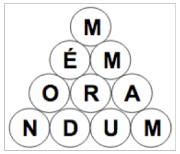

# Tensorflow
# Présentation du MOOC de Google
# et Première vue sur l'API

Romain Jouin




TensorFlow is an Open Source Software Library for Machine Intelligence

UDACITY

# Deep Learning

## Take machine learning to the next level

Advanced

Approx. 3 months

Assumes 6hrs/wk (work at your own pace)

Built by Google

Join 61,607 students

## Prerequisites and Requirements

This is an intermediate to advanced level course. Prior to taking this course, and in addition to the prerequisites and requirements outlined for the Machine Learning Engineer Nanodegree program, you should possess the following experience and skills:

- Minimum 2 years of programming experience (preferably in Python)
- Git and GitHub experience (assignment code is in a GitHub repo)
- Basic machine learning knowledge (especially supervised learning)
- Basic statistics knowledge (mean, variance, standard deviation, etc.)
- Linear algebra (vectors, matrices, etc.)
- Calculus (differentiation, integration, partial derivatives, etc.)

scipy, numpy, scikit learn...

## Syllabus

**Lesson 1: From Machine Learning to Deep Learning**

- Understand the historical context and motivation for Deep Learning.
- Set up a basic supervised classification task and train a black box classifier on it.
- Train a logistic classifier "by hand"Optimize a logistic classifier using gradient descent, SGD, Momentum and AdaGrad.

**Lesson 2: Deep Neural Networks**

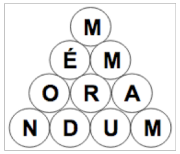
- Train a simple deep network.
- Effectively regularize a simple deep network.
- Train a competitive deep network via model exploration and hyperparameter tuning.

**Lesson 3: Convolutional Neural Networks**

- Train a simple convolutional neural net.
- Explore the design space for convolutional nets.

**Lesson 4: Deep Models for Text and Sequences**

- Train a text embedding model.
- Train a LSTM model.



Environnement pour la soirée

python

TensorFlow

# THE MNIST DATABASE

## of handwritten digits

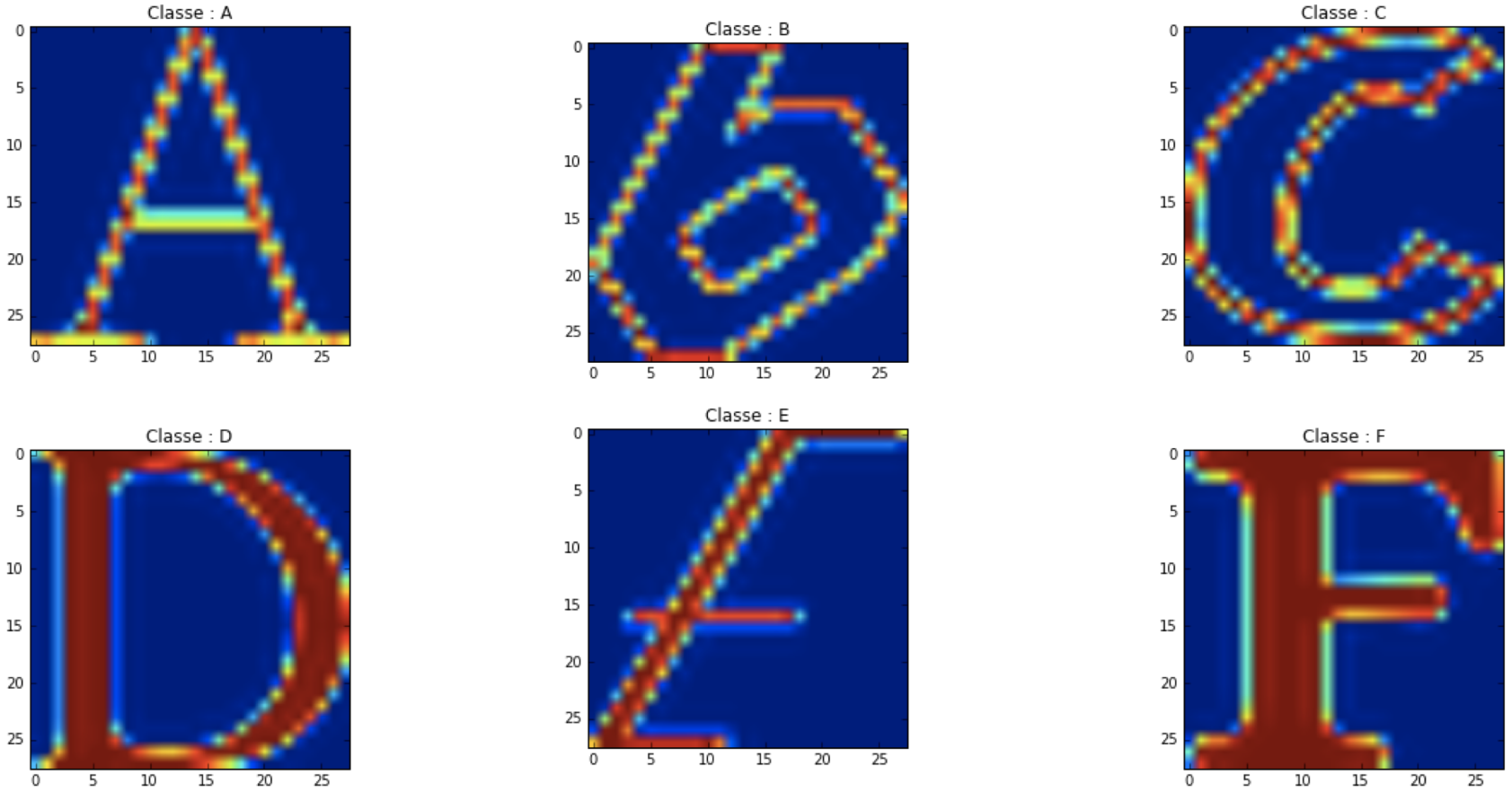
Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

