Institut
québécois
d'intelligence
artificielle

Mila

Université de Montréal

McGill

Institut
québécois
d'intelligence
artificielle

Mila

Introduction aux librairies d'apprentissage machine

Jeremy Pinto
jeremy.pinto@mila.quebec

Contenu de la présentation

- Survol de librairies python pour l’apprentissage machine
- Exemple pratique d’apprentissage machine
- Comparaison de librairies d’apprentissage profond

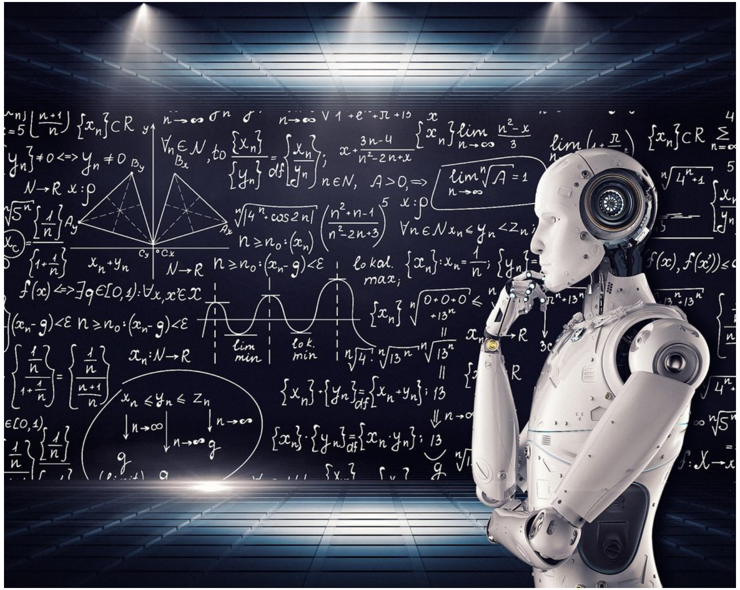


Image via www.vpnsrus.com

Langages

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!

python™

Calcul Scientifique

NumPy

scikit learn

Keras

SciPy

TensorFlow

PYTORCH

Visualisation

plotly

matplotlib

Visdom

seaborn

Gestion de données

pandas $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Environnements

IP[y]: IPython Interactive Computing

jupyter

CO

Gestion des libraires

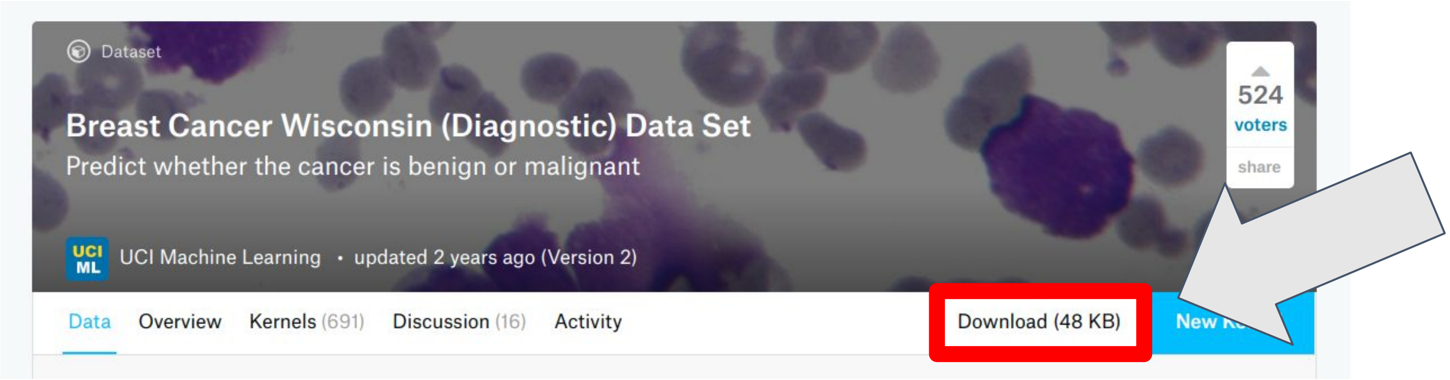
CONDA

PIP

Pipenv

Apprentissage machine avec Python

Exemple - Kaggle.com



“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. ”

Pipeline

Explorer les données

pandas

Structurer les données en Python

NumPy

Traiter les données

scikit learn

Apprentissage machine

seaborn

Visualisation des données

Pipeline

Explorer les données

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Structurer les données en Python

NumPy

Traiter les données

scikit learn

Apprentissage machine

seaborn

Visualisation des données

Explorer les données

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

	# id	diagnosis	# radius_mean	# texture_mean	# perimeter_mean
	ID number	The diagnosis of breast tissues (M = malignant, B = benign)	mean of distances from center to points on the perimeter	standard deviation of gray-scale values	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

breast-cancer....zip

Pipeline

Explorer les données

Structurer les données en Python

Traiter les données

Apprentissage machine

Visualisation des données

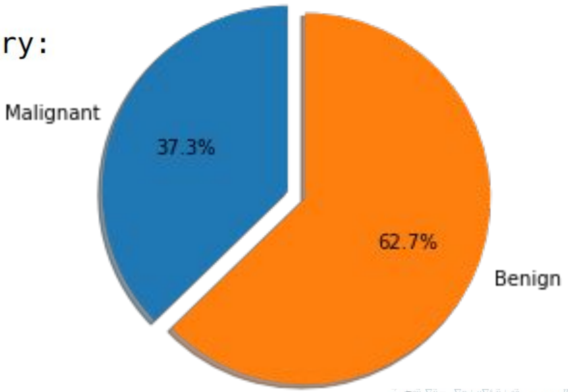
Pandas

```python
import pandas as pd

dataset = pd.read_csv('data.csv')
print("Number of total entries: ", len(dataset))
print("")
print("Entries per category:")
print(dataset["diagnosis"].value_counts())

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

	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	

5 rows × 33 columns

