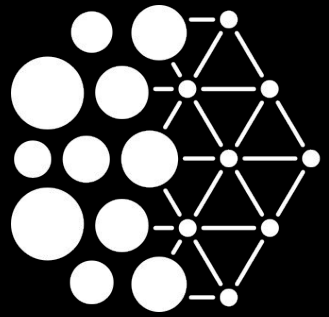
Quebec
Artificial
Intelligence
Institute



Mila

Machine Learning Tools

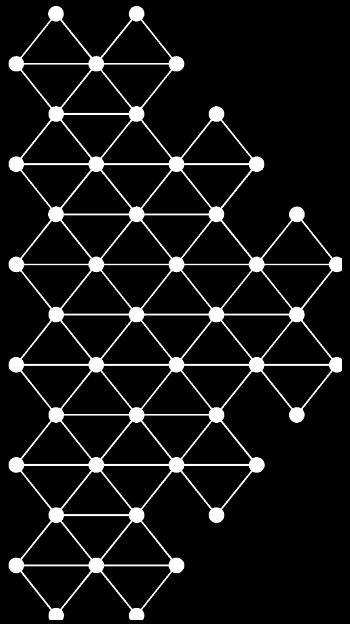
Jeremy Pinto
Applied Research Scientist, Mila
jeremy.pinto@mila.quebec

Contents

- Python
- Machine Learning Tools
- Deep Learning Frameworks
- Project Management
- Hardware

https://www.python.org/

Quebec
Artificial
Intelligence
Institute



Python

Python

Python is a programming language developed in 1991. It is **open-source** and very popular in the machine learning and deep learning community.

Why is python great?

- The zen of python
- Rich ecosystem of libraries
- Easy to get started with (interactive notebooks)

Comparable languages: Lua julia

https://www.python.org/

Python

Python is a programming language developed in 1991. It is **open-source** and very popular in the machine learning and deep learning community.

Why is python great?

- **The zen of python**
- Rich ecosystem of libraries
- Easy to get started with interactive notebooks

Comparable languages: Lua julia

https://www.python.org/

Python

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

“Code is read much more often than it is written” - Guido Van Rossum

https://www.python.org/dev/peps/pep-0020/

https://www.python.org/dev/peps/pep-0008/#a-foolish-consistency-is-the-hobgoblin-of-little-minds

Python

Python is a programming language developed in 1991. It is **open-source** and very popular in the machine learning and deep learning community.

Why is python great?

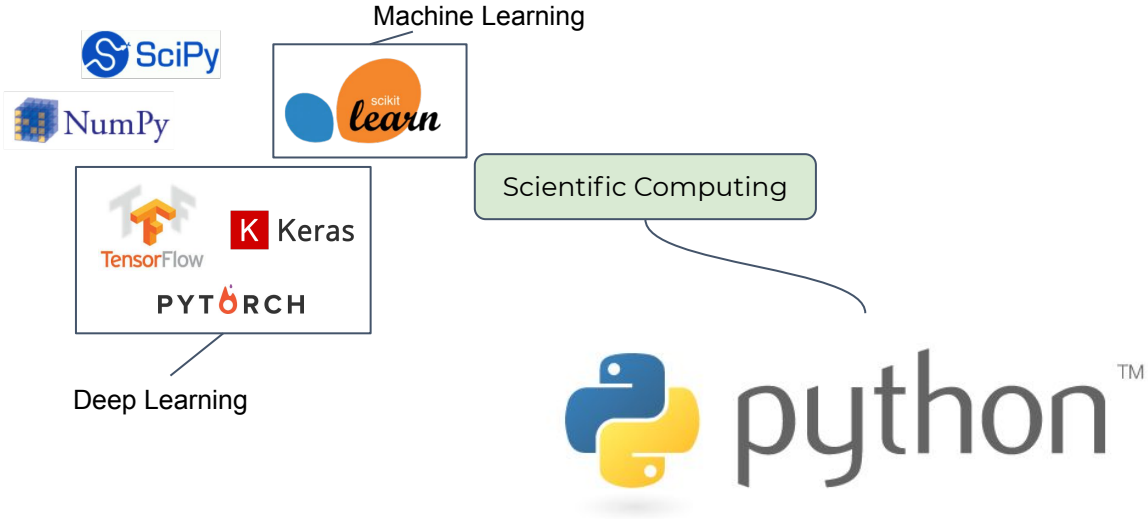
- The zen of python
- **Rich ecosystem of libraries**
- Easy to get started with interactive notebooks

Comparable languages:

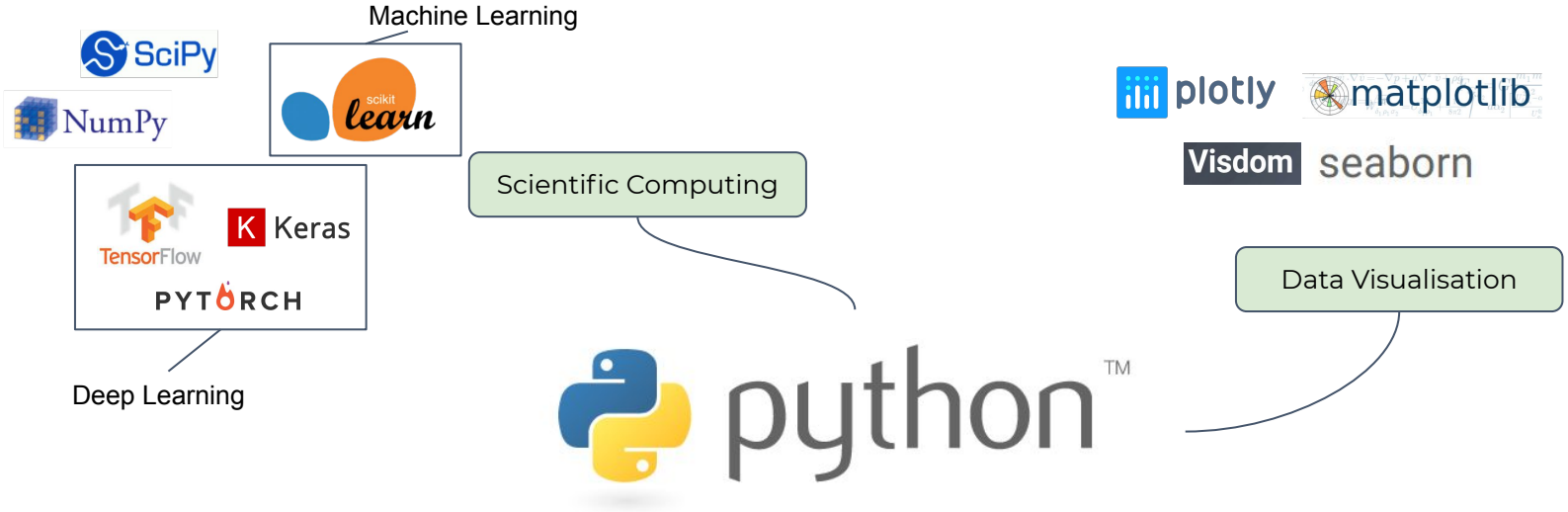
https://www.python.org/

Python Libraries

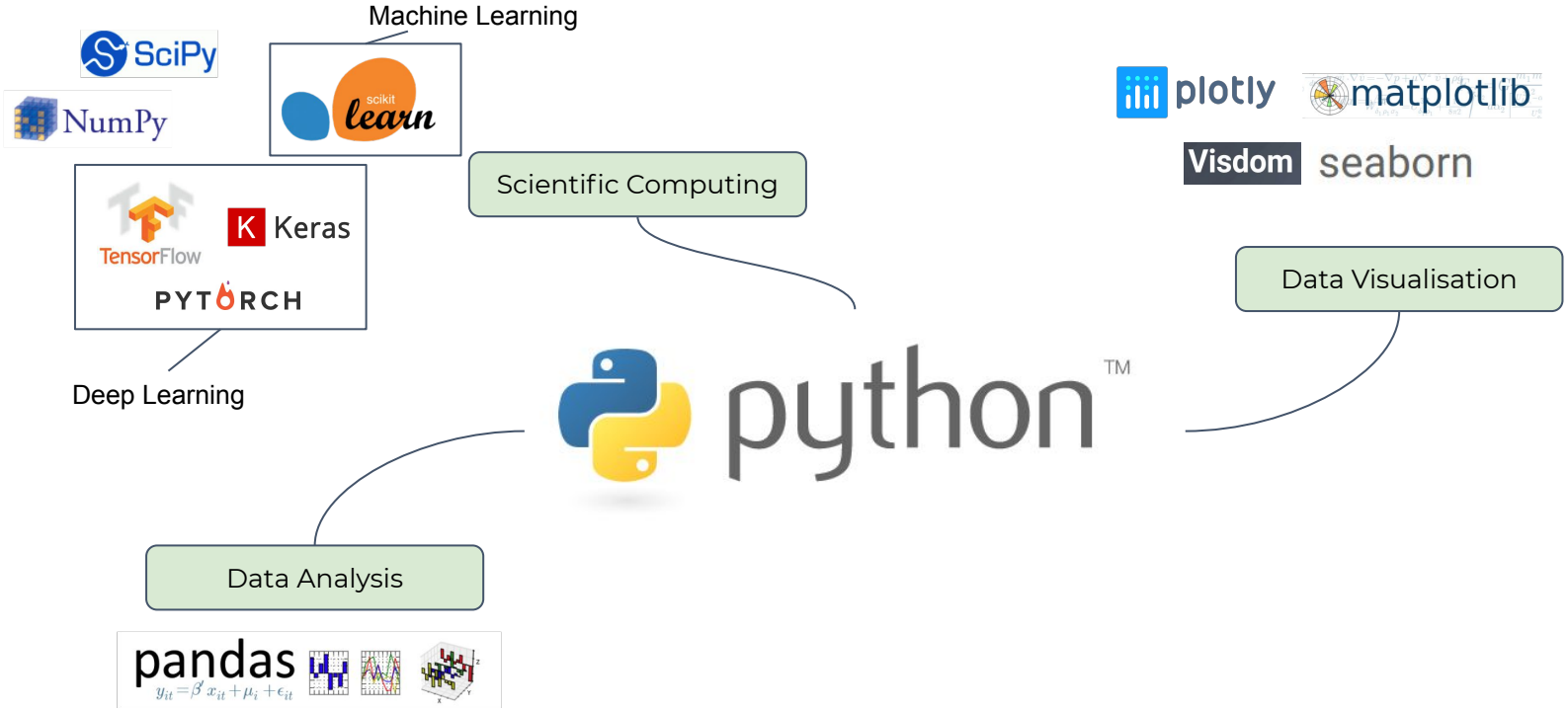
Python Libraries



Python Libraries

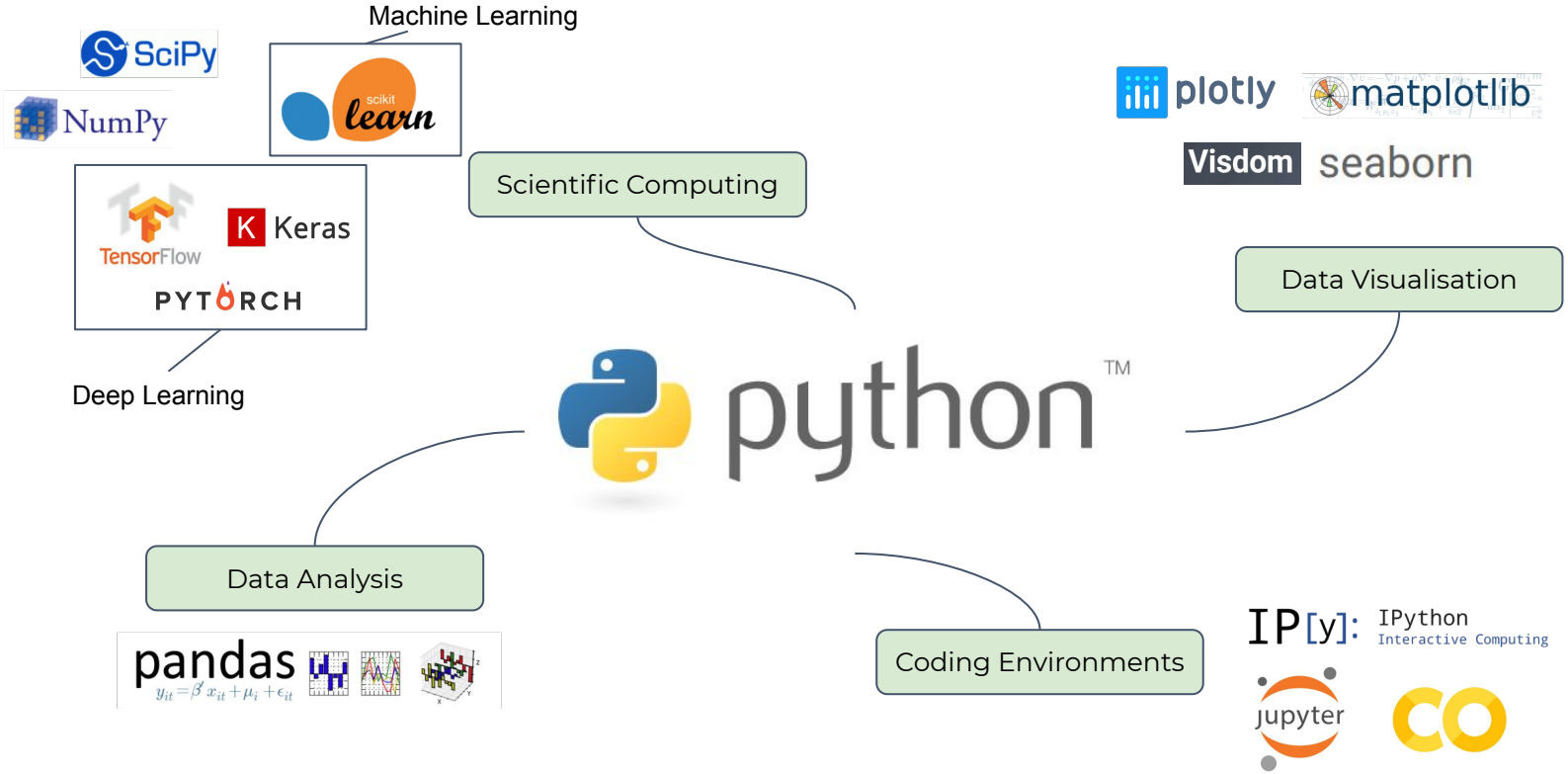


Python Libraries



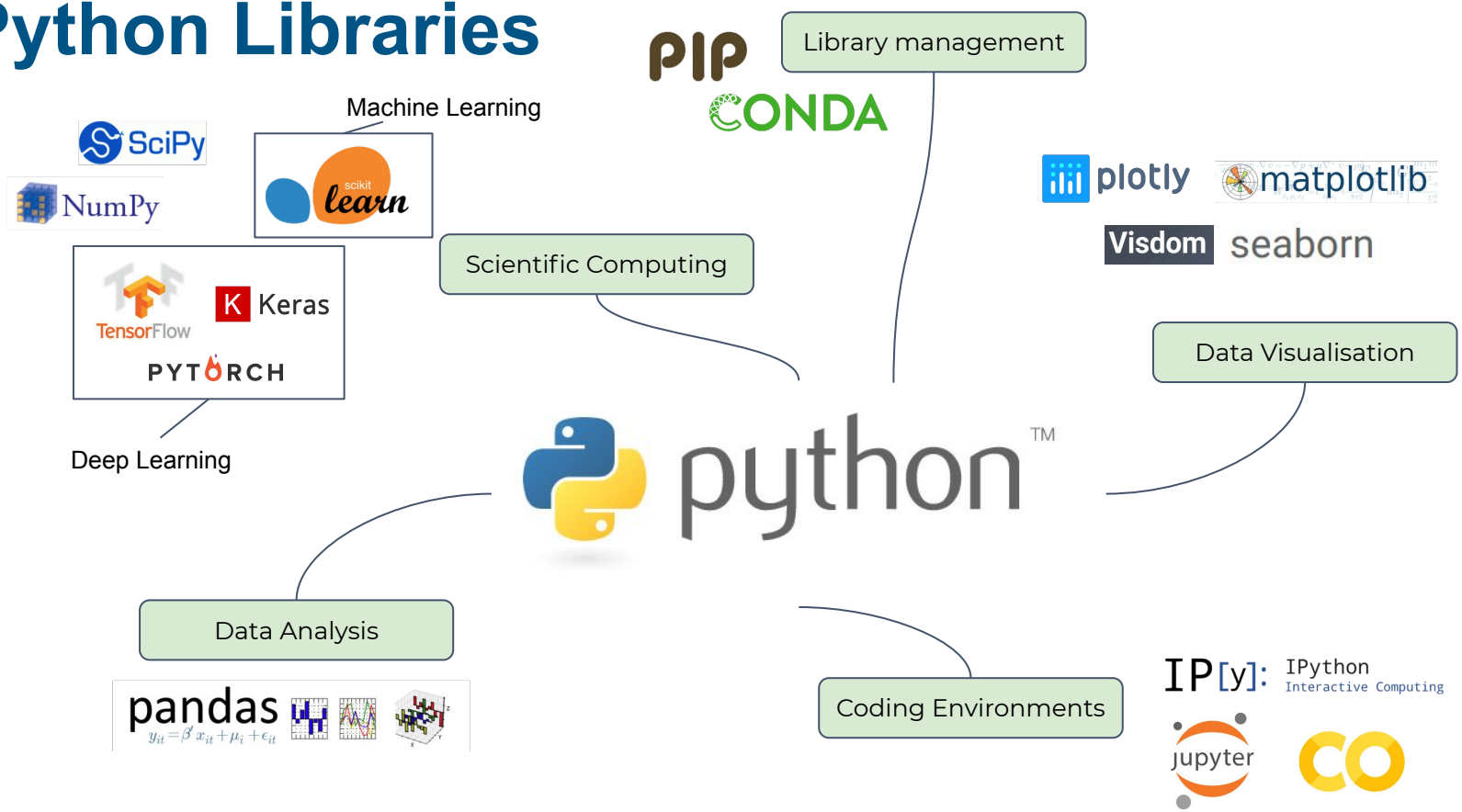
Mila

Python Libraries



Mila

Python Libraries



Mila

Python

Python is a programming language developed in 1991. It is **open-source** and very popular in the machine learning and deep learning community.

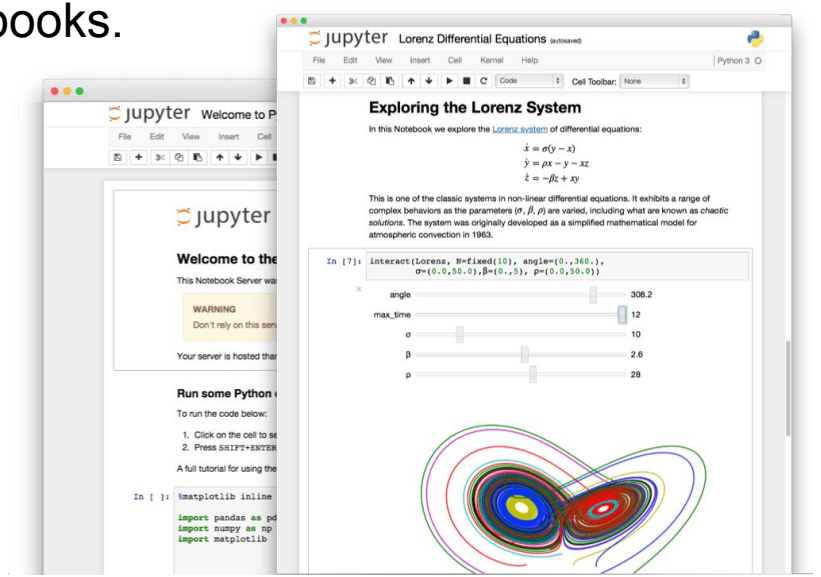
Why is python great?

