE SCALE TRAINING

FACEBOOK

Training ImageNet with ResNet 50 in 1 hour

- 128 * DGX-1
- 10.5 PFLOPS total FP32
- 21 PFLOPS total FP16
- Non-blocking IB fabric

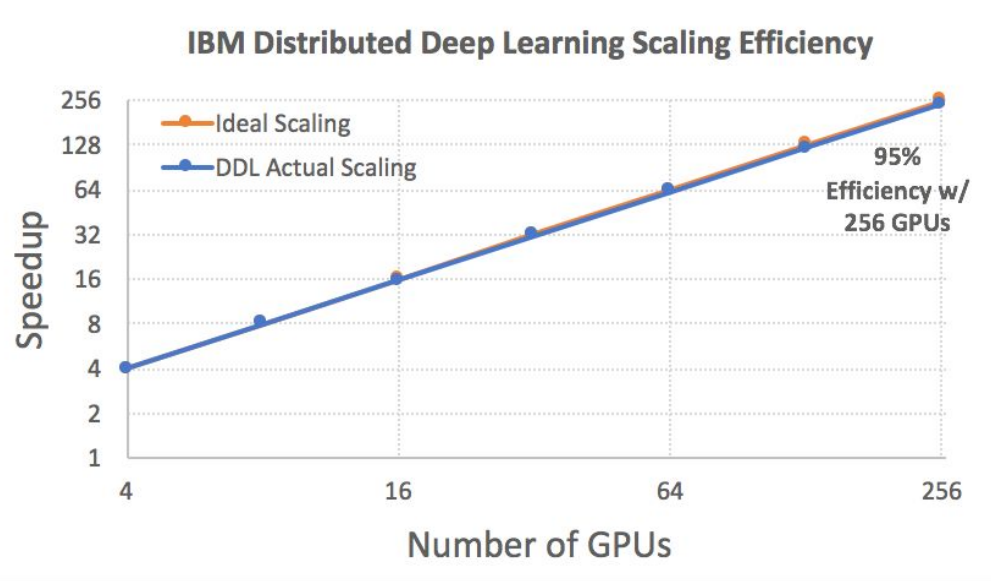


Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... & He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677*.

IBM

Training ImageNet with ResNet 101 in 7 hours

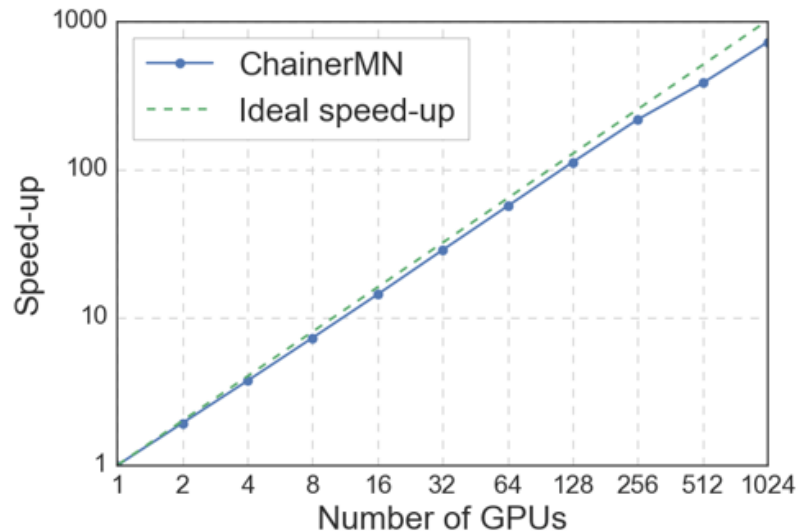
► 64 IBM Power Systems



PREFERRED NETWORKS

Training ImageNet in 15 minutes

- ▶ It consists of 128 nodes with 8 NVIDIA P100 GPUs each, for 1024 GPUs in total.
- ▶ The nodes are connected with two FDR Infiniband links (56Gbps x 2).

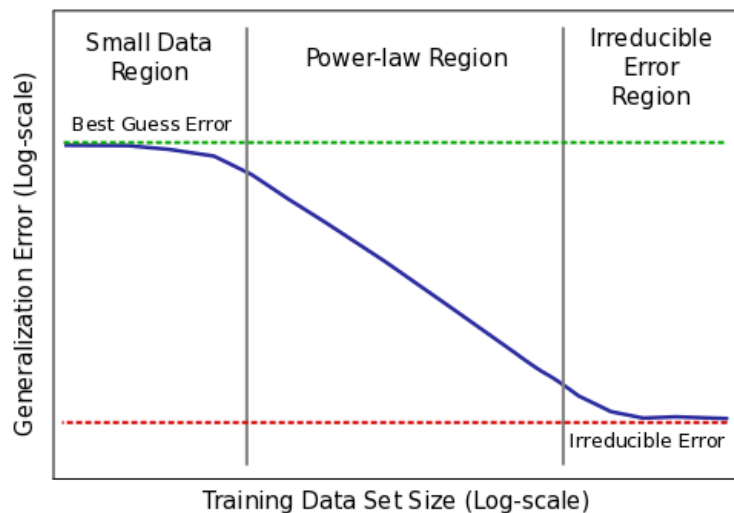


Akiba, T., Suzuki, S., & Fukuda, K. (2017). Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*.

BAIDU SVAIL

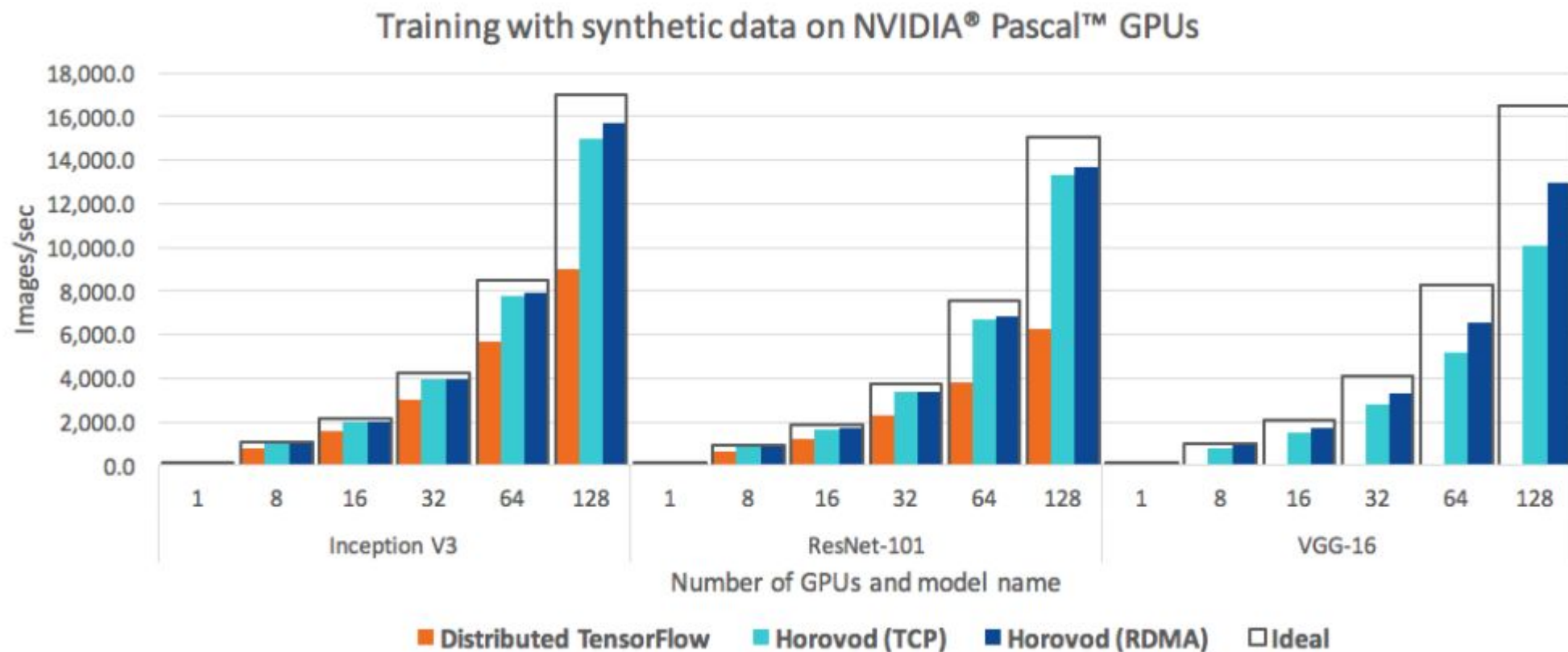
Investigating the log linear nature of the relationship between dataset size and generalization accuracy

► 11 PFLOPS across 1500 GPUs



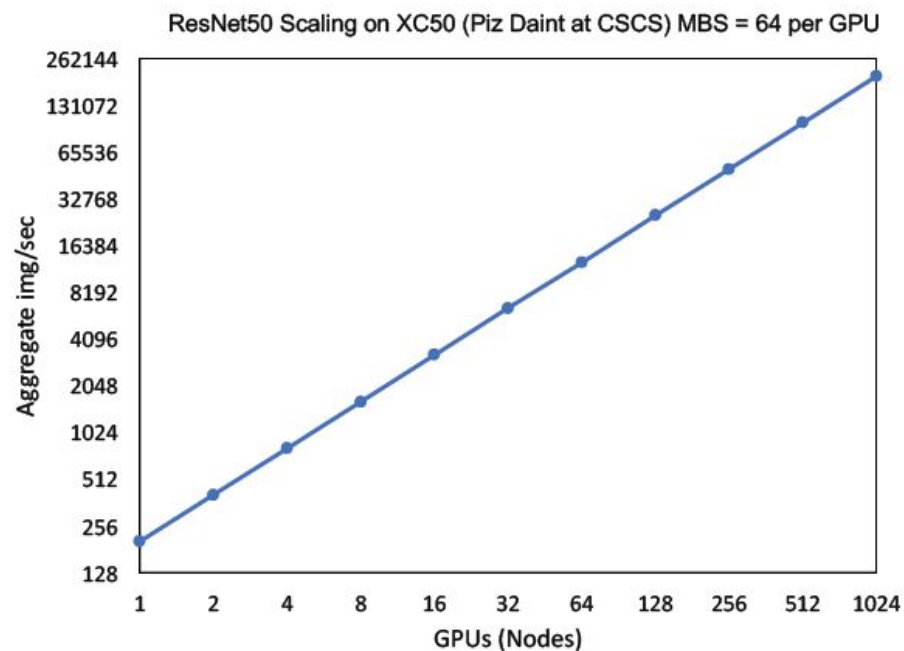
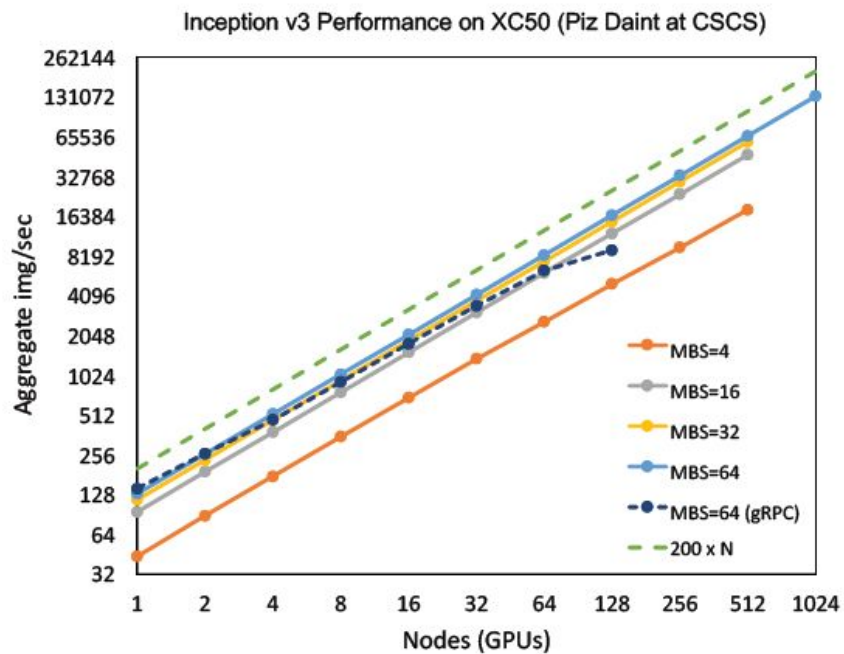
UBER

Investing heavily in Deep Learning scalability



CRAY

Piz Daint at CSCS



SATURN V

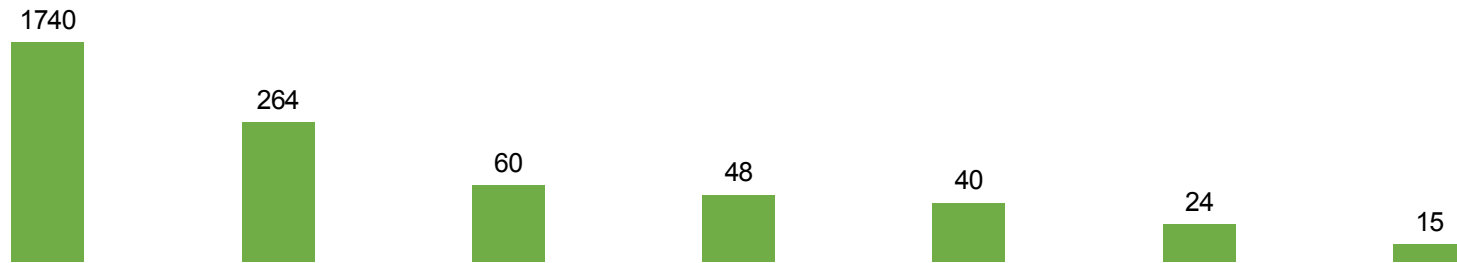
660 DGX-1 Volta Nodes

- ▶ 660 Nodes with a total of 5280 Volta GPUs
- ▶ 660 PFLOPs for AI training



ITERATION TIME

Short iteration time is fundamental for success



**DESPITE
'BLACK BOX'
INTERNALS
-
COMPLEX THEORY
BEHIND
IS
UNDER
RESEARCH NOW**

Stochastic Gradient Descent

More Data and Model Parallelism

Adaptation for Edge Computing

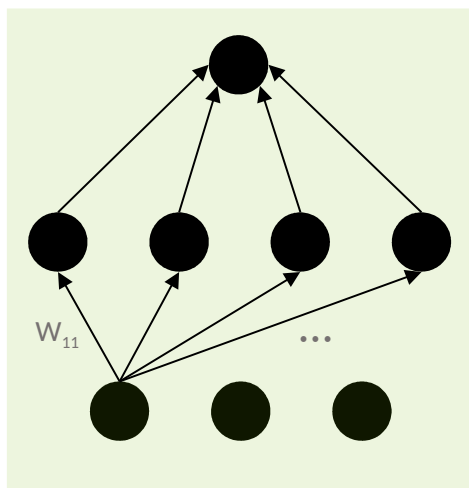
Multi-GPU Scaling

...

GRADIENT BASED OPTIMIZATION
STOCHASTIC GRADIENT DESCENT
(AND ITS VARIANTS)

OPTIMIZATION

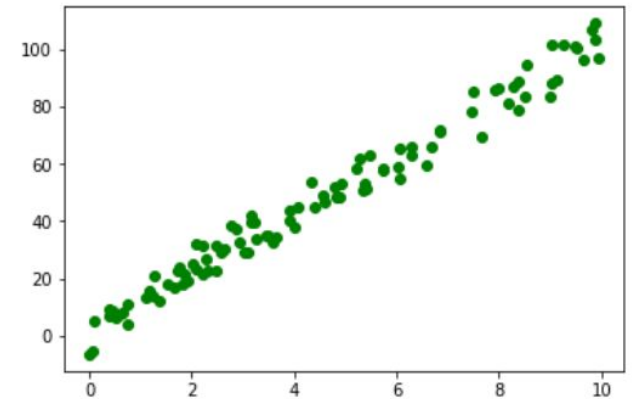
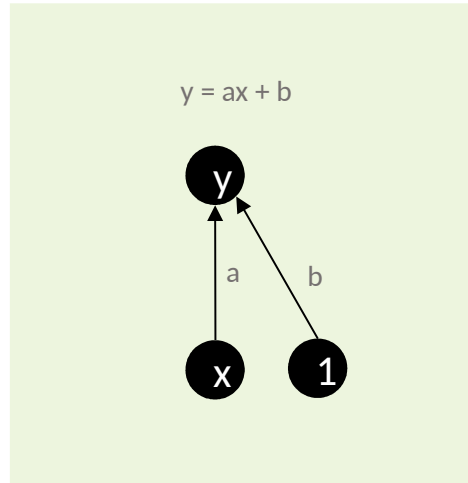
How do we find the parameters of the neural network in the first place?



OPTIMIZATION

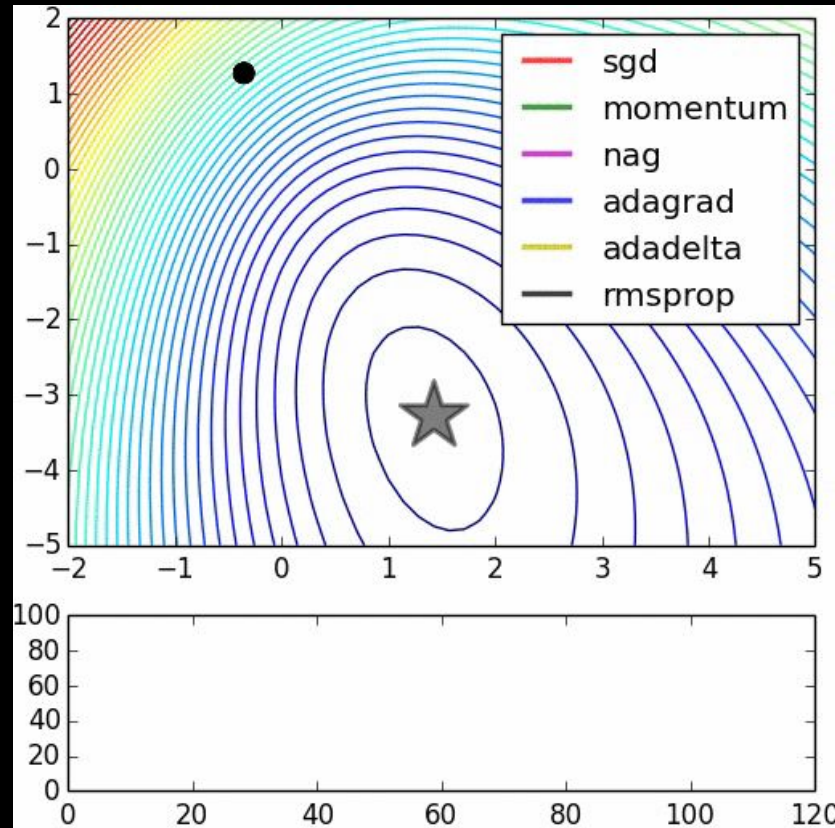
Lets start with the simplest of problems - linear neuron

Our goal is to find best model parameters (combination of a and b) to fit the data



MORE THAN JUST SGD

There exists a wide range of optimization algorithms



SGD FOR MORE COMPLEX NEURAL NETWORKS

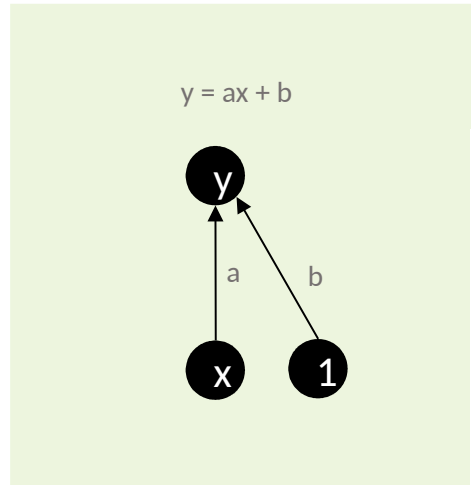
MODERN NEURAL NETWORKS

How do they differ from our trivial example?

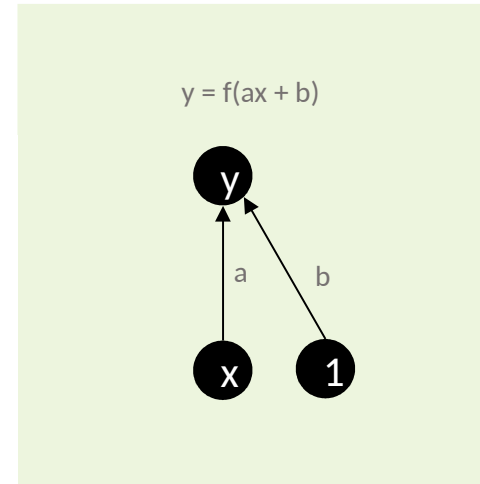
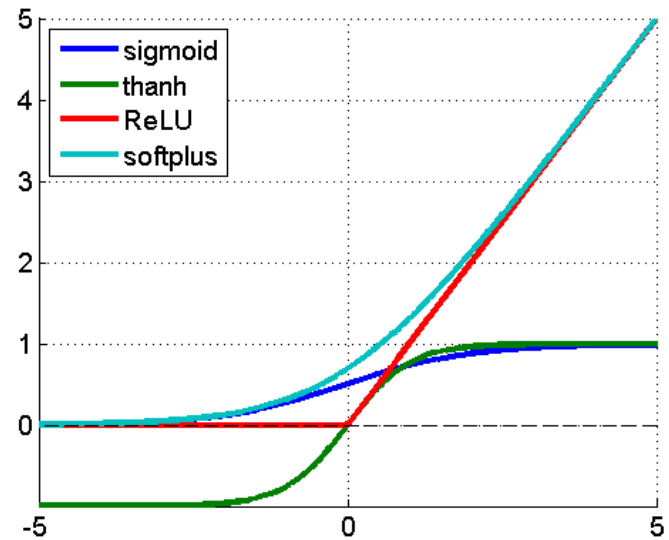
Not significantly!

MODERN NEURAL NETWORKS

How do they differ from our trivial example?



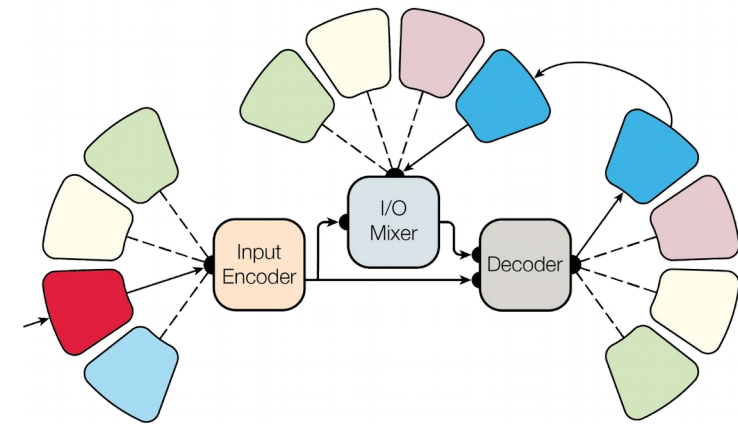
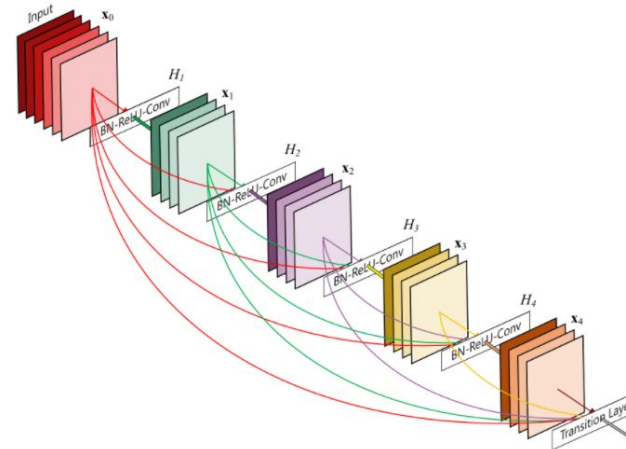
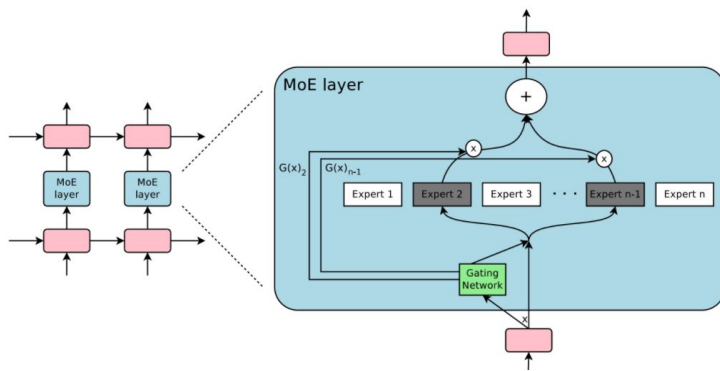
Nonlinearity



MODERN NEURAL NETWORKS

How do they differ from our trivial example?

More complex interconnection and many more parameters



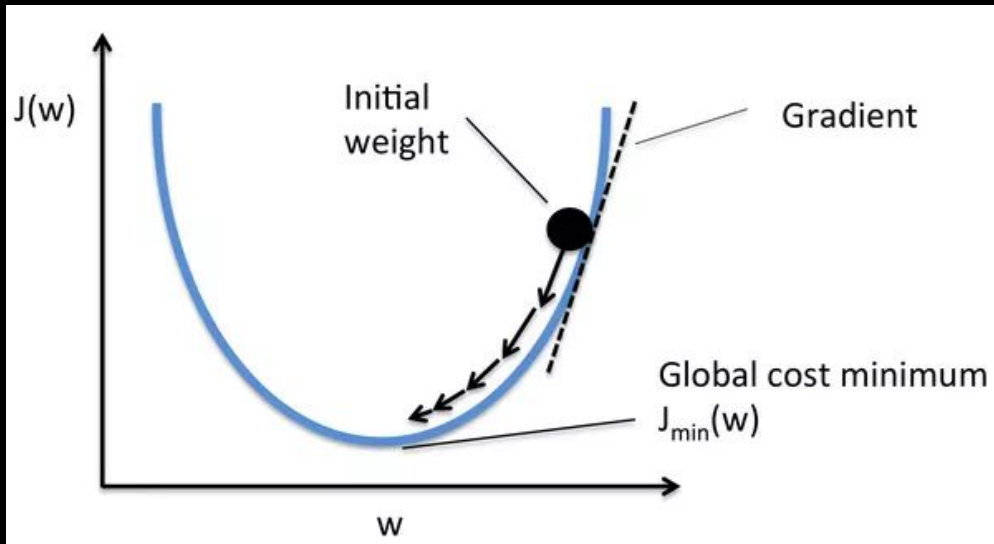
Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., & Uszkoreit, J. (2017). One model to learn them all. *arXiv preprint arXiv:1706.05137*.

