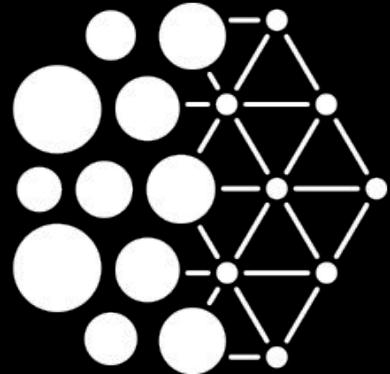


Institut  
québécois  
d'intelligence  
artificielle



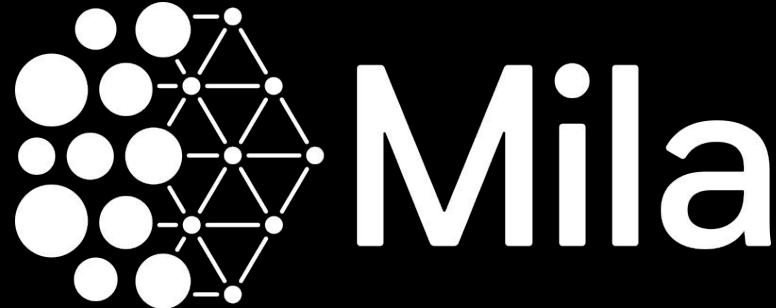
Mila

Université   
de Montréal



McGill

Institut  
québécois  
d'intelligence  
artificielle



# Introduction aux librairies d'apprentissage machine

Jeremy Pinto  
[jeremy.pinto@mila.quebec](mailto: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

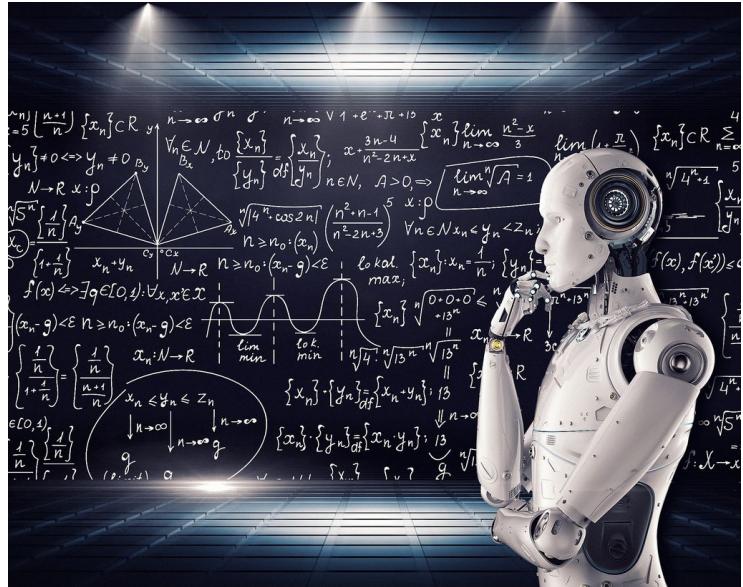


Image via [www.vpnsrus.com](http://www.vpnsrus.com)

# 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!



K Keras



PYTORCH



Calcul Scientifique



Visdom seaborn

Visualisation



Gestion de données



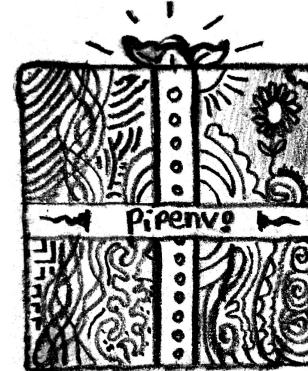
Environnements

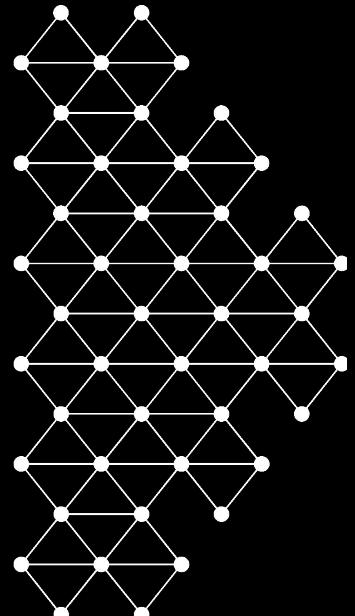


# Gestion des libraires

CONDA

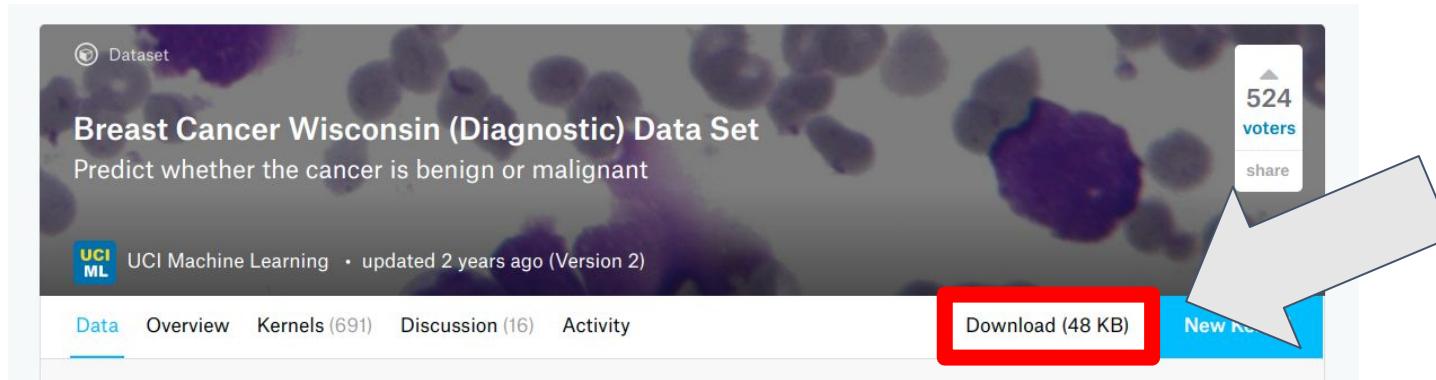
PIP





# 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.”

📁 breast-cancer....zip ^

# Pipeline



Explorer les données

Structurer les données en Python

Traiter les données

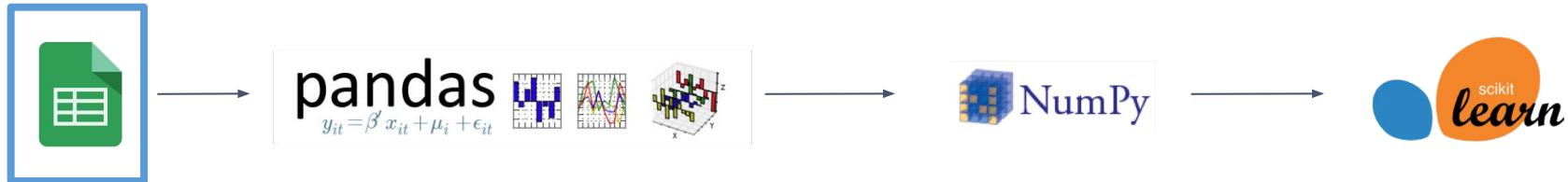
Apprentissage machine



seaborn

Visualisation des données

# Pipeline



Explorer les données

Structurer les données en Python

Traiter les données

Apprentissage machine



seaborn

Visualisation des données

# Explorer les données



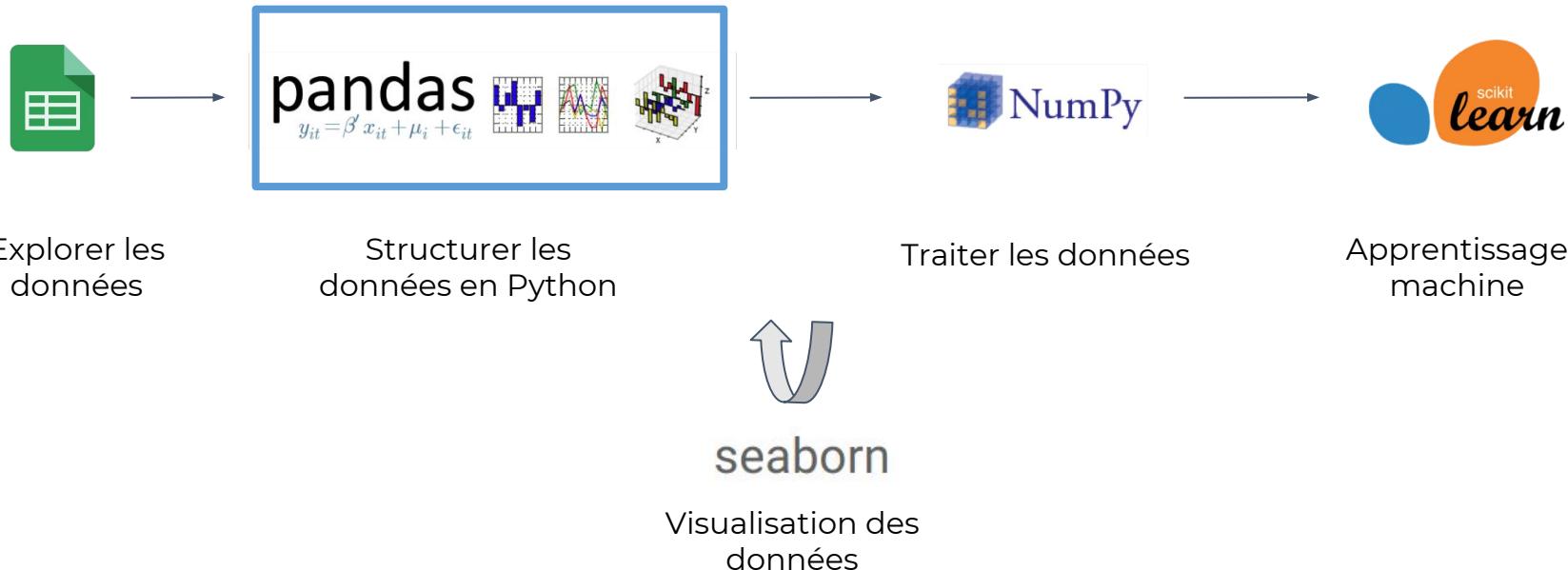
data.csv (122.27 KB)

