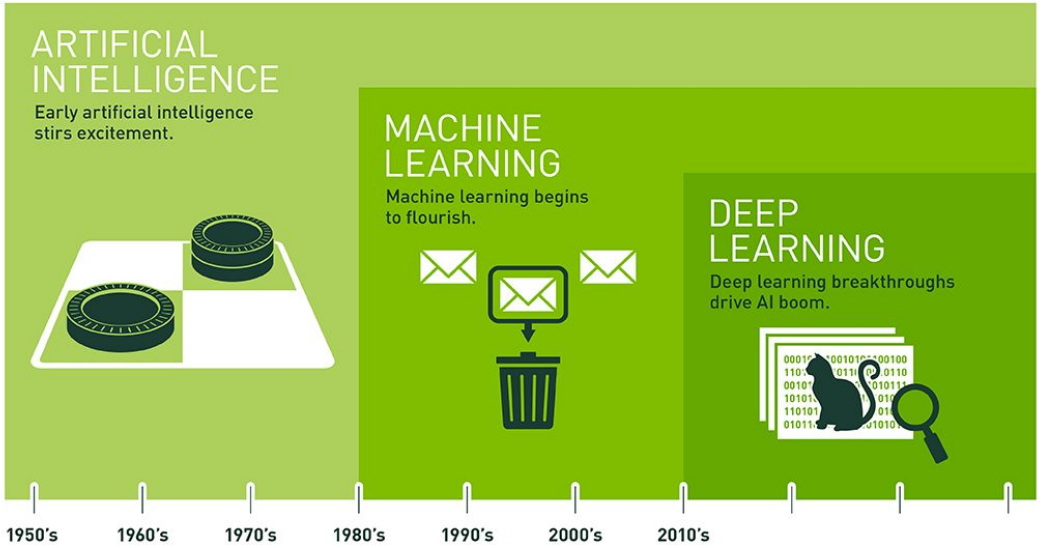
Deep Learning Series

Episode 1 - “La Menace Neurone”

What is Deep Learning ?

Artificial Intelligence, Machine Learning and Deep Learning



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

- Une forme de l'intelligence artificielle
- Dérivé de l'apprentissage automatique
- Représente un ensemble des méthodes d'apprentissage automatique
- Plusieurs particularités qui permettent de se différencier des algorithmes "classiques"

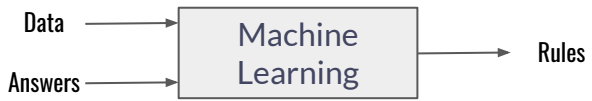
Artificial Intelligence, Machine Learning and Deep Learning

➢ **Artificial Intelligence**

The effort to automate intellectual tasks normally performed by humans.
General field composed of Machine Learning and Deep Learning, but also other fields such as robotics, perception or cognitive systems.

➢ **Machine Learning**

Could a computer go beyond "what we know how to order it to perform" and learn on its own how to perform a specified task ?
Rather than hand-crafting a set of rules, could a computer automatically learn these rules by looking at data ?



A Machine Learning system is Trained rather than explicitly programmed

Learning Representations from data

We need 3 ingredients to do Machine Learning

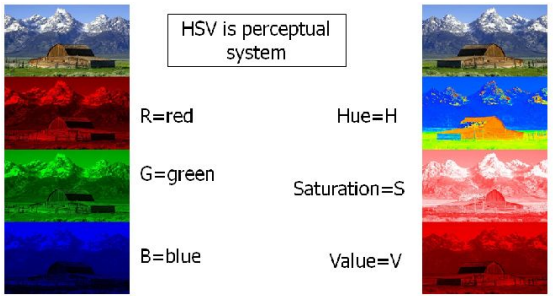
- **Input Data points** : tables, sound files, images, etc.
- **Examples of Expected outputs** : tags for images, transcriptions for speech recognition, etc.
- **A way to measure whether the algorithm is doing a good job**

This measure is used as a feedback signal to adjust the way the algorithms works.
*This adjustment step is called **Learning***

General purpose of a Machine Learning system : **Meaningfully transform data**, to go from the input to the output, by automatically learning **useful representations** of the input data.

Example of representation of the input data

RGB : Useful to filter on color
HSV : Useful to work on saturation

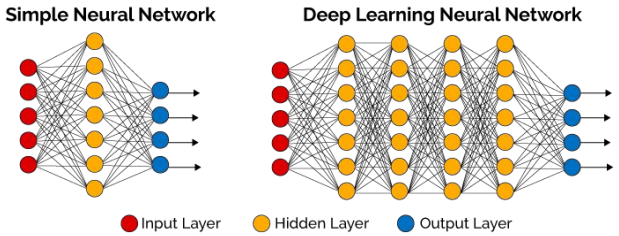


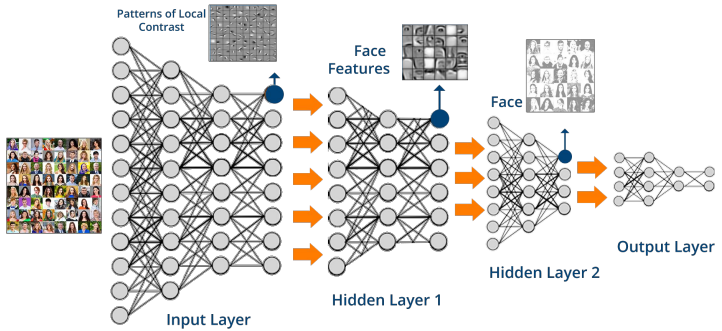
Deep Learning in all this ?