Andola, F., Moskewicz, M., Karayev, S., Girshick, R., Darrell, T., & Keutzer, K. (2014). Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult



global maximum

local maximum

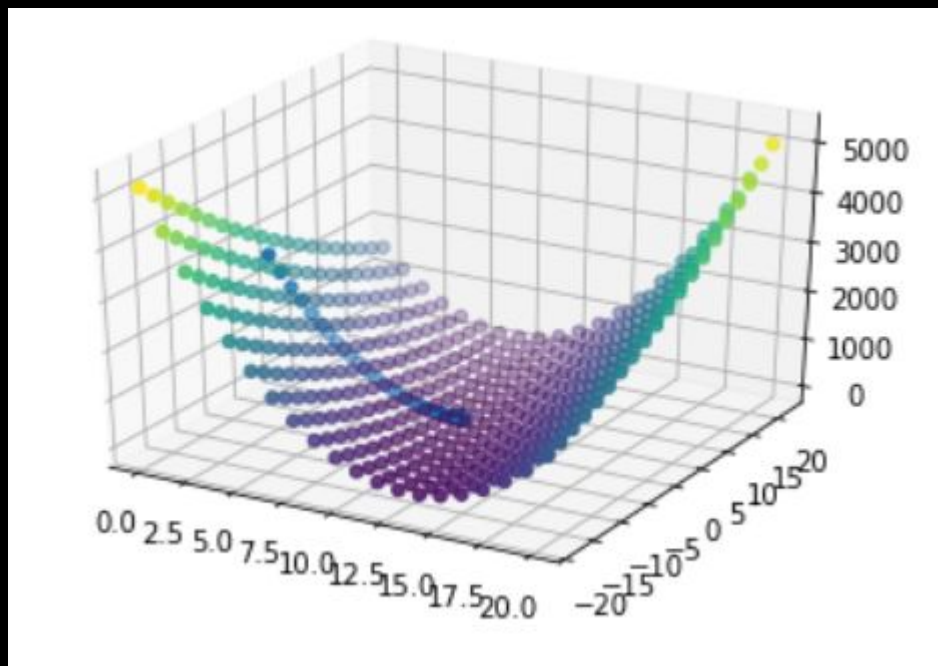
local minimum

global minimum

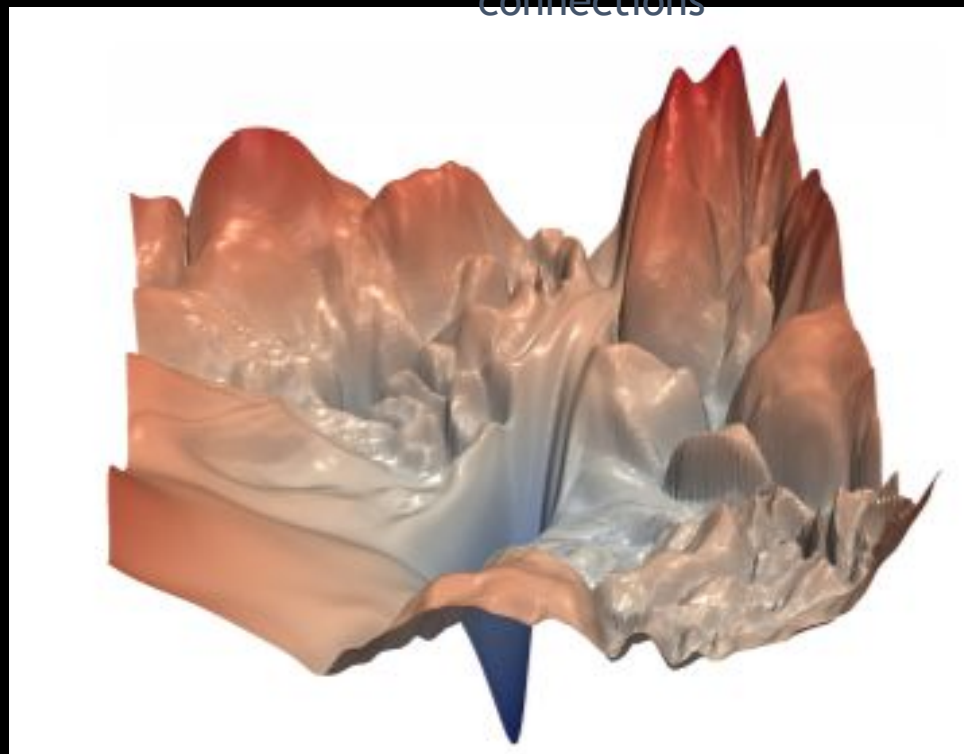
NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

Linear Neuron cost function



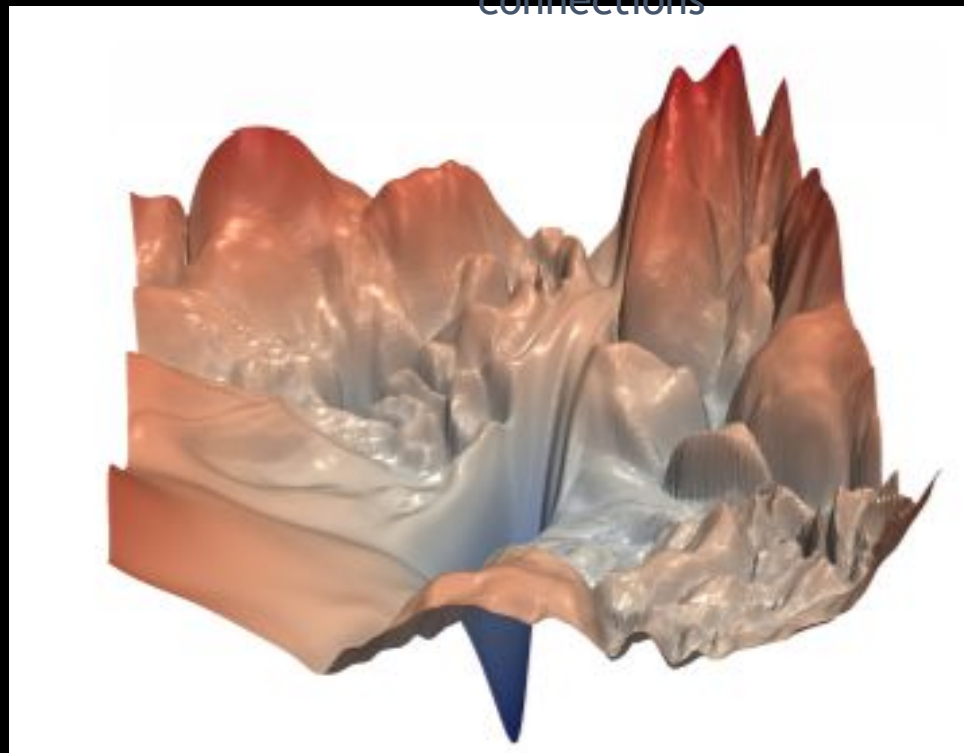
ResNet 56 cost function projection to 3D - no skip connections



NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

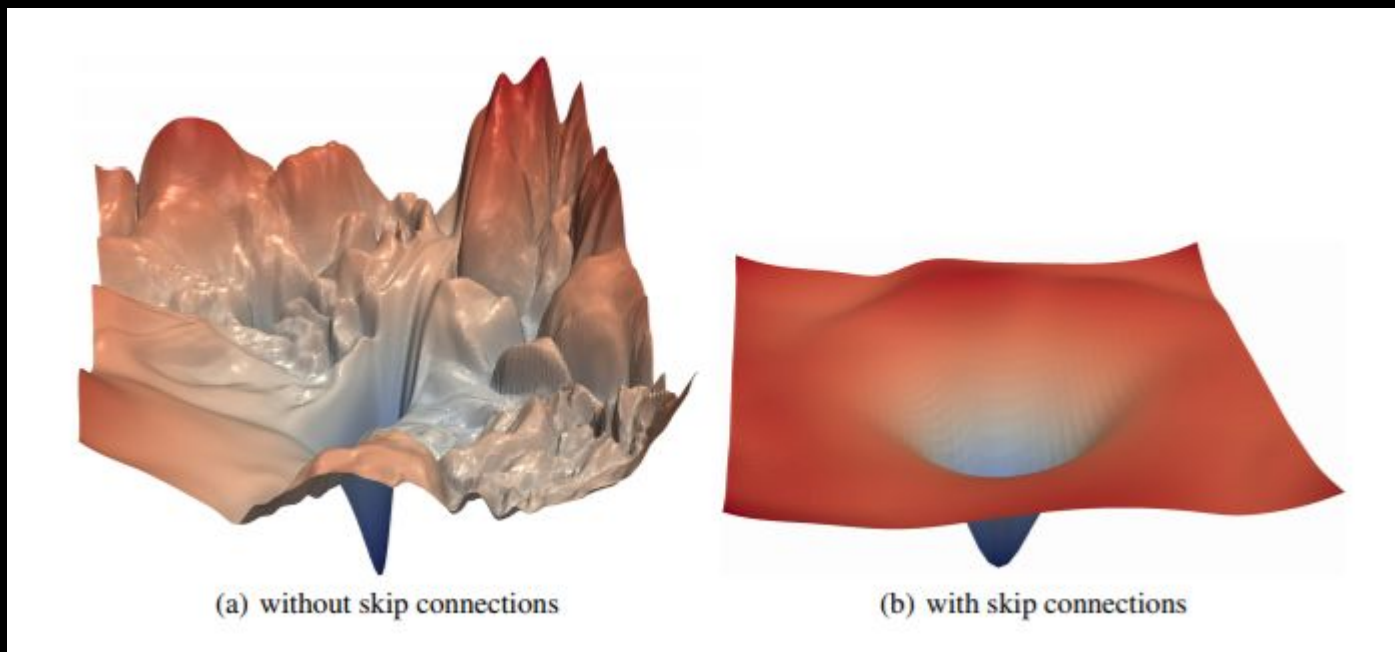
ResNet 56 cost function projection to 3D - no skip connections



Why do we succeed in finding good local minima?

NON CONVEX COST FUNCTIONS

Recent advances such as Residual Connections simplify the optimization problem



OPTIMISATION

The approach is the same though

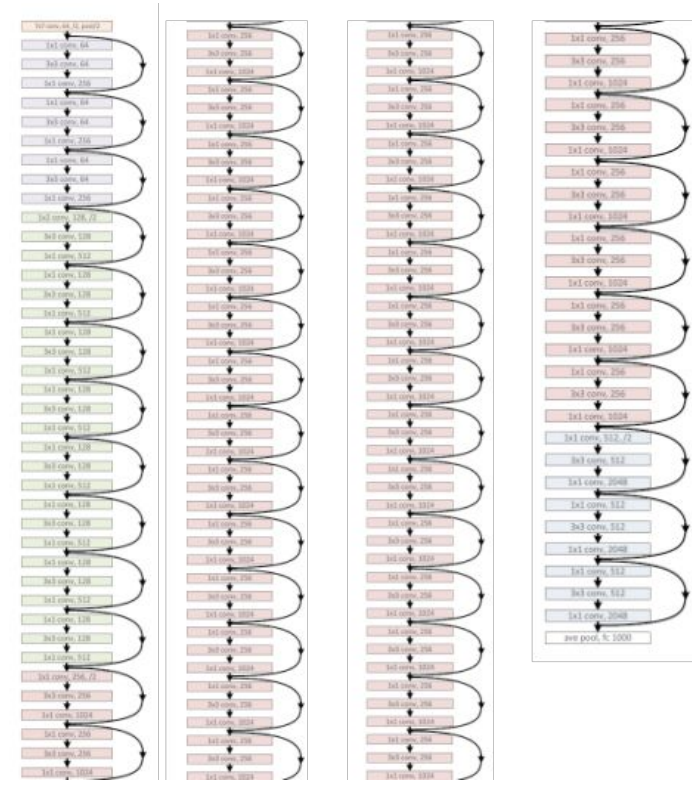
- Define a model (multilayer neural network)
- Define a cost function (problem specific)
- Iteratively:
 - Calculate the gradient of the cost function (the algorithm used to obtain the gradient is called backpropagation)
 - Update the model parameters (again using one of many optimisation algorithms)

BACKPROPAGATION

How do we compute the gradient of such a complex function?

$y = f(u), u = g(x)$
Therefore, $y = (f \circ g)(x)$

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$



BACKPROPAGATION

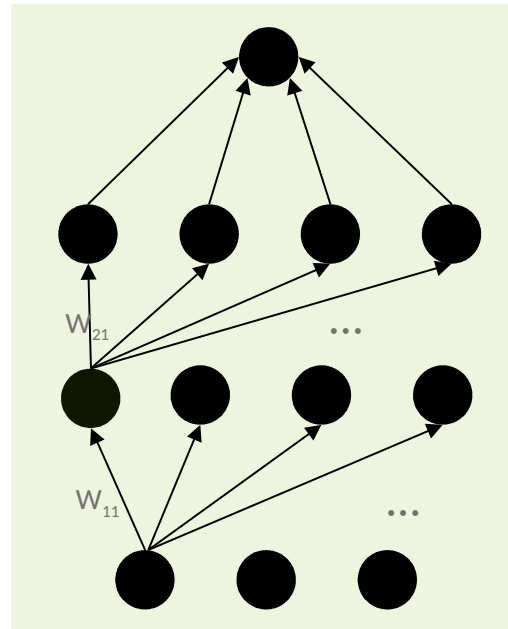
Automatic differentiation

- In practice this is rarely if ever done manually as all of the deep learning frameworks come with:
 - A suite of prebuild optimisation algorithms
 - An automatic differentiation functionality
- A very useful side effect is a fact that you can embed ANY DIFFERENTIABLE code into your neural network!

OPTIMIZATION

Let's start with a simplest neural network - multilayer perceptron

- We will build a simple (2 hidden layers) neural network - multilayer perceptron (no nonlinearity)
- We will work with the MNIST dataset
- Our goal is to find best model parameters to fit the data





DEEP
LEARNING
INSTITUTE

www.nvidia.com/dli

Deep Learning Methods

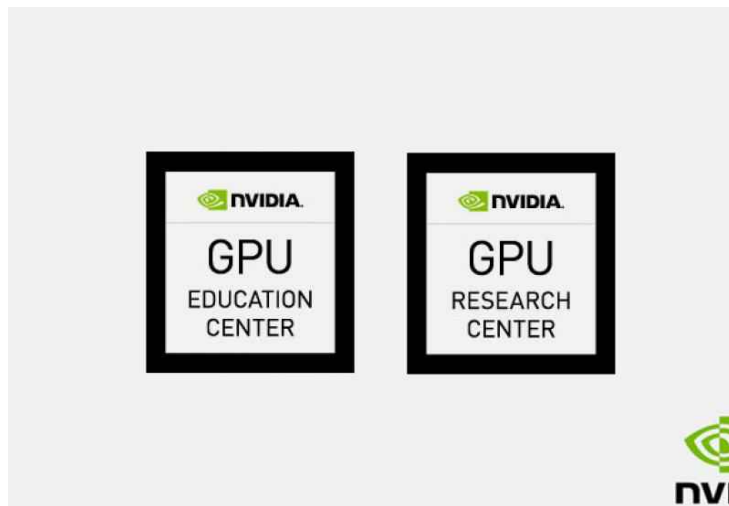
Lecture 04

Lecture Slides + interactive Jupyter-notebooks for Google Colaboratory CPU/GPU/TPU cloud:
<https://cloud.comsys.kpi.ua/s/SMkBSsxRTazoTD6>

Lecture 04 - CATEGORIES, TYPES, ORIGIN, DEVELOPMENT

The course includes materials proposed by NVIDIA Deep Learning Institute (DLI) in the framework of the common

NVIDIA Research Center
and
NVIDIA Education Center.



<https://kpi.ua/nvidia-info>

DEMO 1

CPU version - MNIST digit classification in TensorFlow 2.0

https://drive.google.com/file/d/1XeEckTs4qIFYFa56bYCoSeH_8YA007No/view?usp=sharing

DEMO 2

GPU version - MNIST digit classification in TensorFlow 2.0

https://drive.google.com/file/d/1_whW7Q-gi7TN-NLWIH2AL6CxnkBfvwua/view?usp=sharing

DEMO 3

TPU version - MNIST digit classification in TensorFlow 2.0

https://drive.google.com/file/d/1vESaa6yes2V0dW99vJ0_saM_r2opzmpz/view?usp=sharing

DEMO 4

Main Types of Deep Neural Networks

https://drive.google.com/file/d/1PvGNAGGbC_LB3ytw_vgB8xLsIe-t74Dh/view?usp=sharing

▼ CPU version - MNIST digit classification in TensorFlow 2.0

IMPORTANT: Runtime -> Change runtime -> None

Now, we will see how can we perform the MNIST handwritten digits classification using tensorflow 2.0. It hardly a few lines of code compared to the tensorflow 1.x. As we learned, tensorflow 2.0 uses as keras as its high-level API, we just need to add tf.keras to the keras code.

▼ Enabling and testing the environment

```
! cat /sys/class/dmi/id/product_name
```

```
Google Compute Engine
```

```
! cat /sys/class/dmi/id/sys_vendor
```

```
Google
```

```
! lscpu
```

```
Architecture:           x86_64
CPU op-mode(s):         32-bit, 64-bit
Byte Order:             Little Endian
CPU(s):                 2
On-line CPU(s) list:   0,1
Thread(s) per core:    2
Core(s) per socket:    1
Socket(s):              1
NUMA node(s):          1
Vendor ID:              AuthenticAMD
CPU family:             23
Model:                 49
Model name:             AMD EPYC 7B12
Stepping:              0
CPU MHz:               2249.998
BogoMIPS:              4499.99
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              512K
L3 cache:              16384K
NUMA node0 CPU(s):    0,1
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
```

```
! grep MemTotal /proc/meminfo
```

MemTotal: 13333596 kB

```
! df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
overlay	108G	31G	78G	28%	/
tmpfs	64M	0	64M	0%	/dev
tmpfs	6.4G	0	6.4G	0%	/sys/fs/cgroup
shm	5.9G	0	5.9G	0%	/dev/shm
tmpfs	6.4G	28K	6.4G	1%	/var/colab
/dev/sda1	114G	32G	83G	28%	/etc/hosts
tmpfs	6.4G	0	6.4G	0%	/proc/acpi
tmpfs	6.4G	0	6.4G	0%	/proc/scsi
tmpfs	6.4G	0	6.4G	0%	/sys/firmware

```
! nvidia-smi
```

NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver.



```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
-----
-----
SystemError                                Traceback (most recent
call last)
<ipython-input-7-d1680108c58e> in <module>()
      2 device_name = tf.test.gpu_device_name()
      3 if device_name != '/device:GPU:0':
----> 4     raise SystemError('GPU device not found')
      5 print('Found GPU at: {}'.format(device_name))

SystemError: GPU device not found
```

▼ Import the libraries:

```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
```

▼ Check Tensorflow version

```
print(tf.__version__)
```

2.4.1

▼ Load the dataset:

```
mnist = tf.keras.datasets.mnist
```

▼ Create a train and test set:

```
(x_train,y_train), (x_test, y_test) = mnist.load_data()
```

▼ Normalize data ...

... the x values by dividing with maximum value of x which is 255 and convert them to float:

```
x_train, x_test = tf.cast(x_train/255.0, tf.float32), tf.cast(x_test/255.0, tf.float32)
```

convert y values to int:

```
y_train, y_test = tf.cast(y_train,tf.int64),tf.cast(y_test,tf.int64)
```

▼ Create the model

Define the sequential model:

Define the sequential model:

```
model = tf.keras.models.Sequential()
```

Add the layers - We use a three-layered network. We apply ReLU activation at the first two layers and in the final output layer we apply softmax function:

```
model.add(tf.keras.layers.Flatten())  
model.add(tf.keras.layers.Dense(256, activation="relu"))  
model.add(tf.keras.layers.Dense(128, activation="relu"))  
model.add(tf.keras.layers.Dense(10, activation="softmax"))
```

Compile the model with Stochastic Gradient Descent, that is 'sgd' (we will learn about this in the next chapter) as optimizer and sparse_categorical_crossentropy as loss function and with

accuracy as a metric:

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['a
```

- List item
- List item

▼ Train

Train the model for 10 epochs with batch_size as 32:

```
history = model.fit(x_train, y_train, batch_size=32, epochs=10)
```

```
Epoch 1/10
1875/1875 [=====] - 5s 2ms/step - loss: 1.0382 - acc
Epoch 2/10
1875/1875 [=====] - 5s 2ms/step - loss: 0.3054 - acc
Epoch 3/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.2385 - acc
Epoch 4/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.2010 - acc
Epoch 5/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.1730 - acc
Epoch 6/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.1538 - acc
Epoch 7/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.1348 - acc
Epoch 8/10
1875/1875 [=====] - 5s 2ms/step - loss: 0.1239 - acc
Epoch 9/10
1875/1875 [=====] - 5s 2ms/step - loss: 0.1116 - acc
Epoch 10/10
1875/1875 [=====] - 5s 2ms/step - loss: 0.1017 - acc
```

▼ Show the structure of the model

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(32, 784)	0
dense (Dense)	(32, 256)	200960
dense_1 (Dense)	(32, 128)	32896
dense_2 (Dense)	(32, 10)	1290
Total params: 235,146		

Trainable params: 235,146
Non-trainable params: 0

▼ Evaluate

Evaluate the model on test sets:

```
model.evaluate(x_test, y_test)
```

```
313/313 [=====] - 1s 1ms/step - loss: 0.1082 - accur  
[0.10815522819757462, 0.9682999849319458]
```




```
import tensorflow as tf
```

▼ Check Tensorflow version

```
print(tf.__version__)
```

```
2.4.1
```

▼ Load the dataset:

```
mnist = tf.keras.datasets.mnist
```

▼ Create a train and test set:

```
(x_train,y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data11493376/11490434 [=====] - 0s 0us/step
```

▼ Normalize data ...

... the x values by dividing with maximum value of x which is 255 and convert them to float:

```
x_train, x_test = tf.cast(x_train/255.0, tf.float32), tf.cast(x_test/255.0, tf.float32)
```

convert y values to int:

```
y_train, y_test = tf.cast(y_train,tf.int64),tf.cast(y_test,tf.int64)
```

▼ Create the model

Define the sequential model:

Define the sequential model:

```
model = tf.keras.models.Sequential()
```


Add the layers - We use a three-layered network. We apply ReLU activation at the first two layers

```
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation="relu"))
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="softmax"))
```

Compile the model with Stochastic Gradient Descent, that is 'sgd' (we will learn about this in the next chapter) as optimizer and sparse_categorical_crossentropy as loss function and with accuracy as a metric:

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['a
```

Show the structure of the model

▼ Train

Train the model for 10 epochs with batch_size as 32:

```
history = model.fit(x_train, y_train, batch_size=32, epochs=10)
```

```
Epoch 1/10
1875/1875 [=====] - 5s 2ms/step - loss: 1.0041 - acc
Epoch 2/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.2966 - acc
Epoch 3/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.2382 - acc
Epoch 4/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.2003 - acc
Epoch 5/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.1774 - acc
Epoch 6/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.1537 - acc
Epoch 7/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.1370 - acc
Epoch 8/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.1232 - acc
Epoch 9/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.1089 - acc
Epoch 10/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.1030 - acc
```



```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(32, 784)	0

dense (Dense)	(32, 256)	200960
dense_1 (Dense)	(32, 128)	32896
dense_2 (Dense)	(32, 10)	1290
=====		
Total params: 235,146		
Trainable params: 235,146		
Non-trainable params: 0		

▼ Evaluate

Evaluate the model on test sets:

```
model.evaluate(x_test, y_test)
```

```
313/313 [=====] - 1s 2ms/step - loss: 0.1084 - accur  
[0.10837740451097488, 0.96670001745224]
```



▼ TPU version - MNIST digit classification in TensorFlow 2.0

IMPORTANT: Runtime -> Change runtime -> TPU

Now, we will see how can we perform the MNIST handwritten digits classification using tensorflow 2.0. It hardly a few lines of code compared to the tensorflow 1.x. As we learned, tensorflow 2.0 uses as keras as its high-level API, we just need to add tf.keras to the keras code.

▼ Enabling and testing the TPU

First, you'll need to enable TPUs for the notebook:

- Navigate to Edit → Notebook Settings
- select TPU from the Hardware Accelerator drop-down

Next, we'll check that we can connect to the TPU:

```
%tensorflow_version 2.x
import tensorflow as tf
print("Tensorflow version " + tf.__version__)

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver() # TPU detection
    print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])
except ValueError:
    raise BaseException('ERROR: Not connected to a TPU runtime; please see the previ

tf.config.experimental_connect_to_cluster(tpu)
tf.tpu.experimental.initialize_tpu_system(tpu)
tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
```

```
Tensorflow version 2.4.1
Running on TPU  ['10.15.163.122:8470']
INFO:tensorflow:Initializing the TPU system: grpc://10.15.163.122:8470
INFO:tensorflow:Initializing the TPU system: grpc://10.15.163.122:8470
INFO:tensorflow:Clearing out eager caches
INFO:tensorflow:Clearing out eager caches
INFO:tensorflow:Finished initializing TPU system.
INFO:tensorflow:Finished initializing TPU system.
WARNING:absl:`tf.distribute.experimental.TPUStrategy` is deprecated, please u
INFO:tensorflow:Found TPU system:
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replic
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replic
```


▼ Normalize data ...

... the x values by dividing with maximum value of x which is 255 and convert them to float:

```
x_train, x_test = tf.cast(x_train/255.0, tf.float32), tf.cast(x_test/255.0, tf.float32)
```

convert y values to int:

```
y_train, y_test = tf.cast(y_train,tf.int64),tf.cast(y_test,tf.int64)
```

▼ Create the model

Define the sequential model:

```
model = tf.keras.models.Sequential()
```

Add the layers - We use a three-layered network. We apply ReLU activation at the first two layers and in the final output layer we apply softmax function:

```
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation="relu"))
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="softmax"))
```

Compile the model with Stochastic Gradient Descent, that is 'sgd' (we will learn about this in the next chapter) as optimizer and sparse_categorical_crossentropy as loss function and with accuracy as a metric:

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

▼ Train

Train the model for 10 epochs with batch_size as 32:

```
history = model.fit(x_train, y_train, batch_size=32, epochs=10)
```

```
Epoch 1/10
1875/1875 [=====] - 21s 11ms/step - loss: 0.9990 - accuracy: 0.0000
Epoch 2/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.2995 - accuracy: 0.4990
Epoch 3/10
```

```

1875/1875 [=====] - 20s 11ms/step - loss: 0.2413 - a
Epoch 4/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.2023 - a
Epoch 5/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.1679 - a
Epoch 6/10
1875/1875 [=====] - 22s 12ms/step - loss: 0.1526 - a
Epoch 7/10
1875/1875 [=====] - 21s 11ms/step - loss: 0.1340 - a
Epoch 8/10
1875/1875 [=====] - 21s 11ms/step - loss: 0.1230 - a
Epoch 9/10
1875/1875 [=====] - 21s 11ms/step - loss: 0.1105 - a
Epoch 10/10
1875/1875 [=====] - 21s 11ms/step - loss: 0.1019 - a

```



▼ Show the structure of the model

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(32, 784)	0
dense (Dense)	(32, 256)	200960
dense_1 (Dense)	(32, 128)	32896
dense_2 (Dense)	(32, 10)	1290

```

Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0

```

Compare with training results on GPU

```
Epoch 1/10 1875/1875 [=====] - 6s 2ms/step - loss: 0.9975 - accuracy: 0.7304
```

```
Epoch 2/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.3008 - accuracy: 0.9134
```

```
Epoch 3/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.2401 - accuracy: 0.9320
```

```
Epoch 4/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.2071 - accuracy: 0.9423
```

Epoch 5/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.1794 - accuracy: 0.9492

Epoch 6/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.1576 - accuracy: 0.9548

Epoch 7/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.1393 - accuracy: 0.9613

Epoch 8/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.1309 - accuracy: 0.9628

Epoch 9/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.1119 - accuracy: 0.9680

Epoch 10/10 1875/1875 [=====] - 4s 2ms/step - loss: 0.1042 -

▼ Evaluate

Evaluate the model on test sets:

```
model.evaluate(x_test, y_test)
```

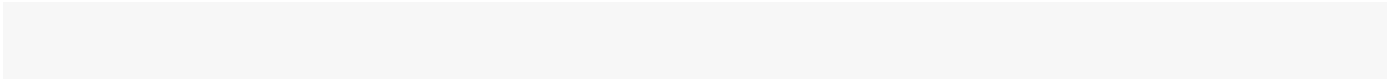
```
313/313 [=====] - 3s 9ms/step - loss: 0.1062 - accuracy: 0.9685  
[0.10615649074316025, 0.968500018119812]
```



▼ Compare with training results on GPU

```
313/313 [=====] - 1s 2ms/step - loss: 0.1077 - accuracy: 0.9671  
[0.10774651169776917, 0.9671000242233276]
```

The results are nearly the same up to 3rd (loss) and 4th (accuracy) significant number after the decimal point.