```
Number of total entries:  569

Entries per category:
B     357
M     212
```



Pipeline

Explorer les données

Structurer les données en Python

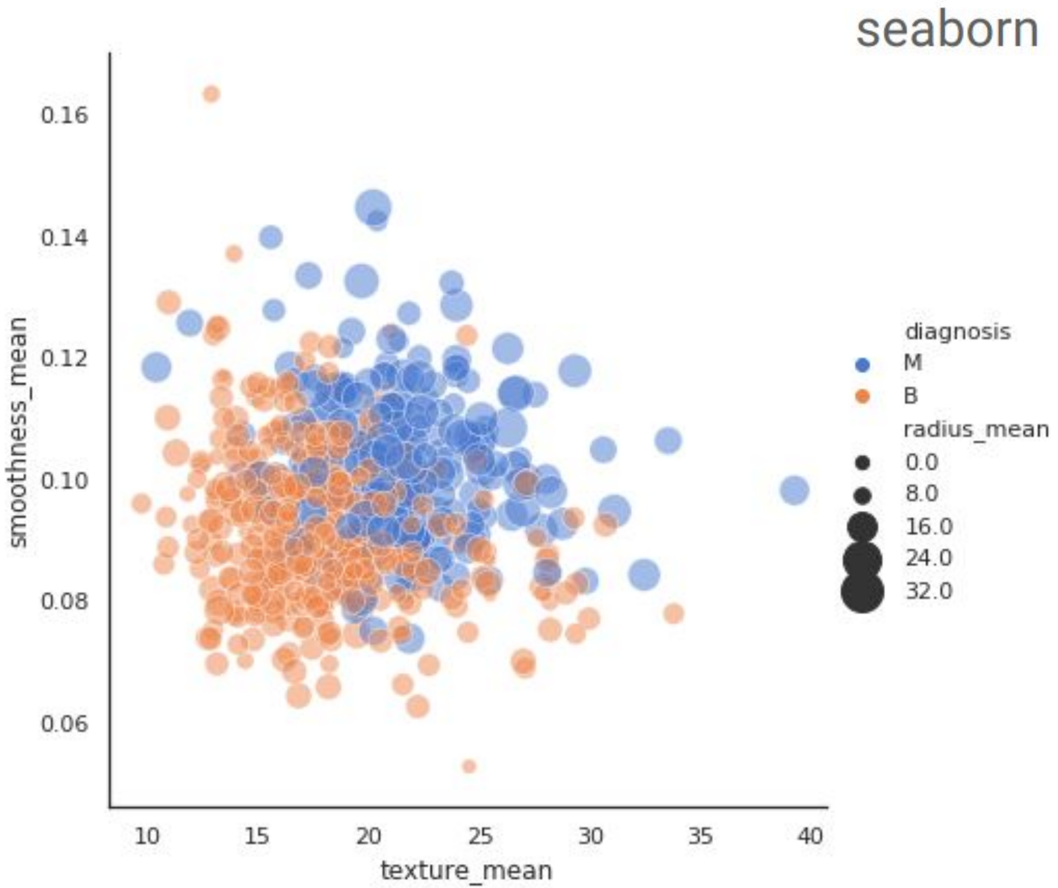
Traiter les données

Apprentissage machine

Visualisation des données

Seaborn

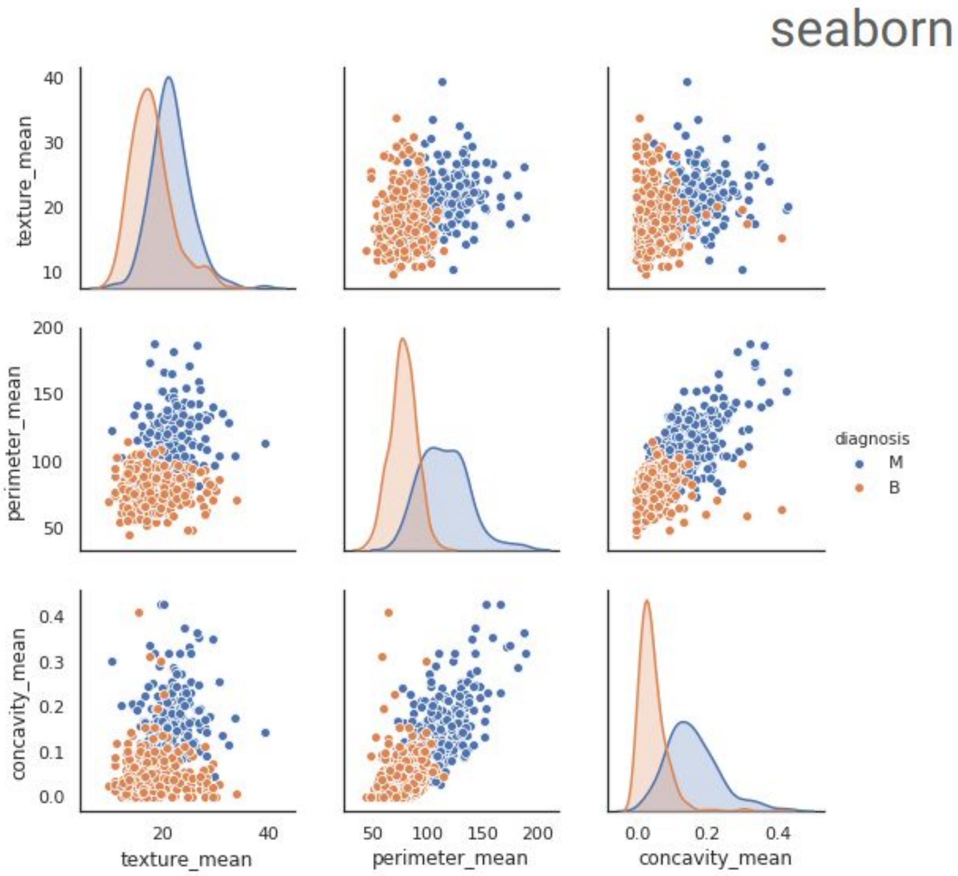
```python
import seaborn as sns
import pandas as pd

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



Mila

Seaborn

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



Pipeline

Explorer les données

Structurer les données en Python

Traiter les données

Apprentissage machine

Visualisation des données

Numpy

- Permet de manipuler des données en N-dimensions (vecteurs, matrices, tenseurs)

- Opérations mathématiques hautement optimisées (multiplication de matrices, FFT, traitement de signal, etc.)