Deep Learning is a subfield of Machine Learning : A specific way of learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations

Other approaches to **Machine Learning** tend to focus on learning only one or two layers of representations of the data -> Shallow Learning

Deep Learning = Deep Sequence of simple data transformations (layers)

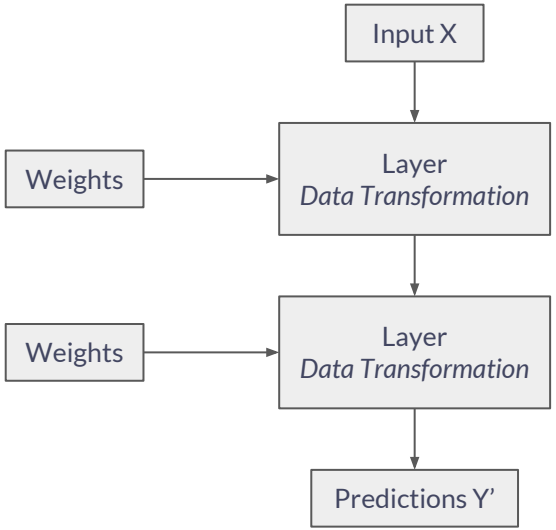




*Layered representations of the data are almost always learned via models called **Neural Networks***

How Deep Learning works ?

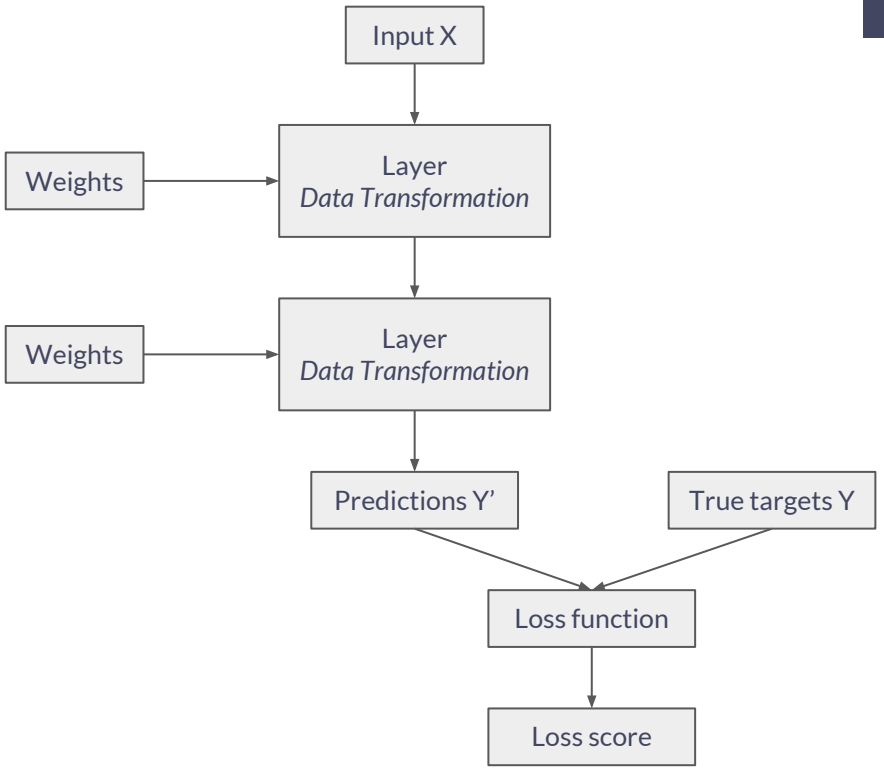
The transformation implemented by a layer is parameterized by its weights.



The Deep Learning workflow

How Deep Learning works ?

The transformation implemented by a layer is parameterized by its weights.