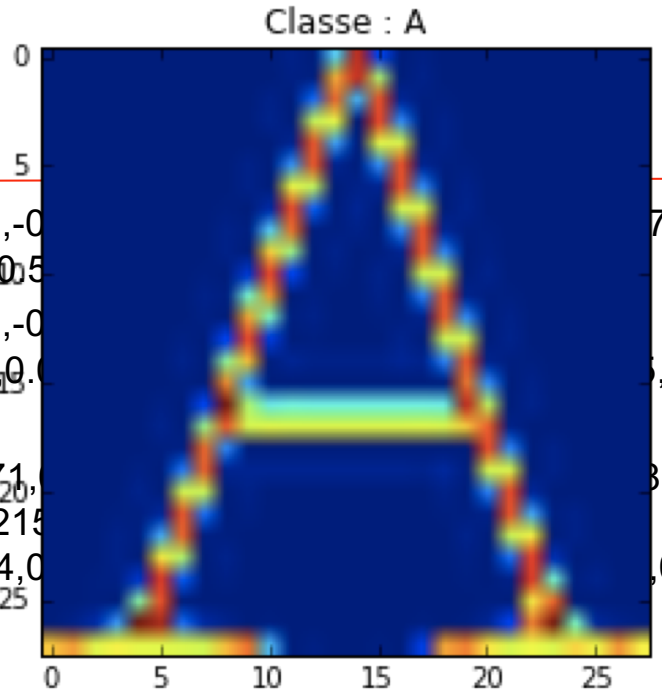
http://yann.lecun.com/exdb/mnist/

MÉMORANDUM

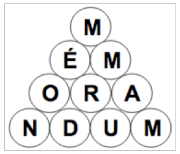
# MNIST Database

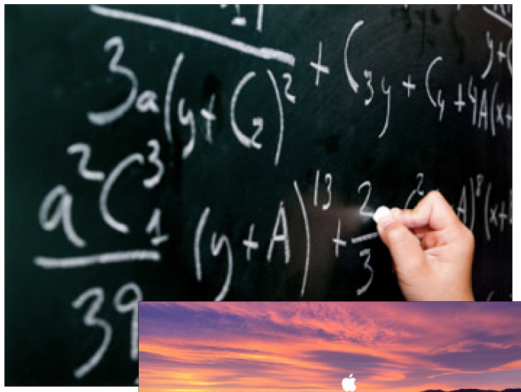


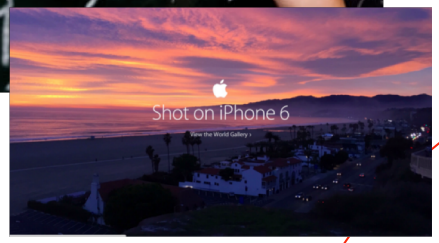
dataset.shape = (52909, 28, 28)
dataset[0].shape = (28, 28)




array(
[
[-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0 ... 7451,0.4254902,-0.36274511,-0.5,-
0.49215686,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5
[-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0
0.49607843,0.2254902,0.42156863,0.0 ... ,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5],
...
[0.19411765,0.24901961,0.11176471,0 ... 3,0.10392157,0.0882353,0.20980392,
0.30392158,-0.19019608,-0.5,-0.49215
0.40196079,0.23333333,0.28431374,0 ... ,0.10392157,0.18235295,0.13137256,
0.26078433,0.14705883]
],
dtype=float32)