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	M	17.99	10.38	122.8
2	M	20.57	17.77	132.9
3	M	19.69	21.25	130
4	M	11.42	20.38	77.58

breast-cancer....zip ^

# Pipeline



# Pandas

```
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	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1000.50	0.1189	0.2781	0.0971	0.0019	0.1812	0.0063	18.49	2.23	3.14	78.69	0.0596	0.0483	0.0157	0.0036	0.0002	0.0002	20.57	2.84	3.08	80.57	0.0537	0.0435	0.0146	0.0033	0.0001	
1	842517	M	20.57	17.77	132.90	1849.00	0.1428	0.3008	0.1259	0.0053	0.2059	0.0084	21.52	2.91	3.31	85.69	0.0619	0.0507	0.0172	0.0040	0.0002	0.0002	21.62	3.04	3.39	84.74	0.0587	0.0475	0.0183	0.0044	0.0001	
2	84300903	M	19.69	21.25	130.00	1789.00	0.1304	0.2914	0.1205	0.0056	0.1974	0.0082	21.48	2.98	3.28	84.32	0.0606	0.0493	0.0169	0.0039	0.0002	0.0002	21.48	3.02	3.32	84.32	0.0574	0.0461	0.0176	0.0042	0.0001	
3	84348301	M	11.42	20.38	77.58	186.00	0.0797	0.2057	0.0849	0.0012	0.1622	0.0054	16.64	2.33	2.81	54.58	0.0355	0.0286	0.0098	0.0023	0.0001	0.0001	11.42	2.33	2.81	54.58	0.0355	0.0286	0.0098	0.0023	0.0001	
4	84358402	M	20.29	14.34	135.10	1991.00	0.1205	0.2838	0.1186	0.0058	0.1997	0.0085	21.42	2.92	3.30	85.69	0.0619	0.0507	0.0172	0.0040	0.0002	0.0002	21.42	3.05	3.39	84.74	0.0587	0.0475	0.0183	0.0044	0.0001	

5 rows × 33 columns

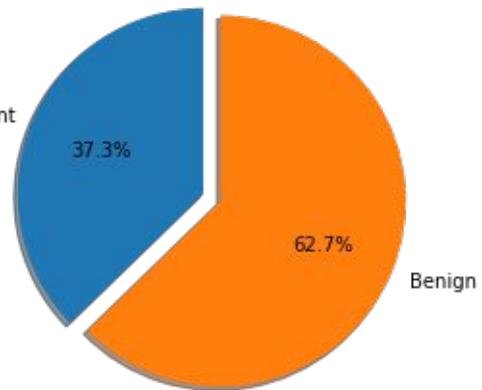
Number of total entries: 569

Entries per category:

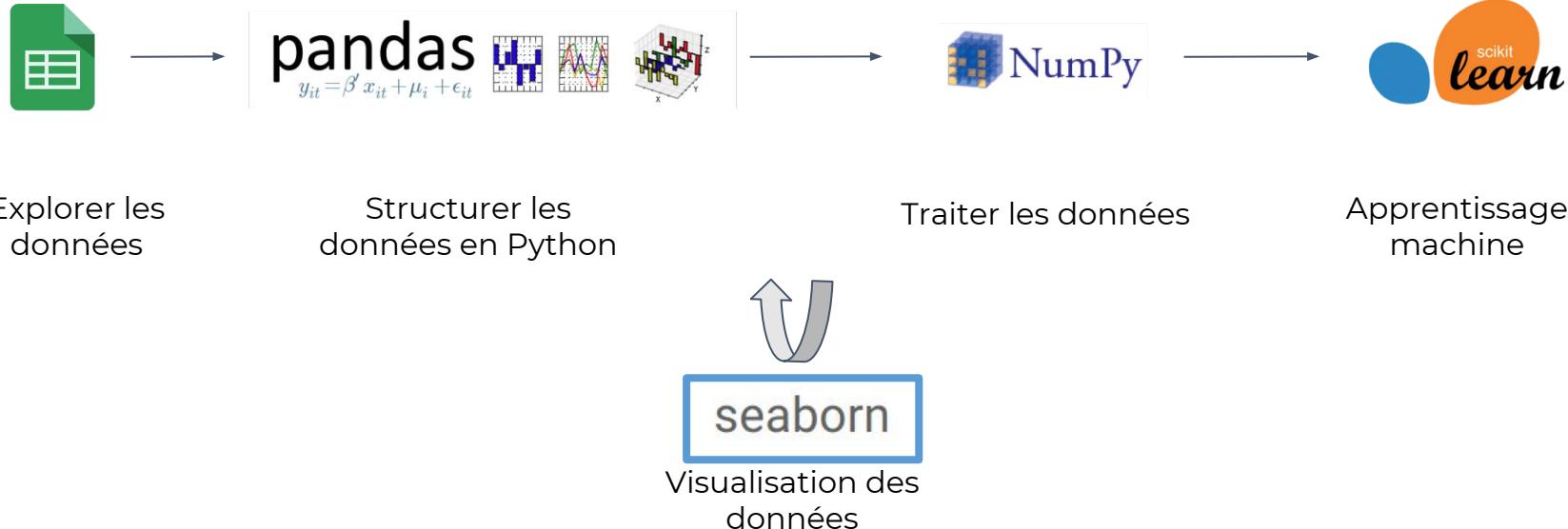
B 357

M 212

Malignant



# Pipeline



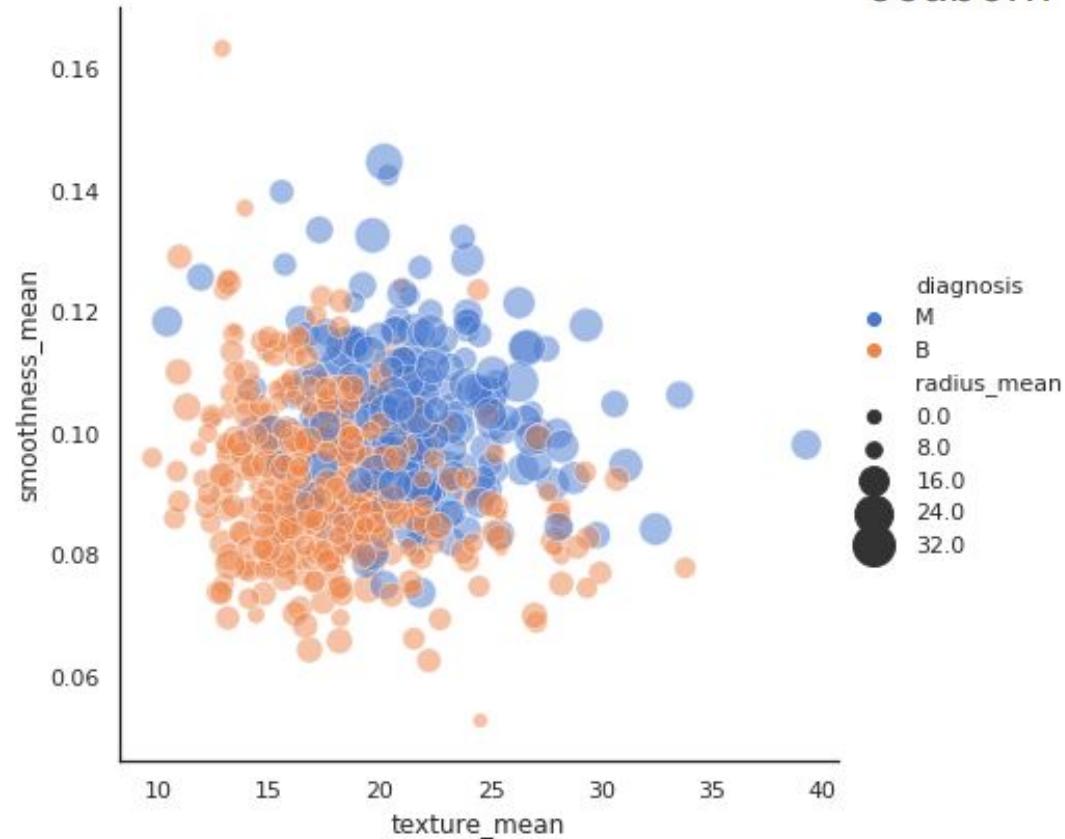
# Seaborn

seaborn