The Deep Learning workflow

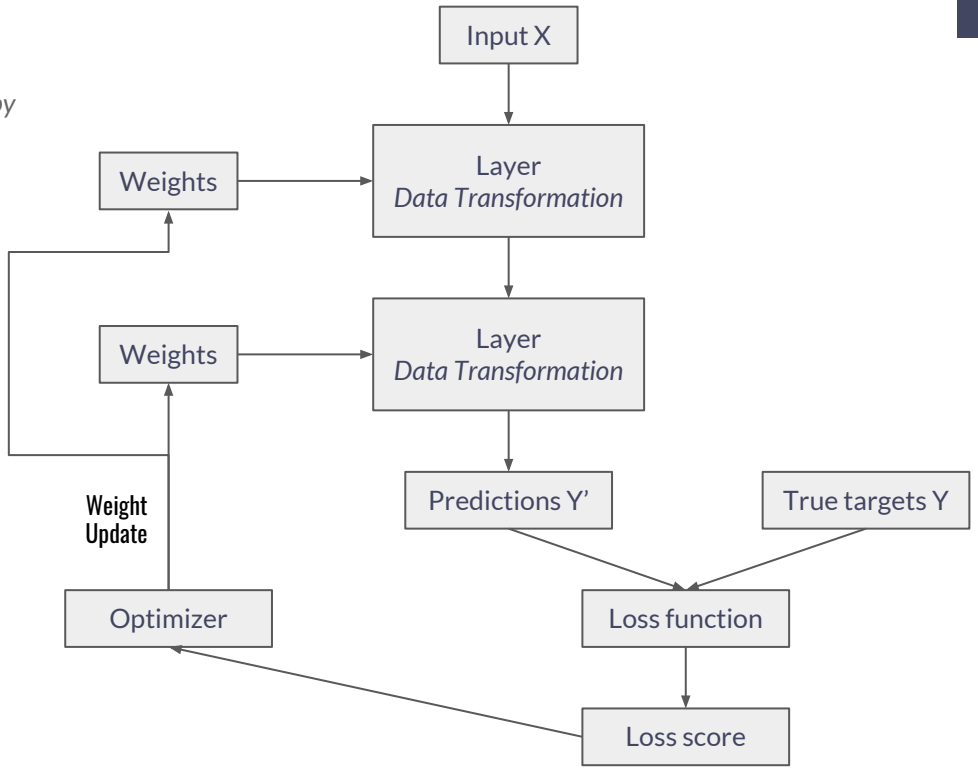
A loss function measures the quality of the network's output.

It computes a distance score between the prediction and the true target

How Deep Learning works ?

The Deep Learning workflow

The transformation implemented by a layer is parameterized by its weights.



Use this score as a feedback signal to adjust the value of the weights a little in order to lower the score for the current example.

This job is done by an optimizer, which implements the Backpropagation algorithm

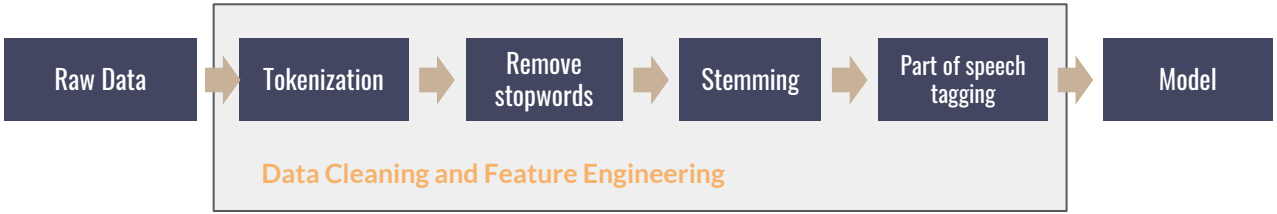
A loss function measures the quality of the network's output.

It computes a distance score between the prediction and the true target

What makes Deep Learning different ?

Deep Learning makes the problem-solving much easier for complex tasks, by completely automating what is the most crucial step in a traditional Machine Learning workflow : Feature Engineering.

Traditional Machine Learning

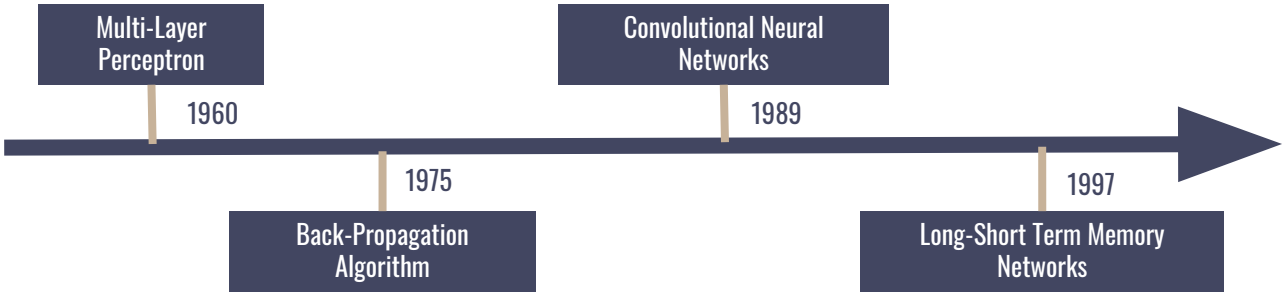


Manually Engineer good layers of representations for the data in order to solve complex problems.

Deep learning

Completely automate the Feature Engineering step : All features are learned in one pass rather than having to engineer them yourself.

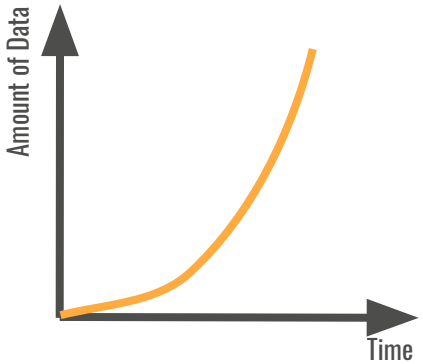
Why now ?

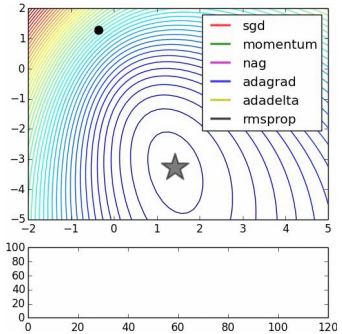


Ideas and main algorithms exist since decades ...

