# Machine Learning : Introduction

D. Cornu

M2-P2N Automne - 2019

# Supports pour le Cours

Tous les supports de cours (slides, codes, exercices, corrections, ...) sont disponibles sur GitHub et seront mis à jour progressivement :

Deyht/ML_M2

Quelques commandes utiles :

```
sudo apt-get install git

git clone https://github.com/Deyht/ML_M2

git pull
```

Pour ne pas risquer de perdre votre travail en cas de mise à jour copiez ces fichiers dans un autre répertoire !

# Références utiles pour réviser ou aller plus loin

Stephen Marsland. **Machine Learning an Algorithmic Perspective**. Chapman and Hall/Crc, 2015. Pédagogique, assez complet et exemples de codes en Python

G. James, D. Witten, T. Hastie et R. Tibshirani .**An Introduction to Statistical Learning with applications in R (ISLR)**. Springer, 2013. Approche plus statistique du machine learning, exemples en R

Christopher M. Bishop. **Pattern Recognition and Machine Learning**. Springer, 2011. Très mathématique, contient toute la théorie necessaire

# Organisation du cours

**Cours 1 : Introduction des concepts du Machine Learning et introduction d'algorithmes classiques**

**Cours 2 à 4 : Introduction sur les réseaux de neurones artificiels simples (1 couche) et implémentation guidée, puis réseaux multi-chouches et implémentation matricielle ...**

**Cours 5 : Programmation GPU - CUDA (kernels et cuBLAS)**

# Introduction

**Problématiques :**

- Traiter un nombre toujours plus grand de données
- Extraire de l'information difficilement visualisable ($>$ 2 dimensions)
- Reproduire l'intelligence de manière artificielle

# Introduction

**Problématiques :**

- Traiter un nombre toujours plus grand de données
- Extraire de l'information difficilement visualisable ($> 2$ dimensions)
- Reproduire l'intelligence de manière artificielle

**Solution possible : Machine learning**
**Objectif :**
**$\rightarrow$ apprendre à un ordinateur à extraire une information statistique et à s'adapter à celle-ci via un processus d'apprentissage.**

**Question : Comment définir "l'apprentissage" ?**

# Définition de l'apprentissage

« Acquérir la connaissance d'une chose par l'exercice de l'intelligence, de la mémoire, des mécanismes gestuels appropriés, etc. »

* *Trésor de la Langue Française informatisé TLFi*

# Définition de l'apprentissage

« Acquérir la connaissance d'une chose par l'exercice de l'intelligence, de la mémoire, des mécanismes gestuels appropriés, etc. »

** Trésor de la Langue Française informatisé TLFi*

D'une manière générale, on peut représenter l'apprentissage animal avec trois capacités :

# Définition de l'apprentissage

« Acquérir la connaissance d'une chose par l'exercice de l'intelligence, de la mémoire, des mécanismes gestuels appropriés, etc. »

** Trésor de la Langue Française informatisé TLFi*

D'une manière générale, on peut représenter l'apprentissage animal avec trois capacités :

- **Se souvenir** - **S'adapter** - **Généraliser**

# Définition de l'apprentissage

« Acquérir la connaissance d'une chose par l'exercice de l'intelligence, de la mémoire, des mécanismes gestuels appropriés, etc. »

** Trésor de la Langue Française informatisé TLFi*

D'une manière générale, on peut représenter l'apprentissage animal avec trois capacités :

- **Se souvenir** - **S'adapter** - **Généraliser**

Savoir **reconnaître une situation** dans laquelle on a déjà été (avoir vu ces données) et se **souvenir** de notre **réaction** (avoir donné cette sortie) et déterminer si celle-ci était **appropriée** (solution correcte).

# Intelligence

Dernière étape du processus : la **généralisation** $\rightarrow$ trouver des similitudes entre les situations et supposer que des solutions adaptées dans certains cas peuvent être appliquées à d'autres.

Pour parler **d'intelligence** il manque encore :

- **Le raisonnement**
(Capacité à poser un problème)

- **La déduction logique**
(Capacité à ordonner des propositions qui s'impliquent)

# Intelligence

Dernière étape du processus : la **généralisation** $\rightarrow$ trouver des similitudes entre les situations et supposer que des solutions adaptées dans certains cas peuvent être appliquées à d'autres.