```
import seaborn as sns
import pandas as pd

dataset = pd.read_csv('data.csv')
sns.set(style="white")

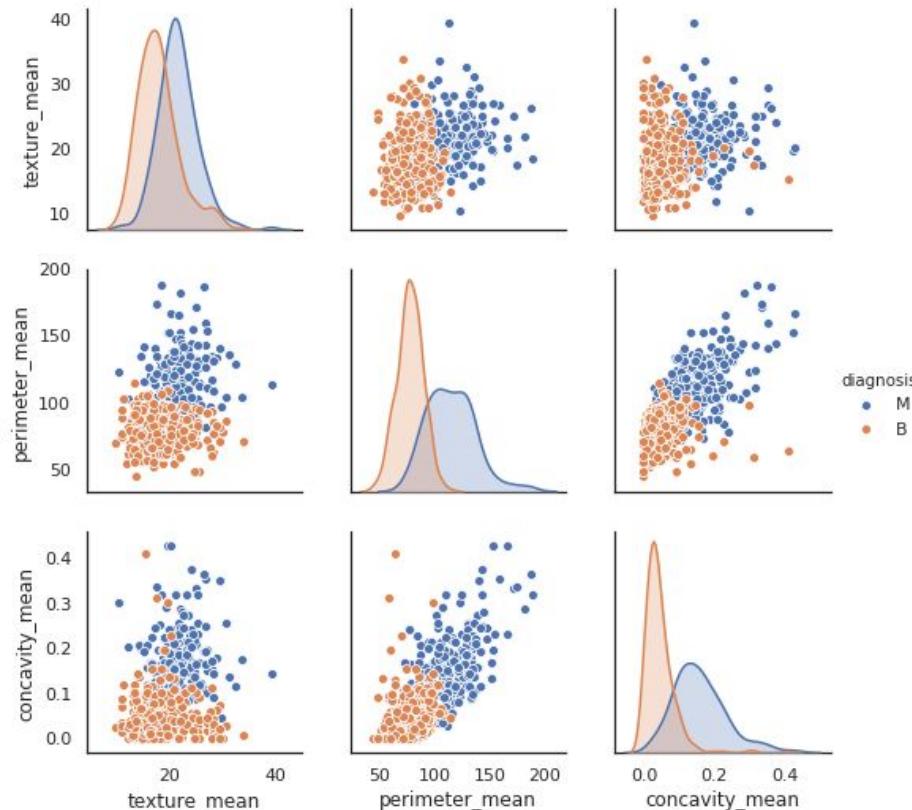
# Plot texture against smoothness
# with radius and class
sns.relplot(x="texture_mean",
             y="smoothness_mean",
             hue="diagnosis",
             size="radius_mean",
             sizes=(40, 400),
             alpha=.5, palette="muted",
             height=6, data=dataset)
```



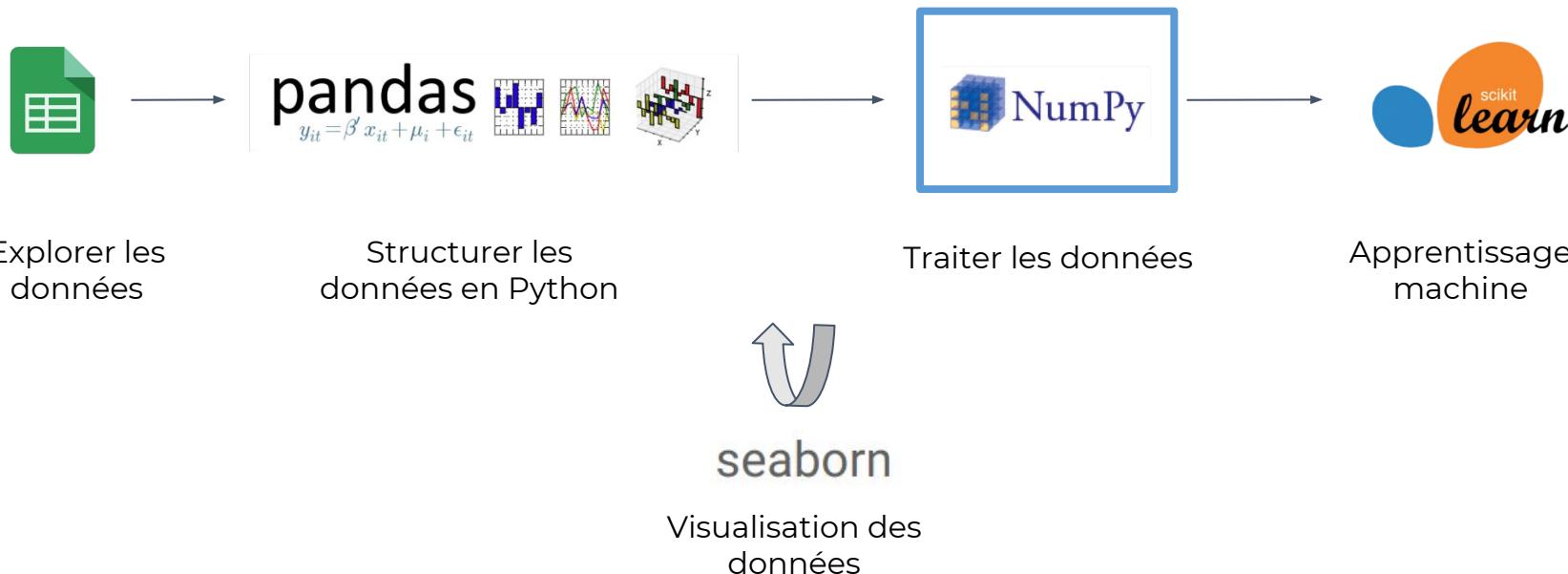
# Seaborn

seaborn

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



# Pipeline



# 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.

```
import numpy as np

n_columns = len(dataset.columns)

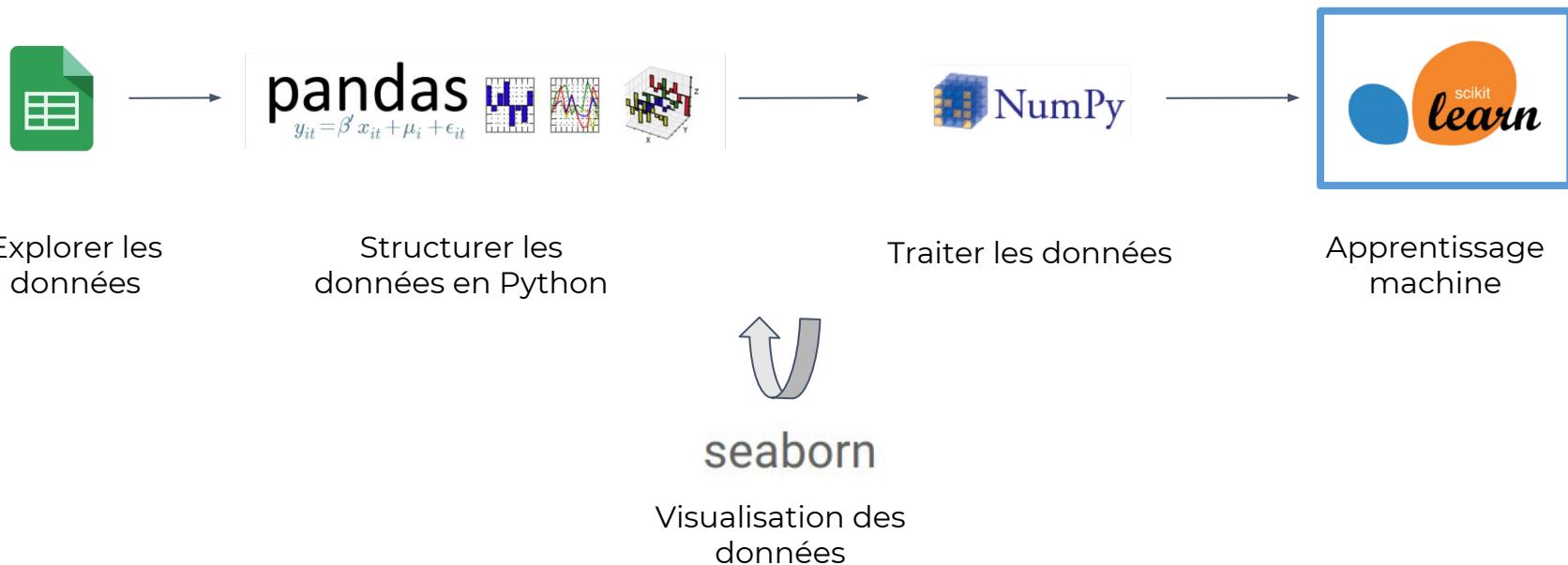
# Convert data to ndarray
# Be careful not to include your labels in your data!
X = np.asarray(dataset.iloc[:, 2:n_columns-1])