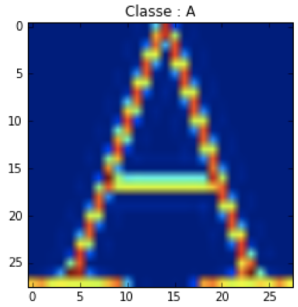




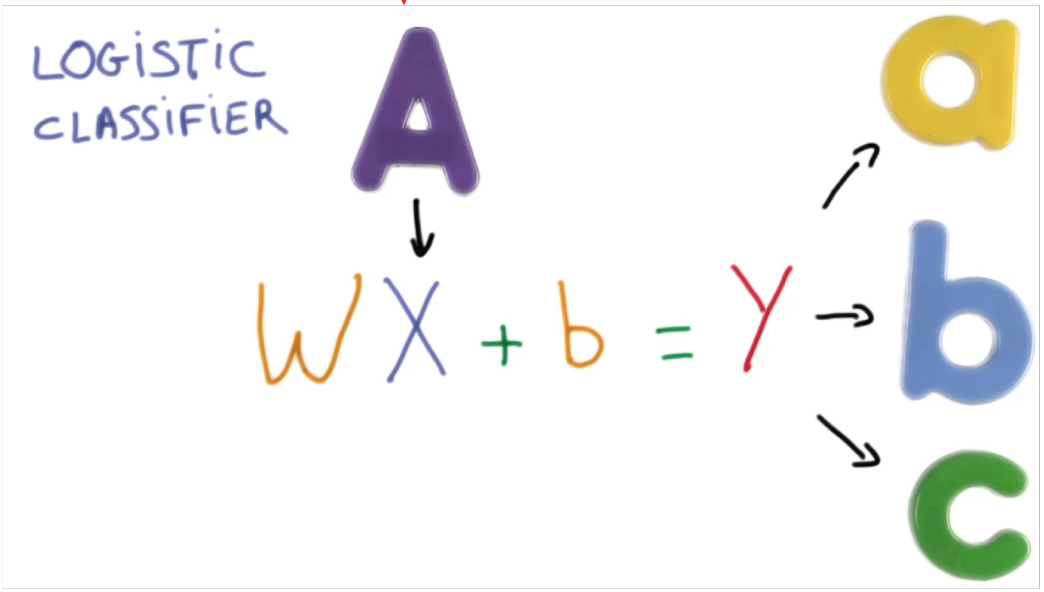



12 millions de paramètres

## Logistic classifier et softmax




28 * 28 = 756



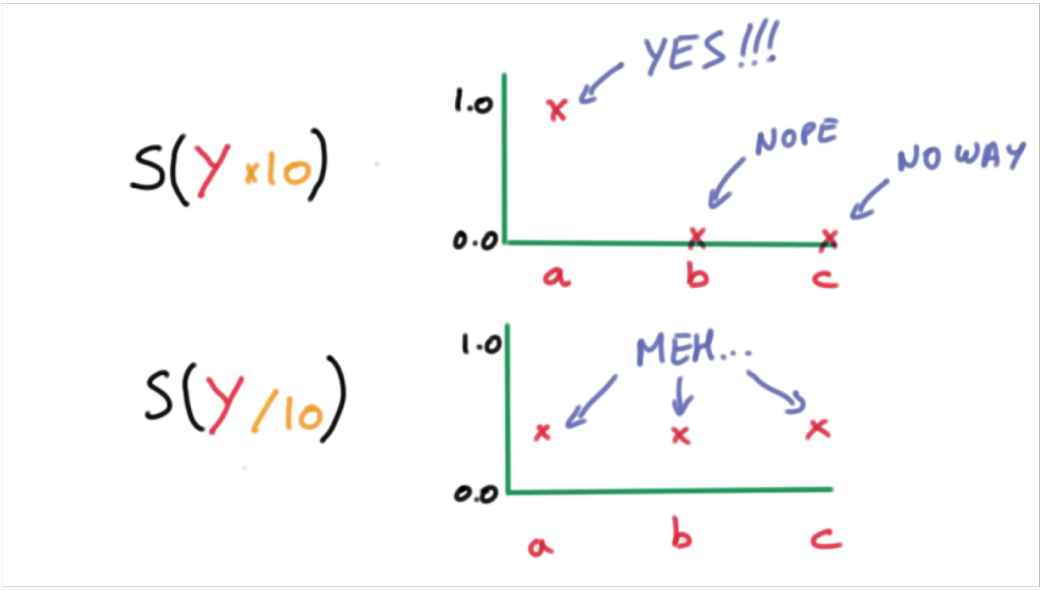
# One hot encoding

Pour prendre une décision binaire sur la classification, on peut transformer les probabilités en valeurs "0" ou "1". C'est le "One hot encoding".

Il faut alors un vecteur par classe, qui contiendra plein de zéros, et une seule valeur "1".

Ainsi chaque classe a son vecteur signature qui lui permet d'être reconnaissable.

Le problème du one-hot encoding c'est que ça créé des matrices très "creuses" s'il y a beaucoup de classes : on a quasiment que des zéros, pour un seul "1" par colonne.



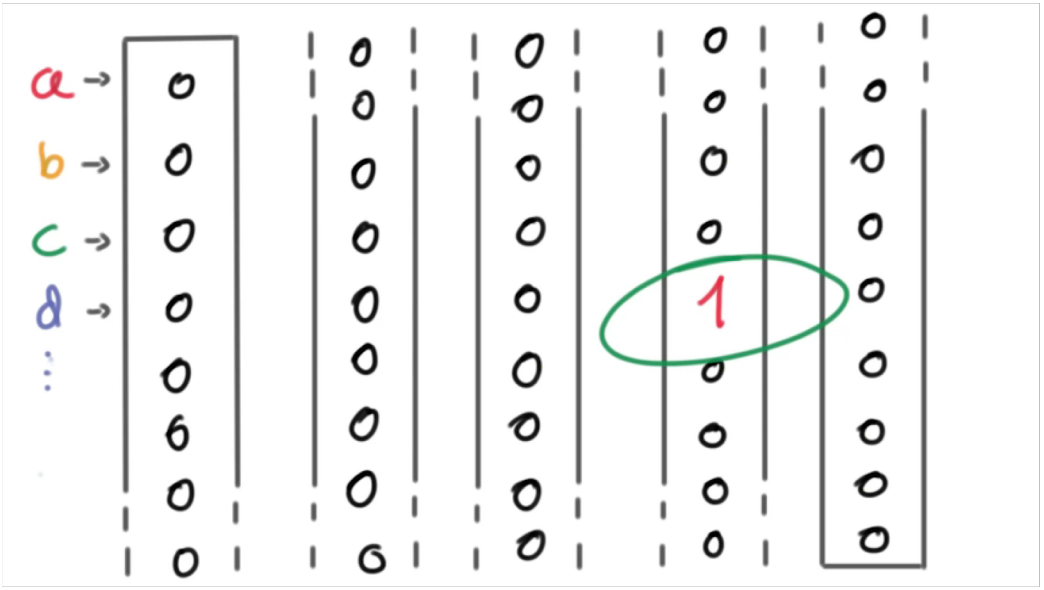
## Cross entropy

Cette fonction permet de transformer l'ensemble des probabilités en un vecteur binaire avec un seul "1", qui sera la classe élue.

On a donc :