Pour parler **d'intelligence** il manque encore :

- **Le raisonnement**
(Capacité à poser un problème)

- **La déduction logique**
(Capacité à ordonner des propositions qui s'impliquent)

Nécessite ce qu'on appelle le symbolic processing $\rightarrow$ représentation par des symboles qui varient dans le temps ou selon le contexte.

**Le machine learning "classique" reste une représentation Subsymbolic.**

# Types d'apprentissage

## Supervisé

Apprend à partir d'un **échantillon d'entraînement** qui contient des exemples ainsi que les réponses associées. L'algorithme tente de généraliser suite à un entrainement.

## Non Supervisé

Les réponses ne sont pas fournies, l'algorithme cherche **les similitudes** entre les données qui lui sont fournies pour former **des catégories**. L'approche statistique donne des estimations de densité.

## Renforcé

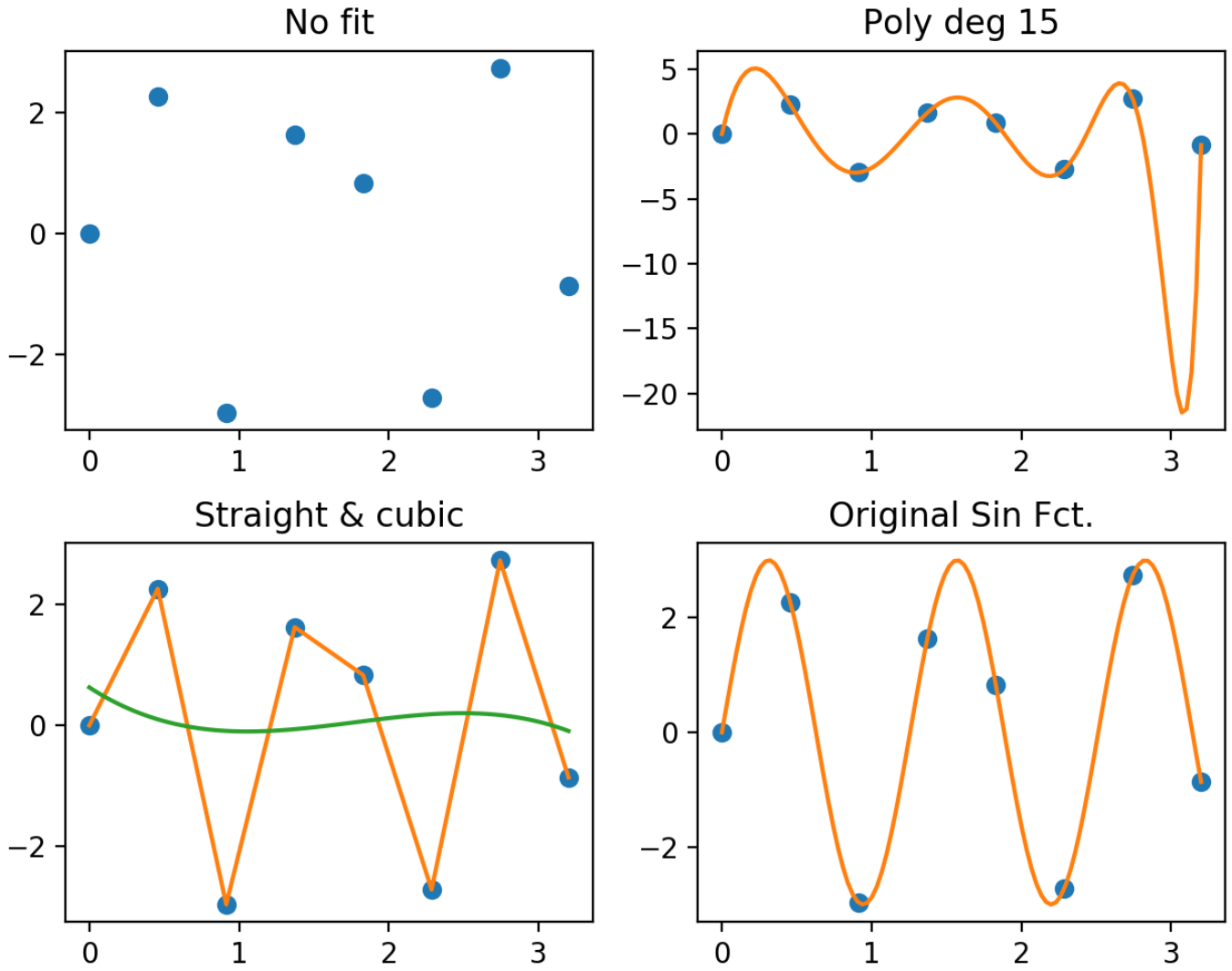
Mélange des méthodes supervisées et non supervisées. Nécessite un échantillon d'entraînement avec les réponses mais **aucune indication** n'est donnée sur **la manière de s'améliorer**.