- Intégration avec Scikit-Learn, Pandas, etc.

```python
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')
```

X =

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
```

y =

```
1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
```

Pipeline

Explorer les données → Structurer les données en Python → Traiter les données → Apprentissage machine

Visualisation des données

Scikit-Learn

- Librairie de Machine Learning
- Intégration avec Numpy
- API simple et réutilisable

```python
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 - Régression Logistique

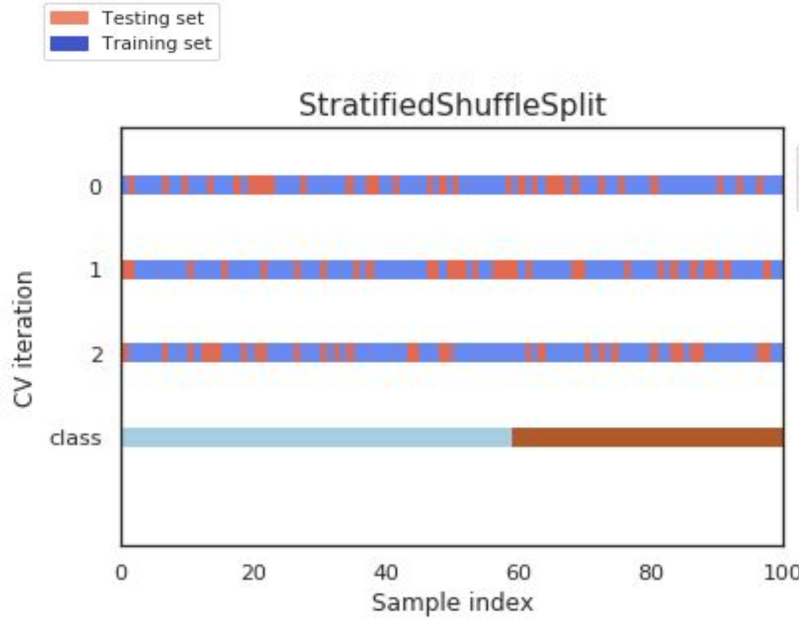
- Librairie de Machine Learning
- Intégration avec Numpy
- API simple et réutilisable

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

```python
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_score(y, y_pred))
```

Accuracy : 96%

Validation croisée




```python
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):

    logreg.fit(X[train_index],y[train_index])

    y_pred = logreg.predict(X[test_index])
    acc = accuracy_score(y[test_index], y_pred)

    acc_tot += acc


print("Average accuracy :", acc_tot/n_splits)
```

Traitement de données

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

$$x_{std} = \frac{(x - \mu)}{\sigma}$$

Traitement de données

```
fit_model(X, y, n_splits=30, scale=False)
```

```
Number of splits used : 30
Average Accuracy: 94.8 %
```

```
fit_model(X, y, n_splits=30, scale=True)
```

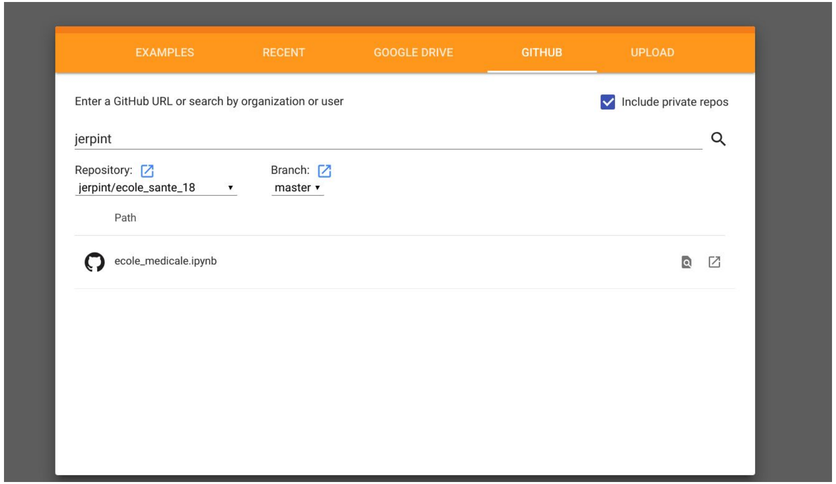
```
Number of splits used : 30
Average Accuracy: 97.4 %
```

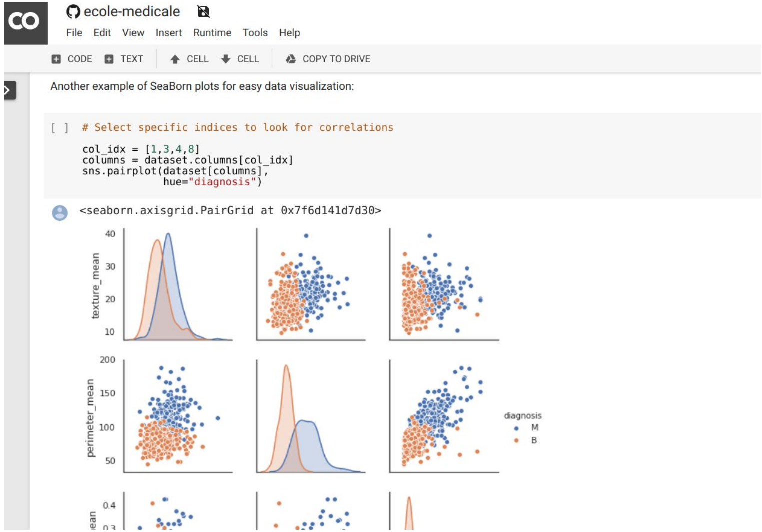
Algorithmes disponibles

1. Supervised learning

Exemple en ligne

Pour recréer toutes les expériences et figures, rendez-vous au https://github.com/jerpint/ecole_sante_18





Mila

Apprentissage profond avec python

Librairies d'apprentissage profond

Torch
(2002)

MILA
theano
(2008)

Caffe
(2013)

cuDNN
DL4J
(2014)

mxnet
Chainer
Keras
TensorFlow
(2015)

Cognitive Toolkit
(2016)