What changed : Hardware, Data and algorithmic advances

GPU





First Look at a Neural Network

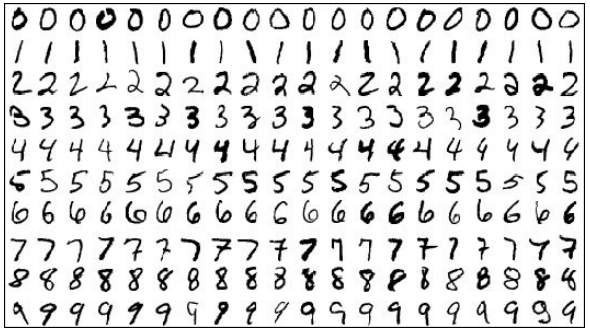
No mathematical formulas, just code :)

Hello World

"Hello World" of Neural Networks : Classifying handwritten digits (MNIST dataset).

Goal : Classify grayscale images of handwritten digits (28x28 pixels) into 10 categories (0 through 9).

Dataset : 60000 images in train set, 10000 images in test set.



MNIST dataset samples

Loading the MNIST dataset

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Training set *Test set*

Network architecture

- The core building block of Neural Networks is the layer : data processing module that acts as a filter for the data.
- The layer extracts representations out of the data, hopefully more meaningful for the problem at hand.
- Deep Learning = Chaining together simple layers in order to learn complex representations

Network Architecture

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

The network above is composed of a sequence of two Dense layers, which are densely connected.

- First layer : Dense hidden layer with 512 neurons
- Second layer : 10-way softmax layer -> *Returns an array of 10 probability scores, one for each class.*

Compilation step

To make the network ready for training, we need to pick three more things :

- A loss function : How to measure performance on the training data ?
- An optimizer : How the network will update itself based on the data it sees and the loss function ?
- One or more metrics : How to monitor the performance during training and testing ?

Compilation step

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

The choice of the optimizer and the loss function will be made clearer later on ...

Prepare the data

The data needs to be preprocessed to meet the network's expectations :

- Correct shape
- Scaled to [0, 1] interval
- Categorically encode the labels

Data types : Tensors

Preparing the image data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Preparing the labels

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

What is a Tensor ?

A Tensor is a container for numbers.

- Scalar -> 0D Tensor
- Vector -> 1D Tensor (numpy vector)
- Matrix -> 2D Tensor
- Pack of matrices -> 3D Tensor

Classical data representations in Deep Learning

- 2D Tensor -> Vector data (*num_samples*, *num_features*)
- 3D Tensor -> Time series or sequence data (*num_samples*, *timesteps*, *num_features*)
- 4D Tensor -> Batch of images (*num_samples*, *width*, *height*, *num_channels*)
- 5D Tensor -> Batch of video data (*num_samples*, *frame*, *width*, *height*, *num_channels*)

Train the network

- Fit the model to its training data, through batches.
- Iterate several times (epochs) through the whole dataset.

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
60000/60000 [==============================] - 2s - loss: 0.2577 - acc: 0.9245
Epoch 2/5
60000/60000 [==============================] - 1s - loss: 0.1042 - acc: 0.9690
Epoch 3/5
60000/60000 [==============================] - 1s - loss: 0.0687 - acc: 0.9793
Epoch 4/5
60000/60000 [==============================] - 1s - loss: 0.0508 - acc: 0.9848
Epoch 5/5
60000/60000 [==============================] - 1s - loss: 0.0382 - acc: 0.9890
```

loss of the network over training data

accuracy of the network over training data

Evaluate on the test set

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

```
 9536/10000 [===========================>..] - ETA: 0s
```

```
print('test_acc:', test_acc)
```

```
test_acc: 0.9777
```

The accuracy is lower on the test set -> This is a sign of overfitting

The engine of Neural Networks : *Gradient-Based optimization*

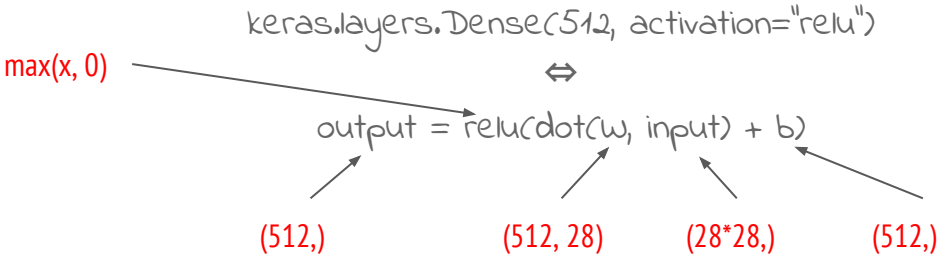
Just a little taste of mathematical intuition

Dive into a layer

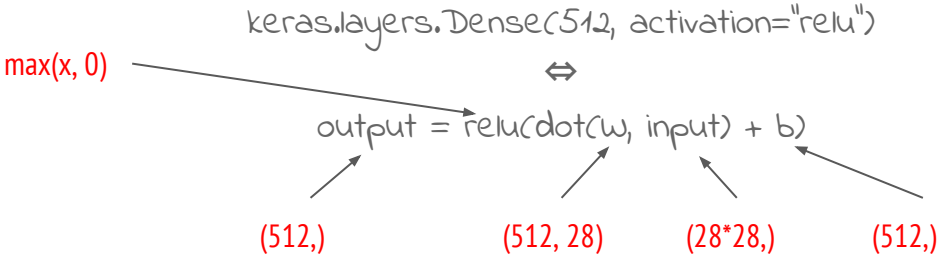
```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

In our example, we were building our network by stacking Dense layers on top of each other.
This layer can be interpreted as a function, which takes as input a 2D Tensor and returns another 2D Tensor (*a new representation for the input tensor*).