- The zen of python
- Rich ecosystem of libraries
- **Easy to get started with interactive notebooks**

Comparable languages:

Interactive Notebooks

Interactive notebooks are a way to run python **code in cells** and view the generated output directly **inline**. They also support **markdown** to display inline text. It is very useful for generating **graphs**, displaying results, sharing ideas, etc. The most common notebook is Jupyter Notebooks.



https://jupyter.org/

Colab

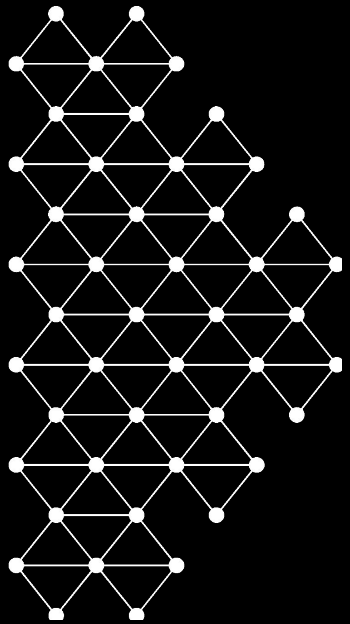
An even easier way to get started with python is using **Google Colab**, an interactive python notebook that works directly from your browser. It comes pre-configured with python3, popular scientific computing libraries (numpy, pytorch, tensorflow, etc.) and you can even have access to a **free GPU** (k80). We will use it for our tutorials.

Python 101

Running a hello world program with Colab is literally this easy:

Quebec
Artificial
Intelligence
Institute

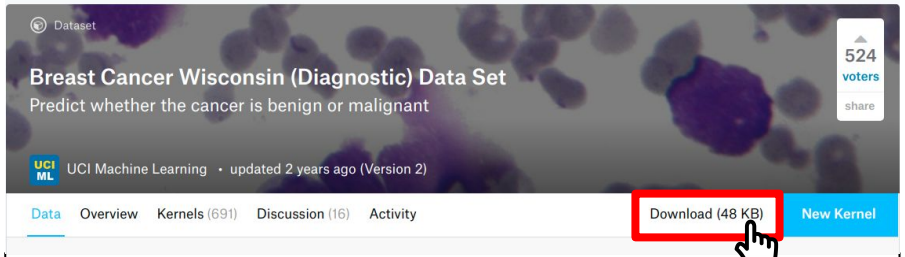
Mila



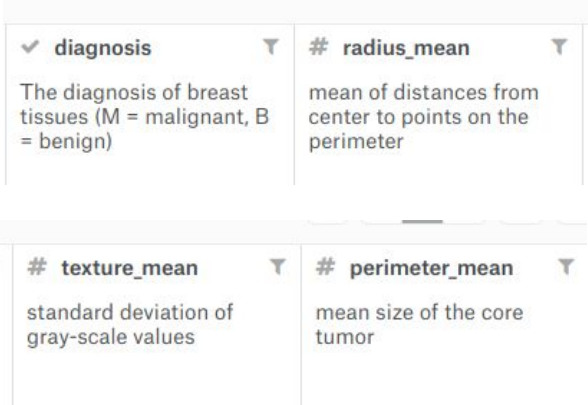
Machine Learning 101

Machine Learning 101

Let’s take a look at a standard machine learning pipeline using python. We will use the **Breast Cancer Wisconsin Data Set** which identifies measurements of tumors as malignant or benign



“Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. ”

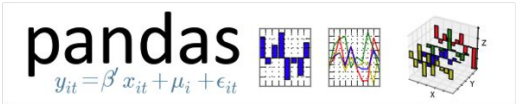


https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Pipeline

This is what our pipeline will look like:

Explore the raw data



Import the data to python

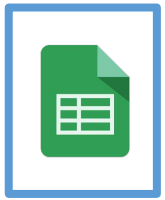
Process the data

Make predictions using machine learning

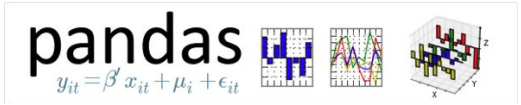
Visually explore the data

Pipeline

We begin by exploring the data with tools we are familiar with:



Explore the raw data



Import the data to python

Process the data

Make predictions using machine learning

Visually explore the data

Data Exploration

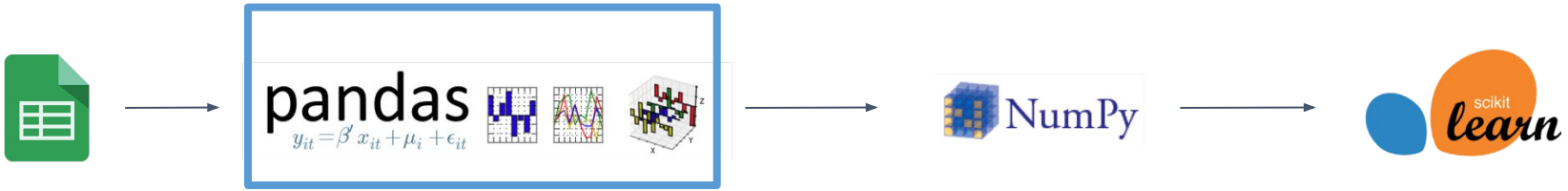
The data is a spreadsheet (.csv) with various measurements and two diagnoses: **M**alignant and **B**enign. This is a binary classification problem.

data.csv (122.27 KB) — 20 of 32 columns

	# id ID number	diagnosis The diagnosis of breast tissues (M = malignant, B = benign)	# radius_mean mean of distances from center to points on the perimeter	# texture_mean standard deviation of gray-scale values	# perimeter_mean mean size of the core tumor
1	842302	M	17.99	10.38	122.8
2	842517	M	20.57	17.77	132.9
3	84300903	M	19.69	21.25	130
4	84348301	M	11.42	20.38	77.58

Pipeline

Now that we know what kind of data to expect, we can import it to python using the pandas library.



Explore the raw data

Import the data to python

Process the data

Make predictions using machine learning

Visually explore the data

Data Import

We use pandas to import the csv file and quickly explore some useful information such as the data quantity and distribution. Pandas will allow us to easily manipulate our data structure and preserve it in its tabular form.

```
[ ]  import pandas as pd

     dataset = pd.read_csv('data.csv')

     # Print useful stats about the dataset
     print("Number of total entries: ", len(dataset))
     print("")

     print("Entries per category:")
     print(dataset["diagnosis"].value_counts())

     dataset.head() # Explore the first 5 rows of the data
```

```
Number of total entries:  569

Entries per category:
B    357
M    212
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	ar
0	842302	M	17.99	10.38	122.80	
1	842517	M	20.57	17.77	132.90	
2	84300903	M	19.69	21.25	130.00	
3	84348301	M	11.42	20.38	77.58	
4	84358402	M	20.29	14.34	135.10	

Link to the example

Pipeline

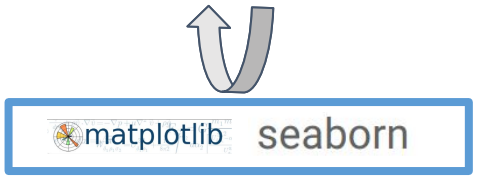
Let’s start looking at our data visually.

Explore the raw data

Import the data to python

Process the data

Make predictions using machine learning

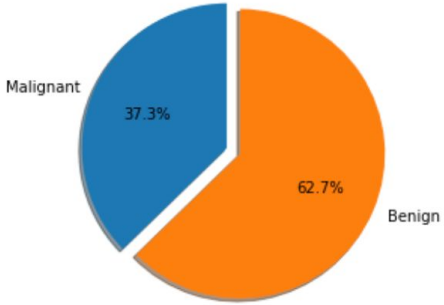


Visually explore the data

Matplotlib

We can use matplotlib to draw a pie chart. So far this seems like something that could easily be done with spreadsheets, but perhaps overly complicated.

```
[ ]  import matplotlib.pyplot as plt
     %matplotlib inline

     # Order the data by class, this is not necessary but can be useful
     dataset = dataset.sort_values(by='diagnosis', ascending=False)

     n_benign = dataset["diagnosis"].value_counts().values[0]
     n_malig = dataset["diagnosis"].value_counts().values[1]

     # Pie chart, where the slices will be ordered and plotted counter-clockwise:
     labels = 'Malignant', 'Benign'
     sizes = [n_malig, n_benign]
     explode = (0, 0.1)  # only "explode" the 2nd slice

     fig1, ax1 = plt.subplots()
     ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
             shadow=True, startangle=90)
     ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

     plt.show()
```



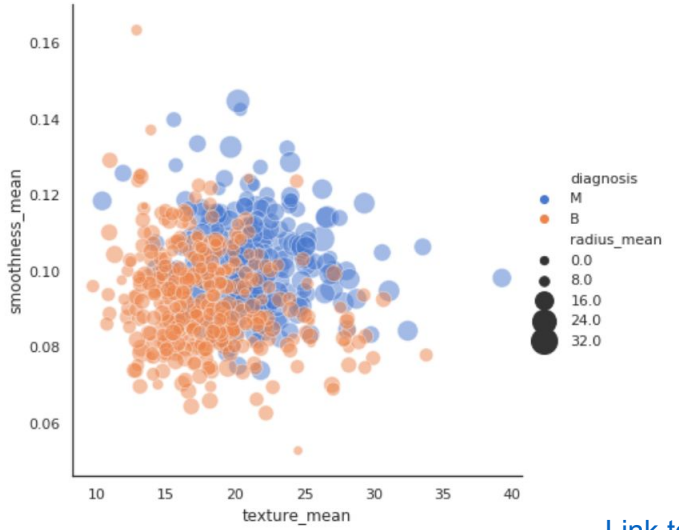
Link to the example

Seaborn

Libraries like seaborn use the rich features of matplotlib combined with pandas to provide an interface that can produce rich data interpretation with few lines of code.