Colab paid products - Cancel contracts here



Deep Learning Basics

Main Types of Deep Neural Networks

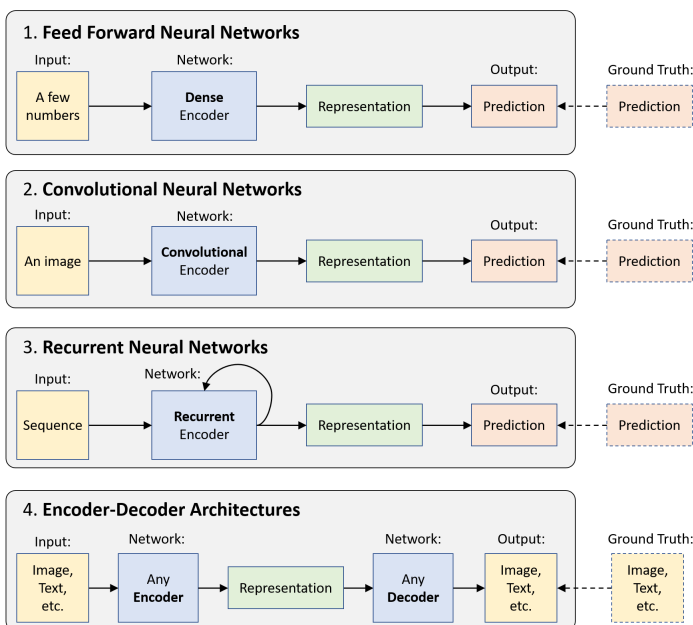
based on (C) [MIT Deep Learning](#) course

This tutorial accompanies the [lecture on Deep Learning Basics](#) given as part of [MIT Deep Learning](#). Acknowledgement to amazing people involved is provided throughout the tutorial and at the end. You can watch the video on YouTube:

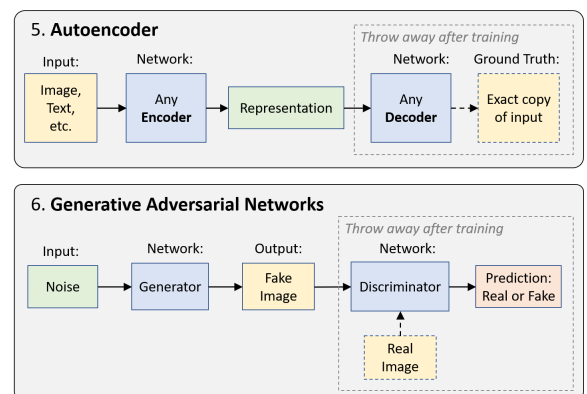


In this tutorial, we mention seven important types/concepts/approaches in deep learning, introducing the first 2 and providing pointers to tutorials on the others. Here is a visual representation of the seven:

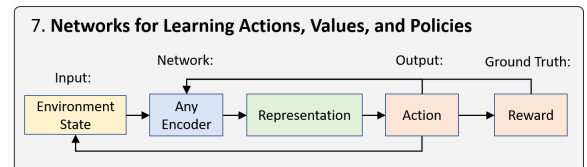
Supervised Learning



Unsupervised Learning



Reinforcement Learning



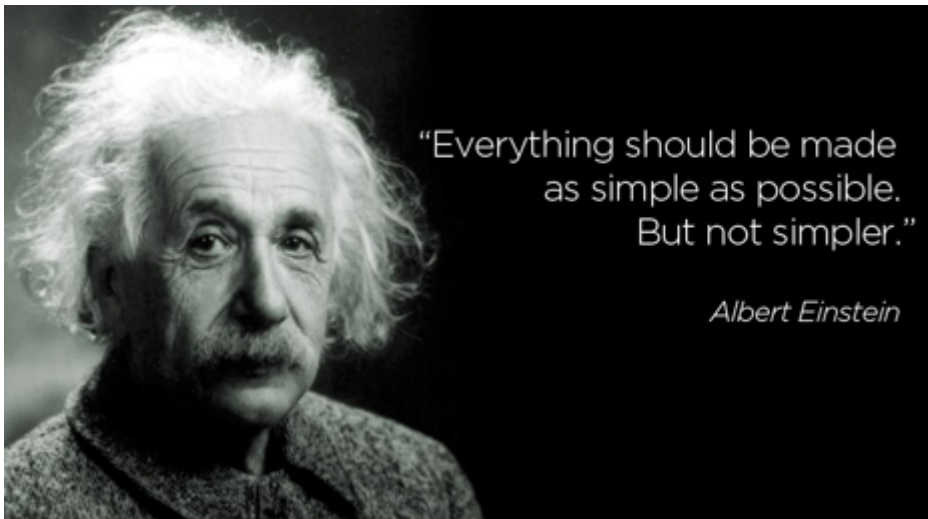
At a high-level, neural networks are either encoders, decoders, or a combination of both. Encoders find patterns in raw data to form compact, useful representations. Decoders generate new data or high-resolution useful information from those representations. As the lecture describes, deep learning discovers ways to **represent** the world so that we can reason about it. The rest is clever methods that help use deal effectively with visual information, language, sound (#1-6) and even act in a world based on this information and occasional rewards (#7).

1. **Feed Forward Neural Networks (FFNNs)** - classification and regression based on features. See [Part 1](#) of this tutorial for an example.
2. **Convolutional Neural Networks (CNNs)** - image classification, object detection, video action recognition, etc. See [Part 2](#) of this tutorial for an example.
3. **Recurrent Neural Networks (RNNs)** - language modeling, speech recognition/generation, etc. See [this TF tutorial on text generation](#) for an example.
4. **Encoder Decoder Architectures** - semantic segmentation, machine translation, etc. See [our tutorial on semantic segmentation](#) for an example.
5. **Autoencoder** - unsupervised embeddings, denoising, etc.
6. **Generative Adversarial Networks (GANs)** - unsupervised generation of realistic images, etc. See [this TF tutorial on DCGANs](#) for an example.
7. **Deep Reinforcement Learning** - game playing, robotics in simulation, self-play, neural architecture search, etc. We'll be releasing notebooks on this soon and will link them here.

There are selective omissions and simplifications throughout these tutorials, hopefully without losing the essence of the underlying ideas. See Einstein quote...

▼ Part 0: Prerequisites:

We recommend that you run this this notebook in the cloud on Google Colab (see link with icon at the top) if you're not already doing so. It's the simplest way to get started. You can also [install TensorFlow locally](#). But, again, simple is best (with caveats):



[tf.keras](#) is the simplest way to build and train neural network models in TensorFlow. So, that's what we'll stick with in this tutorial, unless the models necessitate a lower-level API.

Note that there's [tf.keras](#) (comes with TensorFlow) and there's [Keras](#) (standalone). You should be using [tf.keras](#) because (1) it comes with TensorFlow so you don't need to install anything extra and (2) it comes with powerful TensorFlow-specific features.

```

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

# Commonly used modules
import numpy as np
import os
import sys

# Images, plots, display, and visualization
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import cv2
import IPython
from six.moves import urllib

print(tf.__version__)

```

2.4.1

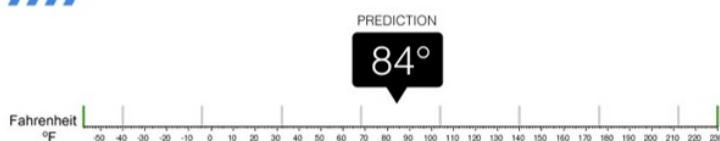
Part 1: Boston Housing Price Prediction with Feed Forward Neural Networks

Let's start with using a fully-connected neural network to do predict housing prices. The following image highlights the difference between regression and classification (see part 2). Given an observation as input, **regression** outputs a continuous value (e.g., exact temperature) and classification outputs a class/category that the observation belongs to.



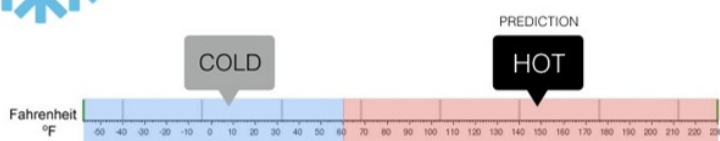
Regression

What is the temperature going to be tomorrow?



Classification

Will it be Cold or Hot tomorrow?



For the Boston housing dataset, we get 506 rows of data, with 13 features in each. Our task is to build a regression model that takes these 13 features as input and output a single value prediction of the "median value of owner-occupied homes (in \$1000)."

Now, we load the dataset. Loading the dataset returns four NumPy arrays:

- The `train_images` and `train_labels` arrays are the *training set*—the data the model uses to learn.
- The model is tested against the *test set*, the `test_images`, and `test_labels` arrays.

```
(train_features, train_labels), (test_features, test_labels) = keras.datasets.boston_housing.load_data()

# get per-feature statistics (mean, standard deviation) from the training set to normalize
train_mean = np.mean(train_features, axis=0)
train_std = np.std(train_features, axis=0)
train_features = (train_features - train_mean) / train_std
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston\_housing\_train.npy:
57344/57026 [=====] - 0s 0us/step
```

▼ Build the model

Building the neural network requires configuring the layers of the model, then compiling the model. First we stack a few layers together using `keras.Sequential`. Next we configure the loss function, optimizer, and metrics to monitor. These are added during the model's compile step:

- *Loss function* - measures how accurate the model is during training, we want to minimize this with the optimizer.
- *Optimizer* - how the model is updated based on the data it sees and its loss function.
- *Metrics* - used to monitor the training and testing steps.

Let's build a network with 1 hidden layer of 20 neurons, and use mean squared error (MSE) as the loss function (most common one for regression problems):

```
def build_model():
    model = keras.Sequential([
        Dense(20, activation=tf.nn.relu, input_shape=[len(train_features[0])]),
        Dense(1)
    ])

    # for TF1
    #model.compile(optimizer=tf.train.AdamOptimizer(),
    # for TF2: tf.train.AdamOptimizer() => tf.optimizers.Adam()
    model.compile(optimizer=tf.optimizers.Adam(),
                  loss='mse',
                  metrics=['mae', 'mse'])
    return model
```

```
# this helps makes our output less verbose but still shows progress
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
```

```
print('.', end='')  
  
model = build_model()
```

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 20)	280
dense_1 (Dense)	(None, 1)	21

Total params: 301
Trainable params: 301
Non-trainable params: 0

▼ Train the model

Training the neural network model requires the following steps:

1. Feed the training data to the model—in this example, the `train_features` and `train_labels` arrays.
2. The model learns to associate features and labels.
3. We ask the model to make predictions about a test set—in this example, the `test_features` array. We verify that the predictions match the labels from the `test_labels` array.

To start training, call the `model.fit` method—the model is "fit" to the training data:

```
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=50)  
history = model.fit(train_features, train_labels, epochs=1000, verbose=0, validation  
                    callbacks=[early_stop, PrintDot()])  
  
hist = pd.DataFrame(history.history)  
hist['epoch'] = history.epoch
```

```
.....  
.....  
.....  
.....  
.....
```



```
hist.head()
```

	loss	mae	mse	val_loss	val_mae	val_mse	epc
0	587.850525	22.342070	587.850525	497.338501	21.296099	497.338501	
1	578.270081	22.107761	578.270081	488.346130	21.056395	488.346130	
2	568.495544	21.872797	568.495544	479.016754	20.807596	479.016754	
3	558.563049	21.626896	558.563049	469.127014	20.536999	469.127014	

```
# show RMSE measure to compare to Kaggle leaderboard on https://www.kaggle.com/c/b
rmse_final = np.sqrt(float(hist['val_mse'].tail(1)))
print()
print('Final Root Mean Square Error on validation set: {}'.format(round(rmse_final
```

Final Root Mean Square Error on validation set: 2.383

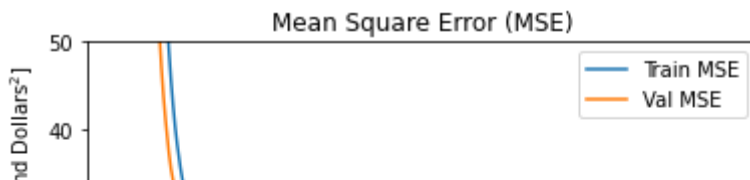
```
# show RMSE measure to compare to Kaggle leaderboard on https://www.kaggle.com/c/b
rmse_final = np.sqrt(float(hist['val_mae'].tail(1)))
print()
print('Final Root Mean Average Error on validation set: {}'.format(round(rmse_fina
```

Final Root Mean Average Error on validation set: 1.441

Now, let's plot the loss function measure on the training and validation sets. The validation set is used to prevent overfitting ([learn more about it here](#)). However, because our network is small, the training convergence without noticeably overfitting the data as the plot shows.

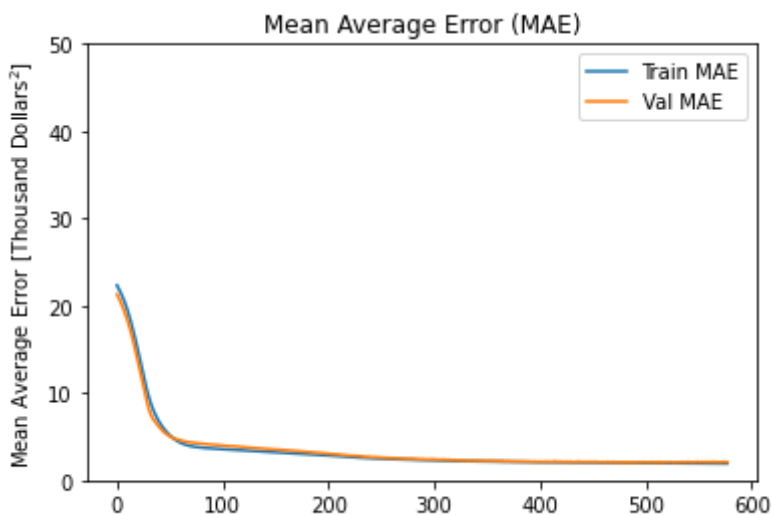
```
def plot_history():
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Square Error [Thousand Dollars2 $]')
    plt.plot(hist['epoch'], hist['mse'], label='Train MSE')
    plt.plot(hist['epoch'], hist['val_mse'], label = 'Val MSE')
    plt.legend()
    plt.title("Mean Square Error (MSE)")
    plt.ylim([0,50])

plot_history()
```



```
def plot_history():
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Average Error [Thousand Dollars$^2$]')
    plt.plot(hist['epoch'], hist['mae'], label='Train MAE')
    plt.plot(hist['epoch'], hist['val_mae'], label = 'Val MAE')
    plt.legend()
    plt.title("Mean Average Error (MAE)")
    plt.ylim([0,50])
```

```
plot_history()
```



Next, compare how the model performs on the test dataset:

```
print(model.metrics_names)

['loss', 'mae', 'mse']
```

```
test_features_norm = (test_features - train_mean) / train_std
loss, mae, mse = model.evaluate(test_features_norm, test_labels)
print('Loss on test set: {}'.format(round(loss, 3)))
print('Mean Average Error on test set: {}'.format(round(mae, 3)))
print('Mean Square Error on test set: {}'.format(round(mse, 3)))
rmse = np.sqrt(mse)
print('Root Mean Square Error on test set: {}'.format(round(rmse, 3)))
```

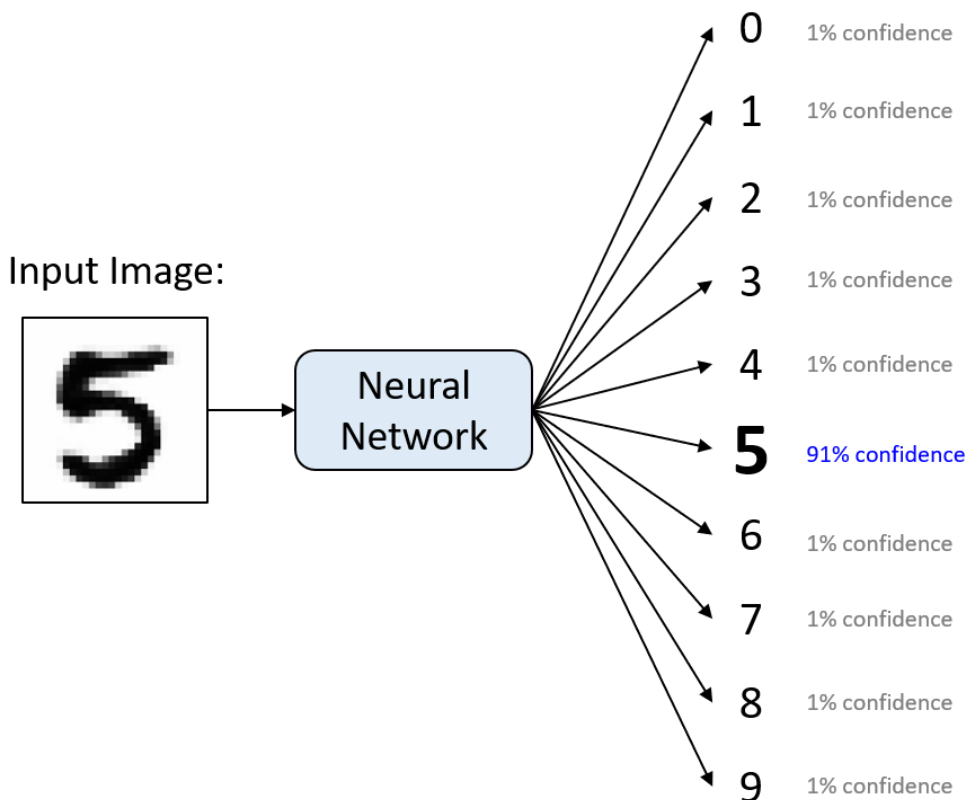
```
4/4 [=====] - 0s 4ms/step - loss: 18.3065 - mae: 2.686
Loss on test set: 18.306
Mean Average Error on test set: 2.686
Mean Square Error on test set: 18.306
Root Mean Square Error on test set: 4.279
```

Compare the RMSE measure you get to the [Kaggle leaderboard](#). An RMSE of 4.105 puts us in 24th place.

Part 2: Classification of MNIST Digits with Convolutional Neural Networks

Next, let's build a convolutional neural network (CNN) classifier to classify images of handwritten digits in the MNIST dataset with a twist where we test our classifier on high-resolution hand-written digits from outside the dataset.

The MNIST dataset contains 70,000 grayscale images of handwritten digits at a resolution of 28 by 28 pixels. The task is to take one of these images as input and predict the most likely digit contained in the image (along with a relative confidence in this prediction):



Now, we load the dataset. The images are 28x28 NumPy arrays, with pixel values ranging between 0 and 255. The *labels* are an array of integers, ranging from 0 to 9.

```
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()

# reshape images to specify that it's a single channel
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-data>

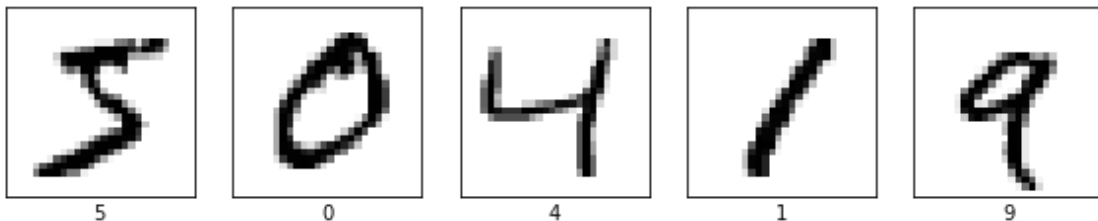
We scale these values to a range of 0 to 1 before feeding to the neural network model. For this, we divide the values by 255. It's important that the *training set* and the *testing set* are preprocessed in the same way:

```
def preprocess_images(imgs): # should work for both a single image and multiple images
    sample_img = imgs if len(imgs.shape) == 2 else imgs[0]
    assert sample_img.shape in [(28, 28, 1), (28, 28)], sample_img.shape # make sure
    return imgs / 255.0

train_images = preprocess_images(train_images)
test_images = preprocess_images(test_images)
```

Display the first 5 images from the *training set* and display the class name below each image. Verify that the data is in the correct format and we're ready to build and train the network.

```
plt.figure(figsize=(10,2))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i].reshape(28, 28), cmap=plt.cm.binary)
    plt.xlabel(train_labels[i])
```



▼ Build the model

Building the neural network requires configuring the layers of the model, then compiling the model. In many cases, this can be reduced to simply stacking together layers:

```
model = keras.Sequential()
# 32 convolution filters used each of size 3x3
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
# 64 convolution filters used each of size 3x3
model.add(Conv2D(64, (3, 3), activation='relu'))
# choose the best features via pooling
model.add(MaxPooling2D(pool_size=(2, 2)))
# randomly turn neurons on and off to improve convergence
model.add(Dropout(0.25))
```

```
# flatten since too many dimensions, we only want a classification output
model.add(Flatten())
# fully connected to get all relevant data
model.add(Dense(128, activation='relu'))
# one more dropout
model.add(Dropout(0.5))
# output a softmax to squash the matrix into output probabilities
model.add(Dense(10, activation='softmax'))
```

Before the model is ready for training, it needs a few more settings. These are added during the model's *compile* step:

- *Loss function* - measures how accurate the model is during training, we want to minimize this with the optimizer.
- *Optimizer* - how the model is updated based on the data it sees and its loss function.
- *Metrics* - used to monitor the training and testing steps. "accuracy" is the fraction of images that are correctly classified.

```
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
conv2d_1 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 12, 12, 64)	0
dropout (Dropout)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense_2 (Dense)	(None, 128)	1179776
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1290
Total params: 1,199,882		
Trainable params: 1,199,882		
Non-trainable params: 0		

```
model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

▼ Train the model

Training the neural network model requires the following steps:

1. Feed the training data to the model—in this example, the `train_images` and `train_labels` arrays.
2. The model learns to associate images and labels.
3. We ask the model to make predictions about a test set—in this example, the `test_images` array. We verify that the predictions match the labels from the `test_labels` array.

To start training, call the `model.fit` method—the model is "fit" to the training data:

```
history = model.fit(train_images, train_labels, epochs=5, validation_split = 0.1)
```

```
Epoch 1/5
1688/1688 [=====] - 11s 3ms/step - loss: 0.3720 - acc
Epoch 2/5
1688/1688 [=====] - 6s 3ms/step - loss: 0.0842 - acc
Epoch 3/5
1688/1688 [=====] - 5s 3ms/step - loss: 0.0678 - acc
Epoch 4/5
1688/1688 [=====] - 5s 3ms/step - loss: 0.0535 - acc
Epoch 5/5
1688/1688 [=====] - 6s 3ms/step - loss: 0.0418 - acc
```



```
print(model.metrics_names)
```

```
['loss', 'accuracy']
```

As the model trains, the loss and accuracy metrics are displayed. This model reaches an accuracy of about 98.68% on the training data.

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
```

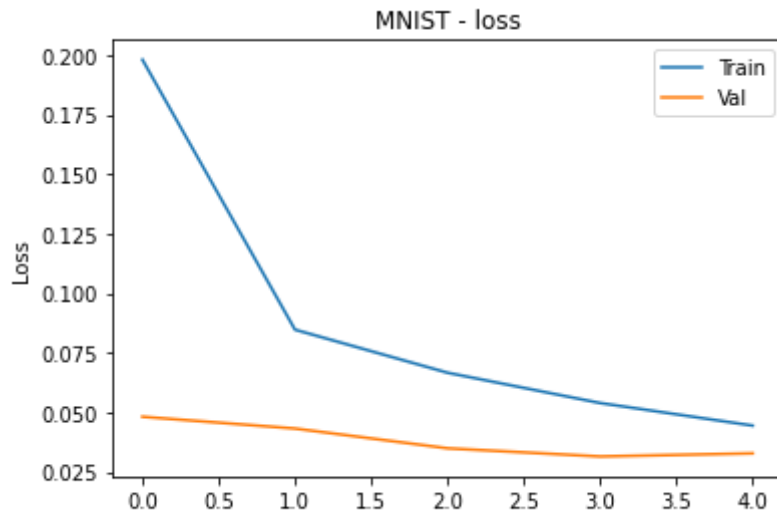
```
hist.head()
```

	loss	accuracy	val_loss	val_accuracy	epoch
0	0.198095	0.940278	0.048189	0.986333	0
1	0.084753	0.975000	0.043236	0.988167	1
2	0.066670	0.979667	0.034934	0.990000	2
3	0.053950	0.983907	0.031506	0.990000	3
4	0.044473	0.985796	0.032814	0.991000	4

```
def plot_history():
```

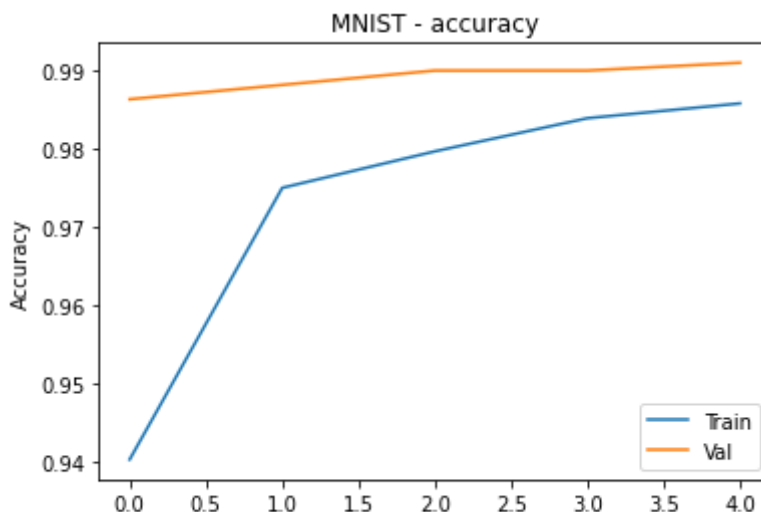
```
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(hist['epoch'], hist['loss'], label='Train')
plt.plot(hist['epoch'], hist['val_loss'], label = 'Val')
plt.legend()
plt.title("MNIST - loss")
#plt.ylim([0,50])
```

plot_history()



```
def plot_history():
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(hist['epoch'], hist['accuracy'], label='Train')
    plt.plot(hist['epoch'], hist['val_accuracy'], label = 'Val')
    plt.legend()
    plt.title("MNIST - accuracy")
    #plt.ylim([0,50])
```

plot_history()



▼ Evaluate accuracy

Next, compare how the model performs on the test dataset:

```
print(test_images.shape)
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('MNIST - Test accuracy:', test_acc)

(10000, 28, 28, 1)
313/313 [=====] - 1s 2ms/step - loss: 0.0291 - accur
MNIST - Test accuracy: 0.9905999898910522
```

▼ Compare with ... after 5 epochs

```
(10000, 28, 28, 1) 313/313 [=====] - 1s 2ms/step - loss: 0.0302 -
accuracy: 0.9910
```

Test accuracy: 0.9909999966621399

Often times, the accuracy on the test dataset is a little less than the accuracy on the training dataset. This gap between training accuracy and test accuracy is an example of *overfitting*. In our case, the accuracy is better at 99.19%! This is, in part, due to successful regularization accomplished with the Dropout layers.