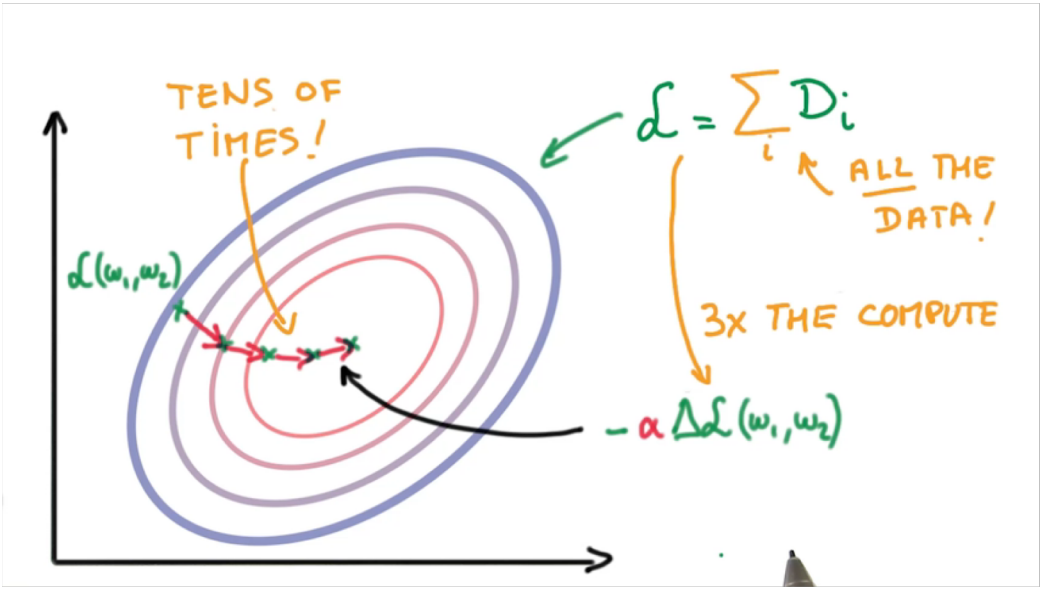
input => modèle => score => softmax =>probabilité => cross-entropy => classe choisie.

C'est la classification logistique multinomiale.



## Schotastic gradient descent (vidéo 36)

Comment trouver la matrice des poids [W] et la matrice de biais [b] qui nous permettent d'atteindre notre objectif, à savoir :
1) d'avoir une petite distance aux classes correctes
2) d'avoir une grande distances aux classes incorrectes



UDACITY

# Jeu de donnée utilisé pendant le MOOC

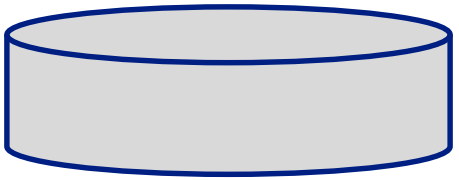
NOT

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

Téléchargement
10 classes : A -> J



training set : 500k images
**train_datasets**

```
notMNIST_large/A.pickle
notMNIST_large/B.pickle
notMNIST_large/C.pickle
notMNIST_large/D.pickle
notMNIST_large/E.pickle
notMNIST_large/F.pickle
notMNIST_large/G.pickle
notMNIST_large/H.pickle
notMNIST_large/I.pickle
notMNIST_large/J.pickle
```

Test set : 19k images
**test_datasets**

```
notMNIST_small/A.pickle
notMNIST_small/B.pickle
notMNIST_small/C.pickle
notMNIST_small/D.pickle
notMNIST_small/E.pickle
notMNIST_small/F.pickle
notMNIST_small/G.pickle
notMNIST_small/H.pickle
notMNIST_small/I.pickle
notMNIST_small/J.pickle
```

Chaque répertoire
contient environ
50 000 images

```
['notMNIST_small/A.pickle',
 'notMNIST_small/B.pickle',
 'notMNIST_small/C.pickle',
 'notMNIST_small/D.pickle',
 'notMNIST_small/E.pickle',
 'notMNIST_small/F.pickle',
 'notMNIST_small/G.pickle',
 'notMNIST_small/H.pickle',
 'notMNIST_small/I.pickle',
 'notMNIST_small/J.pickle']
```

MÉMORANDUM

UDACITY

Lesson 1 – classification logistique et descente de gradient

Lesson 2  - Réseau de neurone profond

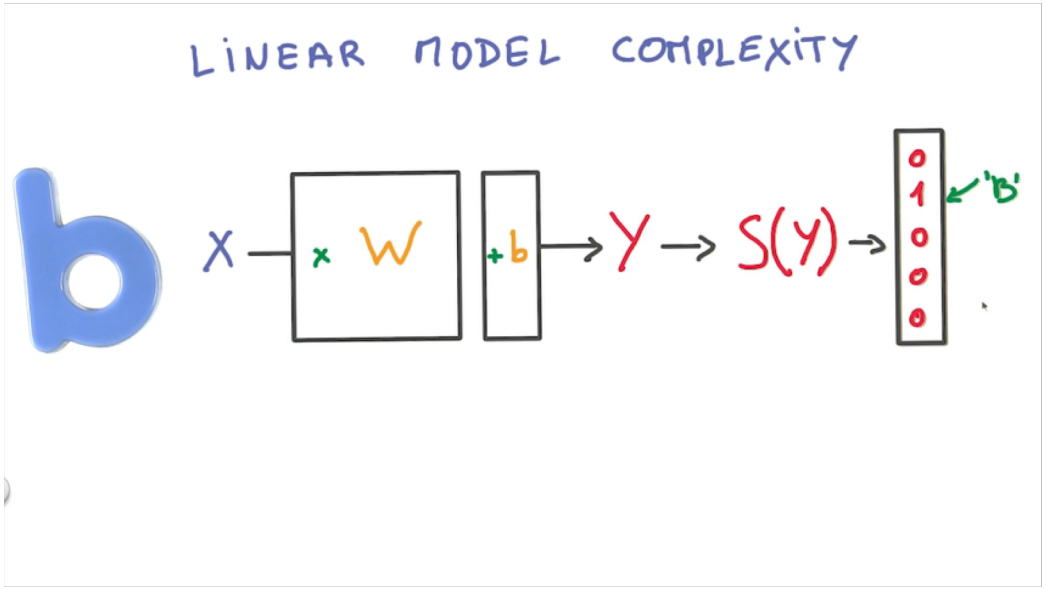
Lesson 3  - Réseau de convolution

Lesson 4 – Réseau de neurone profond pour l'analyse de texte et de séquences (to do)