Trainable parameters



W and **b** are the weights (or trainable parameters) of the layer.
They contain the information learned by the network from exposure to data.

To get an output prediction from an input, we need to go through the calculations of all the layers, step by step.
This is called inference, or forward pass.

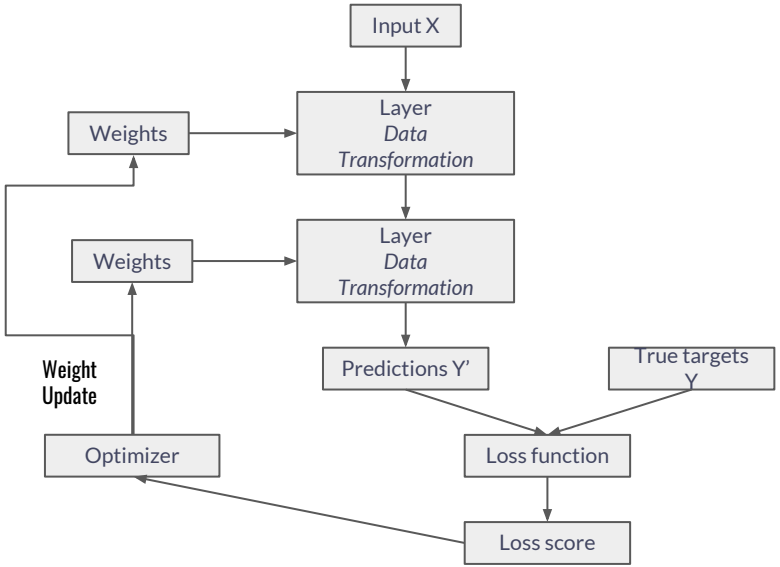
Random Initialization

The weights are initialized with **small random values**.

- At first, there's no chance that those weights would yield to any useful representations.

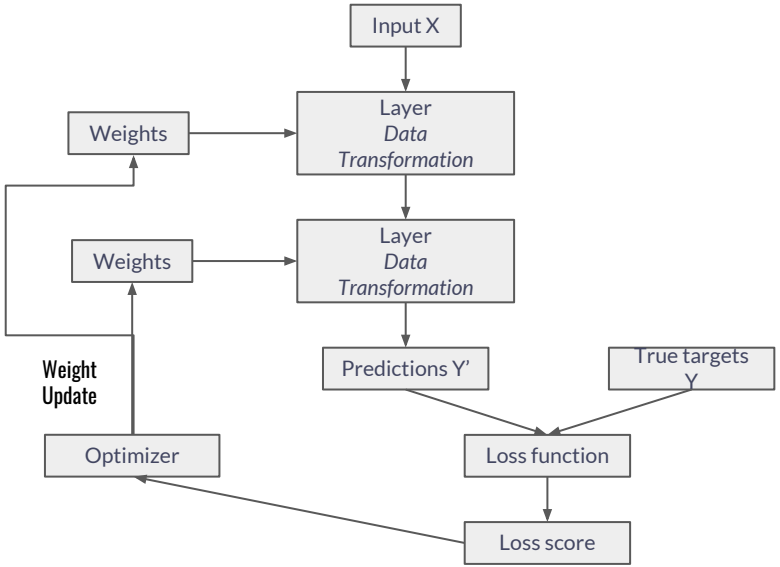
What comes next is to **gradually update the weights**, based on feedback signal.

- This is the training step
- *The "learning" in "Machine Learning"*



Training loop

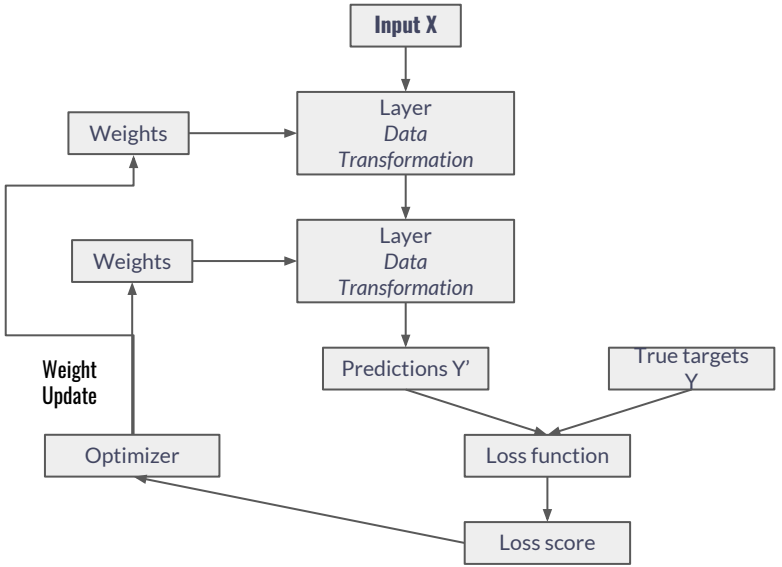
The learning step corresponds to applying the 4 following instructions in a loop :