▼ Make predictions

With the model trained, we can use it to make predictions about some images. Let's step outside the MNIST dataset for that and go with the beautiful high-resolution images generated by a mixture of CPPN, GAN, VAE. See [great blog post by hardmaru](#) for the source data and a description of how these morphed animations are generated:



```

# Set common constants
this_repo_url = 'https://github.com/lexfridman/mit-deep-learning/raw/master/'
this_tutorial_url = this_repo_url + 'tutorial_deep_learning_basics'

mnist_dream_path = 'images/mnist_dream.mp4'
mnist_prediction_path = 'images/mnist_dream_predicted.mp4'

# download the video if running in Colab
if not os.path.isfile(mnist_dream_path):
    print('downloading the sample video...')
    vid_url = this_tutorial_url + '/' + mnist_dream_path

    mnist_dream_path = urllib.request.urlretrieve(vid_url)[0]

def cv2_imshow(img):
    ret = cv2.imencode('.png', img)[1].tobytes()
    img_ip = IPython.display.Image(data=ret)
    IPython.display.display(img_ip)

cap = cv2.VideoCapture(mnist_dream_path)
vw = None
frame = -1 # counter for debugging (mostly), 0-indexed

# go through all the frames and run our classifier on the high res MNIST images as
while True: # should 481 frames
    frame += 1
    ret, img = cap.read()
    if not ret: break

    assert img.shape[0] == img.shape[1] # should be a square
    if img.shape[0] != 720:
        img = cv2.resize(img, (720, 720))

    #preprocess the image for prediction
    img_proc = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_proc = cv2.resize(img_proc, (28, 28))
    img_proc = preprocess_images(img_proc)
    img_proc = 1 - img_proc # inverse since training dataset is white text with bl

    net_in = np.expand_dims(img_proc, axis=0) # expand dimension to specify batch
    net_in = np.expand_dims(net_in, axis=3) # expand dimension to specify number o

    preds = model.predict(net_in)[0]
    guess = np.argmax(preds)
    perc = np rint(preds * 100).astype(int)

    img = 255 - img
    pad_color = 0
    img = np.pad(img, ((0,0), (0,1280-720), (0,0)), mode='constant', constant_valu

```

```

line_type = cv2.LINE_AA
font_face = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1.3
thickness = 2
x, y = 740, 60
color = (255, 255, 255)

text = "Neural Network Output:"
cv2.putText(img, text=text, org=(x, y), fontScale=font_scale, fontFace=font_fa
            color=color, lineType=line_type)

text = "Input:"
cv2.putText(img, text=text, org=(30, y), fontScale=font_scale, fontFace=font_f
            color=color, lineType=line_type)

y = 130
for i, p in enumerate(perc):
    if i == guess: color = (255, 218, 158)
    else: color = (100, 100, 100)

    rect_width = 0
    if p > 0: rect_width = int(p * 3.3)

    rect_start = 180
    cv2.rectangle(img, (x+rect_start, y-5), (x+rect_start+rect_width, y-20), c

    text = '{}: {:>3}%'.format(i, int(p))
    cv2.putText(img, text=text, org=(x, y), fontScale=font_scale, fontFace=fon
                color=color, lineType=line_type)
    y += 60

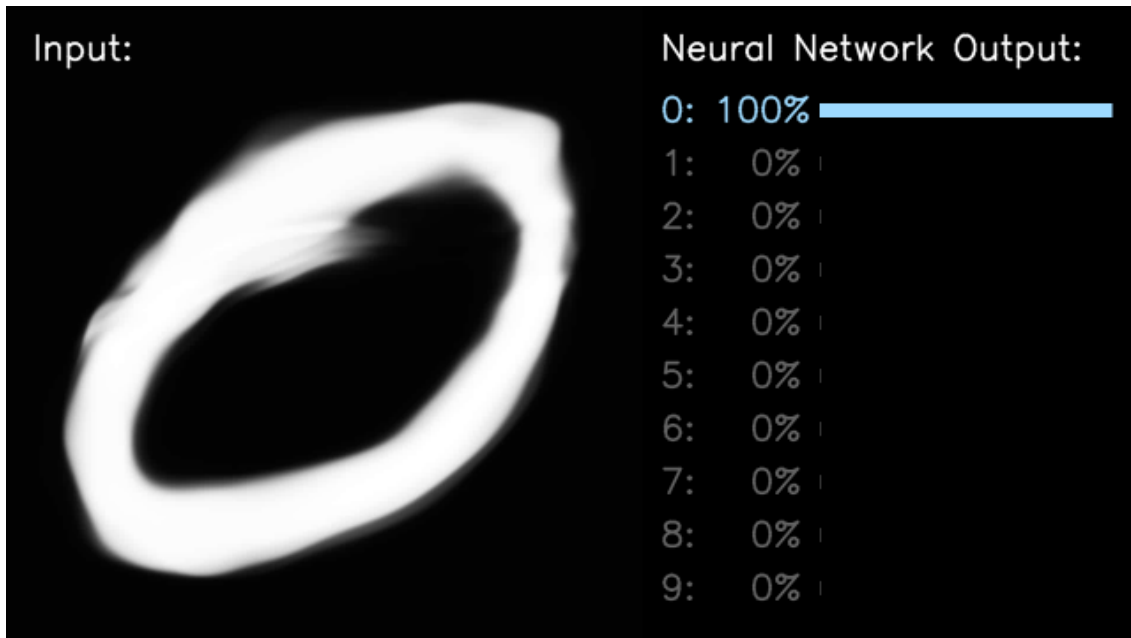
# if you don't want to save the output as a video, set this to False
save_video = True

if save_video:
    if vw is None:
        codec = cv2.VideoWriter_fourcc(*'DIVX')
        vid_width_height = img.shape[1], img.shape[0]
        vw = cv2.VideoWriter(mnist_prediction_path, codec, 30, vid_width_heigh
        # 15 fps above doesn't work robustly so we right frame twice at 30 fps
        vw.write(img)
        vw.write(img)

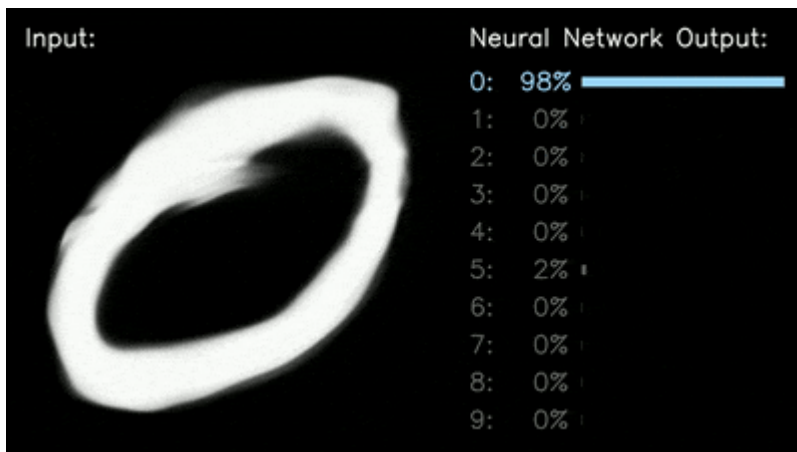
# scale down image for display
img_disp = cv2.resize(img, (0,0), fx=0.5, fy=0.5)
cv2.imshow(img_disp)
IPython.display.clear_output(wait=True)

cap.release()
if vw is not None:
    vw.release()

```



The above shows the prediction of the network by choosing the neuron with the highest output. While the output layer values add 1 to one, these do not reflect well-calibrated measures of "uncertainty". Often, the network is overly confident about the top choice that does not reflect a learned measure of probability. If everything ran correctly you should get an animation like this:



Acknowledgements

The contents of this tutorial is based on and inspired by the work of [TensorFlow team](#)), our [MIT Human-Centered AI team](#), and individual pieces referenced in the [MIT Deep Learning](#) course slides.

Colab paid products - [Cancel contracts here](#)



Методи Deep Learning

-

Deep Learning Methods

-

**Lecture 05. Deep Neural Networks
in TensorFlow**

(based on (C) F.Colliet, Lex Fridman, ... and others
works)

Content

- ◆ Recommended Sources
- ◆ DL Frameworks — Basics
- ◆ DL Frameworks — Workflow
 - * DEMO 1: Workflow in TF2
 - * DEMO 2: How to Monitor Workflow in TF2
- ◆ DL Workflow - Transfer Learning
 - * DEMO 3A: Learning from Scratch in TF2
 - * DEMO 3B: Transfer Learning in TF2

Recommended Sources

— Books

Books (scientific):

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep learning*.
Cambridge: MIT press

Цитовано в 23692 джерелах.

Books (with codes at github):

Alan Fontaine (2018) *Mastering Predictive Analytics with scikit-learn and TensorFlow*. Packt Publishing.

Tanay Agrawal (2021). *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient*, Apress

Recommended Sources

— Papers

Imagenet

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10), 1345-1359.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

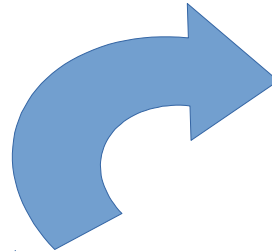
Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

DL Frameworks

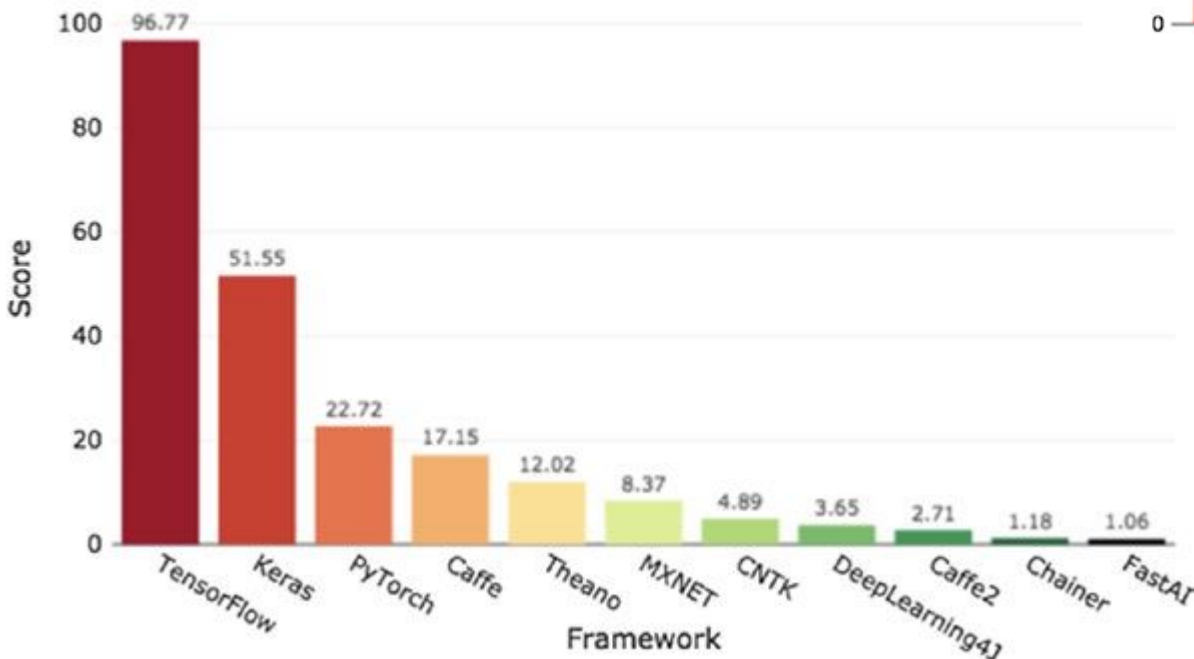
-

Basics

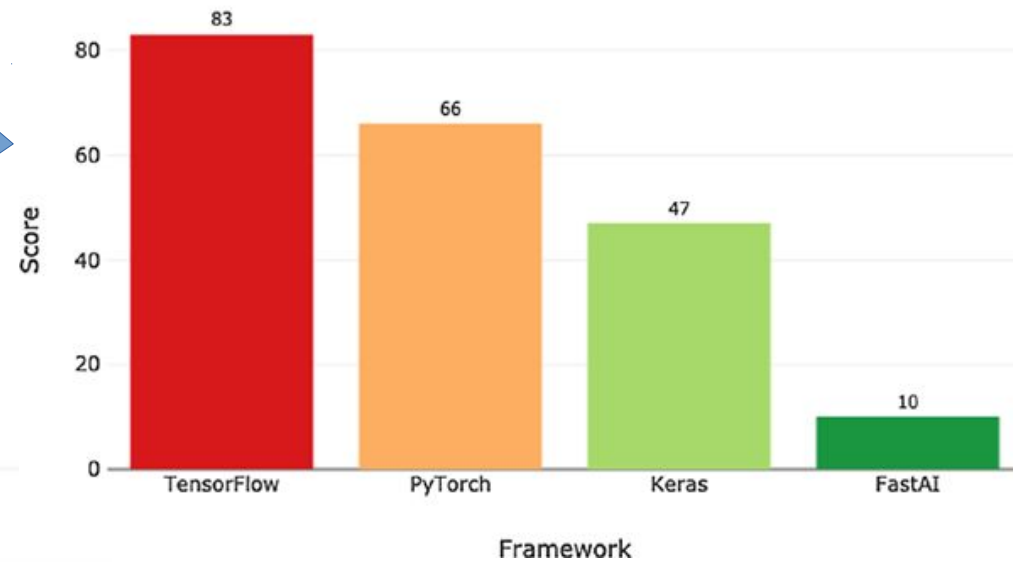
DL Frameworks — Evolution



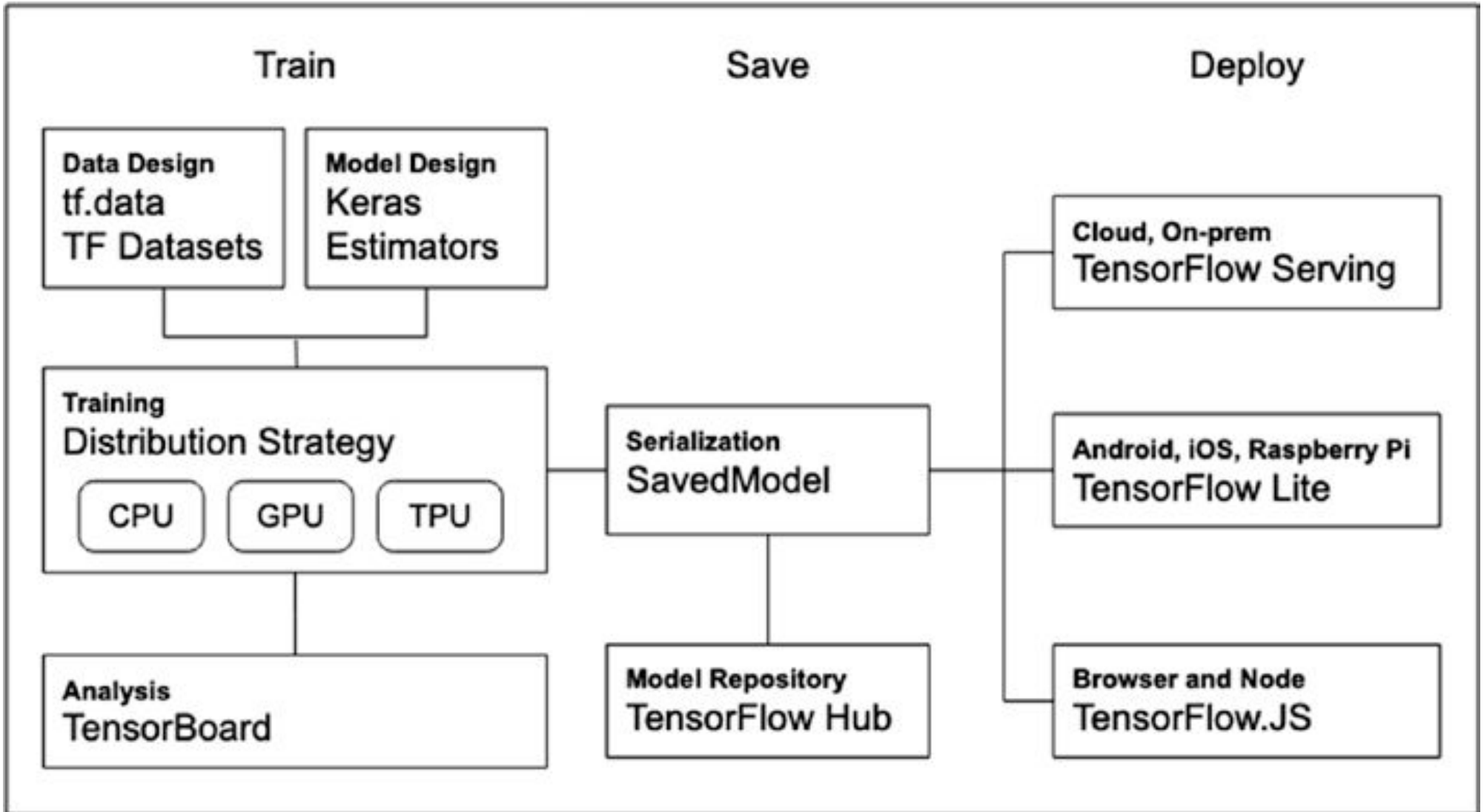
Deep Learning Framework Power Scores 2018



Deep Learning Framework Six-Month Growth Scores 2019



DL Framework — Ecosystem (Tensorflow Example)



DL Framework — **Components** (Tensorflow Example)

Hardware Environment (local, cloud, Edge Computing, ...):

- ◆ CPU

- ◆ GPU

- ◆ TPU

- ◆ Edge Computing Devices ...

Software Environment (local, cloud, Edge Computing, ...):

Operational System: macOS ($\geq 10.12.6$), Ubuntu (≥ 16.04), Windows (≥ 7), Raspbian (≥ 9.0), ...

Programming Language: C, C++, C#, Java, Go, Julia, Ruby, Scala, but ... Python 3

Libraries: ... numerous ...

DL Frameworks

-

Workflow

DL Workflow

(Tensorflow at Google Cloud Example)

- ◆ **Setup Hardware Environment** (at Google VM):
- ◆ select **Runtime Type**: CPU, GPU, TPU — **DEMO 1**.

Setup Software Environment:

Operational System: Ubuntu (pre-installed already).

Programming Language: Python 3 (pre-installed already)

Libraries: Python-libs (many pre-installed already)

Set up (get) dataset: local, cloud (AWS, GC, ... Kaggle, ...)

Get (define or load) model: local, TF Hub, cloud, ...

Compile (configure hyperparameters) model: loss function, optimization method, metrics, duration, callbacks, ...

Train and validate model -> Prediction -> Production? Not yet! :)

DEMO 1: Workflow in TF2

DEMO_1_Workflow_Example_CPU.ipynb

DEMO_1_Workflow_Example_GPU.ipynb

DEMO_1_Workflow_Example_TPU.ipynb

DEMO 2:

How to Monitor Workflow in TF2

DEMO_2_External_Data_Tensorboard_Binary_Classification_Example.ipynb

DL Workflow — Learning/Training Types

- **Learning from scratch:**
 - from **random initial** parameters (weights, ...)
 - from **previously trained** attempts.
- **Transfer learning** — is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.
- **Knowledge distillation** — learning is the process of transferring knowledge from a large model to a smaller one.
- **Federated learning** (also known as collaborative learning) is a machine learning technique that trains an algorithm across multiple decentralized edge devices or servers holding local data samples, without exchanging them.

DL Workflow — Learning/Training Types

— from scratch ... on datasets

Learning from scratch from **random initial** parameters (weights, ...) is very resource-demanding task.

Even for **MNIST** (60K images), it took a time to train the model up to the accuracy 80-90%. For a higher accuracy, more images would be required.

DNN learns better with a **higher volume** of data.

notMNIST, FashionMNIST, ...

<https://www.kaggle.com/yoctoman/graffiti-st-sophia-cathedral-kyiv>

CIFAR10, CIFAR100, ...

Microsoft Common Objects in Context (COCO),
PASCAL Visual Object Classes (PASCAL VOC),

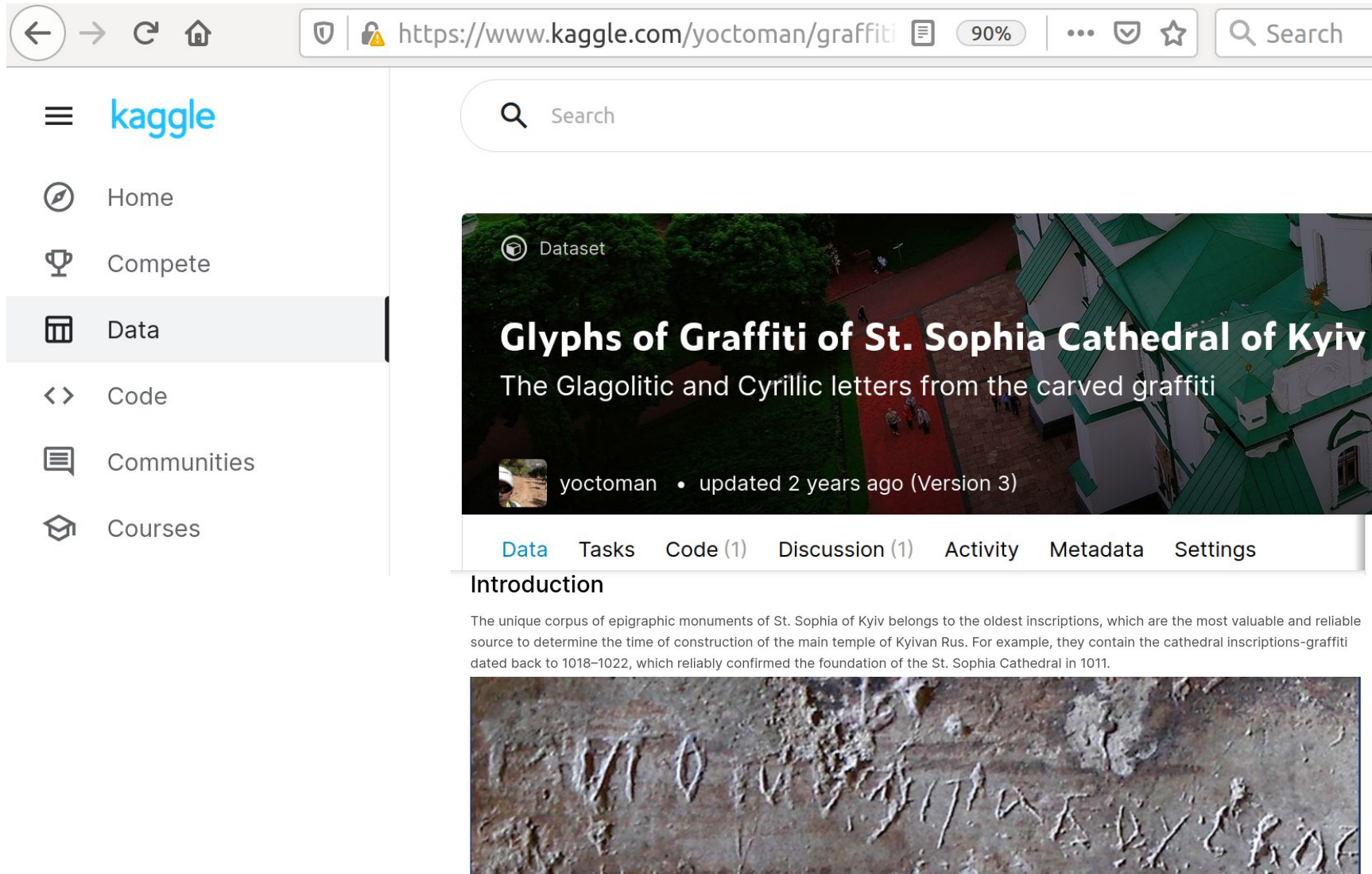
...

ImageNet !!!

DL Workflow — Learning/Training Types

— from scratch ... on datasets

One more dataset with ancient Cyrillic letters ... from Kyiv



The image shows a browser window displaying a Kaggle dataset page. The browser's address bar shows the URL <https://www.kaggle.com/yoctoman/graffiti>. The Kaggle logo is visible in the top left corner of the page. The main content area features a large header image of the St. Sophia Cathedral in Kyiv with the title "Glyphs of Graffiti of St. Sophia Cathedral of Kyiv" and the subtitle "The Glagolitic and Cyrillic letters from the carved graffiti". Below the header, the user "yoctoman" is credited, and it is noted that the dataset was updated 2 years ago (Version 3). A navigation bar includes links for "Data", "Tasks", "Code (1)", "Discussion (1)", "Activity", "Metadata", and "Settings". The "Introduction" section begins with the text: "The unique corpus of epigraphic monuments of St. Sophia of Kyiv belongs to the oldest inscriptions, which are the most valuable and reliable source to determine the time of construction of the main temple of Kyivan Rus. For example, they contain the cathedral inscriptions-graffiti dated back to 1018-1022, which reliably confirmed the foundation of the St. Sophia Cathedral in 1011." Below this text is a photograph of ancient graffiti on a stone surface, showing several lines of carved Cyrillic script.

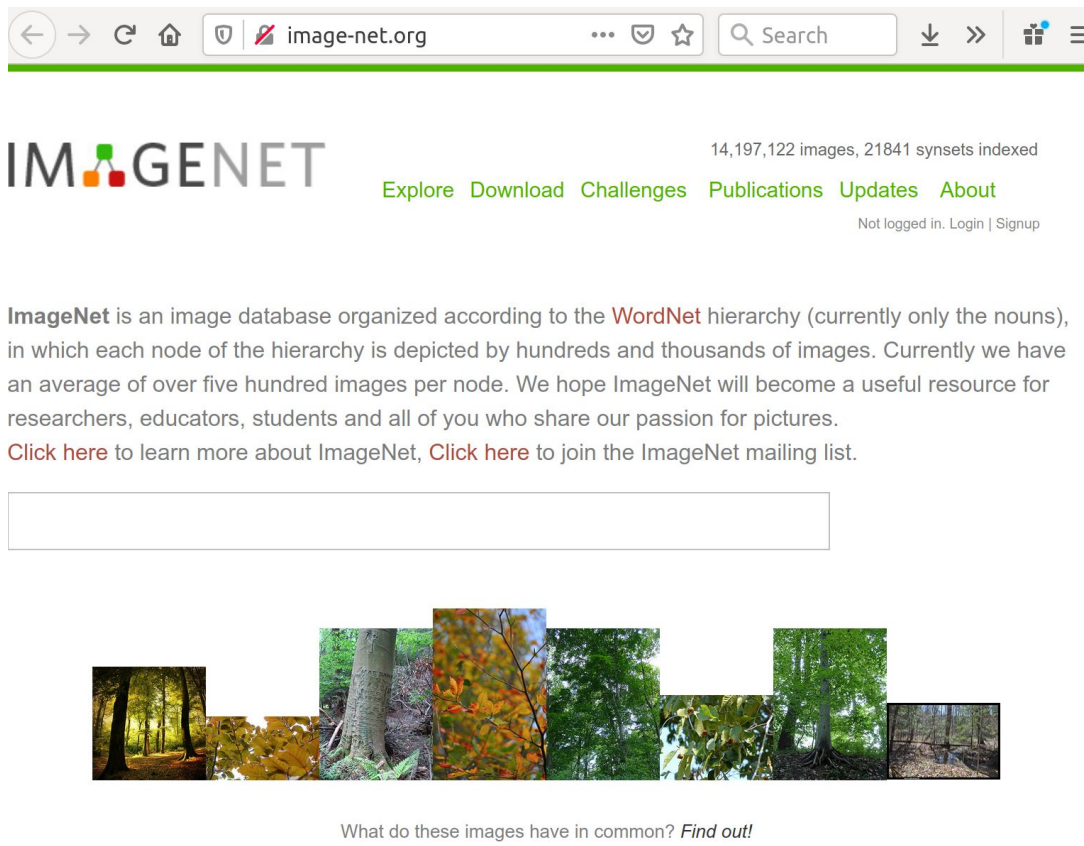
<https://www.kaggle.com/yoctoman/graffiti-st-sophia-cathedral-kyiv>

DL Workflow — Learning/Training Types

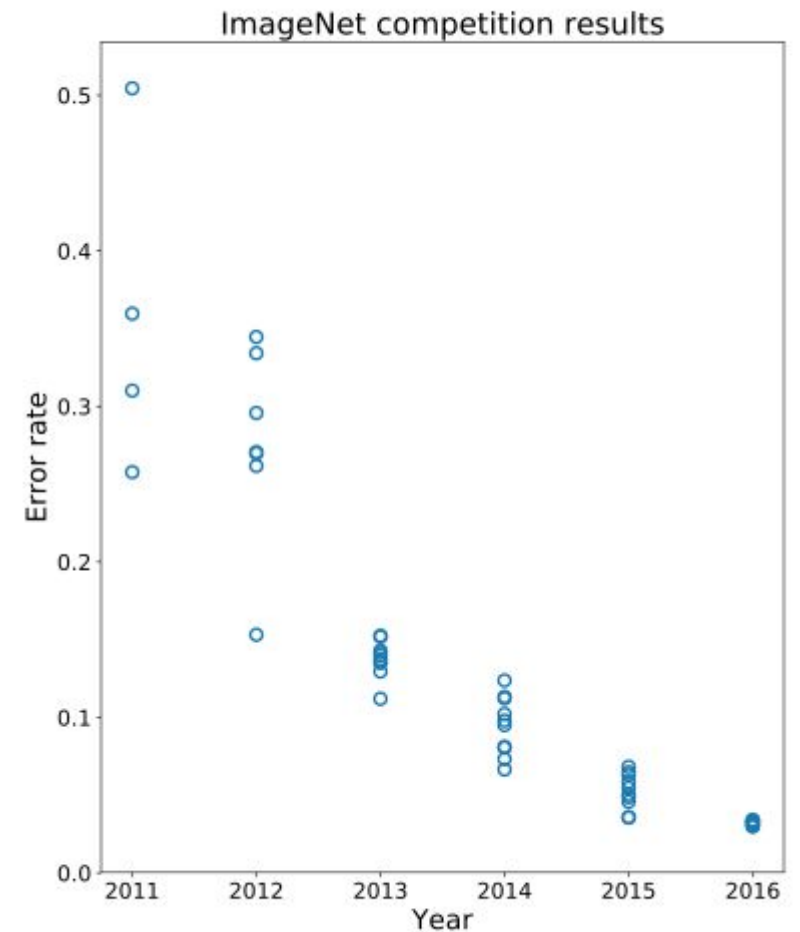
— from scratch ... on Imagenet

ImageNet (<http://image-net.org>): **14,197,122** images into **21,841** subcategories into **27** subtrees.