MÉMORANDUM

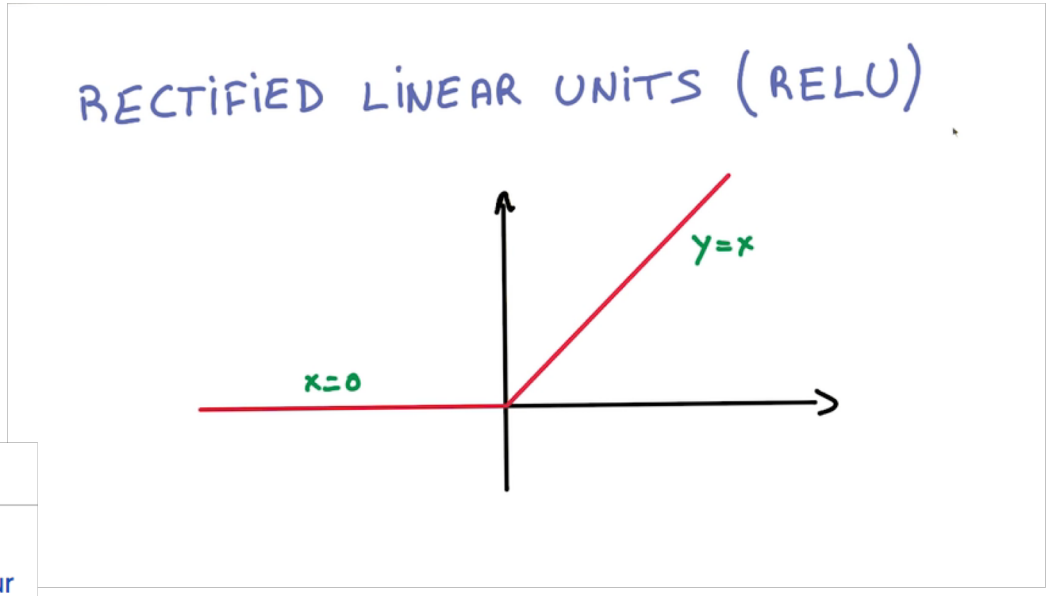
**Nombre de paramètres (vidéo 2)**

Les fonctions linéaires sont très stables, et se calculent bien sur GPU. On voudrait donc les conserver.
Par contre elles ne permettent pas de faire tout type de calculs.
Donc on va chercher à les modifier pour étendre le champ des possibles.



**RELU (vidéo 5)**

Le « relu » est une fonction très simple : c'est le maximum entre 0 et x ....




Rappel - dérivée :

## Définition formelle [ modifier | modifier le code ]

Soit $f$ une fonction réelle à valeurs réelles définie sur une réunion quelconque d'intervalles non triviaux, et $x_0$ appartenant à l'intérieur de l'ensemble de définition $\mathcal{D}_f$.

Pour tout $h \in \mathbb{R}^*$ tel que $[x_0, x_0 + h] \subset \mathcal{D}_f$, on appelle *taux d'accroissement de $f$ en $x_0$ et avec un pas de $h$* la quantité :

$$t_{x_0}(h) = \frac{f(x_0 + h) - f(x_0)}{h}$$

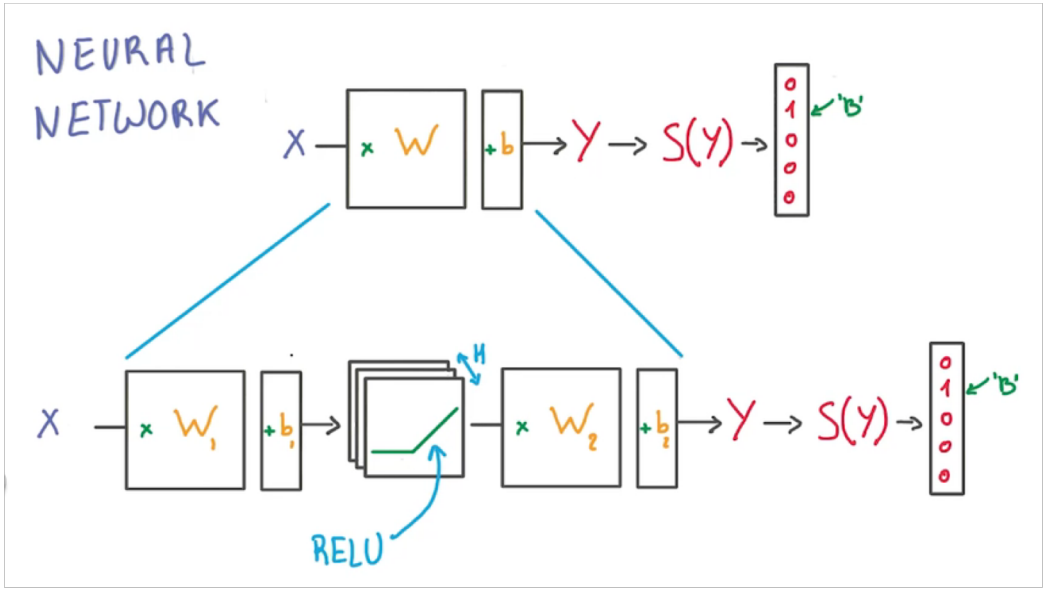
**Réseau de neurone profond (vidéo 7)**

On rend les fonctions non linéaires en insérant une fonction « relu » au milieu de la fonction...