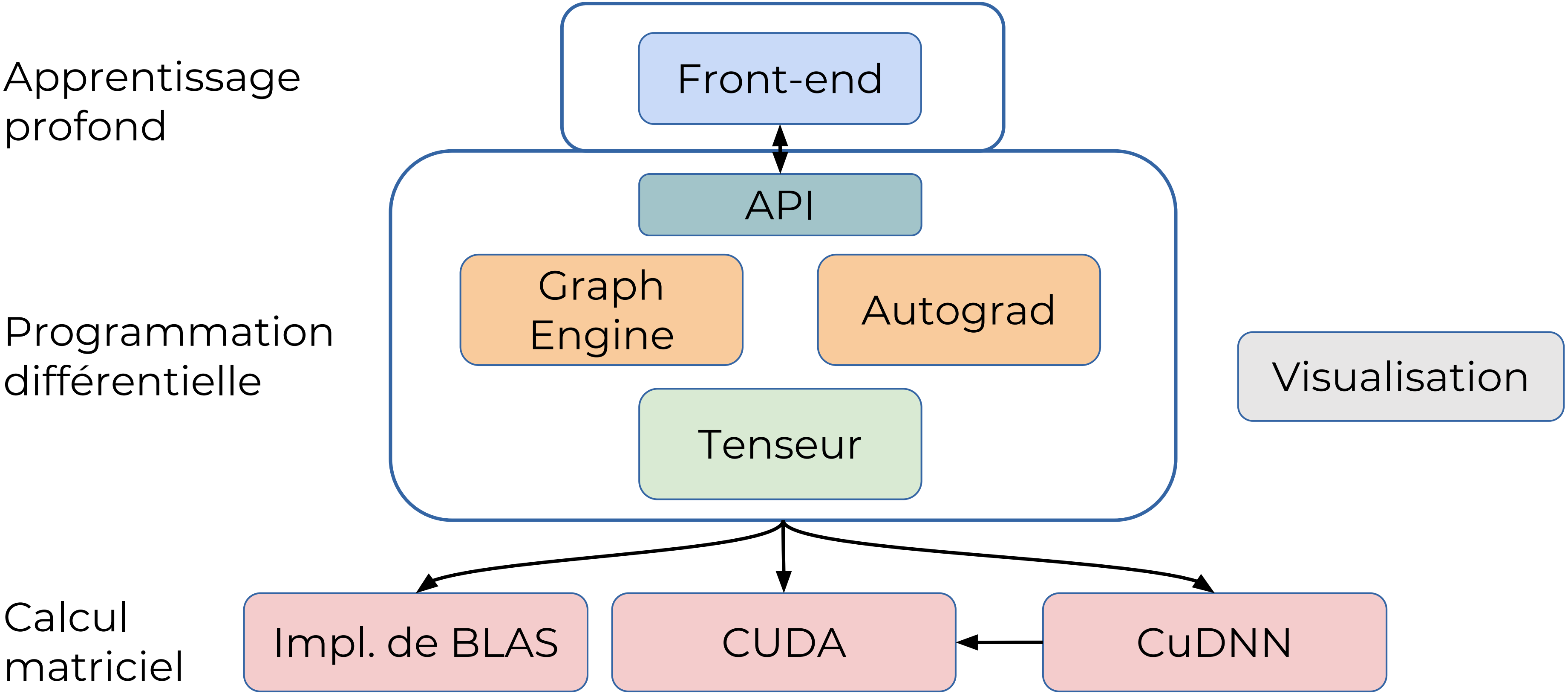
Caffe2
PYTORCH
(2017)

TensorFlow 2.0
(2018?)

PYTORCH 1.0
(2018)

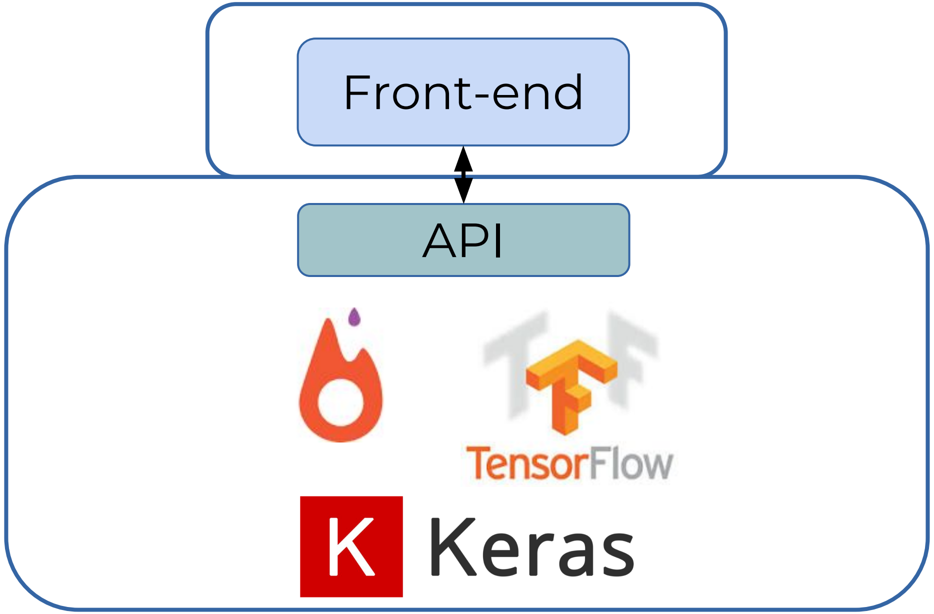
2017/11/15: Release of Theano 1.0.0
Arrêt du développement logiciel par le Mila
Précurseur à beaucoup d'idées qui se retrouvent dans les librairies plus récentes.

Librairies


Apprentissage profond
Programmation différentielle
Calcul matriciel
Front-end
API
Graph Engine
Autograd
Tenseur
Visualisation
Impl. de BLAS
CUDA
CuDNN

Librairies

Apprentissage profond



Visualisation

Caractéristiques désirées

- Une **hiérarchie** d'outils
- Utilisation de **matériel de calcul spécifique** (GPU-TPU)
- **Prototypage** rapide et versatile
- Passage de la recherche à la **production**
- Support de la **communauté** d'utilisateurs + open source

Une hiérarchie d’outils

Front-end

Permet de se concentrer sur les concepts de deep learning. Pas besoin de réinventer la roue (conv2d, ReLu, SoftMax, BatchNorm, etc.)!

torch.Storage

⊟ torch.nn

- Parameters
- ⊞ Containers
- ⊞ Convolution layers
- ⊞ Pooling layers
- ⊞ Padding layers
- ⊞ Non-linear activations (weighted sum, nonlinearity)
- ⊞ Non-linear activations (other)
- ⊞ Normalization layers
- ⊞ Recurrent layers
- ⊞ Linear layers

class **torch.nn.Sequential**(**args*) [source]

A sequential container. Modules will be added to it in the order they are passed in the constructor. Alternatively, an ordered dict of modules can also be passed in.

To make it easier to understand, here is a small example:

```
# Example of using Sequential
model = nn.Sequential(
          nn.Conv2d(1,20,5),
          nn.ReLU(),
          nn.Conv2d(20,64,5),
          nn.ReLU()
        )

# Example of using Sequential with OrderedDict
model = nn.Sequential(OrderedDict([
          ('conv1', nn.Conv2d(1,20,5)),
          ('relu1', nn.ReLU()),
          ('conv2', nn.Conv2d(20,64,5)),
          ('relu2', nn.ReLU())
        ]))
```

source : https://pytorch.org/docs/stable/nn.html

Une hiérarchie d'outils

API

L'API permet de programmer des concepts mathématiques sur les données afin de créer de nouveaux modules.