## Evolutionnaire

Inspiré des organismes biologiques qui s'adaptent pour **maximiser leurs chances de survies**. On représente alors la chance de survie comme la proximité à la solution attendue.

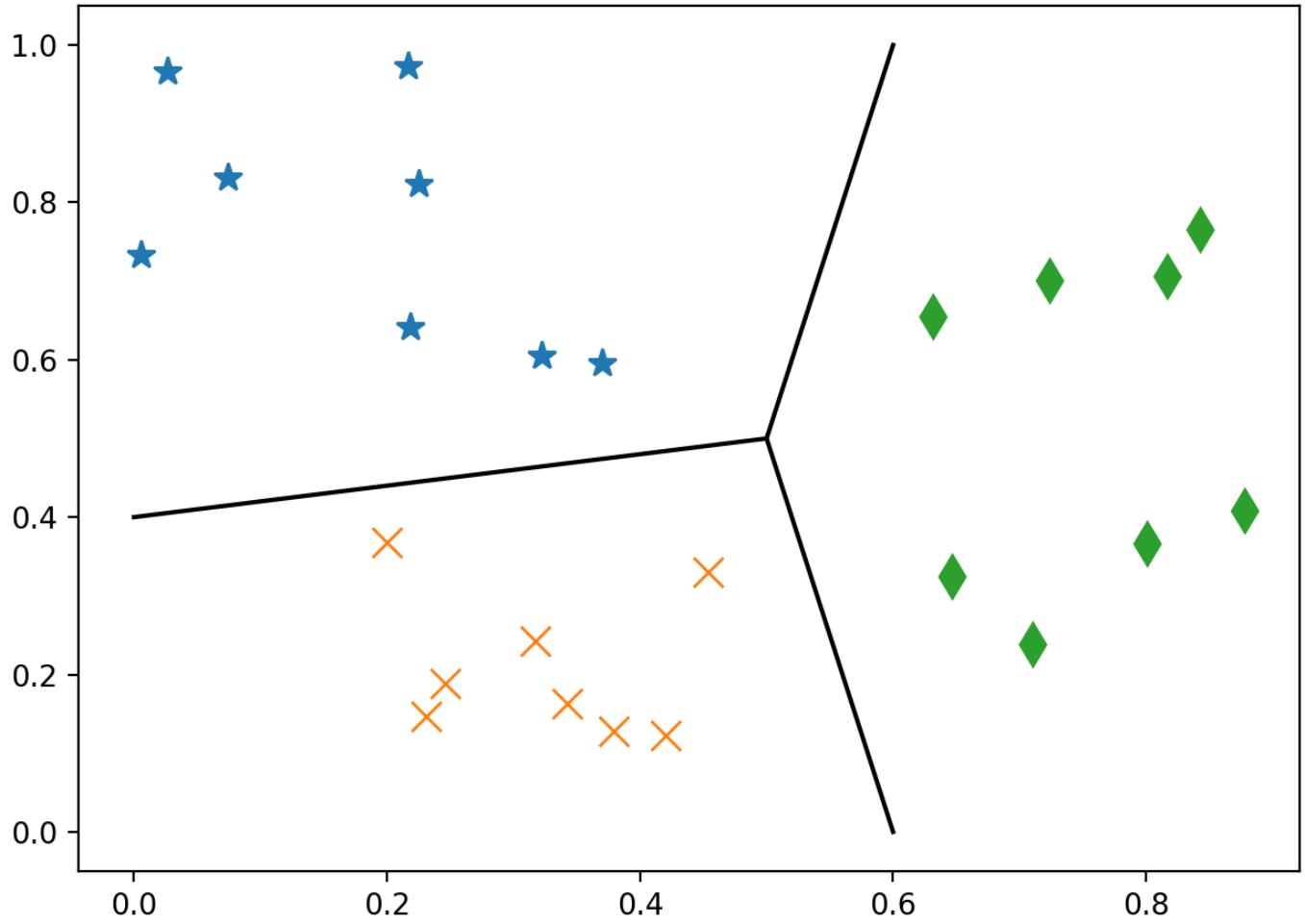
# Apprentissage supervisé : Régression

Retrouver une fonction mathématique qui se rapproche le plus d'un ensemble de points