# 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]]
```

$$y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

# Pipeline



# Scikit-Learn



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

```
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

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

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

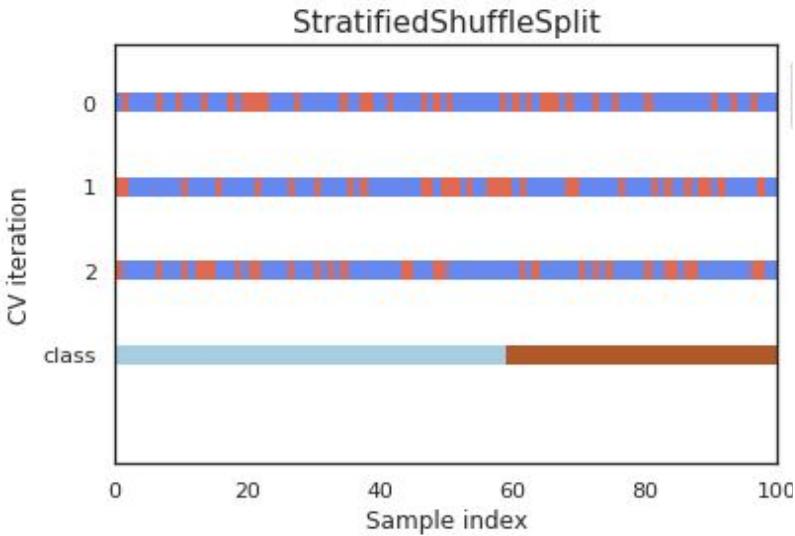
logreg = LogisticRegression(solver='liblinear')
logreg.fit(X,y)

y_pred = logreg.predict(X)