Entraînement du modèle

```python
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

source: https://github.com/yunjey/pytorch-tutorial

PyTorch vs. TensorFlow

	PYTORCH	TensorFlow / Keras
Open Source	Oui (BSD)	Oui (Apache 2.0)
Support GPU + CUDA + CUDNN	Oui	Oui
Visualization	TensorboardX (nouveau), Visdom	Tensorboard
“Pythonic”	Oui, “First-class Python integration”, pdb fonctionne sur le graph directement	Non, pdb en plus de tfdbg
Production	Oui (Torch.jit, nouveau depuis v1.0)	Oui (tensorflow.js, tensorflowlite, etc.)
Modèles pré-entraînés	Oui	Oui
Graphe Computationnel	Dynamique	Statique

PyTorch vs. TensorFlow

	PyTorch	TensorFlow / Keras
Open Source	Oui (BSD)	Oui (Apache 2.0)
Support GPU + CUDA + CUDNN	Oui	Oui
Visualization	TensorboardX (nouveau), Visdom	Tensorboard
“Pythonic”	Oui, “First-class Python integration”, pdb fonctionne sur le graph directement	Non, pdb en plus de tfdbg
Production	Oui (Torch.jit, nouveau depuis v1.0)	Oui (tensorflow.js, tensorflowlite, etc.)
Modèles pré-entraînés	Oui	Oui
Graphe Computationnel	Dynamique	Statique

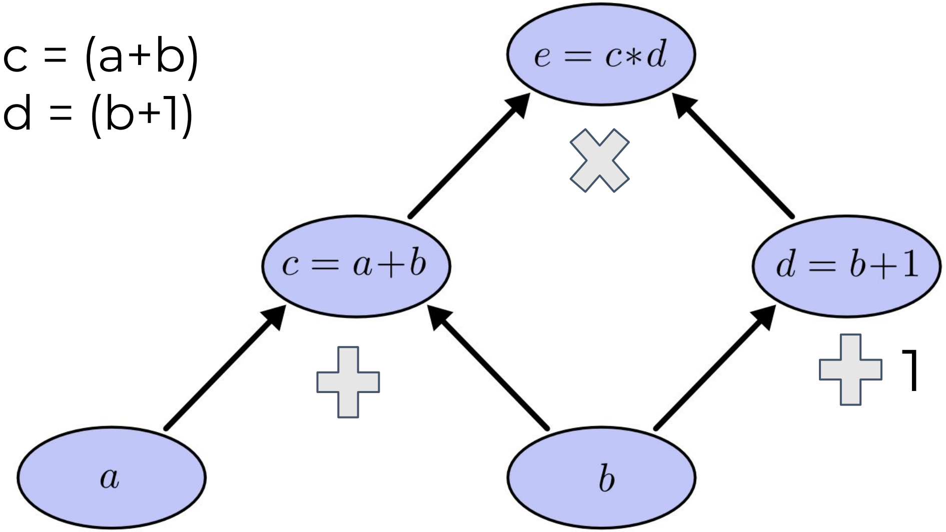
Graphe Computationnel

Graphe Computationnel

Le **graphe computationnel** permet de représenter des opérations mathématiques complexes et de **calculer des dérivées** facilement

Exemple:
e = (a+b)(b+1) = c∘d

c = (a+b)
d = (b+1)

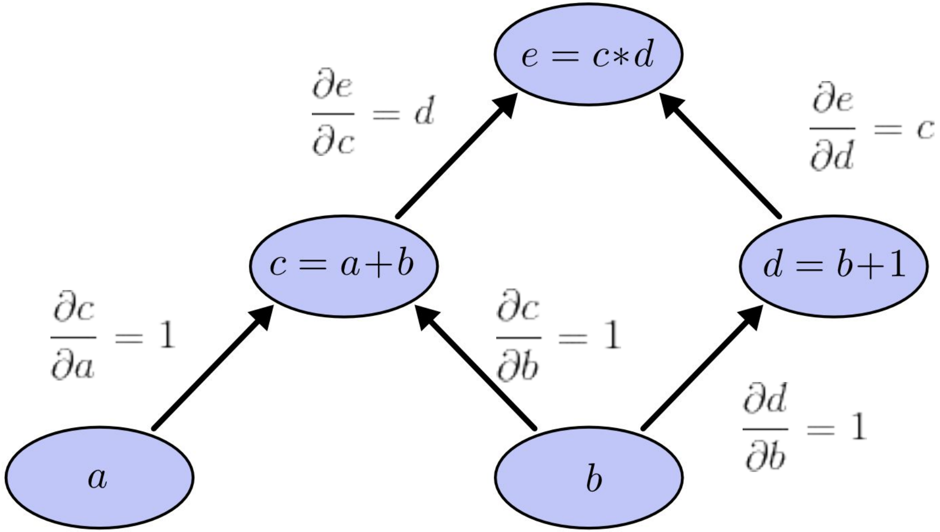


Graphe Computationnel

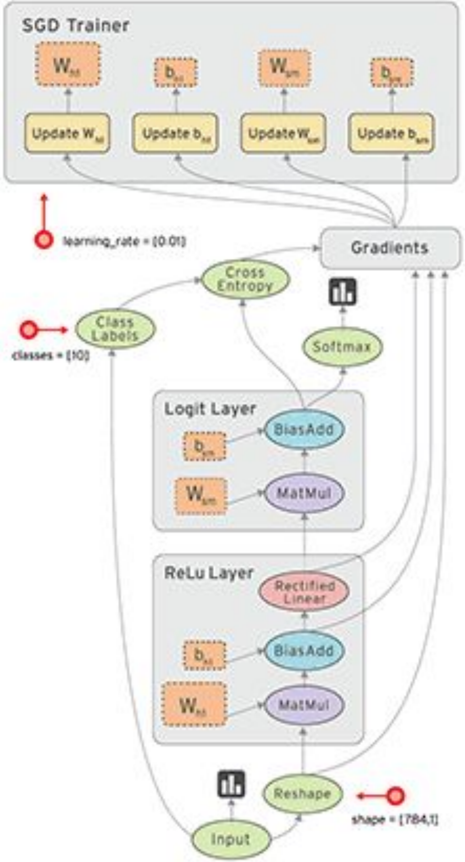
Le **graphe computationnel** permet de représenter des opérations mathématiques complexes et de **calculer des dérivées** facilement

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial a} = d$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = c + d$$



Exemple



SGD Trainer
Gradients
learning_rate = [0.01]
Cross Entropy
Class Labels
classes = [10]
Softmax
Logit Layer
BiasAdd
MatMul
ReLu Layer
Rectified Linear
BiasAdd
MatMul
Reshape
shape = [784,1]
Input

Graphe Computationnel

Graphe Computationnel
Dynamique

VS

Graphe Computationnel
Statique

Graphe Computationnel

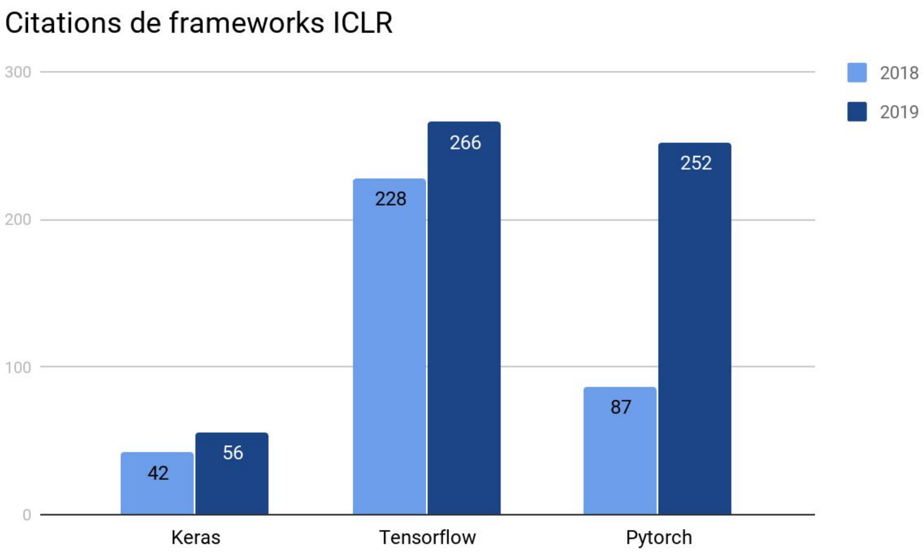
```python
import tensorflow as tf