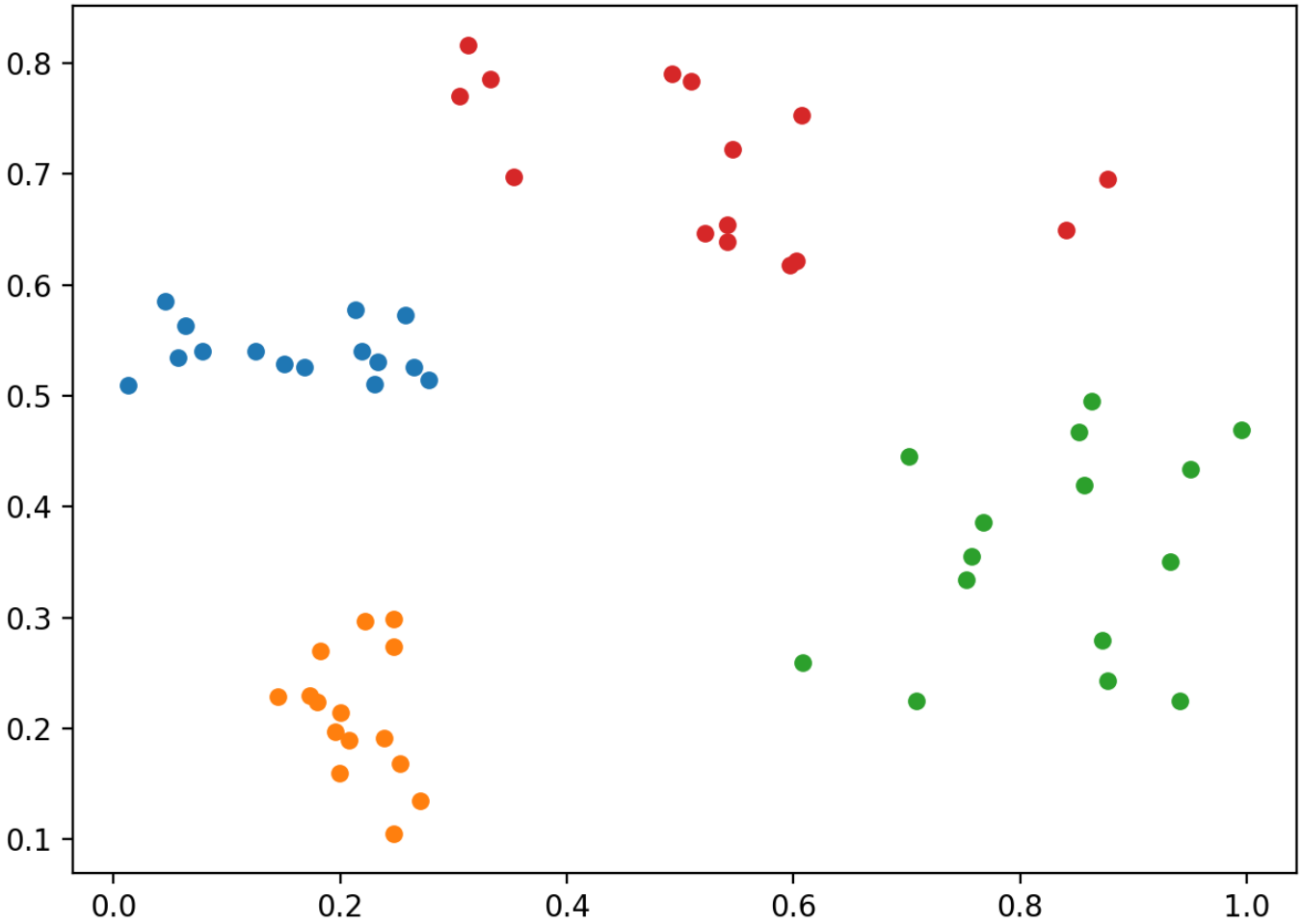
# Apprentissage supervisé : Classification

Décider à partir d'un vecteur d'entrée une des N classes associées.
Difficulté : déterminer la forme des frontières. Parfois des objets n'appartiennent pas à une classe définie (novelty détection)