To classify the images in ImageNet, many ML/DL models were developed. In 2017, one model achieved an error rate of 2.3%.



The screenshot shows the ImageNet website homepage. At the top, the URL is image-net.org. The main header features the "IMAGENET" logo and the text "14,197,122 images, 21841 synsets indexed". Below this are navigation links: "Explore", "Download", "Challenges", "Publications", "Updates", and "About". A login/signup section indicates "Not logged in. Login | Signup". The main content area contains a paragraph describing ImageNet as an image database organized by the WordNet hierarchy, followed by a search bar and a row of image thumbnails. Below the thumbnails is the text: "What do these images have in common? Find out!"



DL Workflow — Learning/Training Types

— Transfer Learning

Transfer learning — is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.

1. Using a Pre-Trained Model

2. Training a Model to Reuse it

To solve task A you have limited data to train a DNN.
One way: to find a related task B with an abundance of data.
Train the DNN on task B and use the model as a starting point for solving task A.

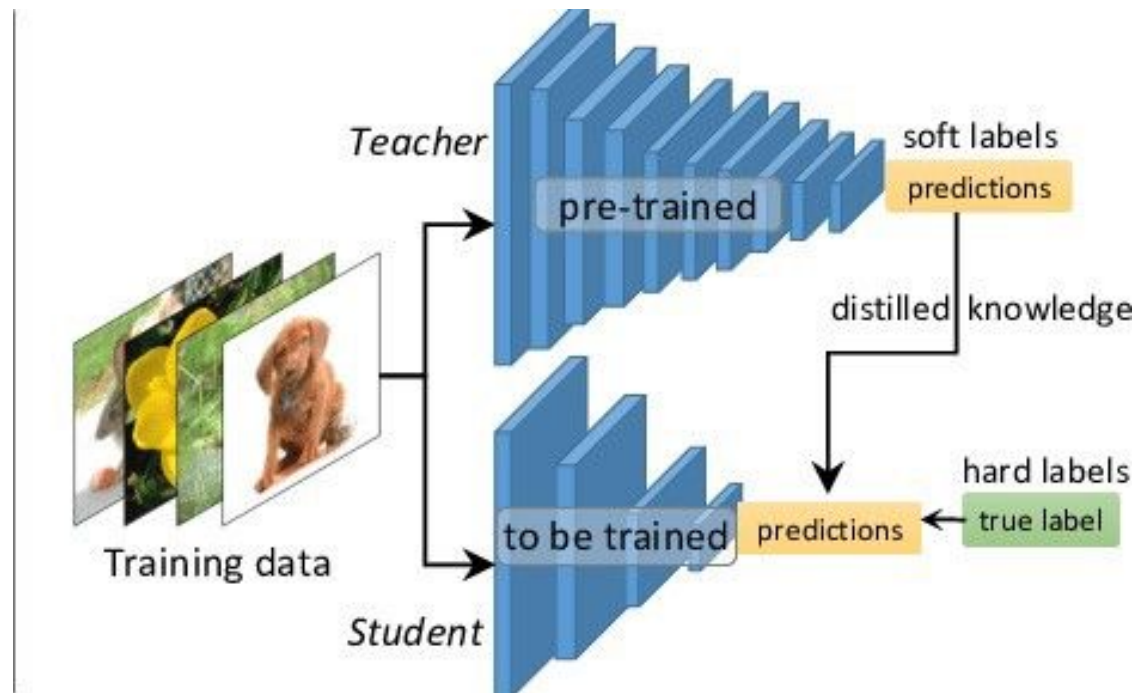
3. Feature Extraction

Another approach is to use DNN to find the best **representation** (most important features) of your problem, and use them.

DL Workflow — Learning/Training Types - Knowledge Distillation

A small model is trained to mimic a pre-trained, larger model (or ensemble of models). This training setting is sometimes referred to as “teacher-student”, where the **large** model is the **teacher** and the **small** model is the **student**.

It has even been observed that classifiers **learn** much **faster** and more **reliably** if trained with the **outputs** of another classifier as **soft** labels, instead of from hard labels (ground truth data).

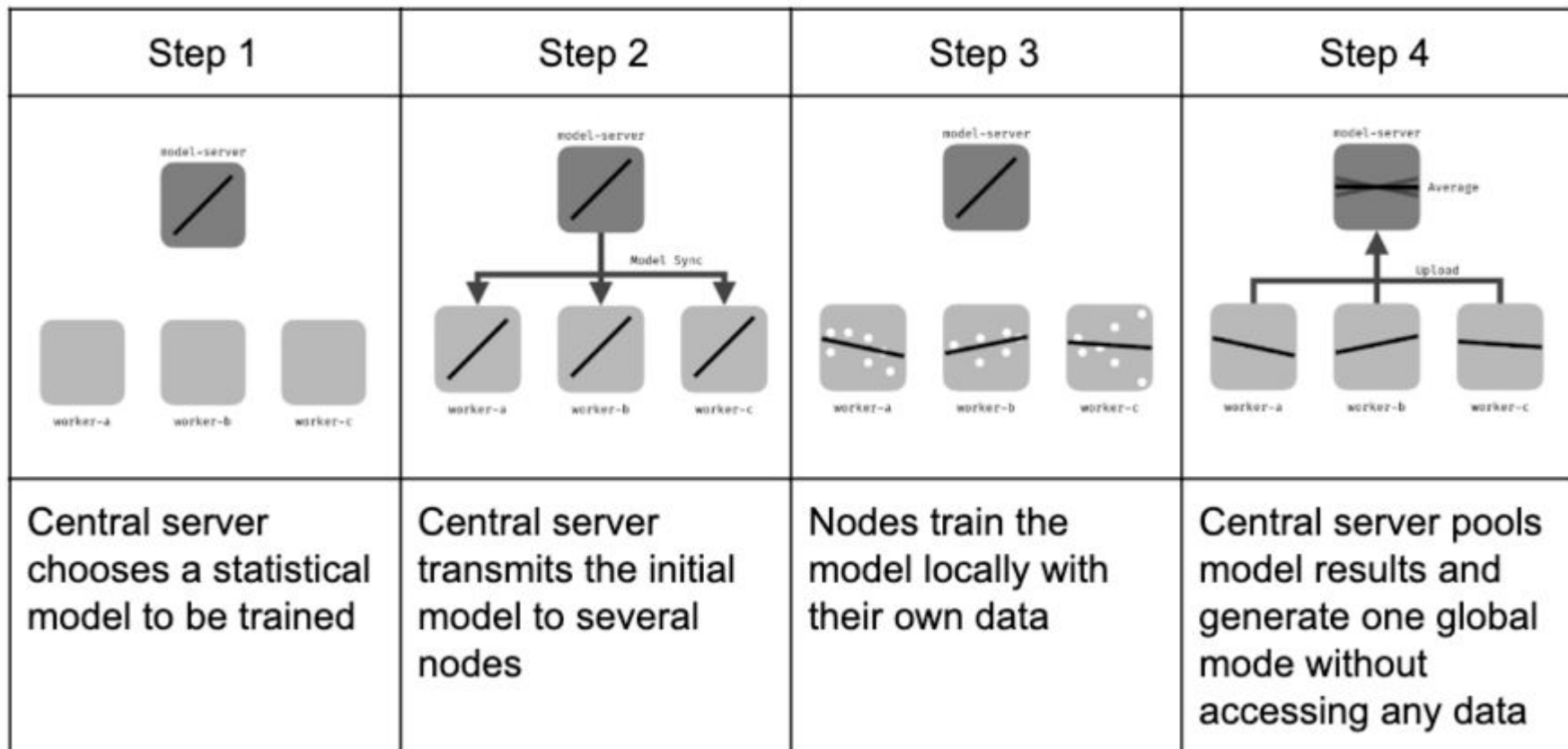


DL Workflow — Learning/Training Types

— Federated Learning

... is in contrast to

- traditional **centralized** learning techniques where all the **local** datasets are **uploaded to one server**,
- and some **classical decentralized** learning techniques where local data samples are identically distributed.



DL Workflow

-

Transfer Learning

DEMO 3A: Learning from Scratch in TF2

`DEMO_3A_Model_from_TF2_Keras_CNN_2_4_6_cifar10_imageClassification.ipynb`

DL Workflow — **Transfer Learning**

Transfer learning

— in general sense: is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned,

— in ML/DL context: it is an important technique of **knowledge transfer** from one to another **ML/DL task**.

Examples:

- ◆ In software engineering: people use binary libraries to reuse the code.
- ◆ In ML/DL: the trained models contain the **algorithms**, the **data**, the **processing power**, and the **expert's domain knowledge**. All these need to be **transferred** to the **new** model. That's what the transfer learning provides.

Transfer Learning — Model Sources

Pre-trained models can be found everywhere:

- from your **colleagues**,
- **github**,
- **Kaggle**,

...

but **Tensorflow Hub** is a the widest and easiest source of pre-trained and trustable DNN models.

Hub

RSVP for your your local TensorFlow Everywhere event today! [Find an event](#)

TensorFlow Hub is a repository of trained machine learning models.

TensorFlow Hub is a repository of trained machine learning models ready for fine-tuning and deployable anywhere. Reuse trained models like BERT and Faster R-CNN with just a few lines of code.

```
!pip install --upgrade tensor
import tensorflow_hub as hub
model = hub.KerasLayer("https
embeddings = model(["The rain
                    "mainly",
```

<https://www.tensorflow.org/hub>

DEMO 3B: Transfer Learning in TF2

`DEMO_3B_Model_from_TF2_Hub_MobileNetV2_ImageNet_imageClassification.ipynb`

Deep Learning Methods

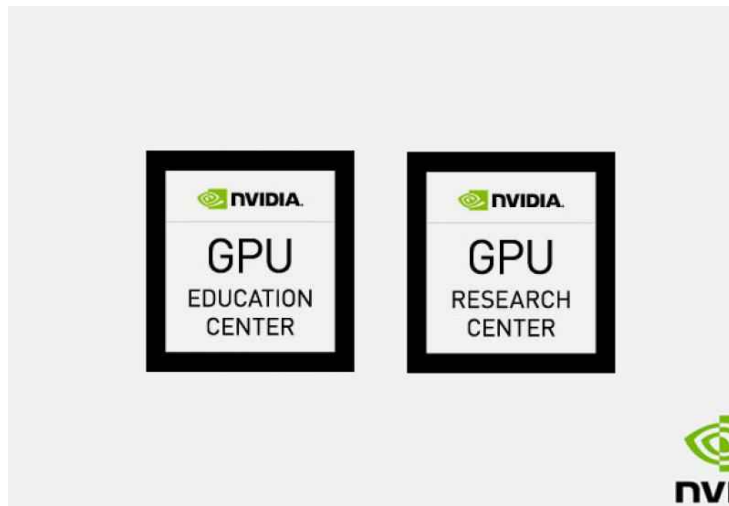
Lecture 07

Lecture Slides + interactive Jupyter-notebooks for Google Colaboratory CPU/GPU/TPU cloud:
<https://cloud.comsys.kpi.ua/s/SMkBSsxRTazoTD6>

Lecture 07 – Deep Learning Workflow - Estimators

The course includes materials proposed by NVIDIA Deep Learning Institute (DLI) in the framework of the common

NVIDIA Research Center
and
NVIDIA Education Center.



<https://kpi.ua/nvidia-info>

Interactive Demonstrations

DEMO A

Introduction to TF Estimators

<https://drive.google.com/file/d/10C-ypmitQmGkPQt-0oStL3OJGgSSvwww/view?usp=sharing>

DEMO B

Create DNN Model by TF Estimators

<https://drive.google.com/file/d/1fro49geaFUoQJ4frgLJy04FRuzNeNA7T/view?usp=sharing>

DEMO C

TF Datasets Benchmark by TF Estimators

<https://drive.google.com/file/d/1dALe-tX9pyMjNKXbBikM9L6sEv34uII4/view?usp=sharing>

DEMO D

Transfer Learning - Rock Paper Scissors (using NASNetMobile)

https://drive.google.com/file/d/1XvXxDE5OSArPgwZ_bcn3ACfFRWu5rBuV/view?usp=sharing

▼ DEMO A - Introduction to TF Estimators

based on (C) Velayudham, Sakranha, TF Authors works

```
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

▼ Connect to Google Drive

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

Mounted at /content/drive

```
! cp /content/drive/MyDrive/2022_COLAB_NN/Lecture_06_TF2_Estimators_CNN_RockPaperS
! ls
```

drive sample_data winequality-white.csv

▼ Loading Data

```
data_url='winequality-white.csv'
data=pd.read_csv(data_url,delimiter=';')
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956

▼ Selecting Features/Labels

```
x = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

```
sc = StandardScaler()
x = sc.fit_transform(x)
```

▼ Creating datasets

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3, random_state
```

```
input_shape = xtrain.shape[1]
```

▼ Defining simple FCN Keras model

```
small_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation = 'relu',
                           input_shape = (input_shape,)),
    tf.keras.layers.Dense(1)
])
```

```
small_model.compile(loss = 'mse', optimizer = 'adam')
```

```
def input_fn(features, labels, training = True, batch_size = 32):
    #converts inputs to a dataset
    dataset = tf.data.Dataset.from_tensor_slices(({'dense_input': features}, labels))

    #shuffle and repeat in a training mode
    if training:
        dataset = dataset.shuffle(1000).repeat()

    #giving inputs in batch for training
    return dataset.batch(batch_size)
```

▼ Convert Keras model to Estimator

```
keras_small_estimator = tf.keras.estimator.model_to_estimator(
    keras_model = small_model, model_dir = 'keras_small_classifier')
```

```
/usr/local/lib/python3.7/dist-packages/keras/backend.py:450: UserWarning: `tf
warnings.warn("`tf.keras.backend.set_learning_phase` is deprecated and '
```

▼ Train

```
keras_small_estimator.train(input_fn = lambda:input_fn(xtrain, ytrain), steps = 20
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/pyt
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
<tensorflow_estimator.python.estimator.estimator.EstimatorV2 at
0x7f96901753d0>
```

▼ Evaluate

```
eval_small_result = keras_small_estimator.evaluate(
    input_fn = lambda:input_fn(xtest, ytest, training = False), steps=1000)
print('Eval result: {}'.format(eval_small_result))
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training_v1.py:2057: User
updates = self.state_updates
Eval result: {'loss': 0.6073161, 'global_step': 2000}
```

▼ Analyze history and metrics

```
%load_ext tensorboard
%tensorboard --logdir ./keras_small_classifier
```

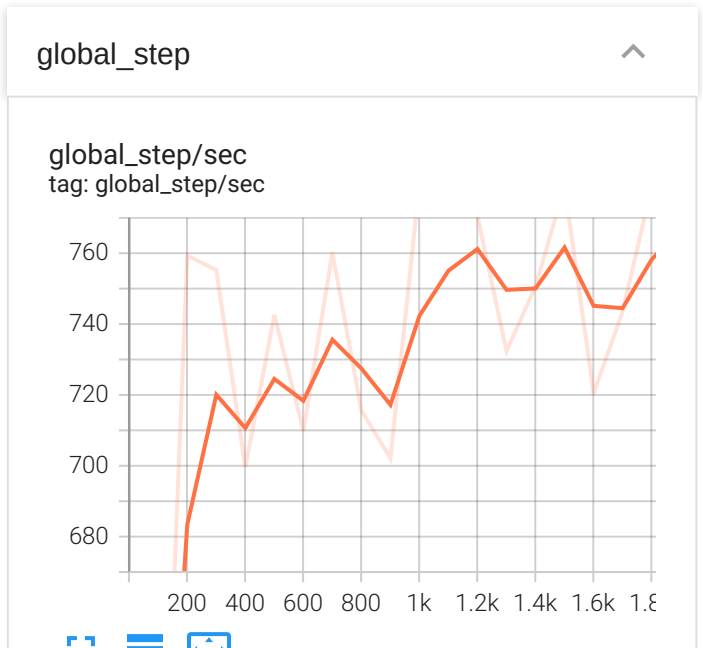
- Show data download links
- Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis
 STEP RELATIVE
 WALL

Filter tags (regular expressions supported)



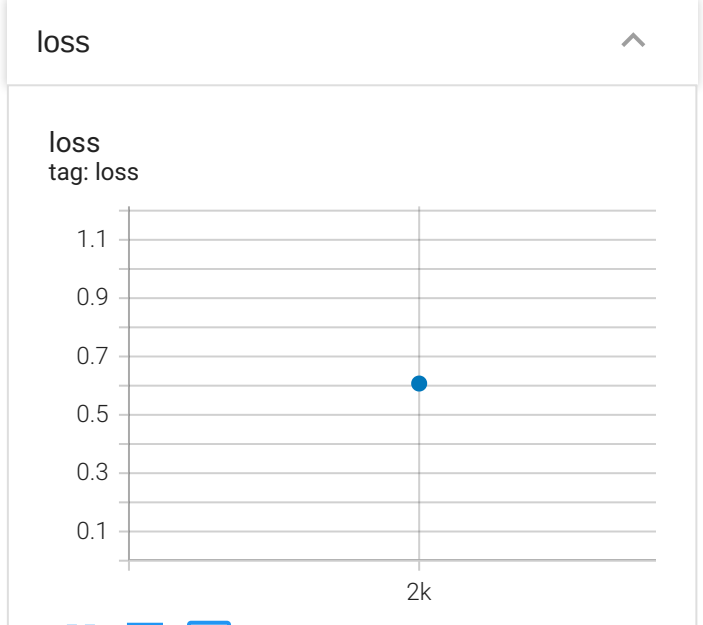
Write a regex to filter runs

.

eval

TOGGLE ALL RUNS

./keras_small_classifier



▼ DEMO B - Create DNN Model by TF Estimators

based on (C) Velayudham, Sakranha, TF Authors works

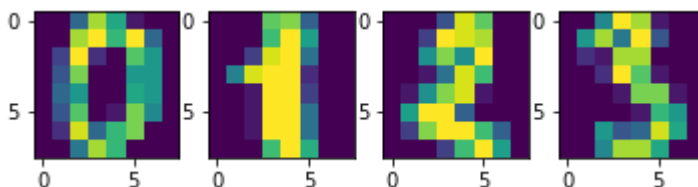
```
import tensorflow as tf
```

▼ Loading Data

```
from sklearn import datasets  
digits = datasets.load_digits()
```

```
#plotting sample image  
import matplotlib.pyplot as plt  
plt.figure(figsize=(1,1))  
fig, ax = plt.subplots(1,4)  
ax[0].imshow(digits.images[0])  
ax[1].imshow(digits.images[1])  
ax[2].imshow(digits.images[2])  
ax[3].imshow(digits.images[3])  
plt.show()
```

<Figure size 72x72 with 0 Axes>



▼ Preprocessing Data

```
# reshape the data to two dimensions  
n_samples = len(digits.images)  
data = digits.images.reshape((n_samples, -1))  
data.shape
```

```
(1797, 64)
```

```
# split into training/testing  
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(  
    data, digits.target, test_size=0.05, shuffle=False)
```


▼ Defining Input Function

```
# create column names for our model input function
columns = ['p_'+ str(i) for i in range(1,65)]
```

```
feature_columns = []
for col in columns:
    feature_columns.append(tf.feature_column.numeric_column(key = col))
```

```
def input_fn(features, labels, training = True, batch_size = 32):
    #converts inputs to a dataset
    dataset = tf.data.Dataset.from_tensor_slices((dict(features),labels))
    #shuffle and repeat in a training mode
    if training:
        dataset=dataset.shuffle(1000).repeat()

    #giving inputs in batches for training
    return dataset.batch(batch_size)
```

▼ Create DNNClassifier Estimator instance

```
classifier = tf.estimator.DNNClassifier(hidden_units = [256, 128, 64],
                                         feature_columns = feature_columns,
                                         optimizer = 'Adagrad',
                                         n_classes = 10,
                                         model_dir = 'classifier')
```

▼ Adding extra hidden layer

▼ without Dropout

```
classifier = tf.estimator.DNNClassifier(hidden_units = [256, 128, 64, 32],
                                         feature_columns = feature_columns,
                                         optimizer = 'Adagrad',
                                         n_classes = 10,
                                         model_dir = 'classifier')
```

▼ with Dropout

```

...
classifier = tf.estimator.DNNClassifier(hidden_units = [256, 128, 64, 32],
                                       feature_columns = feature_columns,
                                       optimizer = 'Adagrad',
                                       n_classes = 10,
                                       dropout = 0.2,
                                       model_dir = 'classifier')
...

```

```

\nclassifier = tf.estimator.DNNClassifier(hidden_units = [256, 12
8, 64, 32],\n
                                       feature_column
s = feature_columns,\n
                                       optim
izer = 'Adagrad'\n
                                       n_classes

```

▼ Model Training

```

# create dataframes for training
import pandas as pd
dftrain = pd.DataFrame(X_train, columns = columns)

```

```

classifier.train(input_fn = lambda:input_fn(dftrain,
                                           y_train,
                                           training = True),
                steps = 2000)

```

```

WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/pyt
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/keras/optimize
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to tf
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has r
<tensorflow_estimator.python.estimator.canned.dnn.DNNClassifierV2 at
0x7f988f1726d0>

```

▼ Model Evaluation

```

# create dataframe for evaluation
dfctest = pd.DataFrame(X_test, columns = columns)

eval_result = classifier.evaluate(
    input_fn = lambda:input_fn(dfctest, y_test, training = False)
)

eval_result

```

```

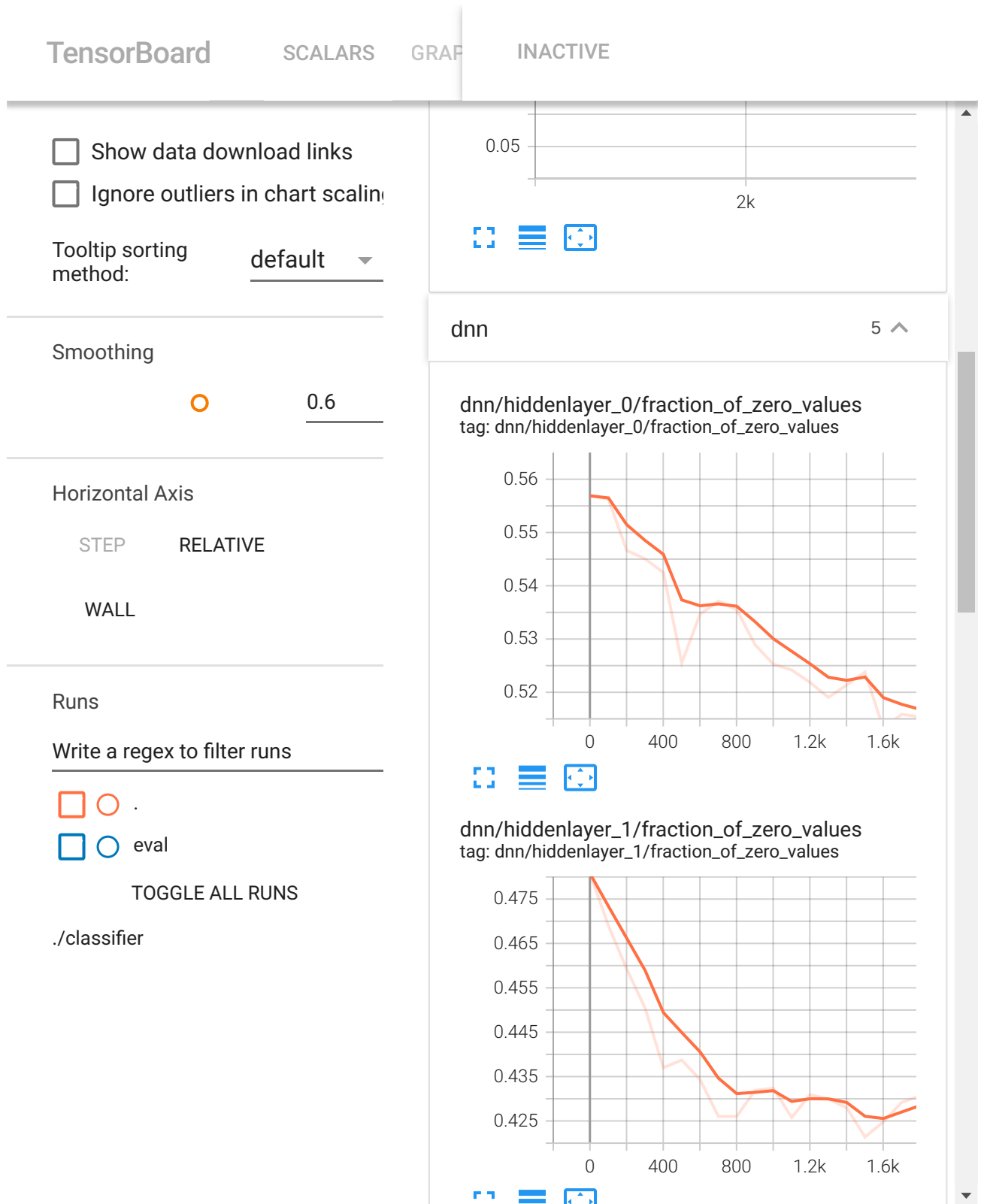
{'accuracy': 0.95555556,
 'average_loss': 0.2061766,

```

```
'loss': 0.19756849,  
'global_step': 2000}
```

▼ Resume at Tensorboard

```
%load_ext tensorboard  
%tensorboard --logdir ./classifier
```



▼ Predicting unseen data

```
# An input function for prediction
def pred_input_fn(features, batch_size = 32):
    # Convert the inputs to a Dataset without labels.
    return tf.data.Dataset.from_tensor_slices(dict(features)).batch(batch_size)

test = df_test.iloc[:2,:] #1st two data points for predictions

expected = y_test[:2].tolist() #expected labels

pred = list(classifier.predict(
    input_fn = lambda: pred_input_fn(test)
))

for pred_dict, expec in zip(pred, expected):
    class_id = pred_dict['class_ids'][0]
    probability = pred_dict['probabilities'][class_id]
    print('predicted class {} , probability of prediction {} , expected label {}'.format(class_id, probability, expec))

predicted class 8 , probability of prediction 0.9750990867614746 , expected label 1
predicted class 4 , probability of prediction 0.95527184009552 , expected label 4
```