Training loop

The learning step corresponds to applying the 4 following instructions in a loop :

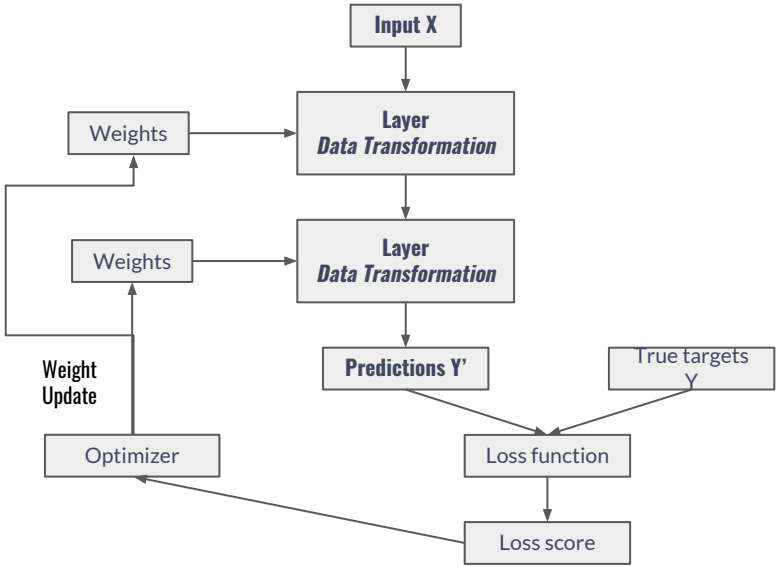
1. Draw a batch of training samples x and corresponding target y



Training loop

The learning step corresponds to applying the 4 following instructions in a loop :

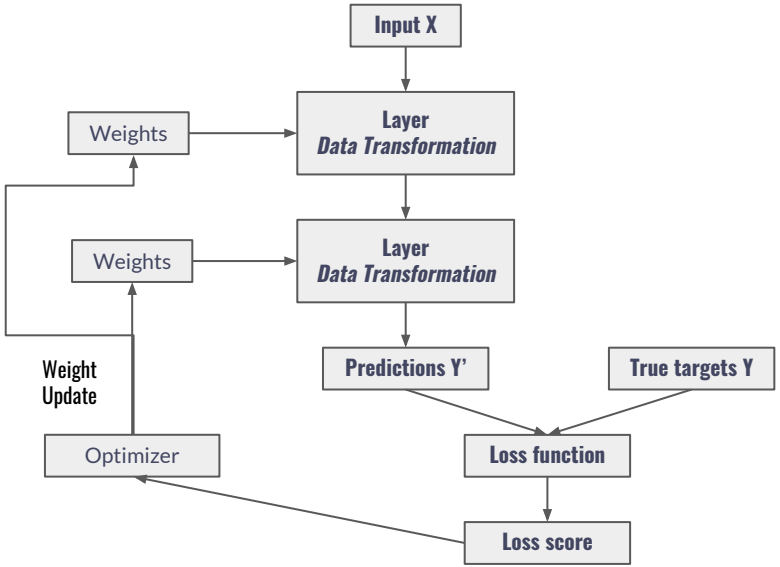
1. Draw a batch of training samples x and corresponding target y
2. Run the network on x (*forward pass*) to obtain predictions y_pred



Training loop

The learning step corresponds to applying the 4 following instructions in a loop :

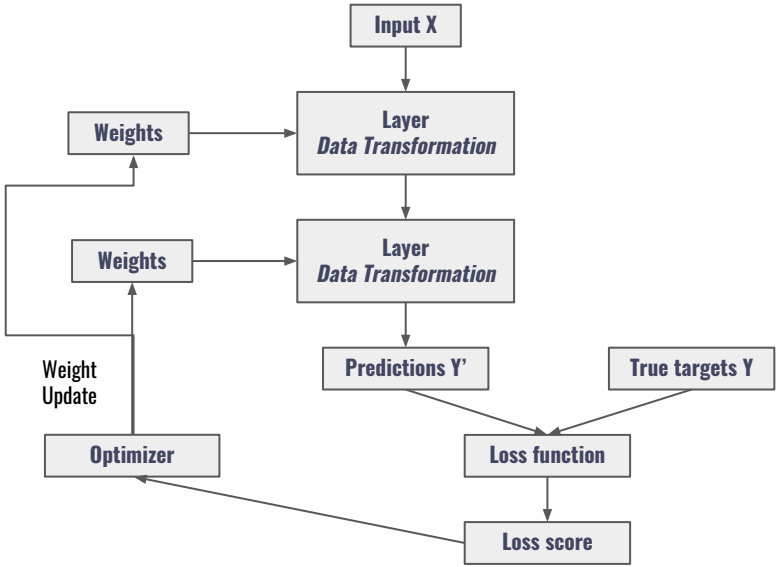
1. Draw a batch of training samples x and corresponding target y
2. Run the network on x (*forward pass*) to obtain predictions y_pred
3. Compute the loss of the network on the batch (measure of mismatch between y and y_pred



Training loop

The learning step corresponds to applying the 4 following instructions in a loop :

1. Draw a batch of training samples x and corresponding target y
2. Run the network on x (*forward pass*) to obtain predictions y_pred
3. Compute the loss of the network on the batch (measure of mismatch between y and y_pred
4. Update all weights of the network in a way that slightly reduces the loss on this batch.