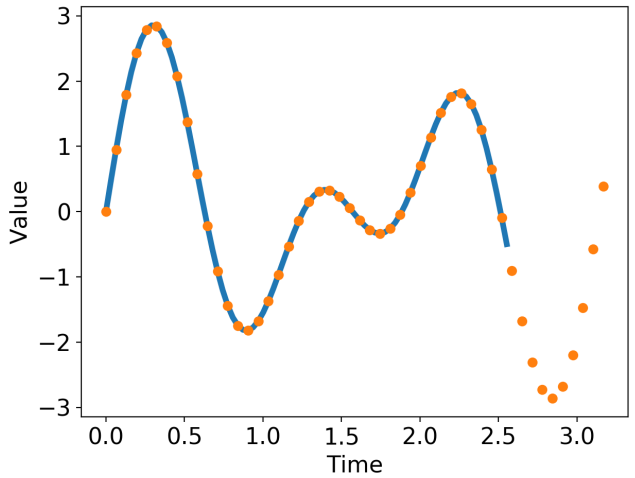
# Apprentissage non supervisé : Clustering

Trouver des groupes dans les données selon une stratégie prédéfinie (par exemple la distance géométrique) sans connaître à l'avance ni le nombre de groupes ni aucune solution pour aucun des objets.



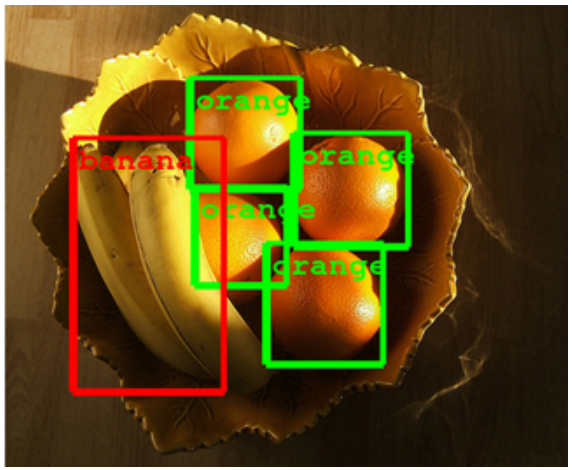
# Autres application du machine learning

**Prédiction de série temporelle**




**Compression de données**

**Reconnaissance d'images**




...

# Frameworks / Outils pour le ML

Maintenant que l'on sait ce dont le ML est capable comment l'utiliser ?

julia

mxnet


DEEP LEARNING


theano

Microsoft CNTK

Caffe2

# Exemple de frameworks

TensorFlow ™ Install Develop Community API r1.11 Ecosystem Search GITHUB

API r1.11

PYTHON JAVASCRIPT C++ JAVA MORE...

Overview

Implementations
- TensorFlow.js
- Go
- Haskell
- Community Supported

Ecosystem
- Hub

## API Documentation

☆☆☆☆☆

TensorFlow has APIs available in several languages both for constructing and executing a TensorFlow graph. The Python API is at present the most complete and the easiest to use, but other language APIs may be easier to integrate into projects and may offer some performance advantages in graph execution.

A word of caution: the APIs in languages other than Python are not yet covered by the API stability promises.

- Python
- JavaScript
- C++
- Java
- Go
- Swift (Early Release)

We encourage the community to develop and maintain support for other languages with the approach recommended by the TensorFlow maintainers. For example, see the bindings for:

- C#,
- Haskell,
- Julia,
- Ruby,
- Rust, and
- Scala.

We also provide the C++ API reference for TensorFlow Serving:

- TensorFlow Serving

# Exemple de frameworks

```
from __future__ import print_function

import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

# Parameters
learning_rate = 0.01
training_epochs = 1000
display_step = 50

# Training Data
train_X = numpy.asarray(
[3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,
2.167,7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.asarray(
[1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,
1.221,2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]

# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples
    )
# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(
      learning_rate).minimize(cost)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X
                , Y:train_Y})
            print("Epoch:", '%04d' % (epoch+1), "
                cost=", "{:.9f}".format(c), \
            "W=", sess.run(W), "b=", sess.run(b))

    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X:
          train_X, Y: train_Y})
    print("Training cost=", training_cost, "W=",
          sess.run(W), "b=", sess.run(b), '\n')

    plt.plot(train_X, train_Y, 'ro', label='Original
          data')
    plt.plot(train_X, sess.run(W) * train_X + sess.
          run(b), label='Fitted line')
    plt.legend()
    plt.show()

    test_X = numpy.asarray([6.83, 4.668, 8.9, 7.91,
          5.7, 8.7, 3.1, 2.1])
    test_Y = numpy.asarray([1.84, 2.273, 3.2, 2.831,
           2.92, 3.24, 1.35, 1.03])

    print("Testing... (Mean square loss Comparison)"
          )
    testing_cost = sess.run(
        tf.reduce_sum(tf.pow(pred - Y, 2)) / (2 *
              test_X.shape[0]),
        feed_dict={X: test_X, Y: test_Y})  # same
              function as cost above
    print("Testing cost=", testing_cost)
    print("Absolute mean square loss difference:",
          abs(
        training_cost - testing_cost))

    plt.plot(test_X, test_Y, 'bo', label='Testing
          data')
    plt.plot(train_X, sess.run(W) * train_X + sess.
          run(b), label='Fitted line')
    plt.legend()
plt.show()
```

# Exemple de frameworks

TensorFlow ™ Install Develop Community API r1.11 Ecosystem Search GITHUB

PYTHON JAVASCRIPT C++ JAVA MORE...

C++
- array_ops
- candidate_sampling_ops
- control_flow_ops
- core
- data_flow_ops
- image_ops
- io_ops
- logging_ops
- math_ops
- nn_ops
- no_op
- parsing_ops
- random_ops
- sparse_ops
- state_ops
- string_ops
- training_ops
- user_ops
- Namespaces

Guides

## The Basics

Let's start with a simple example that illustrates graph construction and execution using the C++ API.

```
// tensorflow/cc/example/example.cc

#include "tensorflow/cc/client/client_session.h"
#include "tensorflow/cc/ops/standard_ops.h"
#include "tensorflow/core/framework/tensor.h"

int main() {
  using namespace tensorflow;
  using namespace tensorflow::ops;
  Scope root = Scope::NewRootScope();
  // Matrix A = [3 2; -1 0]
  auto A = Const(root, { {3.f, 2.f}, {-1.f, 0.f} });
  // Vector b = [3 5]
  auto b = Const(root, { {3.f, 5.f} });
  // v = Ab^T
  auto v = MatMul(root.WithOpName("v"), A, b, MatMul::TransposeB(true));
  std::vector<Tensor> outputs;
  ClientSession session(root);
  // Run and fetch v
  TF_CHECK_OK(session.Run({v}, &outputs));
  // Expect outputs[0] == [19; -3]
  LOG(INFO) << outputs[0].matrix<float>();
  return 0;
}
```

Place this example code in the file `tensorflow/cc/example/example.cc` inside a clone of the TensorFlow github repository. Also place a `BUILD` file in the same directory with the following contents:

```
load("//tensorflow:tensorflow.bzl", "tf_cc_binary")

tf_cc_binary(
    name = "example",
    srcs = ["example.cc"],
    deps = [
```

Contents
- The Basics
- Graph Construction
  - Scope
  - Operation Constructors
  - Constants
- Graph Execution

# Exemple de frameworks

```
# Parameters
TRAINING_SIZE <- 50000
DIGITS <- 2
MAXLEN <- DIGITS + 1 + DIGITS
charset <- c(0:9, "+", "␣")
char_table <- learn_encoding(charset)
examples <- generate_data(size = TRAINING_SIZE,
      digits = DIGITS)
x <- array(0, dim = c(length(examples$questions),
      MAXLEN, length(char_table)))
y <- array(0, dim = c(length(examples$questions),
      DIGITS + 1, length(char_table)))

for(i in 1:TRAINING_SIZE){
  x[i,,] <- encode(examples$questions[i], char_table
      )
  y[i,,] <- encode(examples$results[i], char_table)
}

indices <- sample(1:TRAINING_SIZE, size = TRAINING_
      SIZE)
x <- x[indices,,]
y <- y[indices,,]
# Explicitly set apart 10% for validation data that
      we never train over
split_at <- trunc(TRAINING_SIZE/10)
x_val <- x[1:split_at,,]
y_val <- y[1:split_at,,]
x_train <- x[(split_at + 1):TRAINING_SIZE,,]
y_train <- y[(split_at + 1):TRAINING_SIZE,,]

print('Training␣Data:')
print(dim(x_train))
print(dim(y_train))
print('Validation␣Data:')
print(dim(x_val))
print(dim(y_val))

# Training
HIDDEN_SIZE <- 128
BATCH_SIZE <- 128
```

```
LAYERS <- 1

model <- keras_model_sequential()

model %>%
  layer_lstm(HIDDEN_SIZE, input_shape=c(MAXLEN,
        length(char_table))) %>%
  layer_repeat_vector(DIGITS + 1)

for(i in 1:LAYERS)
  model %>% layer_lstm(HIDDEN_SIZE, return_sequences
        = TRUE)

model %>%
  time_distributed(layer_dense(units = length(char_
        table))) %>%
  layer_activation("softmax")

model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
summary(model)

# Fitting loop
model %>% fit(
  x = x_train,
  y = y_train,
  batch_size = BATCH_SIZE,
  epochs = 70,
  validation_data = list(x_val, y_val)
)

# Predict for a new observation
new_obs <- encode("55+22", char_table) %>%
  array(dim = c(1,5,12))
result <- predict(model, new_obs)
result <- result[1,,]
decode(result, char_table)
```