▼ DEMO C - TF Datasets Benchmark by TF Estimators

based on (C) Velayudham, Sakranha, TF Authors works

```
# To avoid the compatibility issue with Tensorflow, cuda and models repo code.  
# Try installing the below TensorFlow version and cuda version at the start of col  
!pip install tensorflow==2.8  
!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
```

```
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packages (1.12.0)  
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (1.20.0)  
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (1.11.0)  
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (0.4.0)  
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (0.1.1)  
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (9.0.1)  
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (1.1.1)  
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (0.23.1)  
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (3.9.2)  
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (1.24.3)  
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (2.3.2)  
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (1.1.0)  
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (0.23.0)  
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (1.5.2)  
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (1.6.0)  
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (1.6.3)  
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (2.21.0)  
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (2.6.8)  
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (0.11.15)  
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (0.6.0)  
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (0.4.1)  
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (2.0.0)  
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (3.1.4)  
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (0.2.1)  
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (0.7.0)  
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (4.4)  
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (0.5)  
  
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (0.4.6)  
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (2.5)  
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (2.0.0)  
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (2017.4.17)  
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (1.21.1)  
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (3.0.0)  
Installing collected packages: tf-estimator-nightly, tensorboard, keras, tensorflow  
  Attempting uninstall: tensorboard  
    Found existing installation: tensorboard 2.10.1  
    Uninstalling tensorboard-2.10.1:  
      Successfully uninstalled tensorboard-2.10.1  
  Attempting uninstall: keras  
    Found existing installation: keras 2.10.0  
    Uninstalling keras-2.10.0:  
      Successfully uninstalled keras-2.10.0  
  Attempting uninstall: tensorflow  
    Found existing installation: tensorflow 2.10.0  
    Uninstalling tensorflow-2.10.0:  
      Successfully uninstalled tensorflow-2.10.0
```

```
ERROR: pip's dependency resolver does not currently take into account all t
tfx-bsl 1.10.1 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,!=2.4.*,!
tensorflow-serving-api 2.10.0 requires tensorflow<3,>=2.10.0, but you have
tensorflow-data-validation 1.10.0 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.
Successfully installed keras-2.8.0 tensorboard-2.8.0 tensorflow-2.8.0+zzzco
Reading package lists... Done
Building dependency tree
Reading state information... Done
libcudnn8 is already the newest version (8.1.0.77-1+cuda11.2).
The following package was automatically installed and is no longer required
  libnvidia-common-460
```

```
import tensorflow as tf
import tensorflow_datasets as tfds
```

▼ TensorFlow Datasets

```
#To get the list of available
tfds.list_builders()

'huggingface:psc',
'huggingface:ptb_text_only',
'huggingface:pubmed',
'huggingface:pubmed_qa',
'huggingface:py_ast',
'huggingface:qa4mre',
'huggingface:qa_srl',
'huggingface:qa_zre',
'huggingface:qangaroo',
'huggingface:qanta',
'huggingface:qasc',
'huggingface:qasper',
'huggingface:qed',
'huggingface:qed_amara',
'huggingface:quac',
'huggingface:quail',
'huggingface:quarel',
'huggingface:quartz',
'huggingface:quickdraw',
'huggingface:quora',
'huggingface:quoref',
'huggingface:race',
'huggingface:re_dial',
'huggingface:reasoning_bg',
'huggingface:recipe_nlg',
'huggingface:reclor',
'huggingface:red_caps',
'huggingface:reddit',
'huggingface:reddit_tifu',
'huggingface:refresd',
'huggingface:reuters21578',
'huggingface:riddle_sense',
'huggingface:ro_sent',
'huggingface:ro_sts',
'huggingface:ro_sts_parallel',
'huggingface:roman_urdu'.
```

```
'huggingface:roman_urdu_hate_speech',
'huggingface:ronec',
'huggingface:ropes',
'huggingface:rotten_tomatoes',
'huggingface:russian_super_glue',
'huggingface:rvl_cdip',
'huggingface:s2orc',
'huggingface:samsum',
'huggingface:sanskrit_classic',
'huggingface:saudinewsnet',
'huggingface:sberquad',
'huggingface:sbu_captions',
'huggingface:scan',
'huggingface:scb_mt_enth_2020',
'huggingface:scene_parse_150',
'huggingface:schema_guided_dstc8',
'huggingface:scicite',
'huggingface:scielo',
'huggingface:scientific_papers',
'huggingface:scifact',
'huggingface:sciq',
'huggingface:scitail',
'huggingface:scitldr'
```

```
len(tfds.list_builders())
```

1122

```
#ds, ds_info = tfds.load('cifar10', split='train', with_info=True)
ds, ds_info = tfds.load('fashion_mnist', split='train', with_info=True)
fig = tfds.show_examples(ds, ds_info)
```



▼ Benchmark your datasets

Note: This API is new and only available via

! pip install tfds-nightly

```
#! pip install tfds-nightly
```

```
! nvidia-smi
```

```
Mon Oct 3 18:41:21 2022
```

```
+-----+
| NVIDIA-SMI 460.32.03      Driver Version: 460.32.03      CUDA Version: 11.2
+-----+-----+-----+
| GPU  Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M.
|====+=====+====+=====+
|   0  Tesla T4             Off          | 00000000:00:04.0 Off  |
| N/A   61C    P0      30W / 70W | 286MiB / 15109MiB |      0%      Default
+-----+-----+-----+-----+
|
| Processes:
| GPU   GI    CI          PID    Type   Process name                      GPU Memory
|      ID    ID                                   |              Usage
+-----+-----+-----+-----+
|
|
```

```
+-----+
| Processes:
| GPU   GI    CI          PID    Type   Process name                      GPU Memory
|      ID    ID                                   |              Usage
+-----+-----+-----+-----+
|
|
```



```
# Benchmark your datasets - GPU
```

```
ds = ds.batch(32).prefetch(1)
```

```
tfds.benchmark(ds, batch_size=32)
```

```
tfds.benchmark(ds, batch_size=32) # Second epoch much faster due to auto-caching
```



```
***** Summary *****
```

```
100% 1875/1875 [00:04<00:00, 257.26it/s]
Examples/sec (First included) 12025.68 ex/sec (total: 60032 ex, 4.9
Examples/sec (First only) 364.56 ex/sec (total: 32 ex, 0.09 sec)
Examples/sec (First excluded) 12234.40 ex/sec (total: 60000 ex, 4.9
```

```
***** Summary *****
```

```
100% 1875/1875 [00:00<00:00, 2256.08it/s]
```

```
# Benchmark your datasets - TPU
```

```
ds = ds.batch(32).prefetch(1)
```

```
tfds.benchmark(ds, batch_size=32)
```

```
tfds.benchmark(ds, batch_size=32) # Second epoch much faster due to auto-caching
```

```
***** Summary *****
```

```
100% 59/59 [00:00<00:00, 125.63it/s]
Examples/sec (First included) 3414.61 ex/sec (total: 1920 ex, 0.56
Examples/sec (First only) 249.06 ex/sec (total: 32 ex, 0.13 sec)
Examples/sec (First excluded) 4352.14 ex/sec (total: 1888 ex, 0.43
```

```
***** Summary *****
```

```
100% 59/59 [00:00<00:00, 174.71it/s]
Examples/sec (First included) 4876.42 ex/sec (total: 1920 ex, 0.39
Examples/sec (First only) 269.12 ex/sec (total: 32 ex, 0.12 sec)
Examples/sec (First excluded) 6869.81 ex/sec (total: 1888 ex, 0.27
```

```
BenchmarkResult:
```

	duration	num_examples	avg
first+lasts	0.393731	1920	4876.422340
first	0.118906	32	269.121261

```
#! pip install tensorflow_data_validation
```

```
# Display the datasets statistics on a Colab/Jupyter notebook
```

```
tfds.show_statistics(ds_info)
```

Sort by
 Feature order Reverse order
 Feature search (regex enab...
 Features: fixed-length ints(2)

Numeric Features (2)							
	count	missing	mean	std dev	zeros	min	med
image	60.0k	0%	0	0	0%	0	
label							

▼ Import VGG16 module

```
keras_Vgg16 = tf.keras.applications.VGG16(
    input_shape=(160, 160, 3), include_top=False)
keras_Vgg16.trainable = False
```

▼ Create Keras model by adding layers to VGG16 model

```
estimator_model = tf.keras.Sequential([
    keras_Vgg16,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256),
    tf.keras.layers.Dense(1)
])
```

```
keras_Vgg16.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584

```

block2_pool (MaxPooling2D) (None, 40, 40, 128) 0
block3_conv1 (Conv2D) (None, 40, 40, 256) 295168
block3_conv2 (Conv2D) (None, 40, 40, 256) 590080
block3_conv3 (Conv2D) (None, 40, 40, 256) 590080
block3_pool (MaxPooling2D) (None, 20, 20, 256) 0
block4_conv1 (Conv2D) (None, 20, 20, 512) 1180160
block4_conv2 (Conv2D) (None, 20, 20, 512) 2359808
block4_conv3 (Conv2D) (None, 20, 20, 512) 2359808
block4_pool (MaxPooling2D) (None, 10, 10, 512) 0
block5_conv1 (Conv2D) (None, 10, 10, 512) 2359808
block5_conv2 (Conv2D) (None, 10, 10, 512) 2359808
block5_conv3 (Conv2D) (None, 10, 10, 512) 2359808
block5_pool (MaxPooling2D) (None, 5, 5, 512) 0

```

```

=====
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688

```

```
estimator_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 5, 5, 512)	14714688
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 256)	131328
dense_1 (Dense)	(None, 1)	257

```

=====
Total params: 14,846,273
Trainable params: 131,585
Non-trainable params: 14,714,688

```

▼ Compile

```
# Compile the model
estimator_model.compile(
    optimizer = 'adam',
    loss=tf.keras.losses.BinaryCrossentropy(from_logits = True),
    metrics = ['accuracy'])
```

▼ Create Estimator

```
est_vgg16 = tf.keras.estimator.model_to_estimator(keras_model = estimator_model,
                                                  model_dir = 'classifier')
```

▼ Data preprocessing

```
IMG_SIZE = 160
import tensorflow_datasets as tfds
def preprocess(image, label):
    image = tf.cast(image, tf.float32)
    #image = (image/127.5) - 1
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    return image, label
```

▼ Input function

```
def train_input_fn(batch_size):
    data = tfds.load('cats_vs_dogs', as_supervised=True)
    train_data = data['train']
    train_data = train_data.map(preprocess).shuffle(500).batch(batch_size)
    return train_data
```

▼ Training

```
#!/usr/bin/env python
import tensorflow as tf
est_vgg16.train(input_fn = lambda: train_input_fn(32), steps = 500)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/resource_variable_ops.py:435:
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both
/usr/local/lib/python3.7/dist-packages/keras/backend.py:450: UserWarning: `tf
  warnings.warn("`tf.keras.backend.set_learning_phase` is deprecated and '
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/resource_variable_ops.py:435:
Instructions for updating:
Use standard file utilities to get mtimes.
<tensorflow_estimator.python.estimator.estimator.EstimatorV2 at
0x7fba65713650>
```

▼ Evaluation

```
est_vgg16.evaluate(input_fn = lambda: train_input_fn(32), steps=10)
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training_v1.py:2057: UserWarning: Using the training_loss attribute of the Model object is deprecated and will be removed in a future version. Use the loss attribute instead.  
updates = self.state_updates  
{'accuracy': 0.934375, 'loss': 0.3812843, 'global_step': 500}
```

▼ Monitoring

```
%load_ext tensorboard  
%tensorboard --logdir ./classifier
```

- Show data download links
- Ignore outliers in chart scaling

Filter tags (regular expressions su...

Tooltip sorting default

accuracy

Smoothing 0.6

loss

loss_1

Horizontal Axis
STEP RELATIVE

WALL

Runs

Write a regex to filter runs

eval

TOGGLE ALL RUNS

./classifier



0s completed at 9:42 PM



▼ DEMO D - Transfer Learning - Rock Paper Scissors (using NASNetMobile)

based on (C) Ng, Moroney, Mingxing Tan, Quoc V. Le, Rowlani, and **Oleksii Trekhleb** ("Our Man in Uber" :))

▼ Experiment overview

In this experiment we will build a [Convolutional Neural Network](#) (CNN) model using [Tensorflow](#) to recognize Rock-Paper-Scissors signs (gestures) on the photo.

Instead of training the model from scratch we will use **transfer learning** method (look at DEMOs in previous lectures) a family of [TF2-Keras-Applications](#) models. Here we will actually use NASNetMobile and other models which are pre-trained on the [ImageNet](#) dataset, a large dataset of 1.4M images and 1000 classes of web images.

▼ Importing dependencies

```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
import platform
import datetime
import os
import math
import random

print('Python version:', platform.python_version())
print('Tensorflow version:', tf.__version__)
print('Keras version:', tf.keras.__version__)
```

```
Python version: 3.7.14
Tensorflow version: 2.8.2
Keras version: 2.8.0
```

▼ Configuring TensorBoard

We will use TensorBoard as a helper to debug the model training process.

```
# Load the TensorBoard notebook extension.
```

```
# %reload_ext tensorboard
%load_ext tensorboard

# Clear any logs from previous runs.
!rm -rf ./logs/
```

▼ Dataset - Example

We will download Rock-Paper-Scissors dataset from [TensorFlow Datasets](#) collection. To do that we loaded a `tensorflow_datasets` module.

`tensorflow_datasets` defines a collection of datasets ready-to-use with TensorFlow.

Each dataset is defined as a [tfds.core.DatasetBuilder](#), which encapsulates the logic to download the dataset and construct an input pipeline, as well as contains the dataset documentation (version, splits, number of examples, etc.).

```
# See available datasets
tfds.list_builders()

'huggingface:quac',
'huggingface:quail',
'huggingface:quarel',
'huggingface:quartz',
'huggingface:quickdraw',
'huggingface:quora',
'huggingface:quoref',
'huggingface:race',
'huggingface:re_dial',
'huggingface:reasoning_bg',
'huggingface:recipe_nlg',
'huggingface:reclor',
'huggingface:red_caps',
'huggingface:reddit',
'huggingface:reddit_tifu',
'huggingface:refresd',
'huggingface:reuters21578',
'huggingface:riddle_sense',
'huggingface:ro_sent',
'huggingface:ro_sts',
'huggingface:ro_sts_parallel',
'huggingface:roman_urdu',
'huggingface:roman_urdu_hate_speech',
'huggingface:ronec',
'huggingface:ropes',
'huggingface:rotten_tomatoes',
'huggingface:russian_super_glue',
'huggingface:rvl_cdip',
'huggingface:s2orc',
'huggingface:samsum',
'huggingface:sanskrit_classic',
'huggingface:saudinewsnet',
'huggingface:sberquad',
'huggingface:sbu_captions',
'huggingface:scan',
'huggingface:scb_mt_enth_2020'.
```



```
'huggingface:scene_parse_150',  
'huggingface:schema_guided_dstc8',  
'huggingface:scicite',  
'huggingface:scielo',  
'huggingface:scientific_papers',  
'huggingface:scifact',  
'huggingface:sciq',  
'huggingface:scitail',  
'huggingface:scitldr',  
'huggingface:search_qa',  
'huggingface:sede',  
'huggingface:selqa',  
'huggingface:sem_eval_2010_task_8',  
'huggingface:sem_eval_2014_task_1',  
'huggingface:sem_eval_2018_task_1',  
'huggingface:sem_eval_2020_task_11',  
'huggingface:sent_comp',  
'huggingface:senti_lex',  
'huggingface:senti_ws',  
'huggingface:sentiment140',  
'huggingface:sepedi_ner',  
'huggingface:sesotho_ner_corpus',
```

We will use the classic dataset by Moroney:

- Title: rock_paper_scissors
- Description: Images of hands playing rock, paper, scissor game.
- Homepage: <http://laurencemoroney.com/rock-paper-scissors-dataset>
- Source code: `tfds.image_classification.RockPaperScissors`
- Versions: 3.0.0 (default): New split API (<https://tensorflow.org/datasets/splits>)
- Download size: 219.53 MiB
- Image Examples:



Rock



Paper



Scissors

▼ Loading the dataset

```
DATASET_NAME = 'rock_paper_scissors'  
  
(dataset_train_raw, dataset_test_raw), dataset_info = tfds.load(  
    name=DATASET_NAME,  
    data_dir='tmp',  
    with_info=True,  
    as_supervised=True,  
    split=[tfds.Split.TRAIN, tfds.Split.TEST],  
)
```

Downloading and preparing dataset 219.53 MiB (download: 219.53 MiB, generated

DI Completed...: 100% 2/2 [00:05<00:00, 2.84s/ url]

DI Size...: 100% 219/219 [00:05<00:00, 46.67 MiB/s]

Dataset rock paper scissors downloaded and prepared to tmp/rock paper scissor

```

print('Raw train dataset:', dataset_train_raw)
print('Raw train dataset size:', len(list(dataset_train_raw)), '\n')

print('Raw test dataset:', dataset_test_raw)
print('Raw test dataset size:', len(list(dataset_test_raw)), '\n')

```

```

Raw train dataset: <PrefetchDataset element_spec=(TensorSpec(shape=(300, 300,
Raw train dataset size: 2520

```

```

Raw test dataset: <PrefetchDataset element_spec=(TensorSpec(shape=(300, 300,
Raw test dataset size: 372

```



dataset_info

```

tfds.core.DatasetInfo(
  name='rock_paper_scissors',
  full_name='rock_paper_scissors/3.0.0',
  description="""
Images of hands playing rock, paper, scissor game.
""",
  homepage='http://laurencemoroney.com/rock-paper-scissors-dataset',
  data_path='tmp/rock_paper_scissors/3.0.0',
  file_format=tfrecord,
  download_size=219.53 MiB,
  dataset_size=219.23 MiB,
  features=FeaturesDict({
    'image': Image(shape=(300, 300, 3), dtype=tf.uint8),
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=3),
  }),
  supervised_keys=('image', 'label'),
  disable_shuffling=False,
  splits={
    'test': <SplitInfo num_examples=372, num_shards=1>,
    'train': <SplitInfo num_examples=2520, num_shards=2>,
  },
  citation="""@ONLINE {rps,
author = "Laurence Moroney",
title = "Rock, Paper, Scissors Dataset",
month = "feb",
year = "2019",
url = "http://laurencemoroney.com/rock-paper-scissors-dataset"
}""",
)

```

```

NUM_TRAIN_EXAMPLES = dataset_info.splits['train'].num_examples
NUM_TEST_EXAMPLES = dataset_info.splits['test'].num_examples
NUM_CLASSES = dataset_info.features['label'].num_classes

```

```

print('Number of TRAIN examples:', NUM_TRAIN_EXAMPLES)
print('Number of TEST examples:', NUM_TEST_EXAMPLES)
print('Number of label classes:', NUM_CLASSES)

```

```

Number of TRAIN examples: 2520
Number of TEST examples: 372

```

Number of label classes: 3

```
INPUT_IMG_SIZE_ORIGINAL = dataset_info.features['image'].shape[0]
INPUT_IMG_SHAPE_ORIGINAL = dataset_info.features['image'].shape

# For some models only some sizes are possible, for example:
# for NASNetMobile - 224, ...
INPUT_IMG_SIZE_REDUCED = 224
INPUT_IMG_SHAPE_REDUCED = (
    INPUT_IMG_SIZE_REDUCED,
    INPUT_IMG_SIZE_REDUCED,
    INPUT_IMG_SHAPE_ORIGINAL[2]
)

# Here we may switch between bigger or smaller image sized that we will train our
INPUT_IMG_SIZE = INPUT_IMG_SIZE_REDUCED
INPUT_IMG_SHAPE = INPUT_IMG_SHAPE_REDUCED

print('Input image size (original):', INPUT_IMG_SIZE_ORIGINAL)
print('Input image shape (original):', INPUT_IMG_SHAPE_ORIGINAL)
print('\n')
print('Input image size (reduced):', INPUT_IMG_SIZE_REDUCED)
print('Input image shape (reduced):', INPUT_IMG_SHAPE_REDUCED)
print('\n')
print('Input image size:', INPUT_IMG_SIZE)
print('Input image shape:', INPUT_IMG_SHAPE)
```

```
Input image size (original): 300
Input image shape (original): (300, 300, 3)
```

```
Input image size (reduced): 224
Input image shape (reduced): (224, 224, 3)
```

```
Input image size: 224
Input image shape: (224, 224, 3)
```

```
# Function to convert label ID to labels string.
get_label_name = dataset_info.features['label'].int2str
```

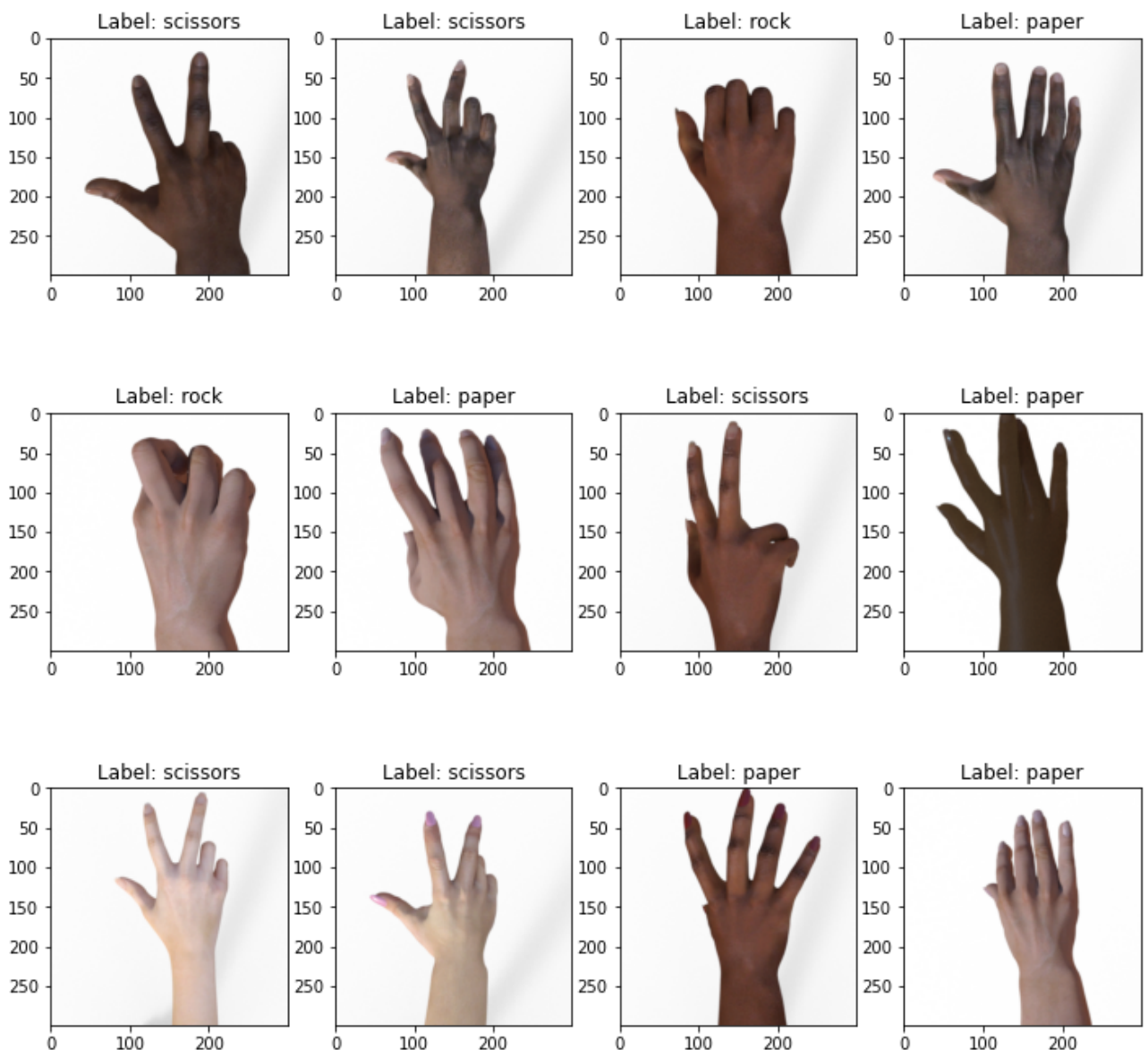
```
print(get_label_name(0));
print(get_label_name(1));
print(get_label_name(2));
```

```
rock
paper
scissors
```

▼ Exploring the dataset

```
def preview_dataset(dataset):
    plt.figure(figsize=(12, 12))
    plot_index = 0
    for features in dataset.take(12):
        (image, label) = features
        plot_index += 1
        plt.subplot(3, 4, plot_index)
        # plt.axis('Off')
        label = get_label_name(label.numpy())
        plt.title('Label: %s' % label)
        plt.imshow(image.numpy())
```

```
# Explore raw training dataset images.
preview_dataset(dataset_train_raw)
```



```
# Explore what values are used to represent the image.
(first_image, first_label) = list(dataset_train_raw.take(1))[0]
print('Label:', first_label.numpy(), '\n')
```

```
print('Image shape:', first_image.numpy().shape, '\n')
print(first_image.numpy())
```

Label: 2

Image shape: (300, 300, 3)

```
[[[254 254 254]
  [253 253 253]
  [254 254 254]
  ...
  [251 251 251]
  [250 250 250]
  [250 250 250]]

 [[254 254 254]
  [254 254 254]
  [253 253 253]
  ...
  [250 250 250]
  [251 251 251]
  [249 249 249]]

 [[254 254 254]
  [254 254 254]
  [254 254 254]
  ...
  [251 251 251]
  [250 250 250]
  [252 252 252]]

 ...

 [[252 252 252]
  [251 251 251]
  [252 252 252]
  ...
  [247 247 247]
  [249 249 249]
  [248 248 248]]

 [[253 253 253]
  [253 253 253]
  [251 251 251]
  ...
  [248 248 248]
  [248 248 248]
  [248 248 248]]

 [[252 252 252]
  [253 253 253]
  [252 252 252]
  ...
  [248 248 248]
  [247 247 247]
  [250 250 250]]]
```

▼ Pre-processing the dataset

```
def format_example(image, label):
    # Make image color values to be float.
    image = tf.cast(image, tf.float32)
    # Make image color values to be in [0..1] range.
    image = image / 255.
    # Make sure that image has a right size
    image = tf.image.resize(image, [INPUT_IMG_SIZE, INPUT_IMG_SIZE])
    return image, label
```

```
dataset_train = dataset_train_raw.map(format_example)
dataset_test = dataset_test_raw.map(format_example)
```

```
# Explore what values are used to represent the image.
(first_image, first_lable) = list(dataset_train.take(1))[0]
print('Label:', first_lable.numpy(), '\n')
print('Image shape:', first_image.numpy().shape, '\n')
print(first_image.numpy())
```

Label: 2

Image shape: (224, 224, 3)

```
[[[0.995526  0.995526  0.995526 ]
  [0.9941408 0.9941408 0.9941408 ]
  [0.99597746 0.99597746 0.99597746]
  ...
  [0.9869748 0.9869748 0.9869748 ]
  [0.98237604 0.98237604 0.98237604]
  [0.97995263 0.97995263 0.97995263]]

 [[0.99607843 0.99607843 0.99607843]
  [0.99509835 0.99509835 0.99509835]
  [0.99578613 0.99578613 0.99578613]
  ...
  [0.98232853 0.98232853 0.98232853]
  [0.98235357 0.98235357 0.98235357]
  [0.9824342  0.9824342  0.9824342  ]]

 [[0.99607843 0.99607843 0.99607843]
  [0.99438554 0.99438554 0.99438554]
  [0.9955736  0.9955736  0.9955736  ]
  ...
  [0.982799   0.982799   0.982799   ]
  [0.97900224 0.97900224 0.97900224]
  [0.98414266 0.98414266 0.98414266]]

 ...

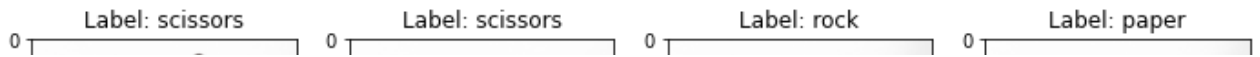
 [[0.9886986  0.9886986  0.9886986 ]
  [0.98788357 0.98788357 0.98788357]
  [0.98773044 0.98773044 0.98773044]
  ...
```

```
[0.97477514 0.97477514 0.97477514]
[0.9725384 0.9725384 0.9725384 ]
[0.96988803 0.96988803 0.96988803]]

[[0.98982257 0.98982257 0.98982257]
 [0.9872209 0.9872209 0.9872209 ]
 [0.98630947 0.98630947 0.98630947]
 ...
 [0.9689198 0.9689198 0.9689198 ]
 [0.97251344 0.97251344 0.97251344]
 [0.9728876 0.9728876 0.9728876 ]]

[[0.98945296 0.98945296 0.98945296]
 [0.9898225 0.9898225 0.9898225 ]
 [0.98757 0.98757 0.98757 ]
 ...
 [0.9692227 0.9692227 0.9692227 ]
 [0.9709499 0.9709499 0.9709499 ]
 [0.9774043 0.9774043 0.9774043 ]]]
```

```
# Explore preprocessed training dataset images.
preview_dataset(dataset_train)
```

▼ Data augmentation

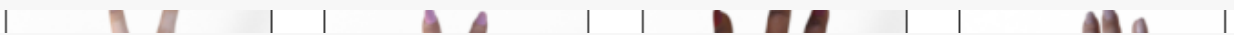
One of the way to fight the [model overfitting](#) and to generalize the model to a broader set of examples is to augment the training data.