```
[ ]  import seaborn as sns

     sns.set(style="white")

     # Plot texture against smoothness
     # with radius and class
     sns.relplot(x="texture_mean",
                 y="smoothness_mean",
                 hue="diagnosis",
                 size="radius_mean",
                 sizes=(40, 400),
                 alpha=.5, palette="muted",
                 height=6, data=dataset)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6d141d7e48>
```



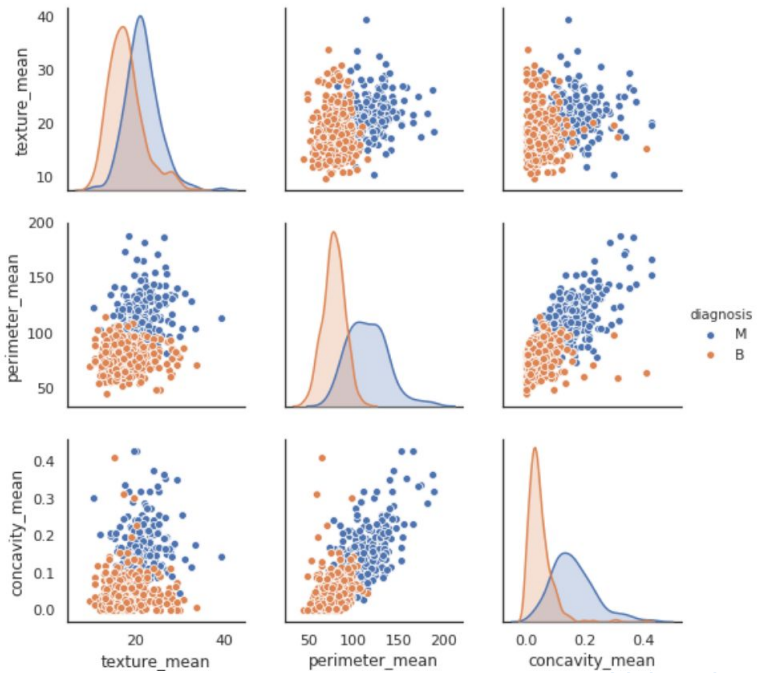
Link to the example

Seaborn

Libraries like seaborn use the rich features of matplotlib combined with pandas to provide an interface that can produce rich data interpretation with few lines of code.

```
[ ]  # Select specific indices to look for correlations

     col_idx = [1,3,4,8]
     columns = dataset.columns[col_idx]
     sns.pairplot(dataset[columns],
                  hue="diagnosis")
```

```
<seaborn.axisgrid.PairGrid at 0x7f6d141d7d30>
```



Link to the example

Mila

Pipeline

Next we will convert our data to Numpy arrays.

Explore the raw data

Import the data to python

Process the data

Make predictions using machine learning

Visually explore the data

Numpy

Numpy allows us to manipulate N-dimensional arrays in python. It is a ubiquitous data structure supported by most python libraries that manipulate data.

Here we convert our pandas data to numpy arrays to be compatible with the scikit-learn API.

```
[ ]  import numpy as np

     n_columns = len(dataset.columns)

     # Convert data to ndarray
     # Be careful not to include your labels in your data!
     X = np.asarray(dataset.iloc[:, 2:n_columns-1])

     # Labels (binary), True is Malignant, False is Benign
     y = np.asarray(dataset.iloc[:, 1] == 'M')


     idx = np.random.randint(len(dataset))

     print('Sample datapoint #', idx)
     print('\nData :')
     print(X[idx,:])
     print('\nLabel :', y[idx])
```

```
Sample datapoint # 289

Data :
[1.160e+01 1.836e+01 7.388e+01 4.127e+02 8.508e-02 5.855e-02 3.367e-02
 1.777e-02 1.516e-01 5.859e-02 1.816e-01 7.656e-01 1.303e+00 1.289e+01
 6.709e-03 1.701e-02 2.080e-02 7.497e-03 2.124e-02 2.768e-03 1.277e+01
 2.402e+01 8.268e+01 4.951e+02 1.342e-01 1.808e-01 1.860e-01 8.288e-02
 3.210e-01 7.863e-02]

Label : False
```

Link to the example

Pipeline

Next we will use the scikit-learn API to train a machine learning model on our data.

Explore the raw data

Import the data to python

Process the data

Make predictions using machine learning

Visually explore the data

Scikit-Learn

Scitkit-Learn is a machine learning library in python that contains many different types of algorithms with a standardized API. It typically looks like this.

```
from sklearn import SomeModel

my_model = SomeModel(important_parameters)
my_model.fit(X_train, y_train)

y_pred = my_model.predict(X_test)

print(score(y_pred, y_test))
```

Scikit-Learn

Let's look at an example of L2-regularized logistic regression on our dataset:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i (X_i^T w + c)) + 1)$$

```
[ ]  from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score

     logreg = LogisticRegression(solver='liblinear')
     logreg.fit(X,y)

     y_pred = logreg.predict(X)
     print("Accuracy: {:.1f} %".format(accuracy_score(y, y_pred)*100))
```

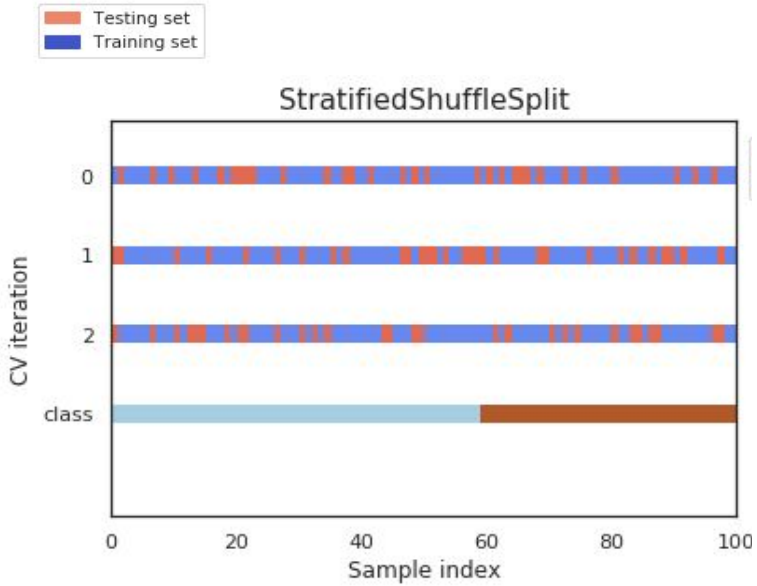
```
Accuracy: 96.0 %
```

Link to the example

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Scikit-Learn

In the previous example we trained and evaluated on the same data. We can use scikit-learn to implement different cross-validation strategies on train and test sets.

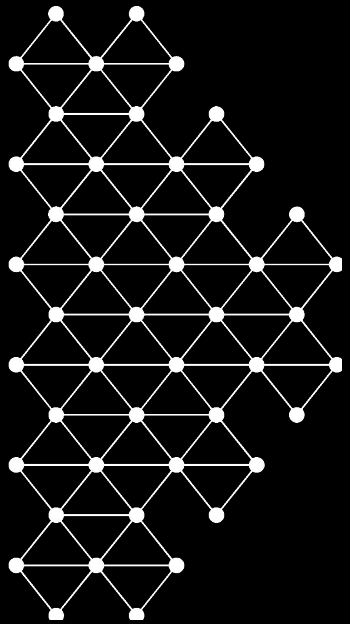



```
[ ]  from sklearn.model_selection import StratifiedShuffleSplit

     acc_tot = 0
     n_splits = 3

     sss = StratifiedShuffleSplit(n_splits=n_splits,
                                  test_size=0.3,
                                  random_state=42)

     logreg = LogisticRegression(solver='liblinear')

     for train_index, test_index in sss.split(X, y):

         X_train = X[train_index]
         y_train = y[train_index]