# Quelle approche choisir ?

**Utiliser des frameworks**

- Beaucoup de supports (Guides, aide au debug, ...)
- Souvent déjà parallelisé
- Performant rapidement
- Mis à jour sans votre intervention (gain de performance dans le temps sans changer de code)

**"Ouvrir la boîte"**

- Meilleure compréhension des mécanismes
- Contrôle plus fin, et niveau d'optimisation contrôlé
- Indépendant du maintien d'une librairie
- Capable à terme d'utiliser les différents frameworks facilement

# Exemple d'algorithmes supervisé : SVM

**Description :** Le Support Vector Machine est un algorithme supervisé qui sert à faire de la classification.

**Objectif :** Généraliser les propriétés propres à certaines classes d'objets pré-établies avec un nombre de dimensions arbitraire.

**Méthode :** trouver les points les plus proches de la séparation étudiée qui définiront les "supports". Une fois ces points établis, l'algorithme cherche la séparation linéaire qui minimise la somme des distances entre les points des différentes classes. Pour cela, il cherche notamment à maximiser la surface autour de cette séparation qui ne contient aucun des "supports".

**Mémoire :** L'algorithme conserve la mémoire de cette classification via la pente et l'ordonnée à l'origine de cette séparation linéaire. $\rightarrow$ Capable de généraliser.

# Support Vector Machine illustré

# Mise en pratique : L'algorithme k-means

**Description :** Cet algorithme est non-supervisé et sert à faire du clustering.

**Objectif :** Trouver des groupes dans des données avec un nombre de dimensions arbitraire selon une méthode géométrique.

**Méthode :** definir un nombre "k" de centres (definie le nombre de groupes maximum). Pour chacun des points il trouve le centre le plus proche selon une distance géométrique. Les centres sont ensuite deplacés vers le centre de gravité du groupe de points qui leur correspond. Cette opération est repétée jusqu'à ce que les centres ne bougent plus.

**Mémoire :** L'algorithme conserve la mémoire des clusters en retenant la position des différents centres. $\rightarrow$ Capable de généraliser.

# k-means clustering illustré

# k-means algorithme

**Boucle principale :** jusqu'à ce que les centres ne bougent plus

- **Phase d'identification :** boucle sur les points
  - Calcul de la distance aux différents centres (k)
  - Associer le point en cours au centre le plus proche
- **Phase de mise à jour :** Boucle sur les centres
  - Calcul du barycentre du cluster defini précédemment
  - Repositioner le centre à la place du barycentre

# k-means : Exercice - Coder le k-means

**Données :** Vous disposez de fichiers de données *kmeans_input_file.dat*, ainsi que du code python ayant servi pour leur génération.

**Code :** un code incomplet est fourni, la lecture du fichier d'entrée y est intégrée, ainsi qu'une fonction de calcul de distance. Plusieurs variables utiles sont fournies mais toutes les variables nécessaires n'ont pas été déclarées.

**Points clés :**

- Identifier les variables supplémentaires dont vous aurez besoin.
- Bien identifier les différentes boucles, et les coder.
- Prendre le temps de réfléchir au moyen de trouver la distance minimum.
- Trouver comment interrompre la boucle principale intelligemment.

**Résultats :** Un outil de visualisation est proposé *kmeans_visual.py*