As you saw from the previous section all training examples have a white background and vertically positioned right hands. But what if the image with the hand will be horizontally positioned or what if the background will not be that bright. What if instead of a right hand the model will see a left hand. To make our model a little bit more universal we're going to flip and rotate images and also to adjust background colors.

You may read more about a [Simple and efficient data augmentations using the Tensorflow tf.Data and Dataset API](#).



```
def augment_flip(image: tf.Tensor) -> tf.Tensor:
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    return image
```



```
def augment_color(image: tf.Tensor) -> tf.Tensor:
    image = tf.image.random_hue(image, max_delta=0.08)
    image = tf.image.random_saturation(image, lower=0.7, upper=1.3)
    image = tf.image.random_brightness(image, 0.05)
    image = tf.image.random_contrast(image, lower=0.8, upper=1)
    image = tf.clip_by_value(image, clip_value_min=0, clip_value_max=1)
    return image
```

```
def augment_rotation(image: tf.Tensor) -> tf.Tensor:
    # Rotate 0, 90, 180, 270 degrees
    return tf.image.rot90(
        image,
        tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32)
    )
```

```
def augment_inversion(image: tf.Tensor) -> tf.Tensor:
    random = tf.random.uniform(shape=[], minval=0, maxval=1)
    if random > 0.5:
        image = tf.math.multiply(image, -1)
        image = tf.math.add(image, 1)
    return image
```

```
def augment_zoom(image: tf.Tensor, min_zoom=0.8, max_zoom=1.0) -> tf.Tensor:
    image_width, image_height, image_colors = image.shape
    crop_size = (image_width, image_height)

    # Generate crop settings, ranging from a 1% to 20% crop.
```

```

scales = list(np.arange(min_zoom, max_zoom, 0.01))
boxes = np.zeros((len(scales), 4))

for i, scale in enumerate(scales):
    x1 = y1 = 0.5 - (0.5 * scale)
    x2 = y2 = 0.5 + (0.5 * scale)
    boxes[i] = [x1, y1, x2, y2]

def random_crop(img):
    # Create different crops for an image
    crops = tf.image.crop_and_resize(
        [img],
        boxes=boxes,
        box_indices=np.zeros(len(scales)),
        crop_size=crop_size
    )
    # Return a random crop
    return crops[tf.random.uniform(shape=[], minval=0, maxval=len(scales), dtype=tf.float32)]

choice = tf.random.uniform(shape=[], minval=0., maxval=1., dtype=tf.float32)

# Only apply cropping 50% of the time
return tf.cond(choice < 0.5, lambda: image, lambda: random_crop(image))

```

```

def augment_data(image, label):
    image = augment_flip(image)
    image = augment_color(image)
    image = augment_rotation(image)
    image = augment_zoom(image)
    image = augment_inversion(image)
    return image, label

```

```

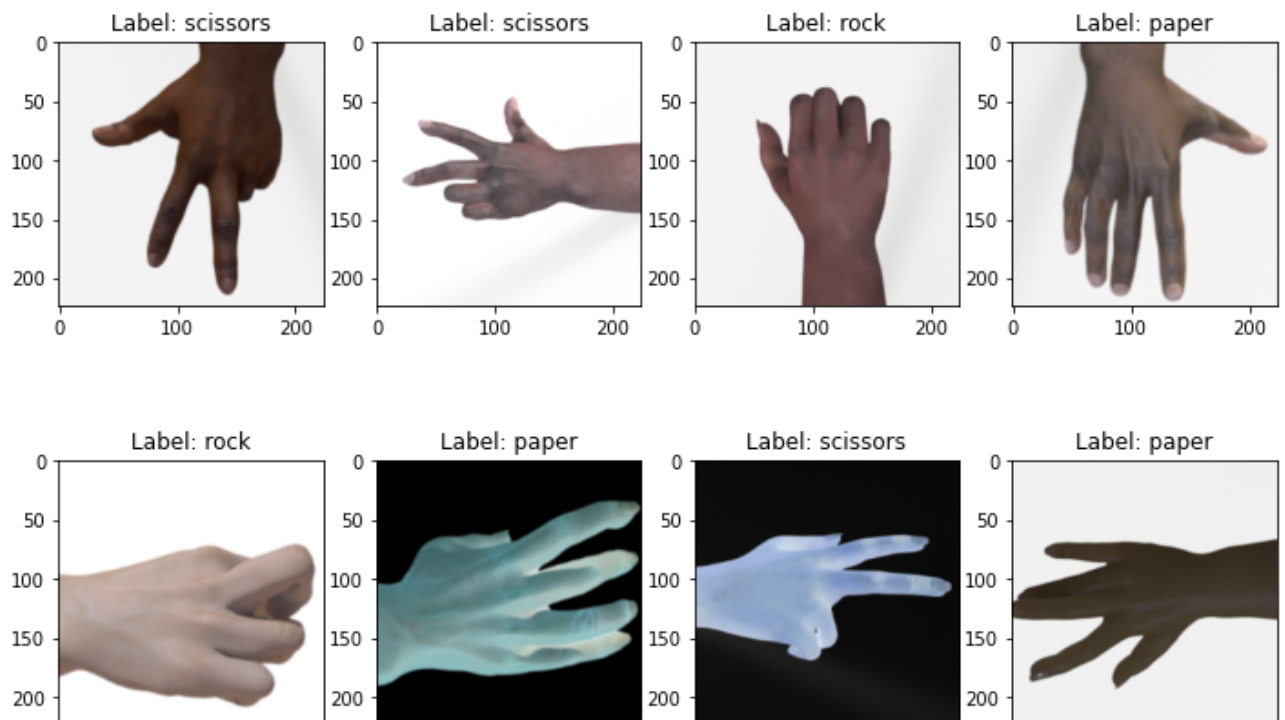
dataset_train_augmented = dataset_train.map(augment_data)

```

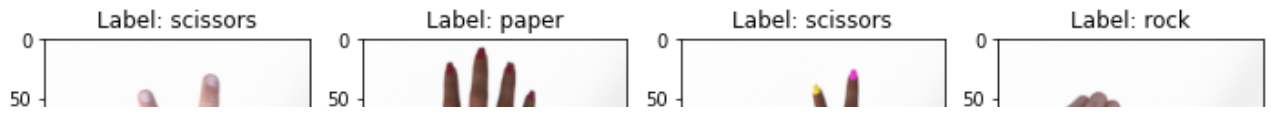
```

# Explore augmented training dataset.
preview_dataset(dataset_train_augmented)

```



```
# Explore test dataset.  
preview_dataset(dataset_test)
```



▼ Data shuffling and batching

We don't want our model to learn anything from the order or grouping of the images in the dataset. To avoid that we will shuffle the training examples. Also we're going to split the training set by batches to speed up training process and make it less memory consuming.

```
BATCH_SIZE = 800

dataset_train_augmented_shuffled = dataset_train_augmented.shuffle(
    buffer_size=NUM_TRAIN_EXAMPLES
)

dataset_train_augmented_shuffled = dataset_train_augmented.batch(
    batch_size=BATCH_SIZE
)

# Prefetch will enable the input pipeline to asynchronously fetch batches while you
dataset_train_augmented_shuffled = dataset_train_augmented_shuffled.prefetch(
    buffer_size=tf.data.experimental.AUTOTUNE
)

dataset_test_shuffled = dataset_test.batch(BATCH_SIZE)

print(dataset_train_augmented_shuffled)
print(dataset_test_shuffled)
```

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32))
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32))
```

```
# Debugging the batches using conversion to Numpy arrays.
batches = tfds.as_numpy(dataset_train_augmented_shuffled)
for batch in batches:
    image_batch, label_batch = batch
    print('Label batch shape:', label_batch.shape, '\n')
    print('Image batch shape:', image_batch.shape, '\n')
    print('Label batch:', label_batch, '\n')

    for batch_item_index in range(len(image_batch)):
        print('First batch image:', image_batch[batch_item_index], '\n')
        plt.imshow(image_batch[batch_item_index])
        plt.show()
        # Break to shorten the output.
        break
# Break to shorten the output.
break
```

Label batch shape: (800,)

Image batch shape: (800, 224, 224, 3)

Label batch: [2 2 0 1 0 1 2 1 2 2 1 1 2 1 1 1 1 1 1 1 1 0 0 0 0 1 1 2 2 2 0 0
2 1 0 0 0 0 0 1 1 2 2 0 0 2 1 1 0 0 1 2 1 0 0 0 0 1 2 1 1 2 2 1 1 1 1 1 2
0 0 2 1 0 1 0 0 1 1 1 1 2 1 1 0 0 2 2 1 0 0 1 1 2 1 1 0 0 0 2 0 0 1 1 2 0
2 0 1 1 1 2 0 1 0 1 2 1 0 1 2 2 0 2 1 0 0 1 0 1 0 1 2 1 2 2 1 0 2 0 1 1 2
0 2 2 1 0 1 2 2 1 1 0 2 0 0 1 1 0 1 2 2 0 0 2 1 1 0 1 2 0 1 1 1 2 0 2 1 2
1 1 1 2 2 2 1 0 2 0 1 0 1 2 0 0 2 0 1 1 0 2 2 2 1 1 1 0 1 0 2 0 0 1 1 1 2
1 2 1 2 2 0 2 1 0 1 0 0 2 1 1 0 2 2 2 0 1 1 1 2 0 1 0 2 1 1 2 1 2 2 0 1 2
2 0 2 1 0 2 0 0 1 0 2 2 0 0 2 2 0 0 2 2 1 0 0 0 2 1 1 0 2 0 1 1 1 2 1 1 0
1 1 2 2 2 1 2 0 0 0 2 0 2 0 0 0 0 2 2 0 1 0 0 1 1 0 1 1 0 2 1 0 2 0 1 1 0
2 1 0 0 1 2 2 0 1 1 2 2 2 0 2 2 2 0 2 1 2 0 2 1 0 1 1 1 0 2 0 1 0 1 0 0 0
0 1 2 1 0 2 2 0 2 2 2 1 2 1 2 0 1 0 0 0 1 1 1 1 1 2 1 1 1 0 1 0 2 0 0 1 0
0 2 0 0 1 1 1 1 2 2 2 1 1 0 1 2 0 1 2 0 0 1 1 0 2 2 1 2 2 1 1 0 2 1 2 2 0
1 0 0 1 2 0 2 1 1 2 1 1 2 0 2 1 1 1 1 1 2 0 0 0 0 1 1 0 0 2 2 2 1 0 2 1 2
0 1 2 0 1 2 0 2 1 0 0 1 1 2 0 2 2 0 1 1 2 0 1 2 2 0 1 1 2 1 2 1 0 1 2 1 1
1 0 1 1 1 2 0 0 1 0 1 2 2 0 0 0 0 2 0 0 0 2 2 2 0 2 2 2 1 2 1 1 1 0 0 1 2
0 2 0 1 1 0 0 2 1 0 0 1 2 0 0 0 1 1 2 1 1 0 2 0 0 1 2 1 1 0 0 0 1 2 1 1 0
2 0 1 1 0 2 0 2 0 2 2 1 0 1 1 1 2 1 0 0 2 0 0 2 0 1 0 2 2 2 1 1 0 2 1 0 0
2 0 1 1 0 2 2 0 1 2 0 2 2 1 2 0 2 2 2 2 2 0 0 2 2 1 2 0 2 0 1 2 0 2 0 0 2
0 2 2 0 1 1 0 0 2 0 1 1 1 1 0 2 1 0 1 1 2 2 1 0 0 1 1 0 2 2 1 1 2 2 2 1 2
1 2 1 2 1 2 1 2 0 1 1 1 1 2 0 1 2 2 1 2 1 2 1 2 1 2 0 0 1 0 0 1 0 2 1 1 2
0 1 1 2 1 0 1 2 0 0 0 2 0 0 2 2 2 0 0 0 1 1 0 1 1 1 1 1 0 1 2 1 1 2 2 1
0 0 0 0 1 1 1 2 2 0 2 0 0 2 2 1 1 2 2 2 2 1 1]

First batch image: [[[0.03961003 0.04078156 0.04158181]
[0.03945327 0.0406248 0.04142505]
[0.03937632 0.04054785 0.0413481]
...
[0.05511546 0.05628705 0.05708724]
[0.05506533 0.05623692 0.05703712]
[0.05474198 0.05591357 0.05671376]]

[[[0.03894454 0.04011607 0.04091632]
[0.03992707 0.04109859 0.04189885]
[0.04030651 0.04147804 0.04227829]
...
[0.05373991 0.05491149 0.05571169]
[0.05399573 0.05516732 0.05596751]
[0.05498832 0.05615991 0.05696011]]

[[[0.03742933 0.03860086 0.03940111]
[0.04090953 0.04208106 0.04288131]
[0.04145235 0.04262388 0.04342413]
...
[0.05520302 0.05637455 0.0571748]
[0.05519873 0.05637026 0.05717051]
[0.05594653 0.05711806 0.05791831]]

...

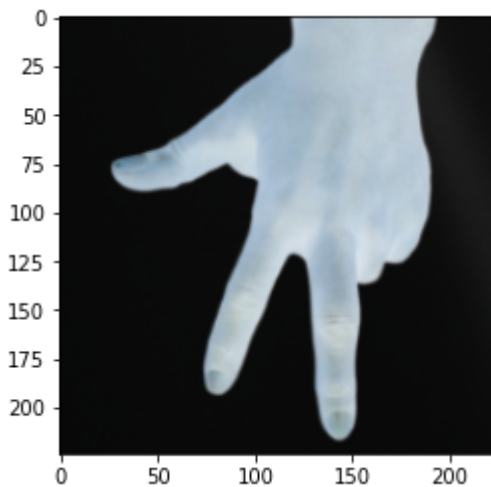
[[[0.03455669 0.03572822 0.03652847]
[0.03422618 0.03539771 0.03619796]
[0.03446275 0.03563428 0.03643453]
...
[0.0422284 0.04339993 0.04420018]
[0.04246092 0.04363251 0.0444327]
[0.0427838 0.04395533 0.04475558]]

```

[[0.03543866 0.03661019 0.03741044]
 [0.03501242 0.03618395 0.03698421]
 [0.03423661 0.03540814 0.03620839]
 ...
 [0.04182118 0.04299271 0.04379296]
 [0.04188967 0.04306126 0.04386145]
 [0.04121393 0.04238546 0.04318571]]

[[0.0322209 0.03339243 0.03419268]
 [0.03201157 0.0331831 0.03398335]
 [0.03320897 0.0343805 0.03518075]
 ...
 [0.04205447 0.043226 0.04402626]
 [0.04196447 0.043136 0.04393625]
 [0.04137576 0.04254729 0.04334754]]]

```



▼ Creating the model

▼ Loading model

We don't want to use the top classification layer of the pre-trained model as it contains 1000 classes when we need only 3 (rock, paper and scissors). We will specify that by setting a `include_top` parameter to `False`.

You may read more about Keras models on [Keras Documentation](#)

```

base_model = tf.keras.applications.NASNetMobile(
    input_shape=INPUT_IMG_SHAPE,
    include_top=False,
    weights='imagenet',
    pooling='avg'
)

```

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/19996672/19993432> [=====] - 1s 0us/step
20004864/19993432 [=====] - 1s 0us/step



```
# Freezing the base model since we don't want to re-train it.  
# We're only interesting in its feature extraction.  
base_model.trainable = False
```

```
base_model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
normal_add_3_10 (Add)	(None, 7, 7, 176)	0	['normal_l 'adjust_b
normal_add_4_10 (Add)	(None, 7, 7, 176)	0	['normal_l 'normal_r
normal_add_5_10 (Add)	(None, 7, 7, 176)	0	['separabl 5_10[0][0] 'normal_b
normal_concat_10 (Concatenate)	(None, 7, 7, 1056)	0	['adjust_b 'normal_a 'normal_a 'normal_a 'normal_a 'normal_a
activation_163 (Activation)	(None, 7, 7, 1056)	0	['normal_c
activation_164 (Activation)	(None, 7, 7, 1056)	0	['normal_c
adjust_conv_projection_11 (Conv2D)	(None, 7, 7, 176)	185856	['activati
normal_conv_1_11 (Conv2D)	(None, 7, 7, 176)	185856	['activati
adjust_bn_11 (Batch Normalization)	(None, 7, 7, 176)	704	['adjust_c '']
normal_bn_1_11 (Batch Normalization)	(None, 7, 7, 176)	704	['normal_c
activation_165 (Activation)	(None, 7, 7, 176)	0	['normal_b
activation_167 (Activation)	(None, 7, 7, 176)	0	['adjust_b
activation_169 (Activation)	(None, 7, 7, 176)	0	['adjust_b
activation_171 (Activation)	(None, 7, 7, 176)	0	['adjust_b
activation_173 (Activation)	(None, 7, 7, 176)	0	['normal_b
separable_conv_1_normal_left1_11 (SeparableConv2D)	(None, 7, 7, 176)	35376	['activati

separable_conv_1_normal_right1_11 (SeparableConv2D)	(None, 7, 7, 176)	32560	['activati
separable_conv_1_normal_left2_11 (SeparableConv2D)	(None, 7, 7, 176)	35376	['activati
separable_conv_1_normal_right2_11 (SeparableConv2D)	(None, 7, 7, 176)	32560	['activati
separable_conv_1_normal_left5_11 (SeparableConv2D)	(None, 7, 7, 176)	32560	['activati

```
tf.keras.utils.plot_model(  
    base_model,  
    show_shapes=True,  
    show_layer_names=True,  
)
```






▼ Adding a classification head

```
model = tf.keras.models.Sequential()

model.add(base_model)

# model.add(tf.keras.layers.GlobalAveragePooling2D())

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(
    units=NUM_CLASSES,
    activation=tf.keras.activations.softmax,
    kernel_regularizer=tf.keras.regularizers.l2(l=0.01)
))
```

```
model.summary()
```

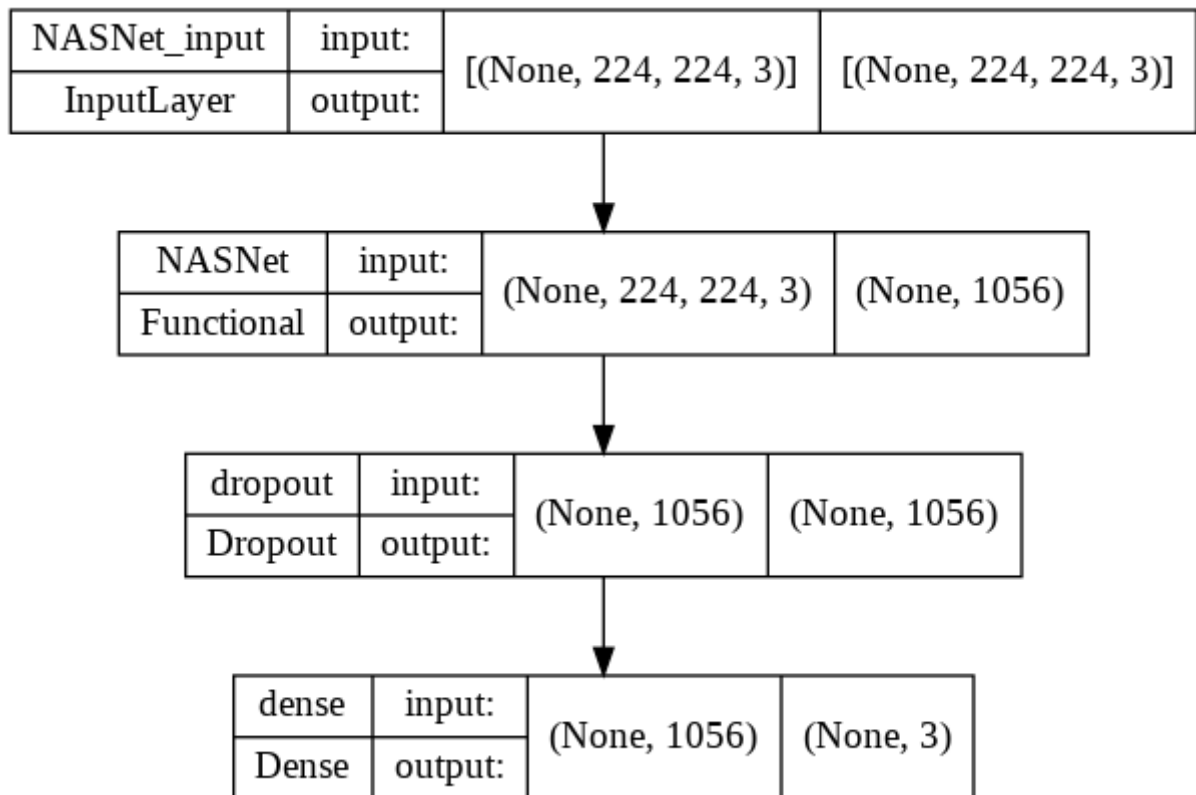
Model: "sequential"

Layer (type)	Output Shape	Param #
NASNet (Functional)	(None, 1056)	4269716
dropout (Dropout)	(None, 1056)	0
dense (Dense)	(None, 3)	3171

```
=====  
Total params: 4,272,887  
Trainable params: 3,171
```

Non-trainable params: 4,269,716

```
tf.keras.utils.plot_model(  
    model,  
    show_shapes=True,  
    show_layer_names=True,  
)
```



▼ Compiling the model

```
# adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)  
rmsprop_optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)  
  
model.compile(  
    optimizer=rmsprop_optimizer,  
    loss=tf.keras.losses.sparse_categorical_crossentropy,  
    metrics=['accuracy']  
)
```

▼ Training the model

```
steps_per_epoch = NUM_TRAIN_EXAMPLES // BATCH_SIZE  
validation_steps = NUM_TEST_EXAMPLES // BATCH_SIZE if NUM_TEST_EXAMPLES // BATCH_S  
  
print('steps_per_epoch:', steps_per_epoch)  
print('validation_steps:', validation_steps)
```

```
steps_per_epoch: 3
validation_steps: 1
```

```
!rm -rf tmp/checkpoints
!rm -rf logs
```

```
# Preparing callbacks.
os.makedirs('logs/fit', exist_ok=True)
tensorboard_log_dir = 'logs/fit/' + datetime.datetime.now().strftime('%Y%m%d-%H%M%S')
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=tensorboard_log_dir,
    histogram_freq=1
)

os.makedirs('tmp/checkpoints', exist_ok=True)
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='tmp/checkpoints/weights.{epoch:02d}-{val_loss:.2f}.hdf5'
)

early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    patience=10,
    monitor='val_accuracy'
    # monitor='val_loss'
)
```

```
initial_epochs = 20
```

```
training_history = model.fit(
    x=dataset_train_augmented_shuffled.repeat(),
    validation_data=dataset_test_shuffled.repeat(),
    epochs=initial_epochs,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    callbacks=[
        # model_checkpoint_callback,
        # early_stopping_callback,
        tensorboard_callback
    ],
    verbose=2
)
```

```
Epoch 1/20
3/3 - 49s - loss: 1.3788 - accuracy: 0.3658 - val_loss: 1.0261 - val_accuracy
Epoch 2/20
3/3 - 25s - loss: 1.1668 - accuracy: 0.4640 - val_loss: 0.9962 - val_accuracy
Epoch 3/20
3/3 - 23s - loss: 1.1169 - accuracy: 0.5017 - val_loss: 0.9978 - val_accuracy
Epoch 4/20
3/3 - 17s - loss: 1.0405 - accuracy: 0.5314 - val_loss: 0.8654 - val_accuracy
Epoch 5/20
3/3 - 31s - loss: 0.9708 - accuracy: 0.5854 - val_loss: 0.9098 - val_accuracy
Epoch 6/20
3/3 - 25s - loss: 0.9220 - accuracy: 0.6110 - val_loss: 0.9252 - val_accuracy
```

```
Epoch 7/20
3/3 - 20s - loss: 0.8553 - accuracy: 0.6547 - val_loss: 0.8942 - val_accuracy
Epoch 8/20
3/3 - 18s - loss: 0.8091 - accuracy: 0.6727 - val_loss: 0.8893 - val_accuracy
Epoch 9/20
3/3 - 30s - loss: 0.8235 - accuracy: 0.6612 - val_loss: 0.8085 - val_accuracy
Epoch 10/20
3/3 - 25s - loss: 0.7650 - accuracy: 0.6948 - val_loss: 0.8547 - val_accuracy
Epoch 11/20
3/3 - 20s - loss: 0.7729 - accuracy: 0.6977 - val_loss: 0.7985 - val_accuracy
Epoch 12/20
3/3 - 17s - loss: 0.7122 - accuracy: 0.7221 - val_loss: 0.8077 - val_accuracy
Epoch 13/20
3/3 - 31s - loss: 0.6899 - accuracy: 0.7346 - val_loss: 0.7447 - val_accuracy
Epoch 14/20
3/3 - 25s - loss: 0.6873 - accuracy: 0.7302 - val_loss: 0.7823 - val_accuracy
Epoch 15/20
3/3 - 20s - loss: 0.6455 - accuracy: 0.7506 - val_loss: 0.7048 - val_accuracy
Epoch 16/20
3/3 - 17s - loss: 0.6388 - accuracy: 0.7750 - val_loss: 0.6474 - val_accuracy
Epoch 17/20
3/3 - 31s - loss: 0.6303 - accuracy: 0.7742 - val_loss: 0.6810 - val_accuracy
Epoch 18/20
3/3 - 26s - loss: 0.6307 - accuracy: 0.7651 - val_loss: 0.7209 - val_accuracy
Epoch 19/20
3/3 - 20s - loss: 0.5980 - accuracy: 0.7913 - val_loss: 0.6859 - val_accuracy
Epoch 20/20
3/3 - 17s - loss: 0.5676 - accuracy: 0.8076 - val_loss: 0.6541 - val_accuracy
```

```
def render_training_history(training_history):
    loss = training_history.history['loss']
    val_loss = training_history.history['val_loss']

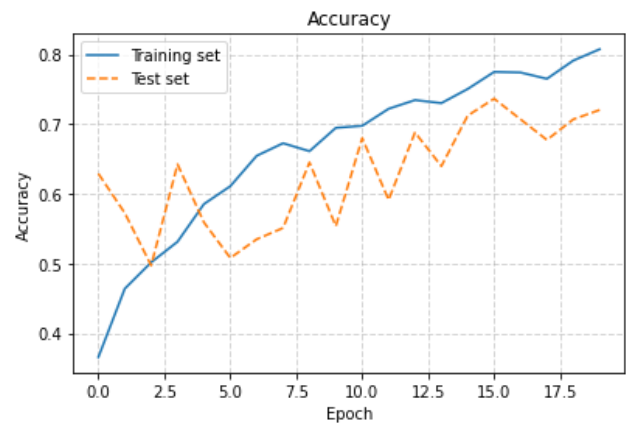
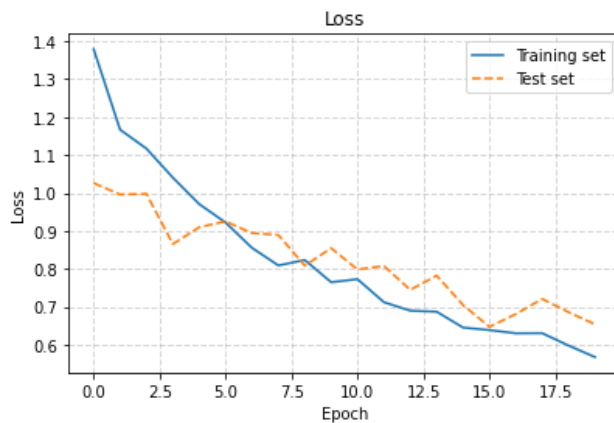
    accuracy = training_history.history['accuracy']
    val_accuracy = training_history.history['val_accuracy']

    plt.figure(figsize=(14, 4))

    plt.subplot(1, 2, 1)
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(loss, label='Training set')
    plt.plot(val_loss, label='Test set', linestyle='--')
    plt.legend()
    plt.grid(linestyle='--', linewidth=1, alpha=0.5)

    plt.subplot(1, 2, 2)
    plt.title('Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(accuracy, label='Training set')
    plt.plot(val_accuracy, label='Test set', linestyle='--')
    plt.legend()
    plt.grid(linestyle='--', linewidth=1, alpha=0.5)
```

```
plt.show()
render_training_history(training_history)
```