print(accuracy_score(y, y_pred))
```

Accuracy : 96%

# Validation croisée



```
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)
```

# Traitements 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

### 1.1. Generalized Linear Models

- 1.1.1. Ordinary Least Squares
  - 1.1.1.1. Ordinary Least Squares Complexity
- 1.1.2. Ridge Regression
  - 1.1.2.1. Ridge Complexity
  - 1.1.2.2. Setting the regularization parameter: generalized Cross-Validation
- 1.1.3. Lasso
  - 1.1.3.1. Setting regularization parameter
    - 1.1.3.1.1. Using cross-validation
    - 1.1.3.1.2. Information-criteria based model selection
    - 1.1.3.1.3. Comparison with the regularization parameter of SVM
- 1.1.4. Multi-task Lasso
- 1.1.5. Elastic Net
- 1.1.6. Multi-task Elastic Net
- 1.1.7. Least Angle Regression
- 1.1.8. LARS Lasso
  - 1.1.8.1. Mathematical formulation
- 1.1.9. Orthogonal Matching Pursuit (OMP)
- 1.1.10. Bayesian Regression
  - 1.1.10.1. Bayesian Ridge Regression
  - 1.1.10.2. Automatic Relevance Determination - ARD
- 1.1.11. Logistic regression
- 1.1.12. Stochastic Gradient Descent - SGD
- 1.1.13. Perceptron
- 1.1.14. Passive Aggressive Algorithms
- 1.1.15. Robust regression: outliers and modeling errors
  - 1.1.15.1. Different scenarios and useful concepts
  - 1.1.15.2. RANSAC: Random Sample Consensus
    - 1.1.15.2.1. Details of the algorithm
  - 1.1.15.3. Theil-Sen estimator: generalized-median-based estimator
    - 1.1.15.3.1. Theoretical considerations
  - 1.1.15.4. Huber Regression
  - 1.1.15.5. Notes
- 1.1.16. Polynomial regression: extending linear models with basis functions

### 1.2. Linear and Quadratic Discriminant Analysis

- 1.2.1. Dimensionality reduction using Linear Discriminant Analysis
- 1.2.2. Mathematical formulation of the LDA and QDA classifiers
- 1.2.3. Mathematical formulation of LDA dimensionality reduction
- 1.2.4. Shrinkage
- 1.2.5. Estimation algorithms

### 1.3. Kernel ridge regression

### 1.4. Support Vector Machines

- 1.4.1. Classification
  - 1.4.1.1. Multi-class classification
  - 1.4.1.2. Scores and probabilities
  - 1.4.1.3. Unbalanced problems
- 1.4.2. Regression
- 1.4.3. Density estimation, novelty detection
- 1.4.4. Complexity
- 1.4.5. Tips on Practical Use
- 1.4.6. Kernel functions
  - 1.4.6.1. Custom Kernels
  - 1.4.6.2. Using Python functions as kernels
  - 1.4.6.3. Using the Gram matrix
  - 1.4.6.13. Parameters of the RBF Kernel
- 1.4.7. Mathematical formulation
  - 1.4.7.1. SVC
  - 1.4.7.2. NuSVC
  - 1.4.7.3. SVR
- 1.4.8. Implementation details

### 1.5. Stochastic Gradient Descent

- 1.5.1. Classification
- 1.5.2. Regression
- 1.5.3. Stochastic Gradient Descent for sparse data
- 1.5.4. Complexity
- 1.5.5. Stopping criterion
- 1.5.6. Tips on Practical Use
- 1.5.7. Mathematical formulation
  - 1.5.7.1. SGD

### 1.6. Nearest Neighbors

- 1.6.1. Unsupervised Nearest Neighbors
  - 1.6.1.1. Finding the Nearest Neighbors
  - 1.6.1.2. KDTree and BallTree Classes
- 1.6.2. Nearest Neighbors Classification
- 1.6.3. Nearest Neighbors Regression
- 1.6.4. Nearest Neighbor Algorithms
  - 1.6.4.1. Brute Force
  - 1.6.4.2. K-D Tree
  - 1.6.4.3. Ball Tree
  - 1.6.4.4. Choice of Nearest Neighbors Algorithm
  - 1.6.4.5. Effect of `leaf_size`
- 1.6.5. Nearest Centroid Classifier
  - 1.6.5.1. Nearest Shrunken Centroid

### 1.7. Gaussian Processes

- 1.7.1. Gaussian Process Regression (GPR)
- 1.7.2. GPR examples
  - 1.7.2.1. GPR with noise-level estimation
  - 1.7.2.2. Comparison of GPR and Kernel Ridge Regressor
  - 1.7.2.3. GPR on Mauna Loa CO<sub>2</sub> data
- 1.7.3. Gaussian Process Classification (GPC)
- 1.7.4. GPC examples
  - 1.7.4.1. Probabilistic predictions with GPC
  - 1.7.4.2. Illustration of GPC on the XOR dataset
  - 1.7.4.3. Gaussian process classification (GPC) on iris d
- 1.7.5. Kernels for Gaussian Processes
  - 1.7.5.1. Gaussian Process Kernel API
  - 1.7.5.2. Basic kernels
  - 1.7.5.3. Kernel operators
  - 1.7.5.4. Matern kernel
  - 1.7.5.5. Rational quadratic kernel
  - 1.7.5.6. Exp-Sine-Squared kernel
  - 1.7.5.7. Dot-Product kernel
  - 1.7.5.8. References

...  
...

### 1.8. Cross decomposition

### 1.9. Naive Bayes

- 1.9.1. Gaussian Naive Bayes
- 1.9.2. Multinomial Naive Bayes
- 1.9.3. Complement Naive Bayes
- 1.9.4. Bernoulli Naive Bayes
- 1.9.5. Out-of-core naive Bayes model fitting

### 1.10. Decision Trees

- 1.10.1. Classification
- 1.10.2. Regression
- 1.10.3. Multi-output problems
- 1.10.4. Complexity
- 1.10.5. Tips on practical use
- 1.10.6. Tree algorithms: ID3, C4.5, C.5.0 and CART
- 1.10.7. Mathematical formulation
  - 1.10.7.1. Classification criteria
  - 1.10.7.2. Regression criteria

### 1.11. Ensemble methods

- 1.11.1. Bagging meta-estimator
  - 1.11.2. Forests of randomized trees
    - 1.11.2.1. Random Forests
    - 1.11.2.2. Extremely Randomized Trees
    - 1.11.2.3. Parameters
    - 1.11.2.4. Parallelization
    - 1.11.2.5. Feature importance evaluation
    - 1.11.2.6. Totally Random Trees Embedding
  - 1.11.3. AdaBoost
- 1.11.3.1. Usage
- 1.11.4. Gradient Tree Boosting
  - 1.11.4.1. Classification
  - 1.11.4.2. Regression
  - 1.11.4.3. Fitting additional weak-learners
  - 1.11.4.4. Controlling the tree size
  - 1.11.4.5. Mathematical formulation
    - 1.11.4.5.1. Loss Functions

◦ 1.11.4.6. Regularization

### 1.12. Multiclass and multilabel algorithms

- 1.12.1. Multilabel classification format
- 1.12.2. One-Vs-The-Rest
  - 1.12.2.1. Multiclass learning
  - 1.12.2.2. Multilabel learning
- 1.12.3. One-Vs-One
  - 1.12.3.1. Multiclass learning
- 1.12.4. Error-Correcting Output-Codes
  - 1.12.4.1. Multiclass learning
- 1.12.5. Multitarget regression
- 1.12.6. Multitarget classification
- 1.12.7. Classifier Chain
- 1.12.8. Regressor Chain

### 1.13. Feature selection

- 1.13.1. Removing features with low variance
- 1.13.2. Univariate feature selection
- 1.13.3. Recursive feature elimination
- 1.13.4. Feature selection using SelectFromModel
  - 1.13.4.1. L1-based feature selection
  - 1.13.4.2. Tree-based feature selection
- 1.13.5. Feature selection as part of a pipeline

### 1.14. Semi-Supervised

- 1.14.1. Label Propagation

### 1.15. Isotonic regression

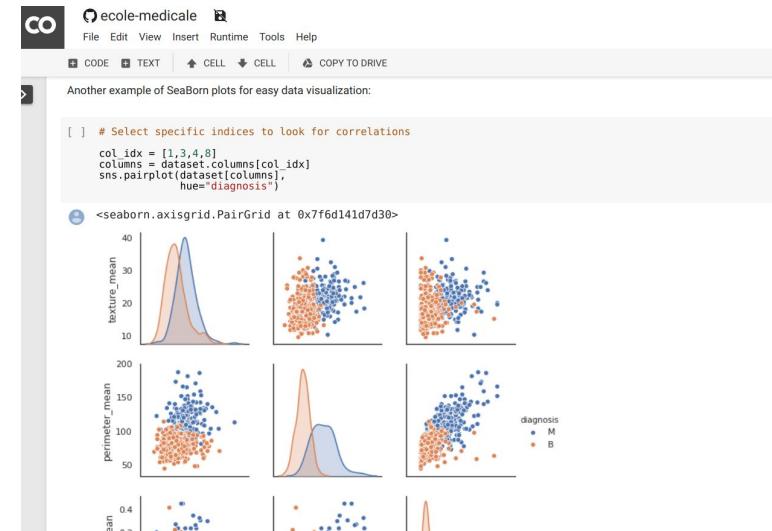
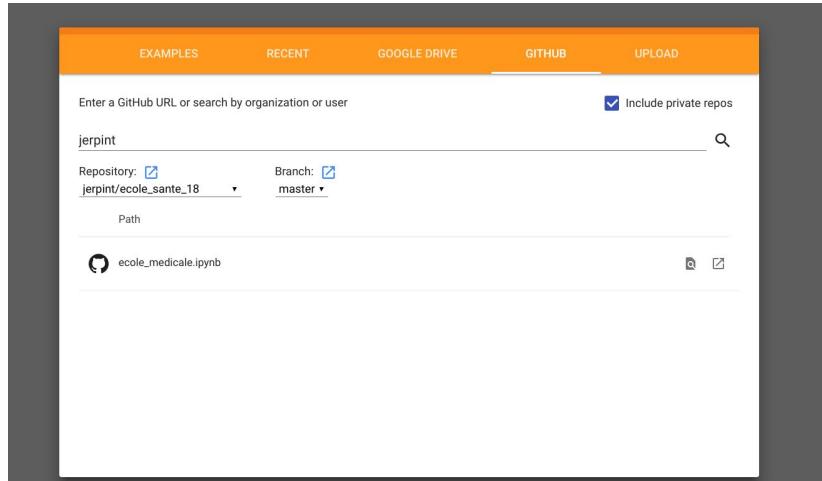
### 1.16. Probability calibration

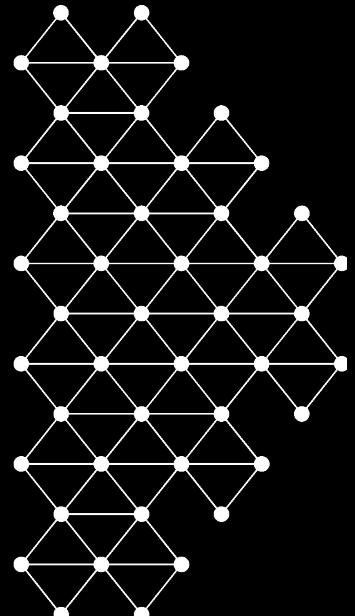
### 1.17. Neural network models (supervised)

- 1.17.1. Multi-layer Perceptron
- 1.17.2. Classification
- 1.17.3. Regression
- 1.17.4. Regularization
- 1.17.5. Algorithms
- 1.17.6. Complexity
- 1.17.7. Mathematical formulation
- 1.17.8. Tips on Practical Use
- 1.17.9. More control with `warm_start`

# Exemple en ligne

Pour recréer toutes les expériences et figures, rendez-vous au  
[https://github.com/jerpint/ecolet\\_sante\\_18](https://github.com/jerpint/ecolet_sante_18)





Apprentissage profond avec  
python

# Librairies d'apprentissage profond



**2017/11/15: Release of Theano 1.0.0**

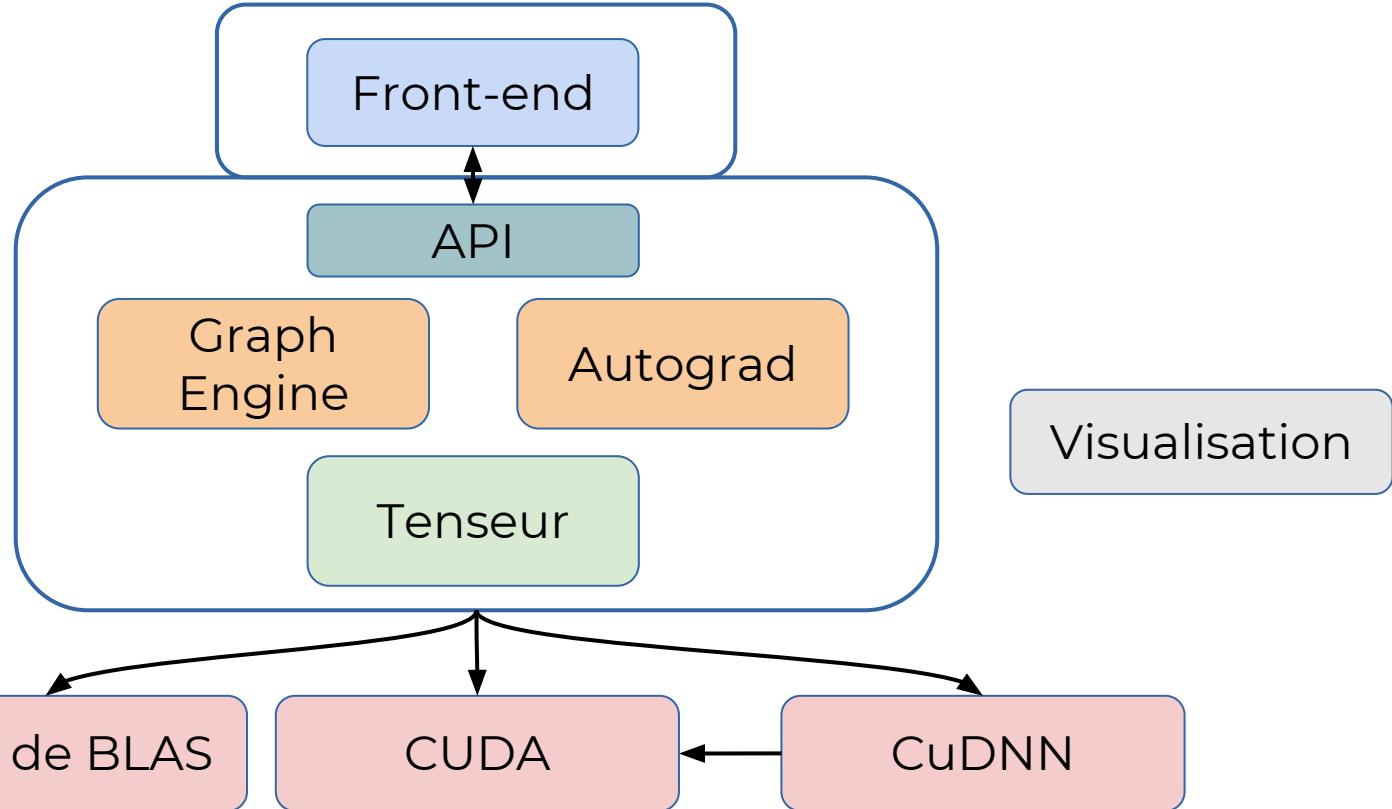
Arrêt du développement logiciel par le Mila  
Précurseur à beaucoup d'idées qui se retrouvent  
dans les librairies plus récentes.

# Librairies

Apprentissage  
profond

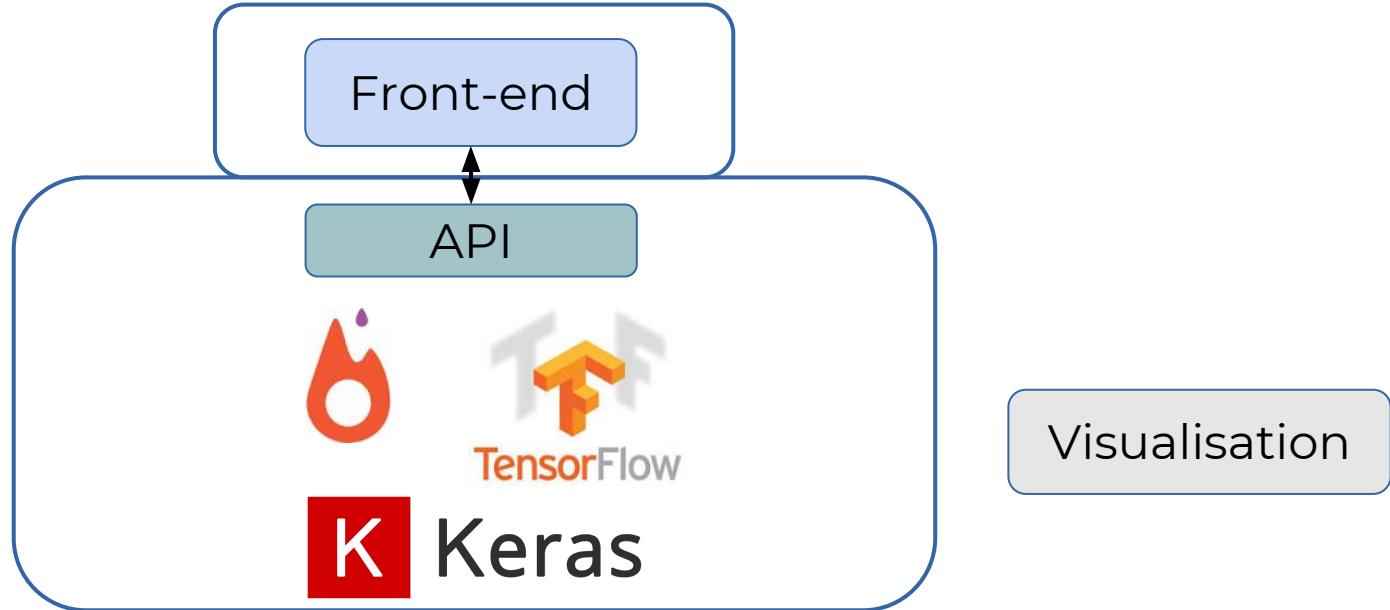
Programmation  
différentielle

Calcul  
matriciel



# Librairies

Apprentissage  
profond



# 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

source : <https://pytorch.org/docs/stable/torch.html>

`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())
]))
```