         X_test = X[test_index]
         y_test = y[test_index]

         logreg.fit(X_train, y_train)

         y_pred = logreg.predict(X_test)
         acc = accuracy_score(y_test, y_pred)

         acc_tot += acc


     print("Average Accuracy: {:.1f} %".format(acc_tot/n_splits*100))
```

```
Average Accuracy: 94.5 %
```

Link to the example

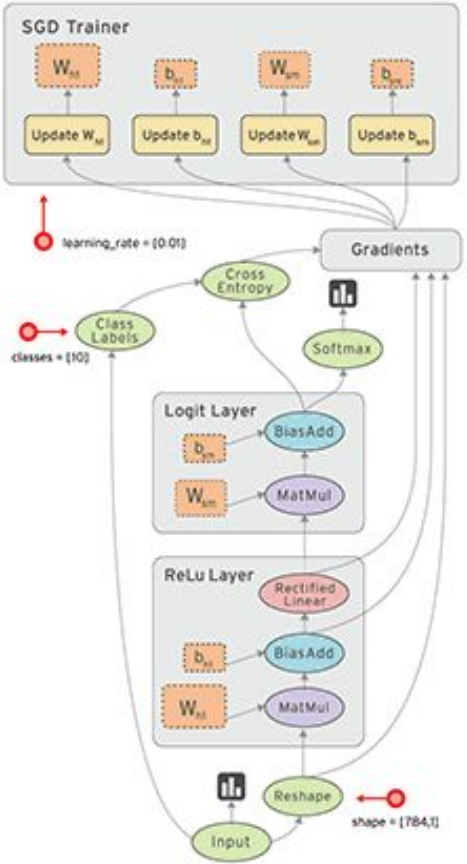
Quebec
Artificial
Intelligence
Institute



Deep Learning Frameworks

Deep Learning Frameworks

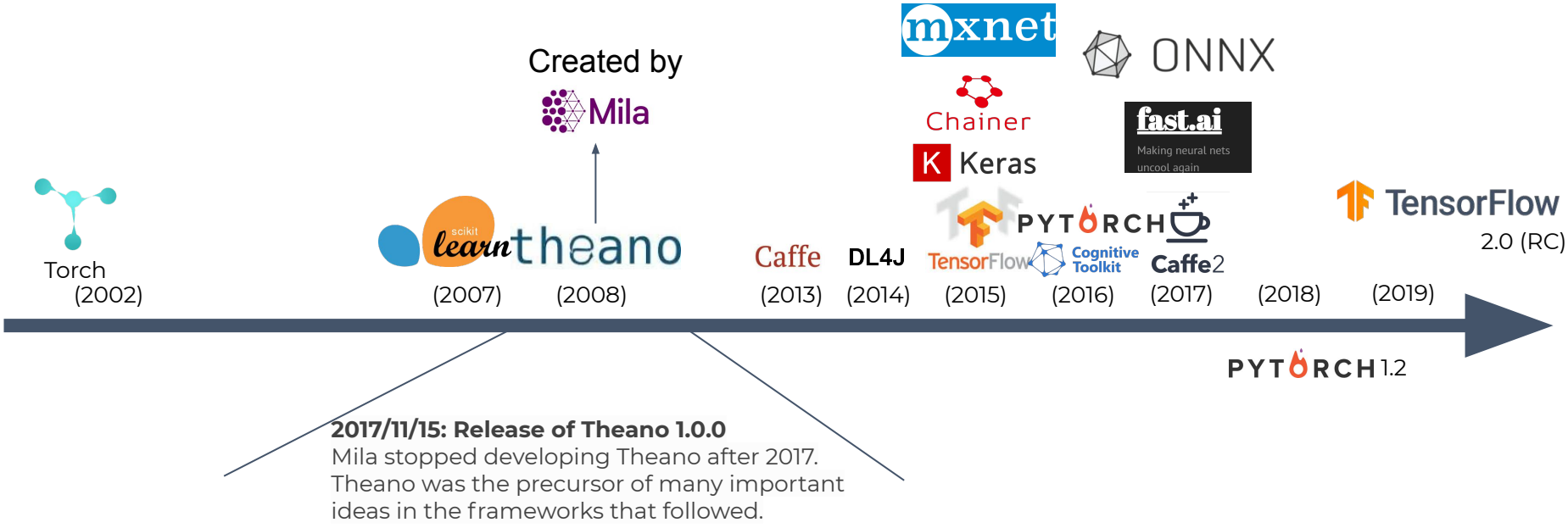
A good deep learning framework will allow users to **easily** implement neural networks and facilitate deployment for **production**. These libraries are built around the idea of **computational graphs** that summarize the operations of a network and facilitate and optimize things like backpropagation and gradient descent.



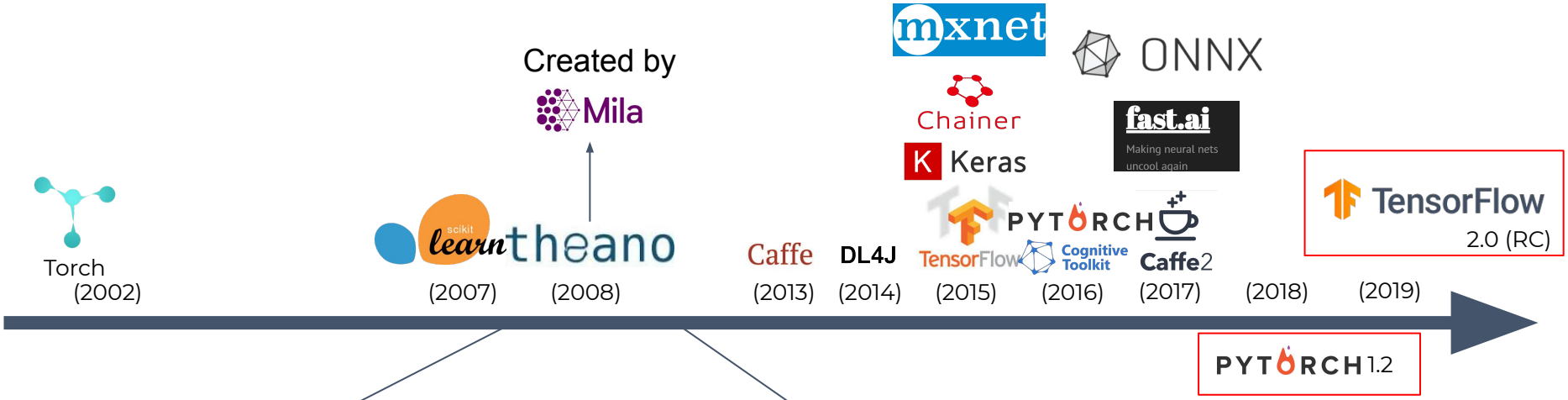
https://www.tensorflow.org/guide/graphs

Mila

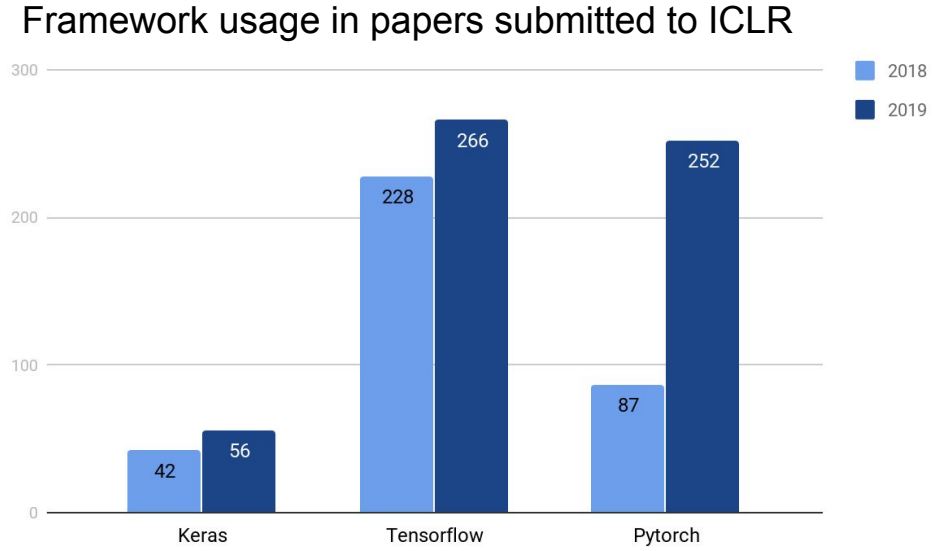
History of Deep Learning Frameworks



History of Deep Learning Frameworks



Pytorch vs. Tensorflow



source:
https://www.reddit.com/r/MachineLearning/comments/9kys38/r_frameworks_mentioned_iclr_20182019_tensorflow/

Pytorch vs. Tensorflow

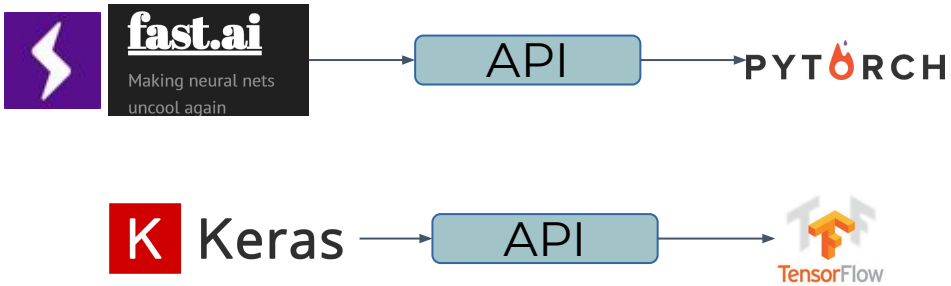
	PYTORCH	TensorFlow
Open Source	Yes (BSD)	Yes (Apache 2.0)
TPU + GPU + CUDA + CUDNN Support	Yes	Yes
Visualization	Tensorboard, TensorboardX, Visdom	Tensorboard, TensorboardX, Visdom
“Pythonic”?	“First-class Python integration” Easy debugging, numpy-style	Harder debugging *(but Eager Mode is default in 2.0)*
Production Ready	TorchScript (torch.jit) or converters	tensorflow.js, tensorflowlite, tf.serving
Pre-trained Models	PyTorch Hub	TensorFlow Hub
Computational Graph	Dynamic *(TorchScript static available)*	Static *(Eager Mode default in 2.0)*

Wrapper APIs

Some libraries offer convenience wrapper functions to make deep learning libraries even more user-friendly and easy to get started with.

"Like most things, API design is not complicated, it just involves following a few basic rules. They all derive from a founding principle: **you should care about your users.** All of them. Not just the smart ones, not just the experts. Keep the user in focus at all times. Yes, including those befuddled first-time users with limited context and little patience. **Every design decision should be made with the user in mind."**

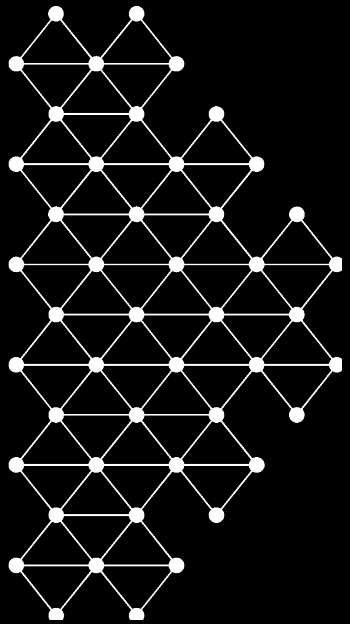
- Francois Chollet, Keras Author



https://williamfalcon.github.io/pytorch-lightning/
https://www.tensorflow.org/guide/keras
https://www.fast.ai/

Quebec
Artificial
Intelligence
Institute

Mila



Project Management

Managing Projects

Colab and **jupyter notebooks** are great tools for quick iteration, data visualization and sharing ideas. **They are not ideal** for scaling and deploying **large projects** and can get cluttered very quickly. Here are some tips for good python code and projects:

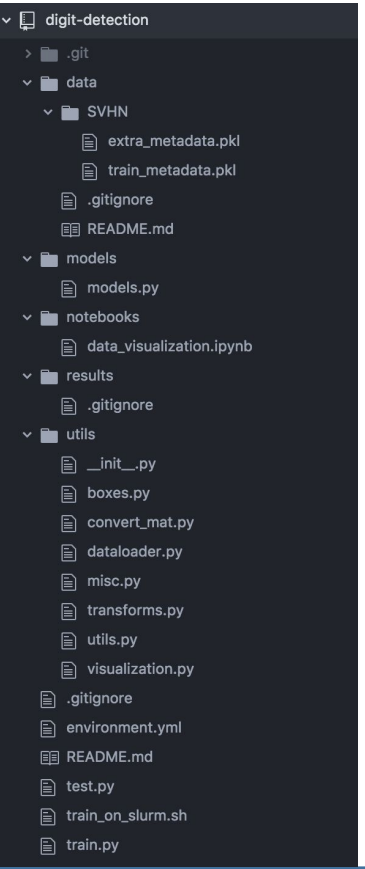
- Maintain an organized codebase
- Use Version Control for your code (Git)
- Use virtual environments
- Use unit tests

Managing Projects

Colab and **jupyter notebooks** are great tools for quick iteration, data visualization and sharing ideas. **They are not ideal** for scaling and deploying **large projects** and can get cluttered very quickly. Here are some tips for good python code and projects:

- **Maintain an organized codebase**
- Use Version Control for your code (Git)
- Use virtual environments
- Use unit tests

Organized Codebase



- Organize your code in a logical hierarchical structure
- Use logical names for variables, filenames, folders etc.
- Avoid code duplication
  - Have the same training/data loading routines regardless of experiments
  - Use object oriented programming paradigms
    - https://realpython.com/python3-object-oriented-programming/
- Follow PEP guidelines (lint your code, think “pythonic”, etc.)
  - https://www.python.org/dev/peps/pep-0008/#a-foolish-consistency-is-the-hobgoblin-of-little-minds
  - http://flake8.pycqa.org/en/latest/#
- Document your code

https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_numpy.html

Managing Projects

Colab and **jupyter notebooks** are great tools for quick iteration, data visualization and sharing ideas. **They are not ideal** for scaling and deploying **large projects** and can get cluttered very quickly. Here are some tips for good python code and projects:

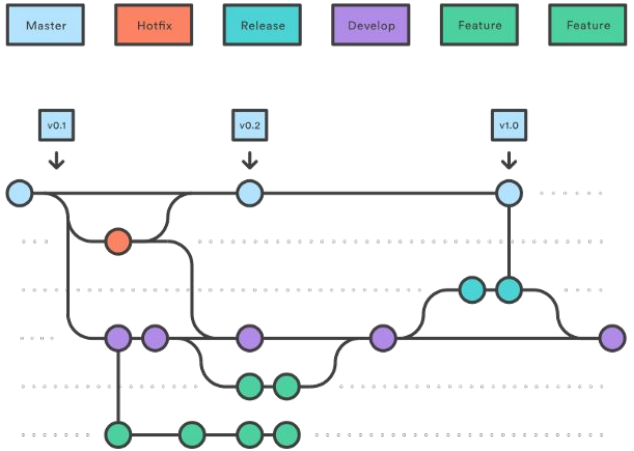
- Maintain an organized codebase
- **Version Control your code (Git)**
- Use virtual environments
- Use unit tests

Version Control Systems

Version Control Systems (VCS) allow users to keep **track** of all **changes** that ever happened in a project. It is great for working collaboratively on large codebases. A very popular (and open source) VCS is **git**.

Version Control Systems

Git focuses on the “branching” paradigm, where changes can be added and merged in a controlled and logical fashion. Each team member works on their own version of the code and can propose or “merge” their changes when new features are ready. Git handles (most) conflicts automatically.



https://www.atlassian.com/git/tutorials/what-is-version-control
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
https://medium.com/gradeup/version-control-system-get-up-to-speed-with-git-ea25b5cb7329

Managing Projects

Colab and **jupyter notebooks** are great tools for quick iteration, data visualization and sharing ideas. **They are not ideal** for scaling and deploying **large projects** and can get cluttered very quickly. Some tips for good python code and projects:

- Maintain an organized codebase
- Version Control your code (Git)
- **Use virtual environments**
- Use unit tests

Virtual Environments

Since python relies on external libraries, versions matter. Libraries might get updated with **breaking changes** which can be dangerous for legacy code. The solution to this is **isolating** each project with their own virtual environment. Even python versions can have an impact on code execution.

CONDA

PIP

https://realpython.com/effective-python-environment/

Managing Projects

Colab and **jupyter notebooks** are great tools for quick iteration, data visualization and sharing ideas. **They are not ideal** for scaling and deploying **large projects** and can get cluttered very quickly. Some tips for good python code and projects:

- Maintain an organized codebase
- Version Control your code (Git)
- Use virtual environments
- **Use unit tests**

Unit tests

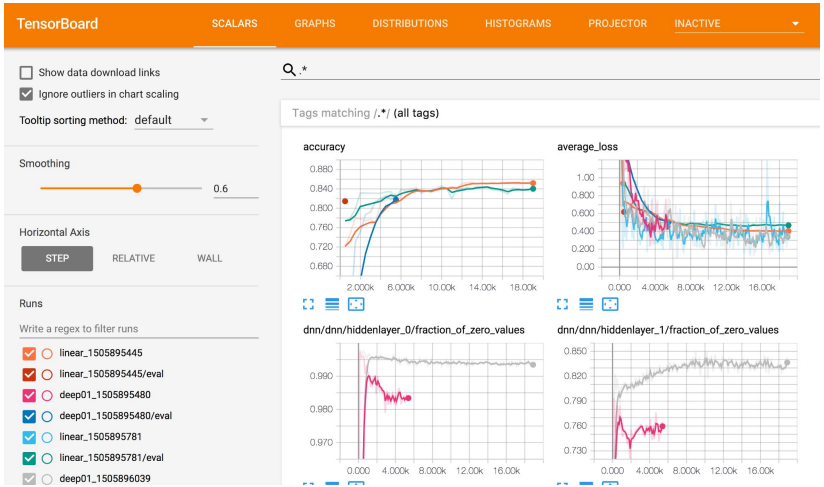
It is good practice to write unit tests that you expect your code to pass every time you run it. This is a great way to catch bugs early and ensure that code works as expected in an explicit way.