Les calculs des dérivées restent alors très simple :

[ g( f(x) ) ]’= g’( f(x) ) * f’(x)

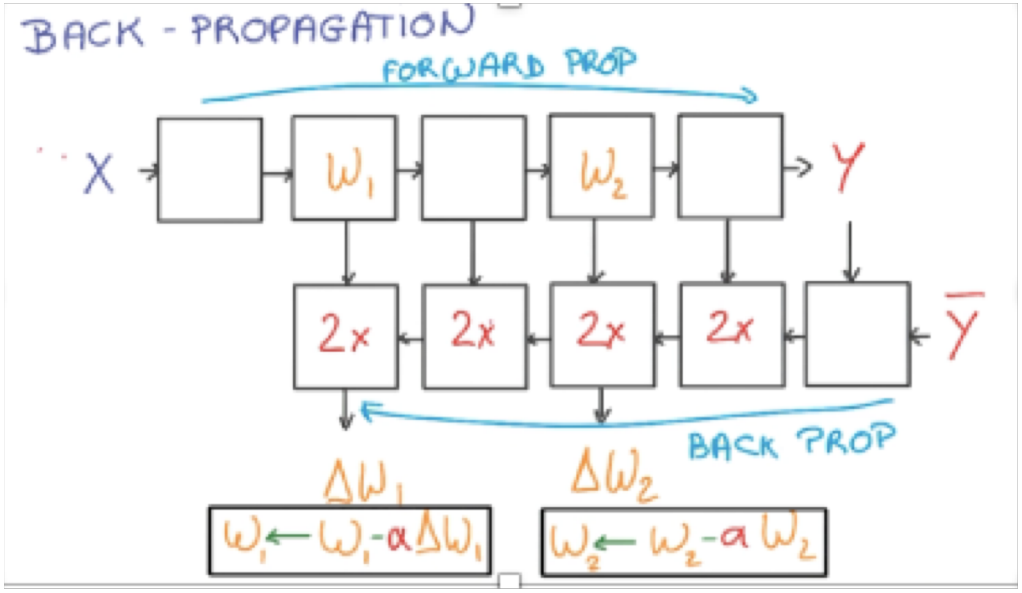
Comme la descente de gradient fait principalement du calcul de dérivée, il est important que cela reste simple.



# Back Propagation

Le calcul du gradient pour un ensemble de poids **W** donné se fait en deux étapes :

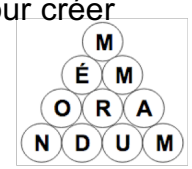
1) **Forward propagation**
    1) relu -> transformation linéaire -> relu -> transformation linéaire
        1) On obtient alors une première prédiction Y
        2) Pour affiner cette prédiction et faire la descente de gradient, il faut trouver les dérivées. Pour trouver les dérivées, on va passer par l'étape 2




**2) Back propagation**

Le graph de back-propagation est calculé par Tensorflow (il est déduit du premier graph), on n'a donc pas en le programmer.
Il calcule des dérivées pour de la matrice de poids **W** que l'on va soustraire de **W** pour créer une nouvelle matrice qui la remplacera.

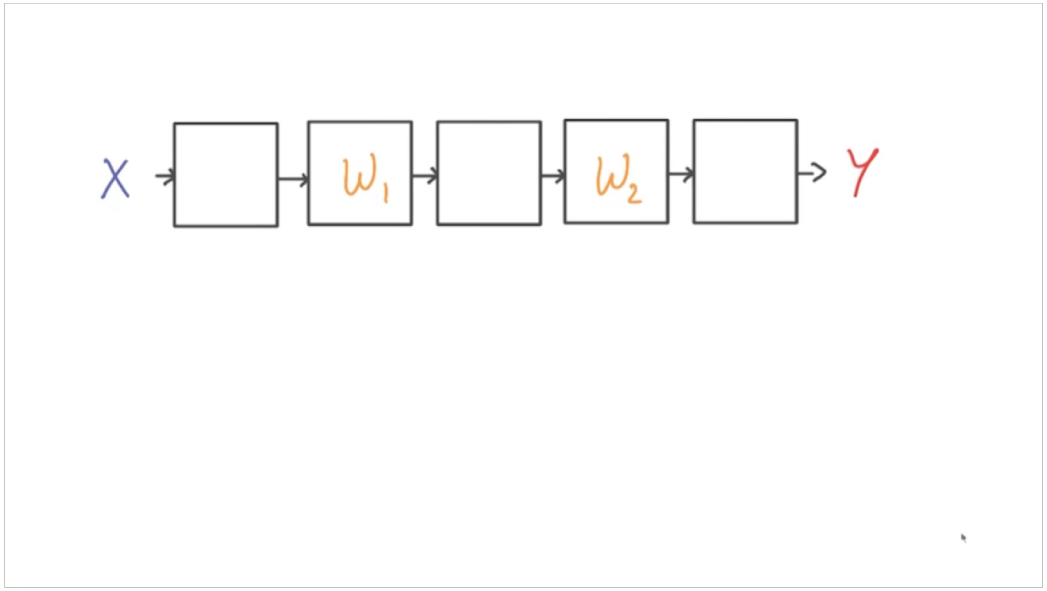
3) **on applique un coefficient alpha sur la dérivée de W avant la soustraction**



# Back Propagation

Le calcul du gradient pour un ensemble de poids **W** donné se fait en deux étapes :

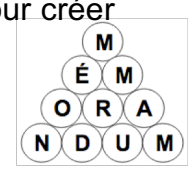
1) **Forward propagation**
    1) relu -> transformation linéaire -> relu -> transformation linéaire
        1) On obtient alors une première prédiction Y
        2) Pour affiner cette prédiction et faire la descente de gradient, il faut trouver les dérivées. Pour trouver les dérivées, on va passer par l'étape 2