# 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



```
# 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

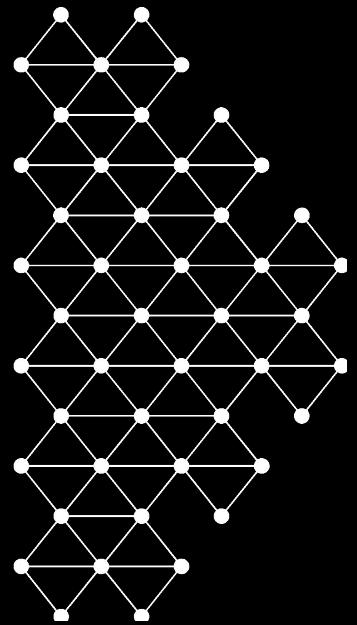


Open Source	Oui (BSD) 	Oui (Apache 2.0) 
Support GPU + CUDA + CUDNN	Oui	Oui
Visualization	TensorboardX (nouveau), Visdom	Tensorboard
“Pythonic”	Oui, “First-class Python integration”, pdb fonctionne sur le graph directement	Non, pdb en plus de tfdbg
Production	Oui (Torch.jit, nouveau depuis v1.0)	Oui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

# PyTorch vs. TensorFlow



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

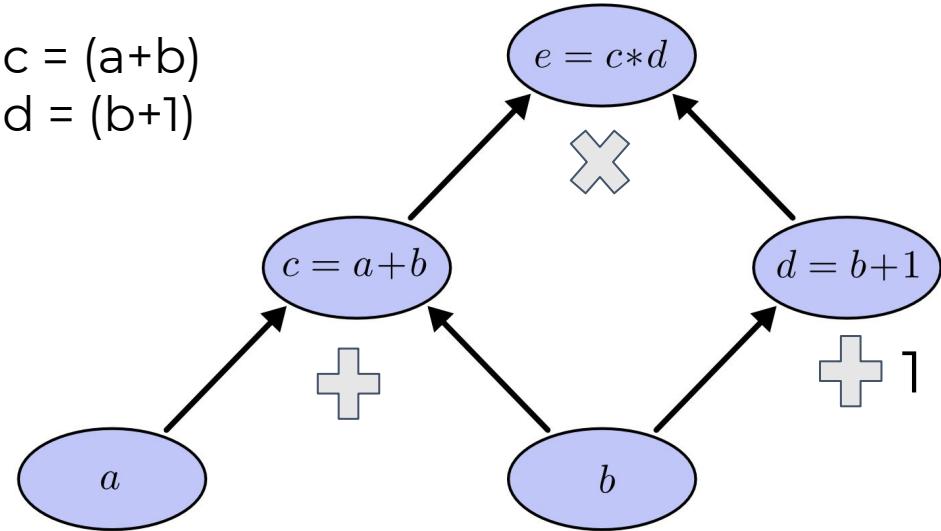
# Graphe Computationnel

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

Exemple:

$$e = (a+b)(b+1) = c \circ d$$

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

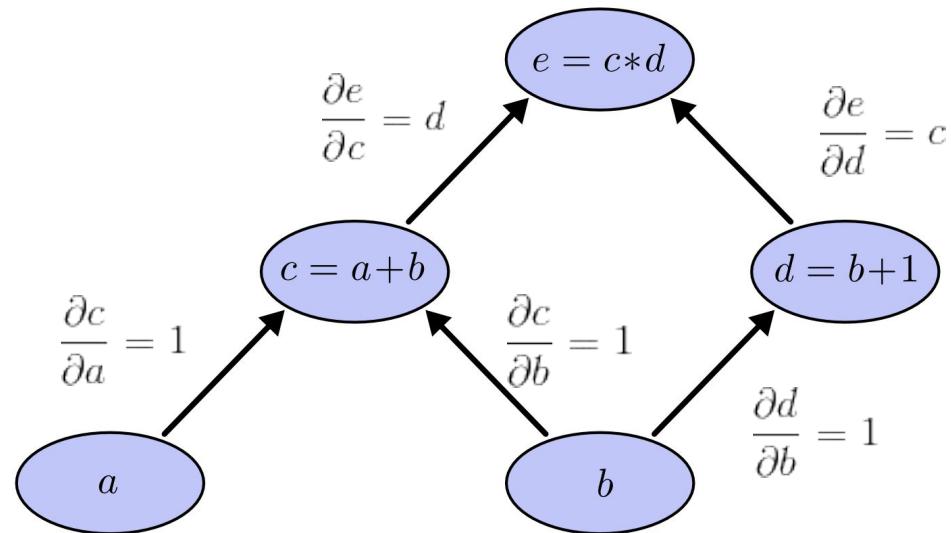


# Graphe Computationnel

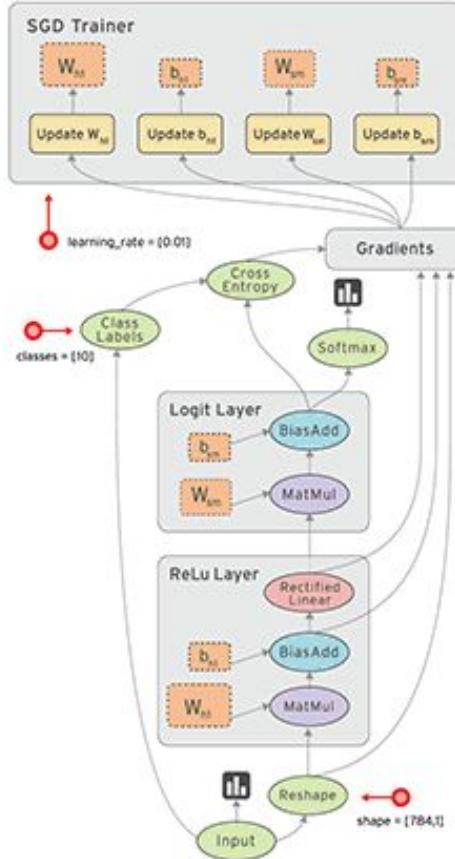
Le **graphe computationnel** permet de représenter des opérations mathématiques complexes et de **calculer des dérivées** facilement

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

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



# Exemple



source : <https://www.tensorflow.org/guide/graphs>

# Graphe Computationnel



Graphe Computationnel  
Dynamique

VS



Graphe Computationnel  
Statique

# Graphe Computationnel