x = tf.constant([[37.0, -23.0], [1.0, 4.0]])
w = tf.Variable(tf.random_uniform([2, 2]))
y = tf.matmul(x, w)
output = tf.nn.softmax(y)
init_op = w.initializer

with tf.Session() as sess:
    # Run the initializer on `w`.
    sess.run(init_op)
```

```python
import torch

xx = torch.tensor([[37.0, -23.0], [1.0, 4.0]])
ww = torch.rand((2,2))
yy = torch.matmul(xx, ww)

output = torch.softmax(yy, dim=0)
```

TensorFlow

VS

PyTorch ou TensorFlow



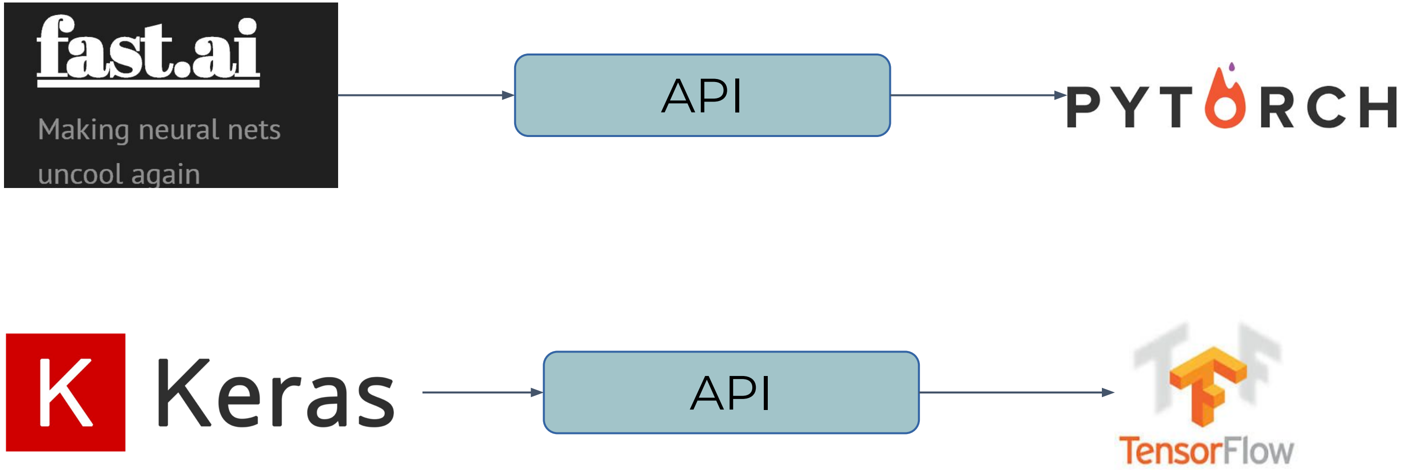
source:
https://www.reddit.com/r/MachineLearning/comments/9kys38/r_frameworks_mentioned_iclr_20182019_tensorflow/

Mila

Une hiérarchie d'outils

"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, auteur de Keras



Une hiérarchie d'outils

MNIST entraîné sur le modèle ResNet18 en **presque** 4 lignes

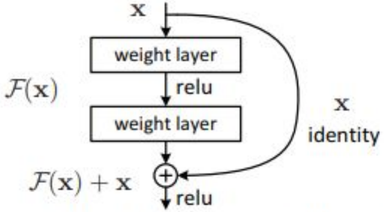


Figure 2. Residual learning: a building block.

fast.ai

Making neural nets uncool again

```
path = untar_data(URLs.MNIST_SAMPLE)
data = ImageDataBunch.from_folder(path)
learn = ConvLearner(data, tvm.resnet18, metrics=accuracy)
learn.fit(1)
```