Updating the weights

*Given an individual weight coefficient in the network, how to know if it should be **increased** or **decreased** ?*

Naïve solution :

- Freeze all weights in the network except the one being considered
- Try different values for this coefficient
- Choose the new value that contribute to minimizing the loss
- Repeat this for all coefficients in the network

Problems with this solution

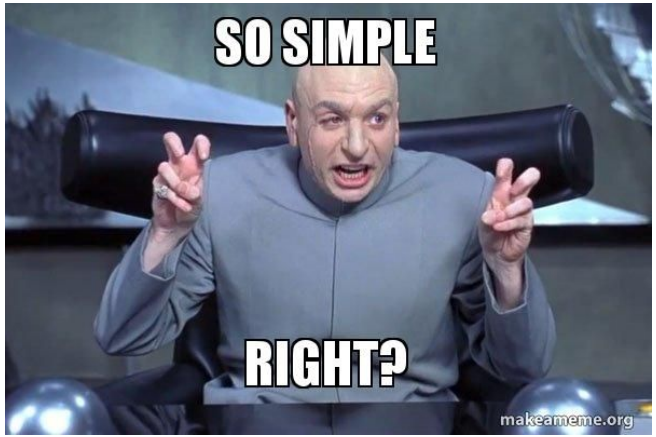
- Horribly inefficient !
- You would need to compute at least two forward passes through all the network for each weight (there can be millions of them)

Updating the weights

*Given an individual weight coefficient in the network, how to know if it should be **increased** or **decreased** ?*

Better approach : Take advantage of the fact that all operations used in the network are differentiable

- Compute the gradient of the loss function with regard to the network's coefficients
- Move the coefficients in the **opposite direction from the gradient**, thus decreasing the loss

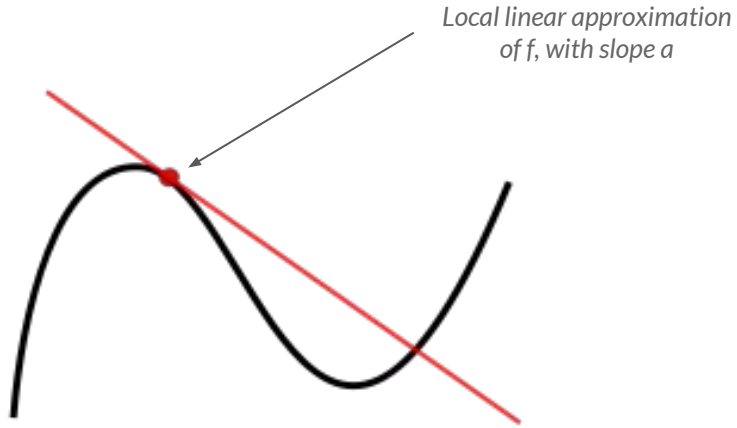


What is a derivative ? A gradient ?

Continuous function -> a small change in x results in a small change in y

Because the function is smooth, it's possible to approximate is as a linear function of slope a

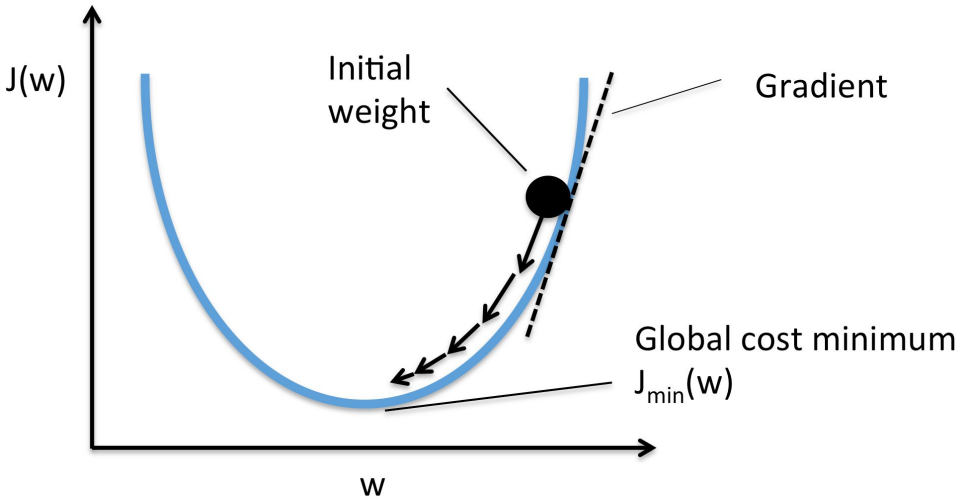
This approximation is only valid when the change is close enough to the original point



You can reduce the value of f(x) by moving x a little in the **opposite direction from the derivative.**

The derivative of a whole tensor is the gradient.
It's the generalization of the concept of **derivatives to functions of multidimensional inputs.**