```
1  def test_tensorflow():
2      '''Basic test to make sure tensorflow is properly installed'''
3
4      import tensorflow as tf
5
6      graph = tf.constant(4) + tf.constant(3)
7
8      with tf.Session() as sess:
9          out = sess.run(graph)
10
11     assert out == 7
```

https://docs.pytest.org/en/latest/

Experiment Management

Deep learning is inherently **empirical**. When solving a task, you will want to try many **variations** of a network, i.e. use different hyperparameters, models, processing, etc. You will therefore need tools to **organize** and **monitor** your experiments.

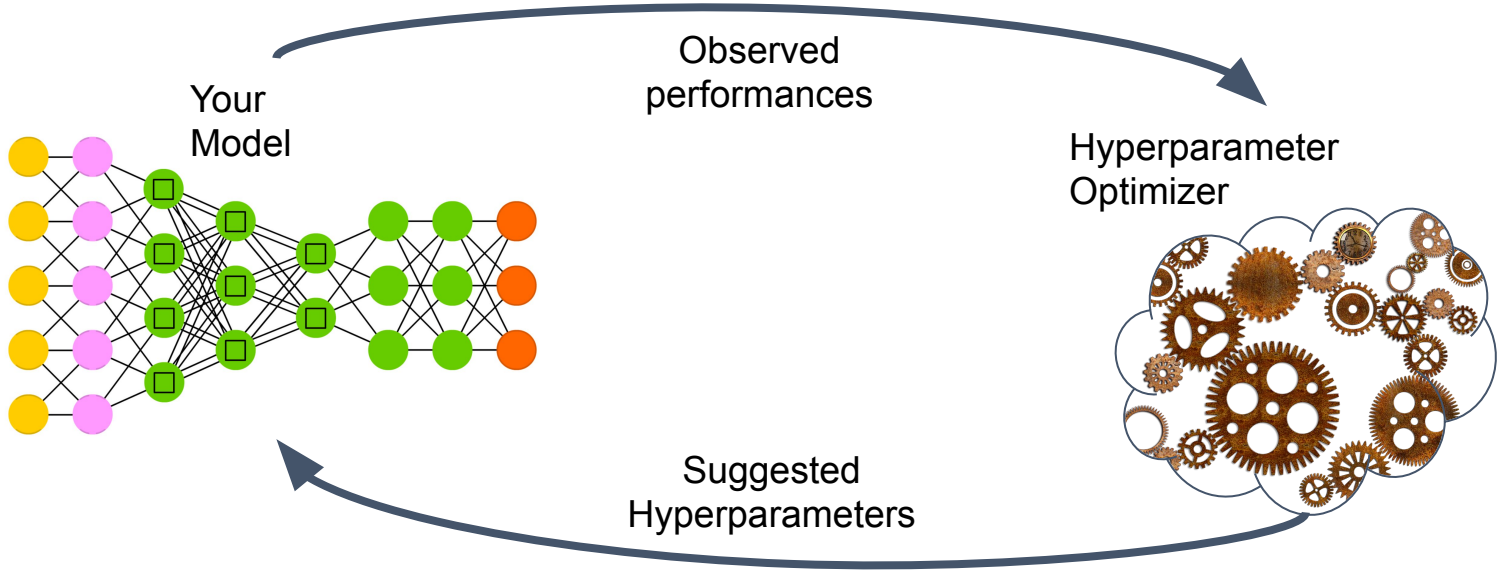


					Parameters		Metrics		
	Date	User	Source	Version	alpha	l1_ratio	mae	r2	rmse
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	1	0.649	0.04	0.862
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.5	0.648	0.046	0.859
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.2	0.628	0.125	0.823
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	1	0	0.619	0.176	0.799
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	1	0.648	0.046	0.859
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.5	0.628	0.127	0.822
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.2	0.621	0.171	0.801
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0	0.615	0.199	0.787
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	1	0.578	0.288	0.742
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.5	0.578	0.288	0.742
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.2	0.578	0.288	0.742
☐	2018-06-04 23:00:08	mlflow	train.py	05e956	0	0	0.578	0.288	0.742

https://www.tensorflow.org/guide/summaries_and_tensorboard
https://mlflow.org/docs/latest/tutorial.html

Hyperparameter Optimization

Manually tweaking parameters can be exhausting and time consuming. Many libraries also offer the possibility to suggest automatically how to update parameters based on empirical results.



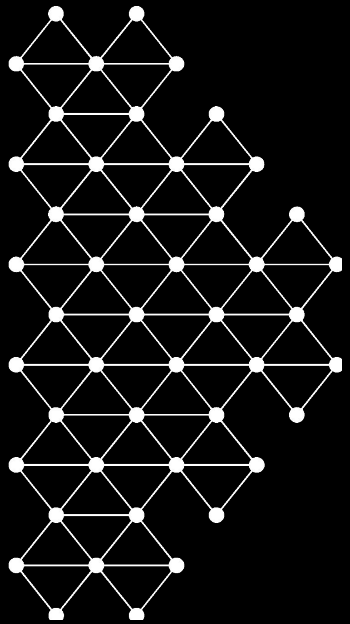
http://www.asimovinstitute.org/neural-network-zoo/

Experiment Management Libraries

Name	Experiment Monitoring	Experiment Organizing	Hyperparameter Optimization	Open Source
Comet.ml	YES	YES	YES	Freemium
Trains	YES	YES	NO	YES
MLFlow	YES	YES	NO	YES
Tensorboard	YES	NO (Limited)	NO	YES
Visdom	YES	NO (Limited)	NO	YES
scikit-optimize	NO	NO	YES	YES
AX.dev	NO	YES (Limited - No GUI)	YES	YES
Orion	NO	NO	YES	YES

Quebec
Artificial
Intelligence
Institute

Mila



Hardware

Hardware

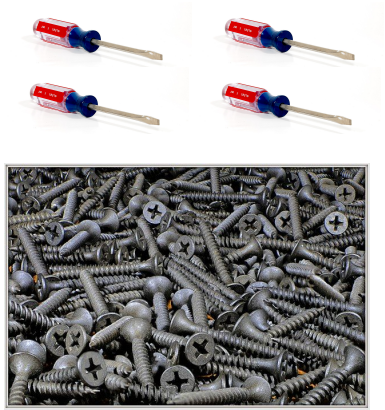
A computer typically consists of a **CPU** and a **GPU**. They were both designed to perform different types of operations. The **CPU** is designed to handle **fewer operations** rapidly and **sequentially** while the **GPU** can handle **many** operations in **parallel** but can be slower.

Hardware

A computer typically consists of a **CPU** and a **GPU**. They were both designed to perform different types of operations. The **CPU** is designed to handle **fewer operations** rapidly and **sequentially** while the **GPU** can handle **many** operations in **parallel** but can be slower.

GPU vs. CPU

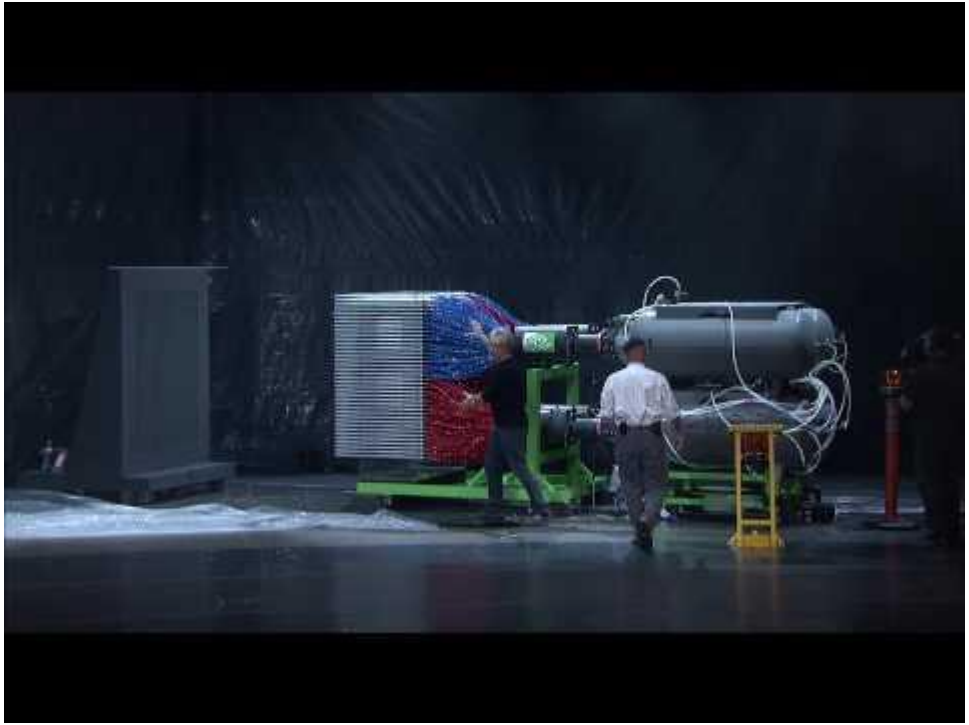
When training deep learning models, most operations are independent and can be massively parallelized. Computations can be done in batches and results aggregated for much faster results.



CPU vs. GPU

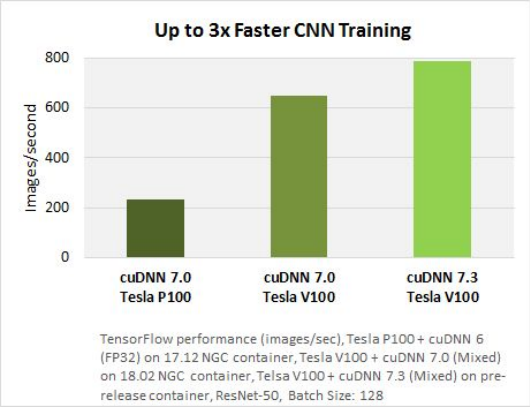