```
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)
```



```
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)
```

VS



# PyTorch ou TensorFlow

Andrej Karpathy ✅  
@karpathy Following

I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved.

11:56 AM - 26 May 2017

384 Retweets 1,519 Likes

33 384 1.5K

Andrej Karpathy ✅  
@karpathy Following

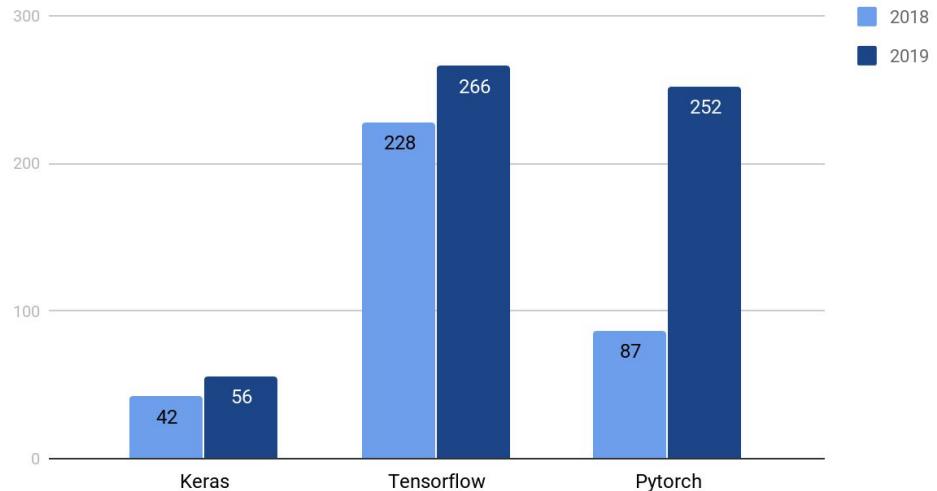
Matlab is so 2012. Caffe is so 2013. Theano is so 2014. Torch is so 2015. TensorFlow is so 2016. :D

RETWEETS 218 LIKES 590

12:08 PM - 8 Feb 2017

45 218 590

Citations de frameworks ICLR

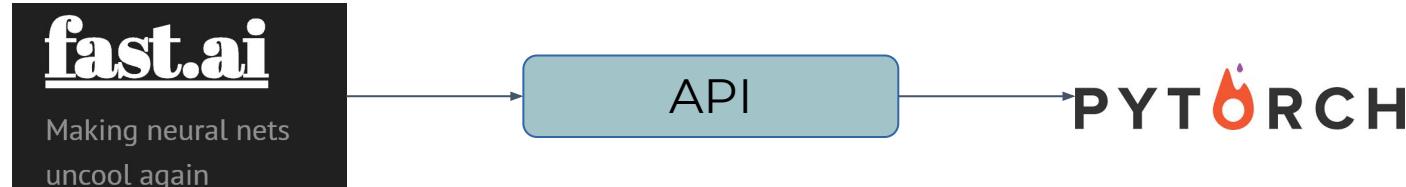


source:  
[https://www.reddit.com/r/MachineLearning/comments/9kys38/r\\_frameworks\\_mentioned\\_iclr\\_20182019\\_tensorflow/](https://www.reddit.com/r/MachineLearning/comments/9kys38/r_frameworks_mentioned_iclr_20182019_tensorflow/)

# 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

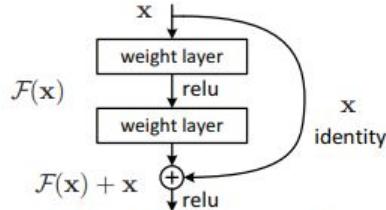


Figure 2. Residual learning: a building block.

**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)

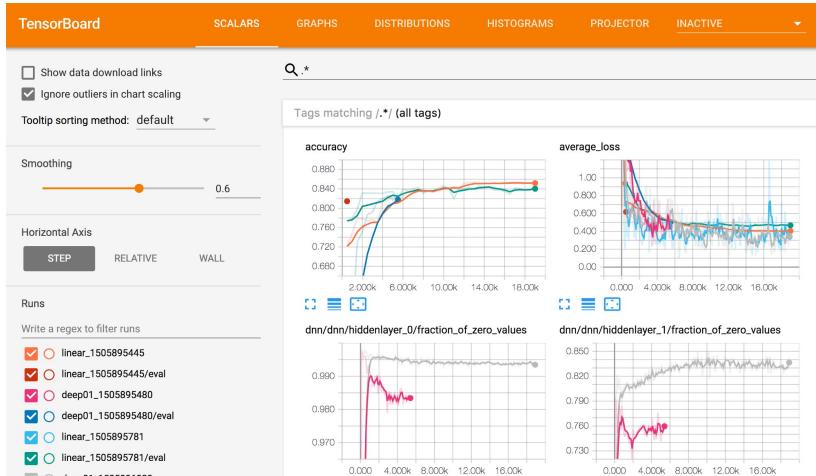
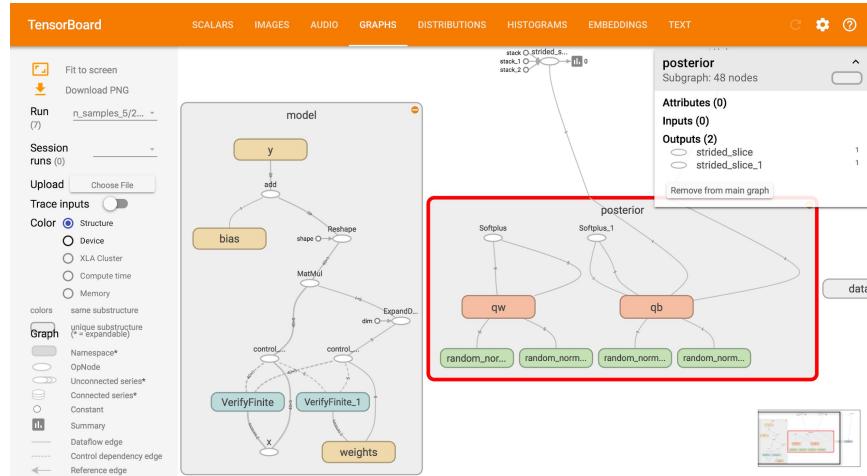
Total time: 00:05
epoch    train loss    valid loss    accuracy
0        0.081393     0.046429     0.985770 (00:05)
```

# Une hiérarchie d'outils

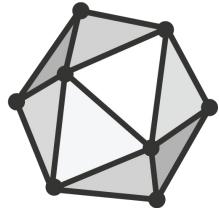


## Visualisation

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



# Interopérabilité de modèles



# ONNX

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



# Installation + Utilisation

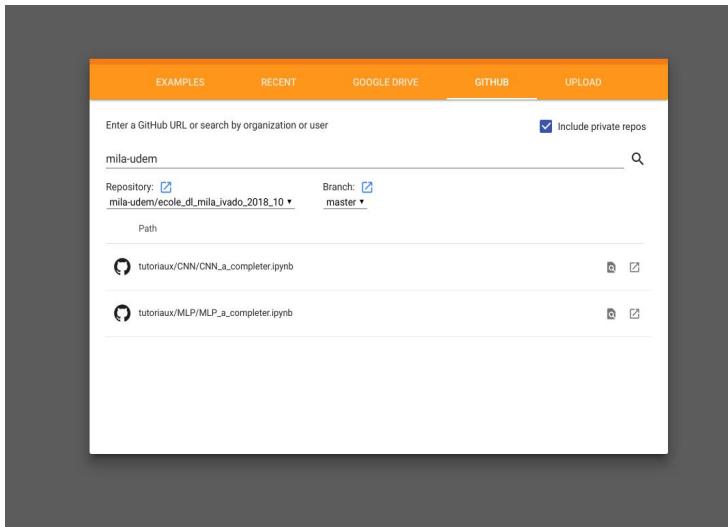
Grâce à Google Colab, PyTorch est facilement installé avec accès à un GPU dans le cloud.