Mini-Batch Stochastic Gradient Descent



1. Draw a batch of training samples x and corresponding target y
2. Run the network on x (*forward pass*) to obtain predictions y_pred
3. Compute the loss of the network on the batch (measure of mismatch between y and y_pred
4. Compute the gradient of the loss with regards to the network's parameters
5. Move the parameters a little in the opposite direction from the gradient

w = w - step * gradient

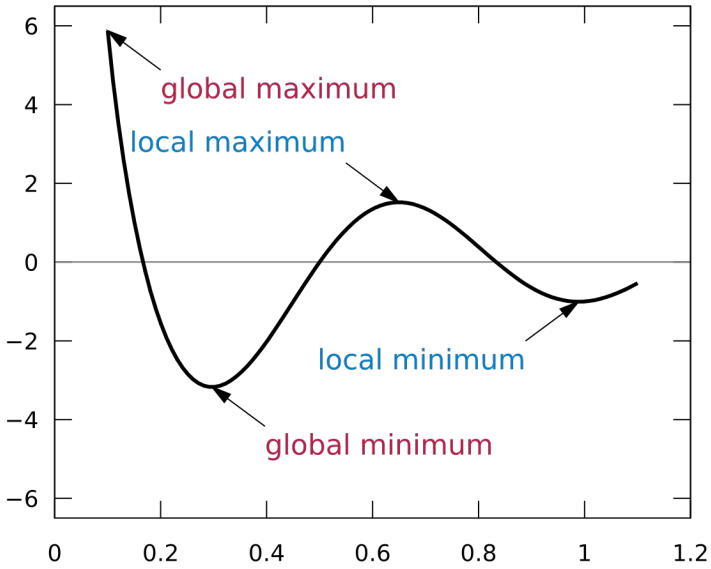
Other optimizers

Other optimizers exist to take into account previous weight updates when computing the next one.

There is for instance SGD with momentum, as well as Adagrad of RMSProp.

The concept of momentum addresses two issues with SGD : convergence speed and local optima.

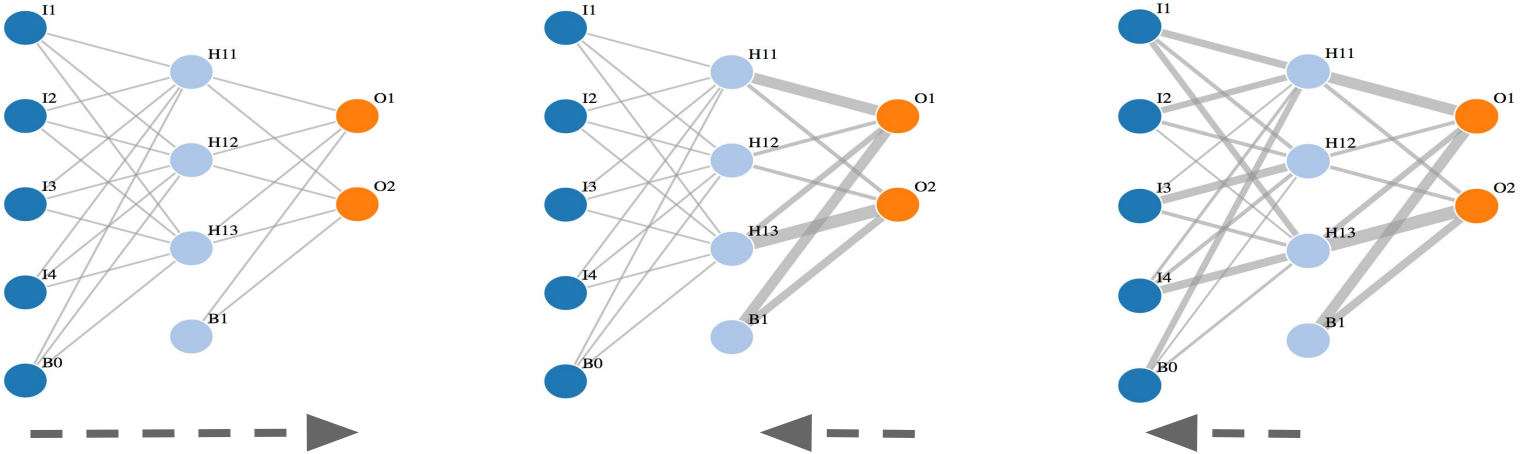
- Inspiration from physics (*ball rolling down a loss curve*)
- With enough momentum, you don't get stuck in a local minima
- Take into account current acceleration but also current velocity



Backpropagation algorithm

In practice, a Neural Network function consists of many tensor operations chained together, each of which has a simple, known derivative.

- Applying the chain rule to the computation of the gradient values of the neural network gives rise of the backpropagation algorithm
- Backpropagation starts with the final loss value and works backwards from the top layers to the bottom layers.



Take away

- Learning means finding a combination of model parameters that minimizes a loss function for a given set of training samples and their corresponding targets
- The learning process is made possible by applying the chain rule of derivation to find the gradient function mapping the current parameters and current batch of data to a gradient value
- The loss is the quantity you'll attempt to minimize during training, so it should represent a measure of success for the task you're trying to solve
- The optimizer specifies the exact way in which the gradient of the loss will be used to update parameters