GPUs were originally developed by the **gaming** industry. The term first appeared with the emergence of the PlayStation 1. They are optimized to perform parallel computations. **GPUs** happen to be very **efficient** at computing **matrix** and **vector** operations in **parallel** which can make them very suitable for machine learning and deep learning.

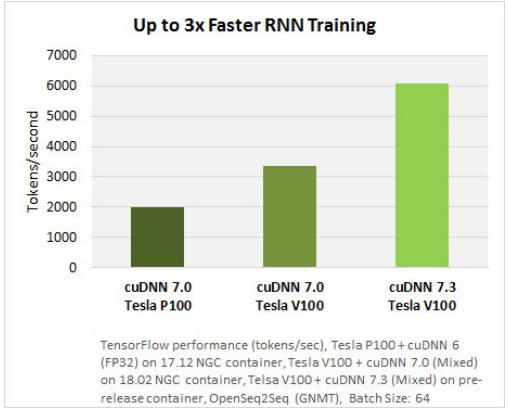
CPU vs. GPU

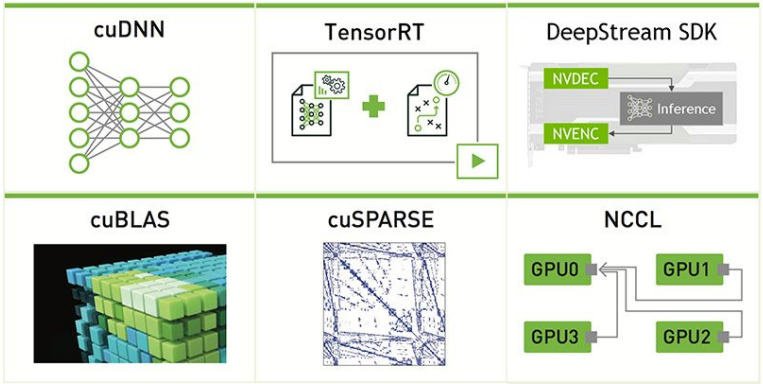


CPU vs. GPU

Modern deep learning frameworks use highly optimized code to parallelize computations automatically on the GPU.





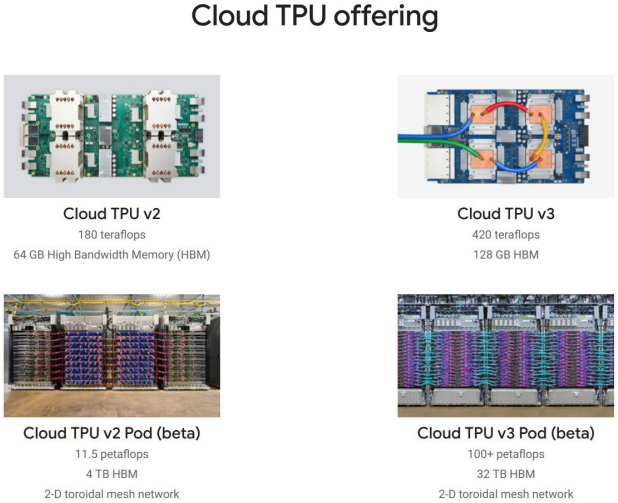


CPU vs. GPU vs. TPU

As deep learning becomes more prevalent, chips have become increasingly specialized for deep learning. Google has developed their own Tensor Processing Unit (TPU) to rival the GPU.

Platform	*Unit*	*Version*	*Mem Type*	*Mem (GB)*	*Mem Bdw (GB/s)*	Peak *FLOPS*
CPU	1 VM	Skylake	DDR4	120	16.6	2T SP†
GPU (DGX-1)	1 Pkg	V100 (SXM2)	HBM2	16	900	125T
TPU	1 Board (8 cores)	v2	HBM	8	2400	180T
TPUv3	8 cores	v3	HBM	16	3600*	420T

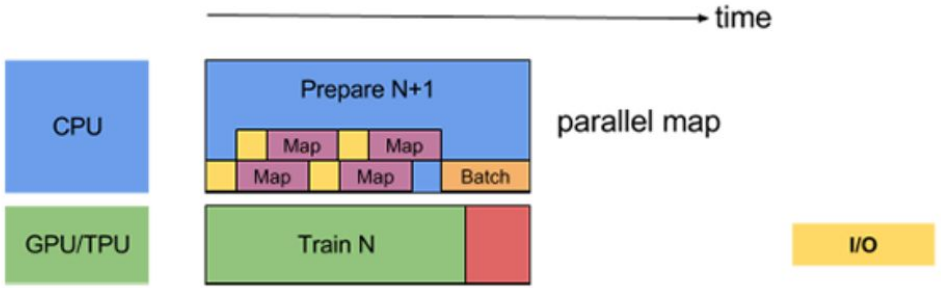
† Single precision: 2 FMA × 32 SP × 16 cores × 2G frequency = 2 SP TFLOPS
* Estimated based on empirical results (Section 4.5).



https://arxiv.org/pdf/1907.10701.pdf

Performance

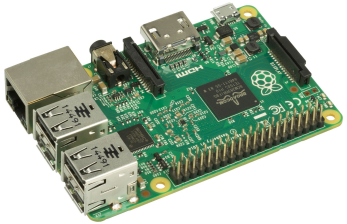
Training deep learning models involves more than just matrix and tensor operations. Data has to be **preprocessed** before it can be computed by the **GPU**. Most modern frameworks will enable preprocessing of a new batch of data to be done on the CPU while the GPU computes results on the previous batch to maximize resource utilization.

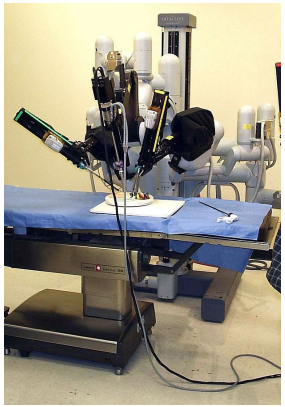


https://www.tensorflow.org/beta/guide/data_performance

Edge Computing

While GPU/TPUs enable batch computations to be sped up, it should be mentioned that CPUs can still be very fast and cost effective for **inference**. It can also be hard to guarantee access to GPUs when doing edge computing, i.e on-device computation.





Cloud Computing

Deploying computations on the cloud has pros and cons.

Pros:

- Get access to the latest hardware
- Low initial cost
- Scale up and down based on demand

Cons

- Can be prohibitively expensive when running 24/7

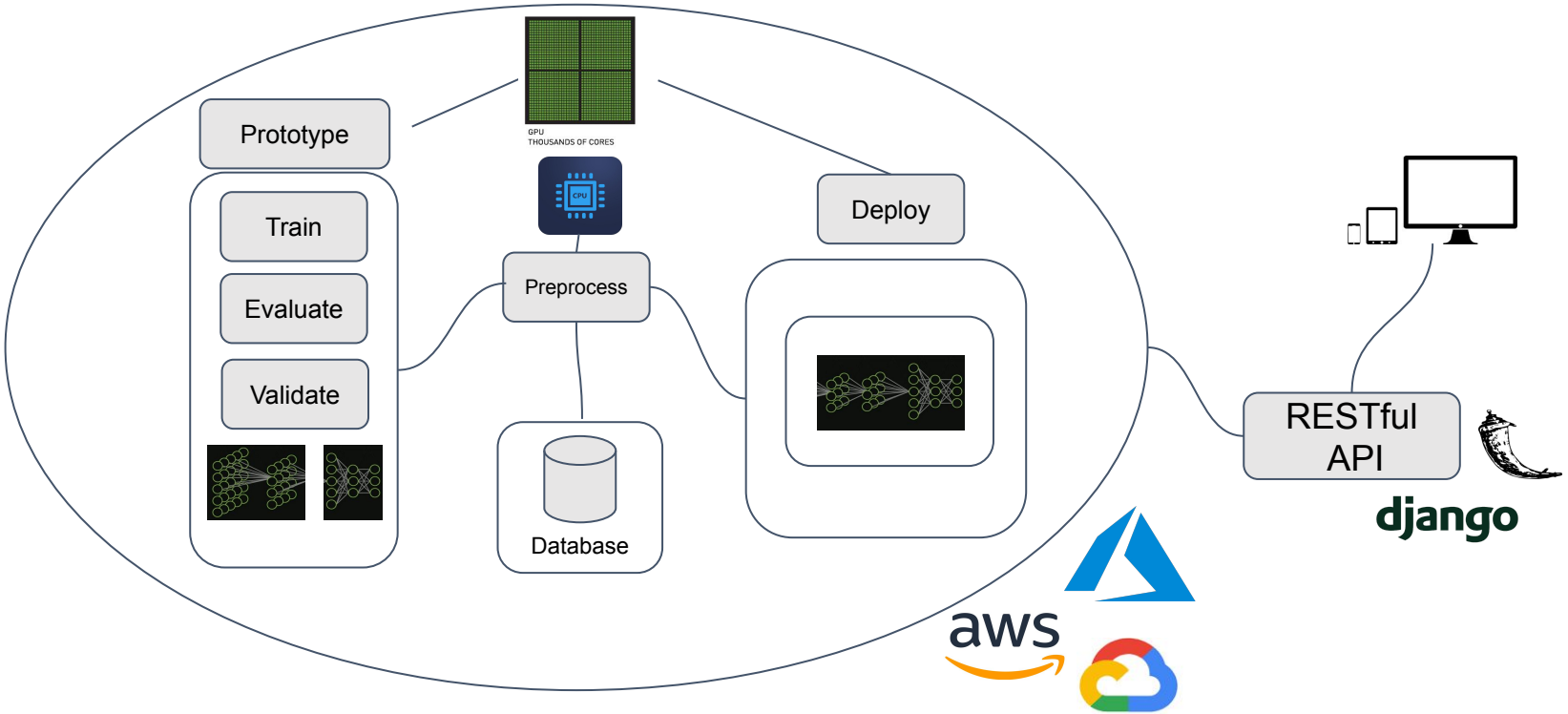
Cloud Computing Cost + Hardware

ImageNet Training

Submission Date	Model	Time to 93% Accuracy	Cost (USD)	Max Accuracy	Hardware	Framework
Sep 2018	ResNet50 *Google Cloud TPU* source	2:44:31	$12.60	93.34%	GCP n1-standard-2, Cloud TPU	TensorFlow v1.11.0
Sep 2018	ResNet-50 *fast.ai/DIUx (Yaroslav Bulatov, Andrew Shaw, Jeremy Howard)* source	0:18:06	$118.07	93.11%	16 p3.16xlarge (AWS)	PyTorch 0.4.1
Sep 2018	Resnet 50 *Andrew Shaw, Yaroslav Bulatov, Jeremy Howard* source	0:18:53	$61.63	93.19%	64 * V100 (8 machines - AWS p3.16xlarge)	ncluster / Pytorch 0.5.0a0+0e8088d
Apr 2018	ResNet50 *Google Cloud TPU* source	8:52:33	$58.53	93.11%	GCP n1-standard-2, Cloud TPU	TensorFlow v1.8rc1
Mar 2018	ResNet50 *Google Cloud TPU* source	12:26:39	$82.07	93.15%	GCP n1-standard-2, Cloud TPU	TensorFlow v1.7rc1
Jan 2018	ResNet50 *DIUX* source	14:37:59	$358.22	93.07%	p3.16xlarge	tensorflow 1.5, tensorpack 0.8.1

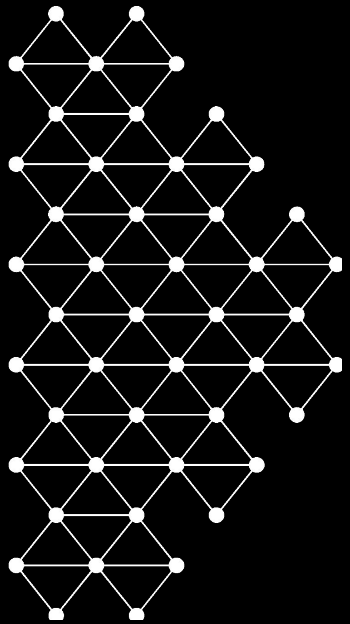
https://dawn.cs.stanford.edu/benchmark/
https://towardsdatascience.com/maximize-your-gpu-dollars-a9133f4e546a

Cloud Computing Pipeline



Quebec
Artificial
Intelligence
Institute

Mila



Questions?