▼ Model fine tuning

We may try to unfreeze some of the top layers of the `base_model` and to train it a little bit more so to adjust top layers to our Rock-Paper-Scissors dataset.

```
# Un-freeze the top layers of the model
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))
```

```
Number of layers in the base model: 770
```

```
# Fine tune from this layer onwards.
# fine_tune_at = 149 # MobileNetV2
fine_tune_at = 752

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

# Compile the model using a much-lower training rate.
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
rmsprop_optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.0001)
model.compile(
    optimizer = rmsprop_optimizer,
    loss=tf.keras.losses.sparse_categorical_crossentropy,
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential"
```


Layer (type)	Output Shape	Param #
NASNet (Functional)	(None, 1056)	4269716
dropout (Dropout)	(None, 1056)	0
dense (Dense)	(None, 3)	3171

=====
Total params: 4,272,887
Trainable params: 70,051
Non-trainable params: 4,202,836
=====

```
# The number of additional epochs during which we're going to fine tune the model.
fine_tuning_epochs = 10
```

```
training_history_fine = model.fit(
    x=dataset_train_augmented_shuffled.repeat(),
    validation_data=dataset_test_shuffled.repeat(),
    epochs=initial_epochs + fine_tuning_epochs,
    initial_epoch=initial_epochs,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    callbacks=[tensorboard_callback],
    verbose=1
)
```

```
Epoch 21/30
3/3 [=====] - 46s 12s/step - loss: 0.5662 - accuracy
Epoch 22/30
3/3 [=====] - 25s 12s/step - loss: 0.5352 - accuracy
Epoch 23/30
3/3 [=====] - 21s 9s/step - loss: 0.5184 - accuracy:
Epoch 24/30
3/3 [=====] - 17s 7s/step - loss: 0.5224 - accuracy:
Epoch 25/30
3/3 [=====] - 31s 11s/step - loss: 0.5036 - accuracy
Epoch 26/30
3/3 [=====] - 24s 12s/step - loss: 0.5107 - accuracy
Epoch 27/30
3/3 [=====] - 20s 9s/step - loss: 0.5002 - accuracy:
Epoch 28/30
3/3 [=====] - 17s 7s/step - loss: 0.4890 - accuracy:
Epoch 29/30
3/3 [=====] - 31s 11s/step - loss: 0.4789 - accuracy
Epoch 30/30
3/3 [=====] - 25s 12s/step - loss: 0.4831 - accuracy
```

```
loss = training_history.history['loss'] + training_history_fine.history['loss']
val_loss = training_history.history['val_loss'] + training_history_fine.history['v

accuracy = training_history.history['accuracy'] + training_history_fine.history['a
val_accuracy = training_history.history['val_accuracy'] + training_history_fine.hi
```

```

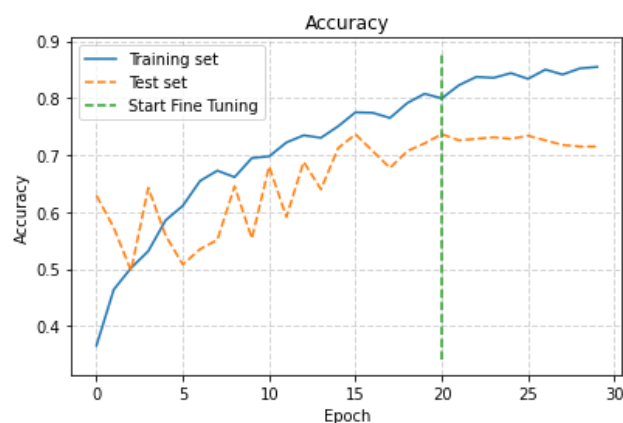
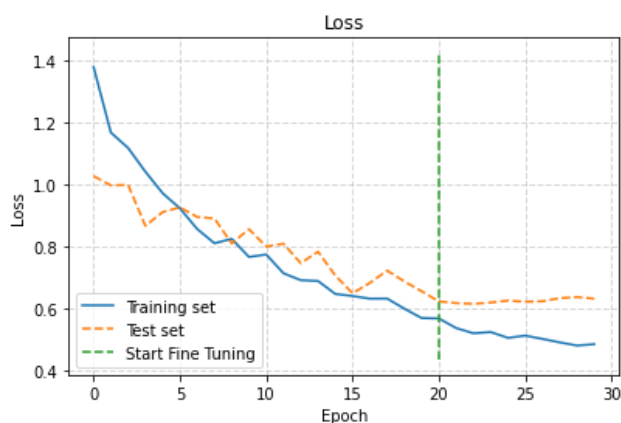
plt.figure(figsize=(14, 4))

plt.subplot(1, 2, 1)
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(loss, label='Training set')
plt.plot(val_loss, label='Test set', linestyle='--')
plt.plot(
    [initial_epochs, initial_epochs],
    plt.ylim(),
    label='Start Fine Tuning',
    linestyle='--'
)
plt.legend()
plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.subplot(1, 2, 2)
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.plot(accuracy, label='Training set')
plt.plot(val_accuracy, label='Test set', linestyle='--')
plt.plot(
    [initial_epochs, initial_epochs],
    plt.ylim(),
    label='Start Fine Tuning',
    linestyle='--'
)
plt.legend()
plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.show()

```



▼ Debugging the training with TensorBoard

Deep Learning Methods

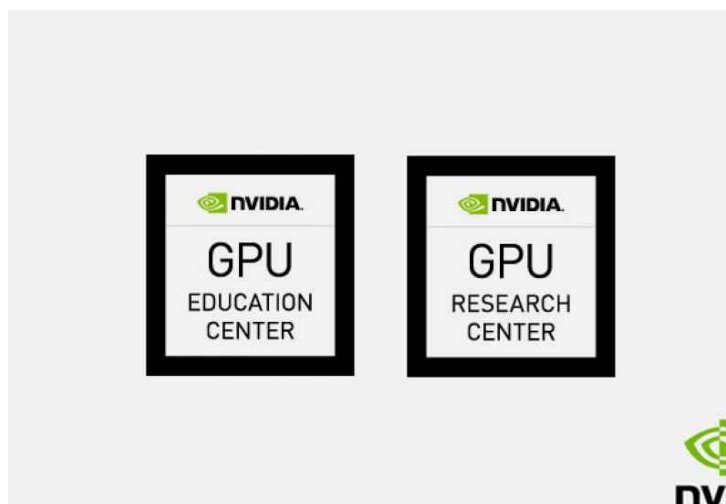
Lecture 08

Lecture Slides + interactive Jupyter-notebooks for Google Colaboratory CPU/GPU/TPU cloud:
<https://cloud.comsys.kpi.ua/s/SMkBSsxRTazoTD6>

Lecture 08 - Deep Learning Methods - Model Deployment

The course includes materials proposed by NVIDIA Deep Learning Institute (DLI) in the framework of the common

NVIDIA Research Center
and
NVIDIA Education Center.



<https://kpi.ua/nvidia-info>

Interactive Demonstrations

DEMO A - CPU

Deep Learning Model Deployment Example - MNIST WebApp (Flask + Google Colab)

<https://drive.google.com/file/d/1ywWNaf8Y2MUG526p1tiKi6lHyDkcmz3C/view?usp=sharing>

DEMO B - GPU

Deep Learning Model Deployment Example - MNIST WebApp (Flask + Google Colab)

<https://drive.google.com/file/d/11eReb0X2kJ3KScPHNM5I1XpLq2R560V0/view?usp=sharing>

DEMO C - TPU

Deep Learning Model Deployment Example - MNIST WebApp (Flask + Google Colab)

<https://drive.google.com/file/d/1X8soRab064l5R0qCv1z8JSDr3JJUBohK/view?usp=sharing>

Lecture 7 - DEMO A - CPU - Deep Learning Model Deployment Example - MNIST WebApp (Flask + Google Colab)

based on (C) Tensorflow Authors Team, Parsaniya, Heaton, Jadhav and other works

▼ Connect to Google Drive

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

Mounted at /content/drive

▼ Go to Project Folder at Google Drive and Check It

```
%cd 'drive/MyDrive/2022_COLAB_NN/Lecture_07_DL_Web-app'
! ls
```

```
/content/drive/MyDrive/2022_COLAB_NN/Lecture_07_DL_Web-app
generated_image          MNIST_test_images
Lecture_07_DL_Web-app.zip model
Lecture_07_MNIST_DEMO_A_web_app_CPU_EMPTY.ipynb static
Lecture_07_MNIST_DEMO_B_web_app_GPU_EMPTY.ipynb templates
Lecture_07_MNIST_DEMO_C_web_app_TPU_EMPTY.ipynb uploads
```

▼ Install Flask

```
!pip install flask-ngrok
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Collecting flask-ngrok
  Downloading flask_ngrok-0.0.25-py3-none-any.whl (3.1 kB)
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/pyth
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/c
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
```

Saved successfully!



```
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Installing collected packages: flask-ngrok
Successfully installed flask-ngrok-0.0.25
```

▼ Import Libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from flask import Flask, flash, redirect, render_template, request, url_for, send_
from flask_ngrok import run_with_ngrok
from tensorflow.keras.models import load_model
```

▼ Load Trained Model

```
mnist_model = load_model('model/mnist.h5')
```

▼ Configure Web-app

```
app = Flask(__name__)
run_with_ngrok(app)
app.secret_key = 'Putin_Huylo'

app.config["MNIST_BAR"] = "generated_image/mnist_vis"
app.config["IMAGES"] = "upload"

@app.route('/')
def home():
    flash("Try CNN Model Trained on MNIST-dataset for Single Digit Prediction...")
    return render_template('index.html')

@app.route('/mnist/')
def mnist_home():
    return render_template('mnist.html')

@app.route('/mnistprediction/', methods=['GET', 'POST'])
def mnist_prediction():
    if request.method == "POST":
        if not request.files['file'].filename:
            flash("No File Found")
        else:
            file' ]
            filename)
            image_gray = cv2.imread("uploads/"+f.filename, cv2.IMREAD_GRAYSCALE)
            img_resize = cv2.resize(image_gray,(28,28))
```

Saved successfully!



```

image_bw = cv2.threshold(img_resize, 75, 255, cv2.THRESH_BINARY)[1]
bitwise_not_image = cv2.bitwise_not(image_bw, mask=None)
pred_img = np.reshape(bitwise_not_image, (1,28,28,1))/255.0

predictions = mnist_model.predict(pred_img)
number = int(np.argmax(predictions))
print(number)

plt.figure()
y_pos = np.arange(10)
plt.bar(y_pos, predictions[0])
plt.savefig('generated_image/mnist_vis/'+f.filename)

return str(number)

@app.route("/get-mnist-image/<image_name>")
def get_mnist_image(image_name):
    try:
        return send_from_directory(app.config["MNIST_BAR"], filename=image_name)
    except FileNotFoundError:
        abort(404)

```

```

# Install pyngrok
!pip install pyngrok==4.1.1

# Register, get 'your authtoken', and replace my token below:
# !ngrok authtoken 'your authtoken'
!ngrok authtoken '2FdbDL8Rak9en9IT4S3pSeMjq0I_6Ntx7LfKFyeS9qLSFAoks'

```

```

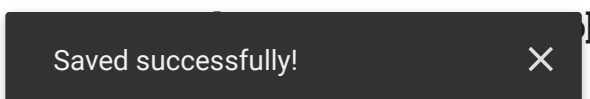
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Collecting pyngrok==4.1.1
  Downloading pyngrok-4.1.1.tar.gz (18 kB)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: pyngrok
  Building wheel for pyngrok (setup.py) ... done
  Created wheel for pyngrok: filename=pyngrok-4.1.1-py3-none-any.whl size=15951 sha256=
  Stored in directory: /root/.cache/pip/wheels/b1/d9/12/045a042fee3127dc40ba6
Successfully built pyngrok
Installing collected packages: pyngrok
Successfully installed pyngrok-4.1.1
Authtoken saved to configuration file: /root/.ngrok2/ngrok.yml

```

▼ Start Web-app

After start ...

- click on the link in the row like:



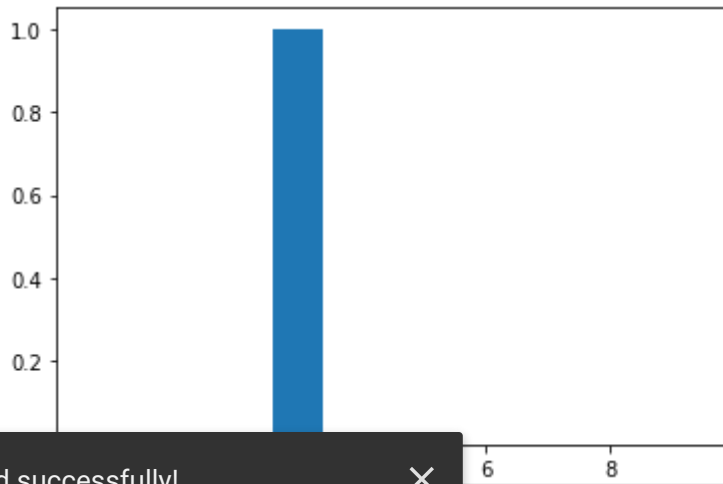
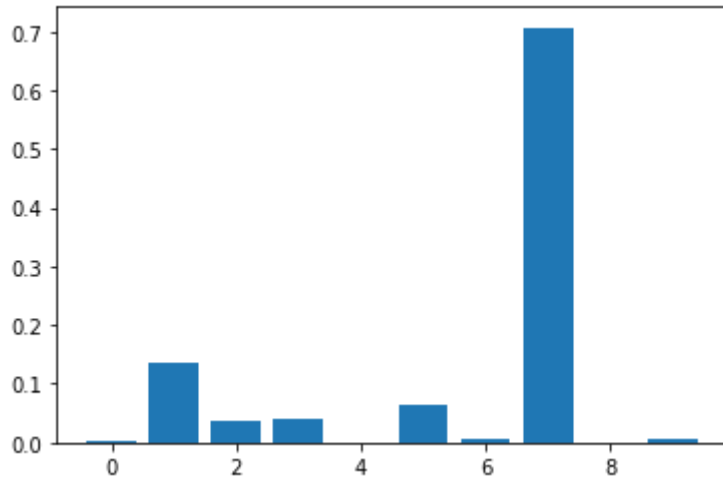
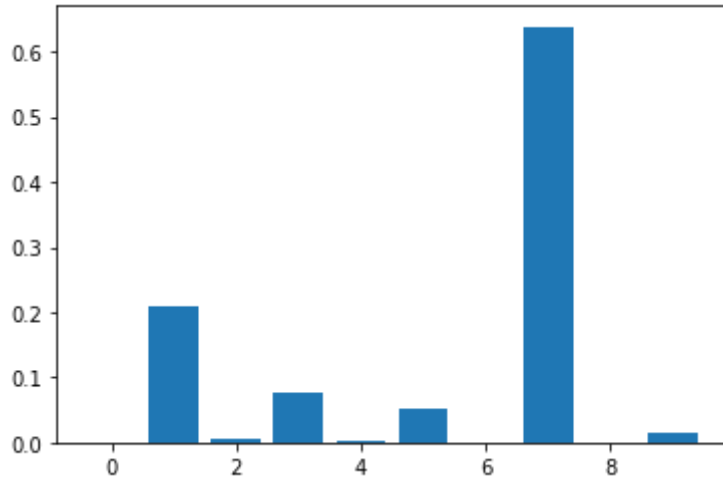
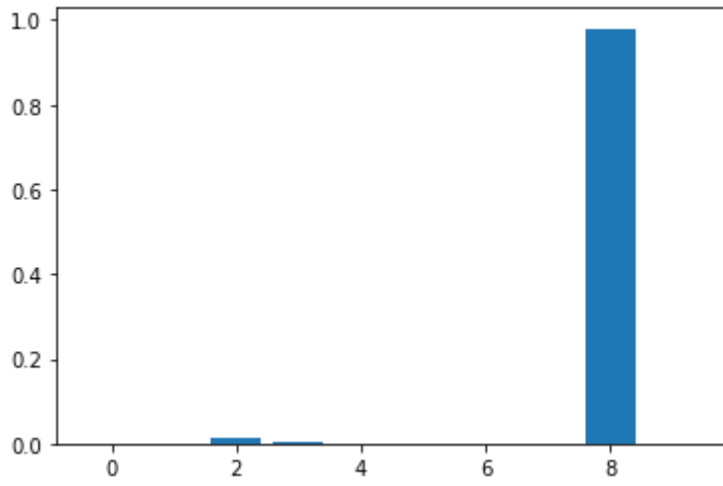
- load the local images of single digit numbers and obtain predictions;
- try images of different quality.

IMPORTANT: this Web-app is cloud-based and ... **some delay can be observed!**

```
▶ app.run()
```

Saved successfully!





Saved successfully! ✕

Lecture 7 - DEMO B - GPU - Deep Learning Model

Deployment Example - MNIST WebApp (Flask + Google Colab)

based on (C) Tensorflow Authors Team, Parsaniya, Heaton, Jadhav and other works

```
! nvidia-smi
```

```
Mon Oct 3 19:43:07 2022
+-----+
| NVIDIA-SMI 460.32.03      Driver Version: 460.32.03      CUDA Version: 11.2
+-----+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M.
|====+=====+====+=====+=====+=====+=====+=====+
|  0   Tesla T4              Off      | 00000000:00:04:0 Off  |    0%      Default
| N/A   43C    P8             9W / 70W |  0MiB / 15109MiB |             MIG M.
|                                         |                      | N/A
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:
| GPU   GI    CI          PID    Type   Process name                      GPU Memory
|      ID    ID                                   |              | Usage
+-----+-----+-----+-----+-----+-----+-----+
| No running processes found
+-----+
```

Connect to Google Drive

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

Mounted at /content/drive

Go to Project Folder at Google Drive and Check It

```
%cd 'drive/MyDrive/2022_COLAB_NN/Lecture_07_DL_Web-app'
! ls
```

```
/content/drive/MyDrive/2022_COLAB_NN/Lecture_07_DL_Web-app
generated_image          MNIST_test_images
Lecture_07_DL_Web-app.zip model
```

Saved successfully!



_app_CPU_EMPTY.ipynb static
_app_GPU_EMPTY.ipynb templates
Lecture_07_MNIST_DEMO_C_web_app_TPU_EMPTY.ipynb uploads

▼ Install Flask

```
!pip install flask-ngrok
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Collecting flask-ngrok
  Downloading flask_ngrok-0.0.25-py3-none-any.whl (3.1 kB)
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/pyth
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/c
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /us
Installing collected packages: flask-ngrok
Successfully installed flask-ngrok-0.0.25
```

▼ Import Libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from flask import Flask, flash, redirect, render_template, request, url_for, send_
from flask_ngrok import run_with_ngrok
from tensorflow.keras.models import load_model
```

▼ Load Trained Model

```
mnist_model = load_model('model/mnist.h5')
```

```
mnist_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320

Saved successfully! X

	(None, 24, 24, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
dropout (Dropout)	(None, 12, 12, 32)	0
conv2d_2 (Conv2D)	(None, 12, 12, 64)	18496
conv2d_3 (Conv2D)	(None, 12, 12, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
conv2d_4 (Conv2D)	(None, 6, 6, 128)	73856
dropout_2 (Dropout)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 128)	589952
batch_normalization (Batch Normalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 730,602
Trainable params: 730,346
Non-trainable params: 256

▼ Configure Web-app

```
app = Flask(__name__)
run_with_ngrok(app)
app.secret_key = 'ACAB_таки_да_ACAB'

app.config["MNIST_BAR"] = "generated_image/mnist_vis"
app.config["IMAGES"] = "upload"

@app.route('/')
def home():
    flash("Try CNN Model Trained on MNIST-dataset for Single Digit Prediction...")
    return render_template('index.html')

@app.route('/mnist/')
def mnist_home():
    return render_template('mnist.html')
```

Saved successfully!



```
methods=['GET', 'POST'])
```

```
def mnist_prediction():
    if request.method == "POST":
        if not request.files['file'].filename:
            flash("No File Found")
        else:
            f = request.files['file']
            f.save("uploads/"+f.filename)
            image_gray = cv2.imread("uploads/"+f.filename, cv2.IMREAD_GRAYSCALE)
            img_resize = cv2.resize(image_gray,(28,28))
            image_bw = cv2.threshold(img_resize, 75, 255, cv2.THRESH_BINARY)[1]
            bitwise_not_image = cv2.bitwise_not(image_bw, mask=None)
            pred_img = np.reshape(bitwise_not_image,(1,28,28,1))/255.0

            predictions = mnist_model.predict(pred_img)
            number = int(np.argmax(predictions))
            print(number)

            plt.figure()
            y_pos = np.arange(10)
            plt.bar(y_pos, predictions[0])
            plt.savefig('generated_image/mnist_vis/'+f.filename)

            return str(number)

@app.route("/get-mnist-image/<image_name>")
def get_mnist_image(image_name):
    try:
        return send_from_directory(app.config["MNIST_BAR"], filename=image_name)
    except FileNotFoundError:
        abort(404)
```

```
# Install pyngrok
!pip install pyngrok==4.1.1

# Register, get 'your authtoken', and replace my token below:
# !ngrok authtoken 'your authtoken'
!ngrok authtoken '2FdbDL8Rak9en9IT4S3pSeMjq0I_6Ntx7LfKFyeS9qLSFAoks'
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Collecting pyngrok==4.1.1
  Downloading pyngrok-4.1.1.tar.gz (18 kB)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: pyngrok
  Building wheel for pyngrok (setup.py) ... done
  Created wheel for pyngrok: filename=pyngrok-4.1.1-py3-none-any.whl size=159
  Stored in directory: /root/.cache/pip/wheels/b1/d9/12/045a042fee3127dc40ba6
Successfully built pyngrok
Installing collected packages: pyngrok
Successfully installed pyngrok-4.1.1
Auth token saved to configuration file: /root/.ngrok2/ngrok.yml
```



Saved successfully! X

After start ...

- click on the link in the row like:

Running on [your_website_at_ngrok.io]

- follow the web-user interface:
 - load the local images of single digit numbers and obtain predictions;
 - try images of different quality.

IMPORTANT: this Web-app is cloud-based and ... **some delay can be observed!**

```
app.run()
```

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deplc
  Use a production WSGI server instead.
* Debug mode: off
INFO:werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Running on http://dbd3-34-72-230-125.ngrok.io
* Traffic stats available on http://127.0.0.1:4040
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:44:12] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:44:12] "GET /static/css/main.css
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:44:12] "GET /static/css/header_fi
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:44:13] "GET /static/image/mnist-s
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:44:13] "GET /favicon.ico HTTP/1.1
```

Saved successfully!



or Self-Guided Experiments:

- try to train, save and use other *.h5 model (like it was described in the previous DEMOs),
- try to use other datasets and related models,
- try to port the web-app to your local environment,
- ...

[Colab paid products](#) - [Cancel contracts here](#)

▶ Executing (2s) Cell > new_run() > run() > run_simple() > inner() > serve_forever() > serve_forever() > select() ... ✕

Lecture 8 - DEMO C - TPU - Deep Learning Model

Deployment Example - MNIST WebApp (Flask + Google Colab)

based on (C) Tensorflow Authors Team, Parsaniya, Heaton, Jadhav and other works

▼ Connect to Google Drive

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

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

▼ Go to Project Folder at Google Drive and Check It

```
%cd 'drive/MyDrive/2022_COLAB_NN/Lecture_07_DL_Web-app'
! ls
```

```
/content/drive/MyDrive/2022_COLAB_NN/Lecture_07_DL_Web-app
generated_image          MNIST_test_images
Lecture_07_DL_Web-app.zip model
Lecture_07_MNIST_DEMO_A_web_app_CPU_EMPTY.ipynb static
Lecture_07_MNIST_DEMO_B_web_app_GPU_EMPTY.ipynb templates
Lecture_07_MNIST_DEMO_C_web_app_TPU_EMPTY.ipynb uploads
```

▼ Install Flask

```
!pip install flask-ngrok
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Requirement already satisfied: flask-ngrok in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /usr
```


▼ Import Libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from flask import Flask, flash, redirect, render_template, request, url_for, send_
from flask_ngrok import run_with_ngrok
from tensorflow.keras.models import load_model
```

▼ Load Trained Model

```
mnist_model = load_model('model/mnist.h5')
```

▼ Configure Web-app

```
app = Flask(__name__)
run_with_ngrok(app)
app.secret_key = 'ACAB_таки_да_ACAB'

app.config["MNIST_BAR"] = "generated_image/mnist_vis"
app.config["IMAGES"] = "upload"

@app.route('/')
def home():
    flash("Try CNN Model Trained on MNIST-dataset for Single Digit Prediction...")
    return render_template('index.html')

@app.route('/mnist/')
def mnist_home():
    return render_template('mnist.html')

@app.route('/mnistprediction/', methods=['GET', 'POST'])
def mnist_prediction():
    if request.method == "POST":
        if not request.files['file'].filename:
            flash("No File Found")
        else:
            f = request.files['file']
            f.save("uploads/"+f.filename)
            image_gray = cv2.imread("uploads/"+f.filename, cv2.IMREAD_GRAYSCALE)
            img_resize = cv2.resize(image_gray, (28,28))
            image_bw = cv2.threshold(img_resize, 75, 255, cv2.THRESH_BINARY)[1]
            bitwise_not_image = cv2.bitwise_not(image_bw, mask=None)
            pred_img = np.reshape(bitwise_not_image, (1,28,28,1))/255.0
```

```

        predictions = mnist_model.predict(pred_img)
        number = int(np.argmax(predictions))
        print(number)

    plt.figure()
    y_pos = np.arange(10)
    plt.bar(y_pos, predictions[0])
    plt.savefig('generated_image/mnist_vis/'+f.filename)

    return str(number)

@app.route("/get-mnist-image/<image_name>")
def get_mnist_image(image_name):
    try:
        return send_from_directory(app.config["MNIST_BAR"], filename=image_name)
    except FileNotFoundError:
        abort(404)

```

```

# Install pyngrok
!pip install pyngrok==4.1.1

```

```

# Register, get 'your authtoken', and replace my token below:
# !ngrok authtoken 'your authtoken'
!ngrok authtoken '2FdbDL8Rak9en9IT4S3pSeMjq0I_6Ntx7LfKFyeS9qLSFAoks'

```

```

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Collecting pyngrok==4.1.1
  Downloading pyngrok-4.1.1.tar.gz (18 kB)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: pyngrok
  Building wheel for pyngrok (setup.py) ... done
  Created wheel for pyngrok: filename=pyngrok-4.1.1-py3-none-any.whl size=159
  Stored in directory: /root/.cache/pip/wheels/b1/d9/12/045a042fee3127dc40ba6
Successfully built pyngrok
Installing collected packages: pyngrok
Successfully installed pyngrok-4.1.1
Authtoken saved to configuration file: /root/.ngrok2/ngrok.yml

```

▼ Start Web-app

After start ...

- click on the link in the row like:

Running on [your_website_at_ngrok.io]

- follow the web-user interface:
 - load the local images of single digit numbers and obtain predictions;
 - try images of different quality.

IMPORTANT: this Web-app is cloud-based and ... **some delay can be observed!**

```
▶ app.run()
```

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deplc
  Use a production WSGI server instead.
* Debug mode: off
INFO:werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Running on http://3a76-35-222-189-46.ngrok.io
* Traffic stats available on http://127.0.0.1:4040
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:48:01] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:48:02] "GET /static/css/main.css
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:48:02] "GET /static/css/header_fi
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:48:02] "GET /static/image/mnist-s
INFO:werkzeug:127.0.0.1 - - [03/Oct/2022 19:48:02] "GET /favicon.ico HTTP/1.1
```



▼ Some Possible Tasks for Self-Guided Experiments:

- try to train, save and use other *.h5 model (like it was described in the previous DEMOs),
- try to use other datasets and related models,
- try to port the web-app to your local environment,
- ...

[Colab paid products](#) - [Cancel contracts here](#)

▶ Executing (21s) C... > new_ru... > run... > run_simpl... > inne... > serve_foreve... > serve_foreve... > select... ●●● ✕