# Exemples

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

[https://github.com/mila-udem/ecoile\\_dl\\_mila\\_ivado\\_2018\\_10](https://github.com/mila-udem/ecoile_dl_mila_ivado_2018_10)



The screenshot shows a Jupyter notebook titled "MLP-AFF-Hiver18Tutorial-Complet.ipynb" running in Google Colab. The notebook is divided into sections:

- INTRODUCTION**: CPU ou GPU, PyTorch en bref, éléments nécessaires pour un projet.
- DÉFINITION DE LA MACHE**: Prédiction de la score soit à un message.
- Le dataset MNIST**: Prétraitement des données, Décomposition en Train / Validation / Test.
- Définition du modèle**: Perceptron Multicouche (MLP).
- Implémentation du modèle en PyTorch**: 1. Boîte à outils, 2. Implémentation.
- Évaluation d'un réseau de neurones**: 1. Boîte à outils, 2. Implementation, 3. Questions.
- Définir la fonction de coût et l'optimiseur**: Fonction de coût.
- Méthode du gradient**: Méthode du gradient.
- ENTRAÎNEMENT**: Epoch, Iteration, Mini-Batch, 1. Définition, 2. MiniBatch en PyTorch, 3. Implementation.
- Basics principale**: 1. CPU ou GPU, 2. Implementation.

Below the notebook content, there are two sections:

- CPU ou GPU**: A note about using CUDA, mentioning that PyTorch supports CUDA but requires a NVIDIA GPU and PyTorch has to be installed with CUDA support. It also mentions that PyTorch can run on CPU if no CUDA is available.
- PyTorch en bref**: A brief introduction to PyTorch, stating it's a library for Python that provides high-level features like operations on tensors (similar to NumPy) with GPU support, and a dynamic graph-based computational graph.

# GPU ou CPU?



**NVIDIA TITAN V**

THE MOST POWERFUL PC GPU EVER CREATED



# Services Cloud avec GPU/TPU



# 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



A screenshot of a Google search results page. The search query is "iclr 2018 github". The results are filtered under the "All" tab. The first result is a link to a GitHub repository titled "GitHub - Chillee/OpenReviewExplorer: Explore OpenReview papers ...". The second result is "GitHub - shahsohil/stableGAN: The code for the ICLR 2018 paper ...". The third result is "GitHub - whyjay/memoryGAN: Repository for our ICLR 2018 paper ...". Each result includes a link to the GitHub repository and a brief description of the project.

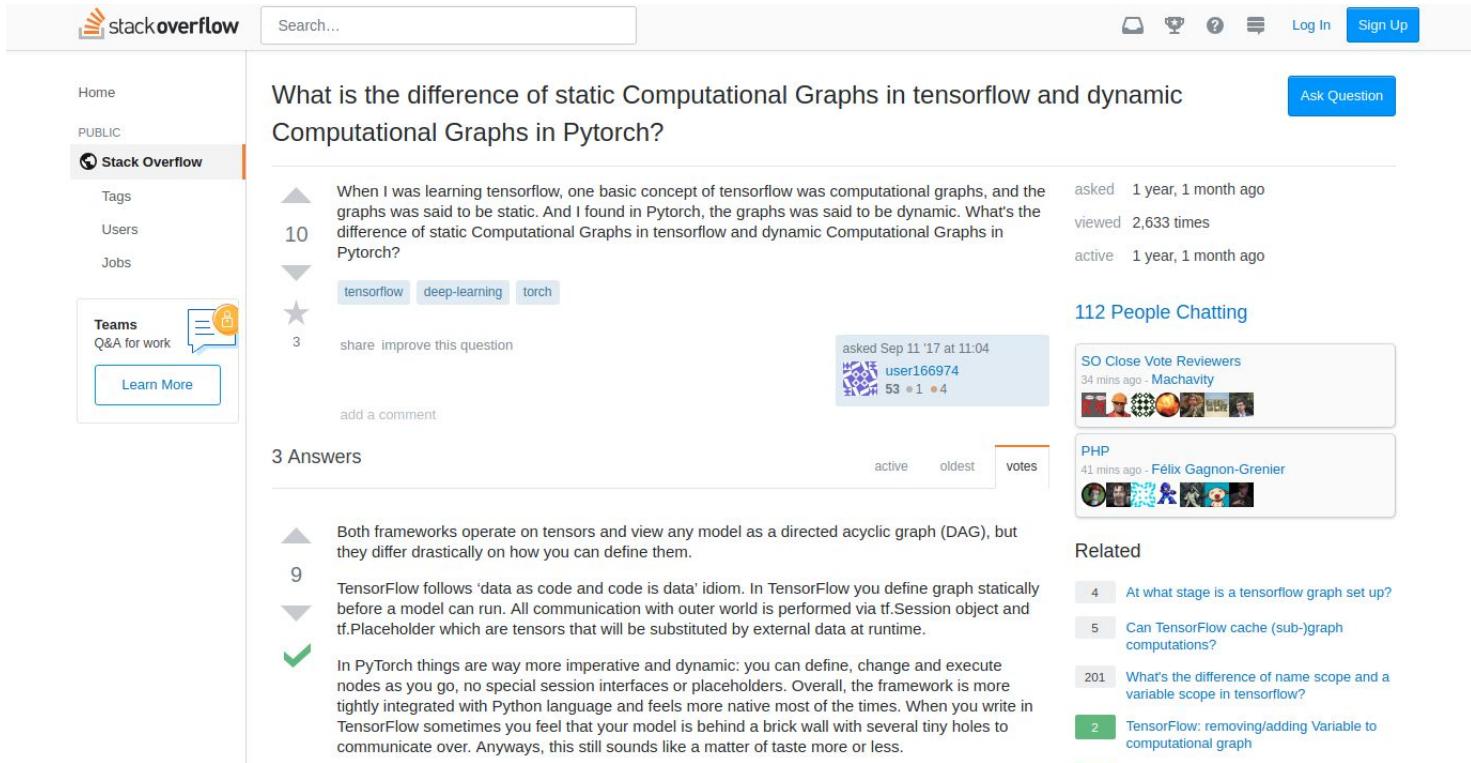
Exemple de Repos utiles:

<https://github.com/google/seq2seq>

<https://github.com/facebookresearch/Detectron>

<https://github.com/openai/gym>

# Outils pratiques - Stack Overflow



The image shows a screenshot of the Stack Overflow website. At the top, there is a navigation bar with the Stack Overflow logo, a search bar containing "Search...", and links for "Log In" and "Sign Up". On the left, there is a sidebar with links for "Home", "PUBLIC", "Stack Overflow" (which is selected), "Tags", "Users", and "Teams". The main content area displays a question titled "What is the difference of static Computational Graphs in tensorflow and dynamic Computational Graphs in Pytorch?". The question has 10 upvotes and is tagged with "tensorflow", "deep-learning", and "torch". It was asked by user166974 on Sep 11 '17 at 11:04. Below the question, there are three answers. The first answer has 9 upvotes and discusses the differences between TensorFlow's static graph and PyTorch's dynamic graph. The second answer has 201 upvotes and provides a detailed comparison. The third answer has 2 upvotes and asks about removing/adding variables to a computational graph. To the right of the main content, there is a sidebar with a "Ask Question" button, a counter for "112 People Chatting", and sections for "SO Close Vote Reviewers" and "PHP". There is also a "Related" section with several more questions.

What is the difference of static Computational Graphs in tensorflow and dynamic Computational Graphs in Pytorch?

When I was learning tensorflow, one basic concept of tensorflow was computational graphs, and the graphs was said to be static. And I found in Pytorch, the graphs was said to be dynamic. What's the difference of static Computational Graphs in tensorflow and dynamic Computational Graphs in Pytorch?

10

tensorflow deep-learning torch

share improve this question

asked Sep 11 '17 at 11:04  
user166974 53 ● 1 ● 4

3 Answers

Both frameworks operate on tensors and view any model as a directed acyclic graph (DAG), but they differ drastically on how you can define them.

TensorFlow follows 'data as code and code is data' idiom. In TensorFlow you define graph statically before a model can run. All communication with outer world is performed via `tf.Session` object and `tf.Placeholder` which are tensors that will be substituted by external data at runtime.

In PyTorch things are way more imperative and dynamic: you can define, change and execute nodes as you go, no special session interfaces or placeholders. Overall, the framework is more tightly integrated with Python language and feels more native most of the times. When you write in TensorFlow sometimes you feel that your model is behind a brick wall with several tiny holes to communicate over. Anyways, this still sounds like a matter of taste more or less.

112 People Chatting

SO Close Vote Reviewers

PHP

Related

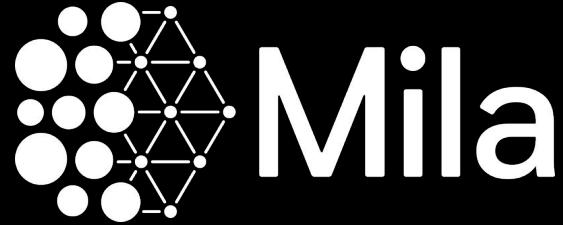
At what stage is a tensorflow graph set up?

Can TensorFlow cache (sub-)graph computations?

What's the difference of name scope and a variable scope in tensorflow?

TensorFlow: removing/adding Variable to computational graph

Institut  
québécois  
d'intelligence  
artificielle



Merci!  
Questions?

Jeremy Pinto  
[jeremy.pinto@mila.quebec](mailto:jeremy.pinto@mila.quebec)