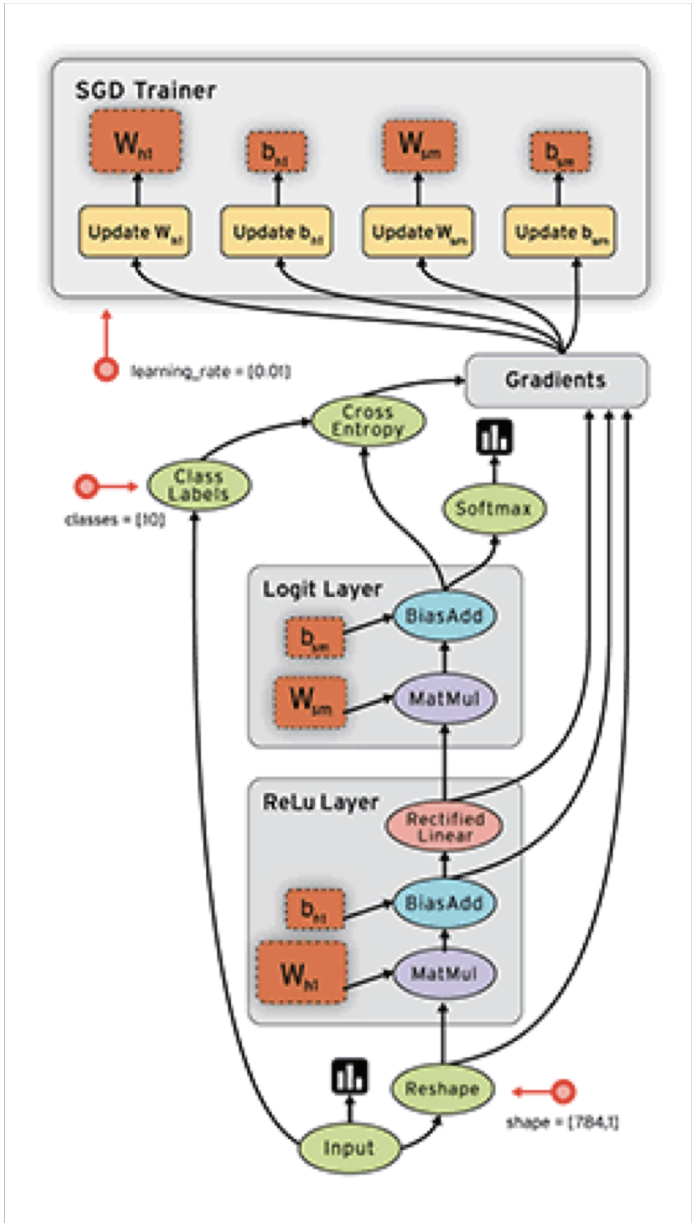

**2) Back propagation**

Le graph de back-propagation est calculé par Tensorflow (il est déduit du premier graph), on n'a donc pas en le programmer.
Il calcule des dérivées pour de la matrice de poids **W** que l'on va soustraire de **W** pour créer une nouvelle matrice qui la remplacera.

3) **on applique un coefficient alpha sur la dérivée de W avant la soustraction**



TensorFlow


SGD Trainer
Update W_h1
Update b_h1
Update W_sm
Update b_sm
learning_rate = [0.01]
Gradients
Cross Entropy
Class Labels
classes = [10]
Softmax
Logit Layer
b_sm
BiasAdd
W_sm
MatMul
ReLu Layer
Rectified Linear
b_h1
BiasAdd
W_h1
MatMul
Reshape
shape = [784,1]
Input