```
Total time: 00:05
epoch  train loss  valid loss  accuracy
0      0.081393    0.046429    0.985770  (00:05)
```

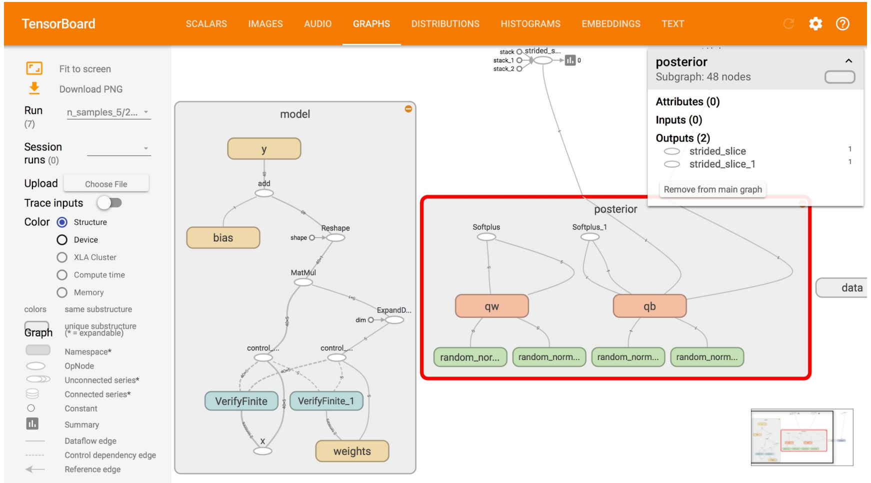

Mila

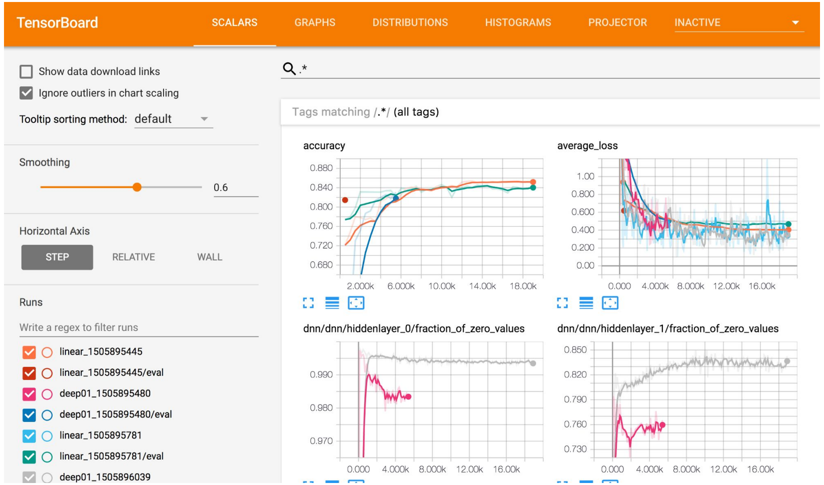
Une hiérarchie d'outils

matplotlib

Visualisation

La visualisation est importante pour diagnostiquer les problèmes d'apprentissage.





Mila

Interopérabilité de modèles

"ONNX enables models to be trained in one framework and transferred to another for inference."

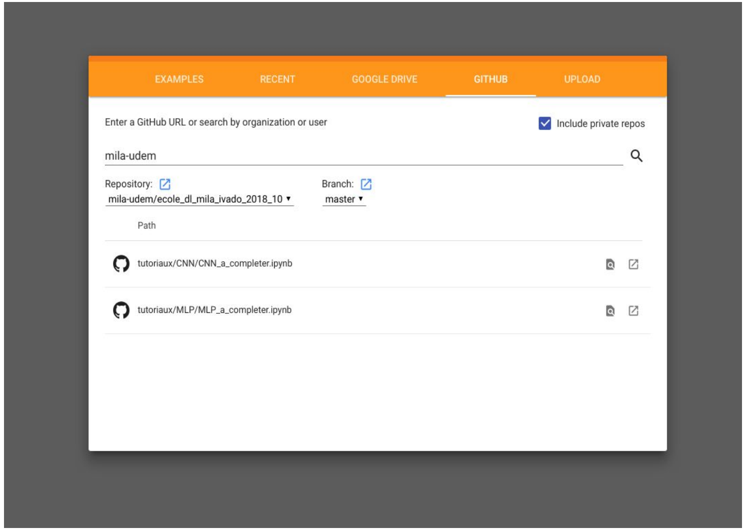
Installation + Utilisation

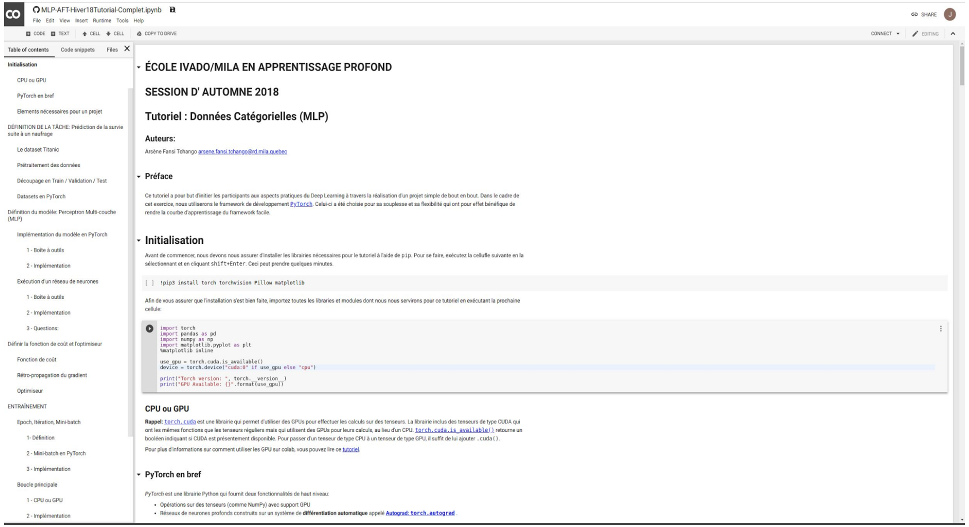
Grâce à Google Colab, PyTorch est facilement installé avec accès à un GPU dans le cloud.

Exemples

Grâce à Google Colab, PyTorch est facilement installé avec accès à un GPU dans le cloud.

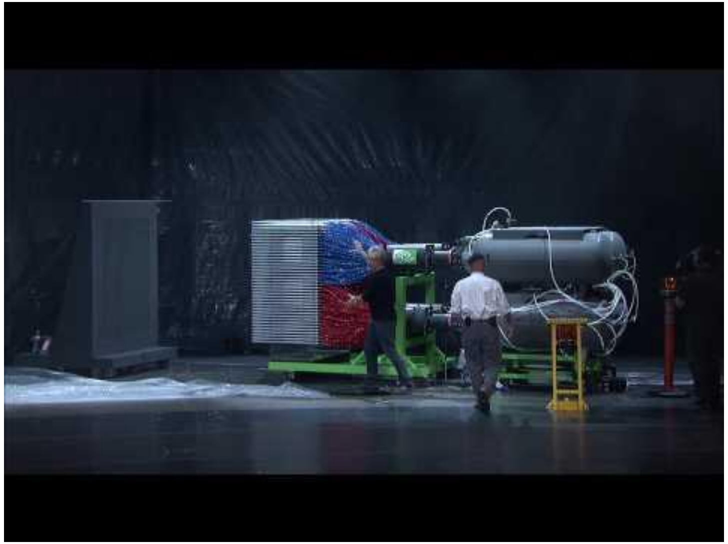
https://github.com/mila-udem/ecole_dl_mila_ivado_2018_10