## https://www.tensorflow.org/versions/r0.7/api_docs/python/state_ops.html#variables

| | |
|---|---|
| Building Graphs: | 26 |
| Constants, Sequences, and Random Values: | 14 |
| Variables: | 32 |
| Tensor Transformations: | 31 |
| Math: | 80 |
| Control Flow: | 26 |
| Images: | 35 |
| Sparse Tensors: | 11 |
| Inputs and Readers: | 30 |
| Data IO (Python functions): | 2 |
| Neural Network: | 34 |
| Running Graphs: | 20 |
| Training: | 35 |
| Wraps python functions: | 1 |
| Testing: | 6 |
| Layers (contrib): | 12 |
| Utilities (contrib): | 2 |

```
#Program that makes up some data in two dimensions, and then fits a line to it.

import tensorflow          as tf
import numpy               as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b  = tf.Variable(tf.zeros([1]))
y   = W * x_data + b

loss        = tf.reduce_mean(tf.square(y - y_data)) # mean squared errors to minimize
optimizer = tf.train.GradientDescentOptimizer(0.5)
train       = optimizer.minimize(loss)

init        = tf.initialize_all_variables()    # Before starting, initialize the variables
sess      = tf.Session()                       # Launch the graph.
sess.run(init)

for step in xrange(201):                       # Fit the line.
   sess.run(train)
   if step % 20 == 0: print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```


**Building Graphs**

add_to_collection as_dtype bytes control_dependencies
convert_to_tensor convert_to_tensor_or_indexed_slices
device Dimension DType get_collection get_default_graph
get_seed **Graph** GraphKeys import_graph_def
load_op_library name_scope NoGradient op_scope
Operation register_tensor_conversion_function
RegisterGradient RegisterShape reset_default_graph Tensor
TensorShape

**Constants, Sequences, and Random Values**

constant fill linspace ones ones_like random_crop
random_normal random_shuffle **random_uniform** range
set_random_seed truncated_normal zeros zeros_like


## Variables

all_variables assert_variables_initialized assign assign_add
assign_sub constant_initializer count_up_to device
get_checkpoint_state get_variable get_variable_scope
IndexedSlices **initialize_all_variables** initialize_variables
latest_checkpoint make_template moving_average_variables
random_normal_initializer random_uniform_initializer Saver scatter_add
scatter_sub scatter_update sparse_mask trainable_variables
truncated_normal_initializer uniform_unit_scaling_initializer
update_checkpoint_state Variable variable_op_scope
variable_scope zeros_initializer

## Tensor Transformations

boolean_mask cast concat depth_to_space dynamic_partition
dynamic_stitch expand_dims gather pack pad rank
reshape reverse reverse_sequence shape shape_n size
slice space_to_depth split squeeze string_to_number tile
to_bfloat16 to_double to_float to_int32 to_int64 transpose
unique_with_counts unpack


**Math**

abs accumulate_n add add_n argmax argmin
batch_cholesky batch_matmul batch_matrix_determinant
batch_matrix_inverse batch_matrix_solve
batch_matrix_solve_ls batch_matrix_triangular_solve
batch_self_adjoint_eig ceil cholesky complex complex_abs
conj cos cross diag div edit_distance erf erfc
exp fft2d floor floordiv ifft2d imag inv
invert_permutation lgamma listdiff log matmul
matrix_determinant matrix_inverse matrix_solve
matrix_solve_ls matrix_triangular_solve maximum minimum
mod mul neg pow real reduce_all reduce_any
reduce_max **reduce_mean** reduce_min reduce_prod
reduce_sum round rsqrt scalar_mul segment_max
segment_mean segment_min segment_prod segment_sum
self_adjoint_eig sign sin sparse_segment_mean
sparse_segment_sqrt_n sparse_segment_sqrt_n_grad
sparse_segment_sum sqrt **square** sub transpose
truediv unique unsorted_segment_sum where


## Control Flow

add_check_numerics_ops Assert check_numerics cond count_up_to
equal greater greater_equal group identity is_finite is_inf
is_nan less less_equal logical_and logical_not
logical_or logical_xor no_op not_equal Print
select tuple verify_tensor_all_finite where

## Images

adjust_brightness adjust_contrast adjust_hue adjust_saturation
convert_image_dtype crop_to_bounding_box decode_jpeg
decode_png draw_bounding_boxes encode_jpeg encode_png
extract_glimpse flip_left_right flip_up_down grayscale_to_rgb
hsv_to_rgb pad_to_bounding_box per_image_whitening
random_brightness random_contrast random_flip_left_right
random_flip_up_down random_hue random_saturation
resize_area resize_bicubic resize_bilinear
resize_image_with_crop_or_pad resize_images
resize_nearest_neighbor rgb_to_grayscale rgb_to_hsv
sample_distorted_bounding_box saturate_cast transpose_image


**Sparse Tensors**

shape sparse_concat sparse_fill_empty_rows sparse_reorder sparse_retain sparse_split sparse_tensor_to_dense sparse_to_dense sparse_to_indicator SparseTensor SparseTensorValue

**Inputs and Readers**

batch batch_join decode_csv decode_json_example decode_raw FIFOQueue FixedLenFeature FixedLengthRecordReader FixedLenSequenceFeature IdentityReader limit_epochs match_filenames_once matching_files parse_example parse_single_example placeholder QueueBase RandomShuffleQueue range_input_producer read_file ReaderBase shuffle_batch shuffle_batch_join size slice_input_producer string_input_producer TextLineReader TFRecordReader VarLenFeature WholeFileReader

**Data IO (Python functions)**

tf_record_iterator TFRecordWriter


**Neural Network**

avg_pool bias_add compute_accidental_hits conv2d
conv2d_transpose depthwise_conv2d dropout elu
embedding_lookup fixed_unigram_candidate_sampler in_top_k l2_loss
l2_normalize learned_unigram_candidate_sampler
local_response_normalization log_uniform_candidate_sampler max_pool
max_pool_with_argmax moments nce_loss relu relu6
sampled_softmax_loss separable_conv2d sigmoid
sigmoid_cross_entropy_with_logits softmax softmax_cross_entropy_with_logits
softplus softsign sparse_softmax_cross_entropy_with_logits tanh top_k
uniform_candidate_sampler

**Running Graphs**

AbortedError AlreadyExistsError CancelledError DataLossError
DeadlineExceededError FailedPreconditionError get_default_session
InteractiveSession InternalError InvalidArgumentError
NotFoundError OpError OutOfRangeError PermissionDeniedError
ResourceExhaustedError **Session** UnauthenticatedError UnavailableError
UnimplementedError UnknownError


**Training**

AdagradOptimizer AdamOptimizer add_queue_runner
AggregationMethod clip_by_average_norm clip_by_global_norm
clip_by_norm clip_by_value Coordinator exponential_decay
ExponentialMovingAverage export_meta_graph FtrlOptimizer
generate_checkpoint_state_proto global_norm global_step
**GradientDescentOptimizer** gradients histogram_summary
image_summary import_meta_graph LooperThread
merge_all_summaries merge_summary MomentumOptimizer
Optimizer QueueRunner RMSPropOptimizer
scalar_summary start_queue_runners stop_gradient
summary_iterator SummaryWriter write_graph zero_fraction

**Wraps python functions**

py_func

**Testing**

assert_equal_graph_def compute_gradient
compute_gradient_error get_temp_dir is_built_with_cuda
main


**Layers (contrib):**

assert_same_float_dtype convolution2d fully_connected
l1_regularizer l2_regularizer summarize_activation
summarize_activations summarize_collection summarize_tensor
summarize_tensors xavier_initializer xavier_initializer_conv2d

**Utilities (contrib):**

constant_value make_tensor_proto


```
tf.train.Optimizer.minimize(loss, global_step=None, var_list=None,
gate_gradients=1, aggregation_method=None,
colocate_gradients_with_ops=False, name=None)
```

Add operations to minimize `loss` by updating `var_list`.

This method simply combines calls `compute_gradients()` and `apply_gradients()`. If you want to process the gradient before applying them call `compute_gradients()` and `apply_gradients()` explicitly instead of using this function.

Args:

- `loss`: A `Tensor` containing the value to minimize.
- `global_step`: Optional `Variable` to increment by one after the variables have been updated.
- `var_list`: Optional list of `Variable` objects to update to minimize `loss`. Defaults to the list of variables collected in the graph under the key `GraphKeys.TRAINABLE_VARIABLES`.
- `gate_gradients`: How to gate the computation of gradients. Can be `GATE_NONE`, `GATE_OP`, or `GATE_GRAPH`.
- `aggregation_method`: Specifies the method used to combine gradient terms. Valid values are defined in the class `AggregationMethod`.
- `colocate_gradients_with_ops`: If True, try colocating gradients with the corresponding op.
- `name`: Optional name for the returned operation.

Returns:

An Operation that updates the variables in `var_list`. If `global_step` was not `None`, that operation also increments `global_step`.