Mila

GPU ou CPU?



Services Cloud avec GPU/TPU

Azure

aws

Paperspace

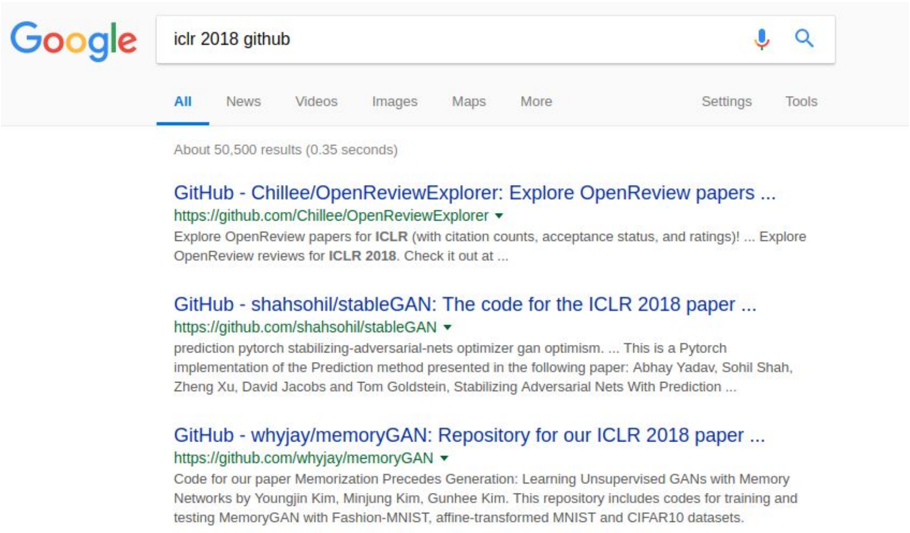
Google Cloud

L'infonuagique

- Avantages
    - Réduction des coûts initiaux
    - Utilisation à grande échelle
    - “Démocratique” (ne dépend pas de la puissance de calcul de l'utilisateur)

- Désavantages
    - Nécessite une connection internet
    - Coût par utilisation
    - Installation n'est pas toujours évidente (beaucoup d'options, beaucoup de choix de design importants)

Outils pratiques - GitHub



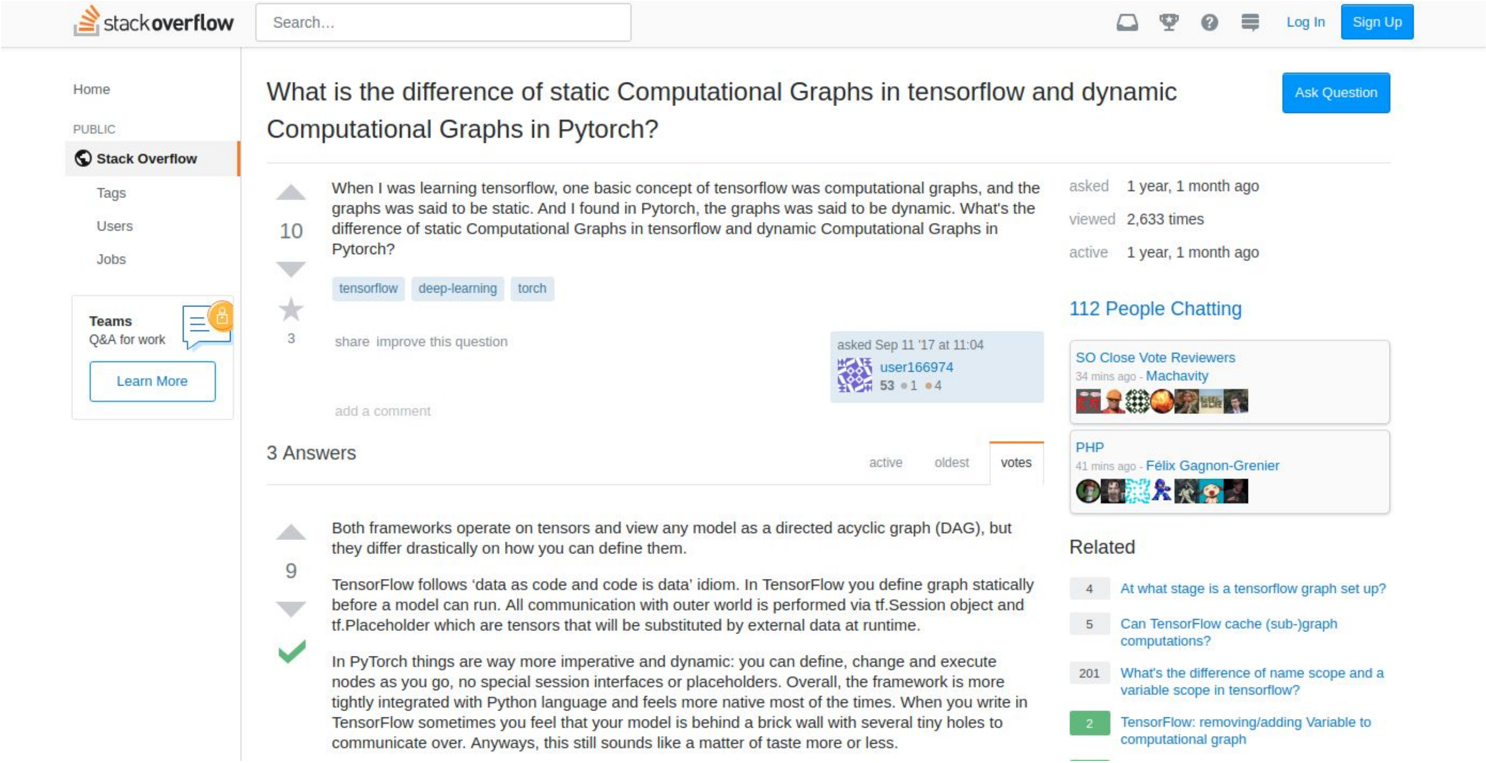
Exemple de Repos utiles:

https://github.com/google/seq2seq

https://github.com/facebookresearch/Detectron

https://github.com/openai/gym

Outils pratiques - Stack Overflow



Institut
québécois
d'intelligence
artificielle

Mila

Merci!
Questions?

Jeremy Pinto
jeremy.pinto@mila.quebec