

Deep Learning par la pratique

Jour 1, après-midi : MLP

Supports de cours



Le github <https://github.com/SkatAI/deeplearning>

- slides
- notebooks
- code
- documents pdf



Google Colab : <https://colab.research.google.com/>

- ou notebooks en local
- mais de préférence avec GPU

Recap

On a vu ce matin

- Rappels :
 - classification / regression,
 - biais / overfitting,
 - métriques, fonction de coût,
 - techniques de régularisation
- Le Perceptron
- Fonctions d'activation, optimiseur
- Descente de Gradient - SGD
- Perceptron Multi Couche avec scikit-learn

Vocabulaire

- Noeuds, Nodes, Neurones : Les unités fonctionnelles des couches
- Layers, Couches : ensemble de noeuds
- **Pré-activations** : Les valeurs des entrées de la couche cachée avant l'application des fonctions d'activation.
- **Activations** : Les valeurs de la couche cachée après l'application des fonctions d'activation.
- Perceptron multicouche (MLP) : réseau de neurones avec au moins une couche cachée.
 - Réseaux de neurones peu profonds (**shallow**) : Réseaux avec 1 seule couche cachée.
 - Réseaux de neurones profonds (**deep**) : Réseaux avec plusieurs couches cachées.
- Réseaux feedforward : Réseaux où les connexions forment un graphe acyclique (sans boucles).
- Réseaux entièrement connectés, dense : Réseaux où chaque élément d'une couche est connecté à tous les éléments de la couche suivante.

Plan

- Le deep learning
- Écosystème et librairies
- Types de réseaux
- Tenseurs
- Cycle de vie d'un modèle
- Backpropagation

émargement

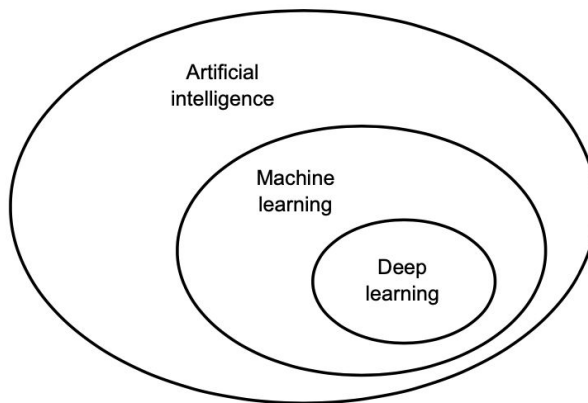
<https://evalcfd.orsys.fr/>

Le deep learning

c'est quoi ?

à partir de quand on est dans du deep learning

- + architecture : deep = beaucoup de couches / au moins 2
- + volume de données important



Le deep learning

on atteint des performances de niveau humain

- images : image classification
- audio : speech to text, text to speech, speech recognition
- texte : traduction, résumé, correction, extraction d'entité, classification
- series temporelles
- IA générative !

Pourquoi ça marche ?

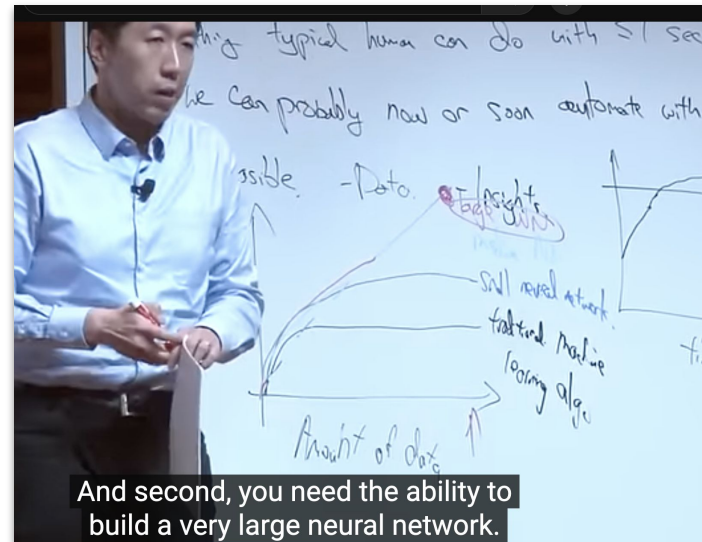
Pourquoi le deep learning

Avec le machine learning classique, on atteint un maximum de performance difficile à dépasser même en rajoutant des données d'entraînement ou avec des modèles plus complexes

Avec le deep learning, la performance continue à augmenter quand on ajoute des données

Andrew Ng

<https://youtu.be/21EiKfQYZXc?t=1224>



Ce qui fait la puissance du deep learning

Comparé au machine learning classique

La combinaison de :

- Décomposition automatisée de la complexité des données de couche en couche.
- Automatisation de la représentation interne des données de plus en plus complexe.
- Ces représentations sont apprises conjointement sous la contrainte de la fonction de coût.
- Apprentissage conjoint de toutes les couches

Francois Chollet - Deep Learning with python p18

d'un point de vue plus math : les réseaux profonds peuvent "apprendre" toute fonction continue par morceau !

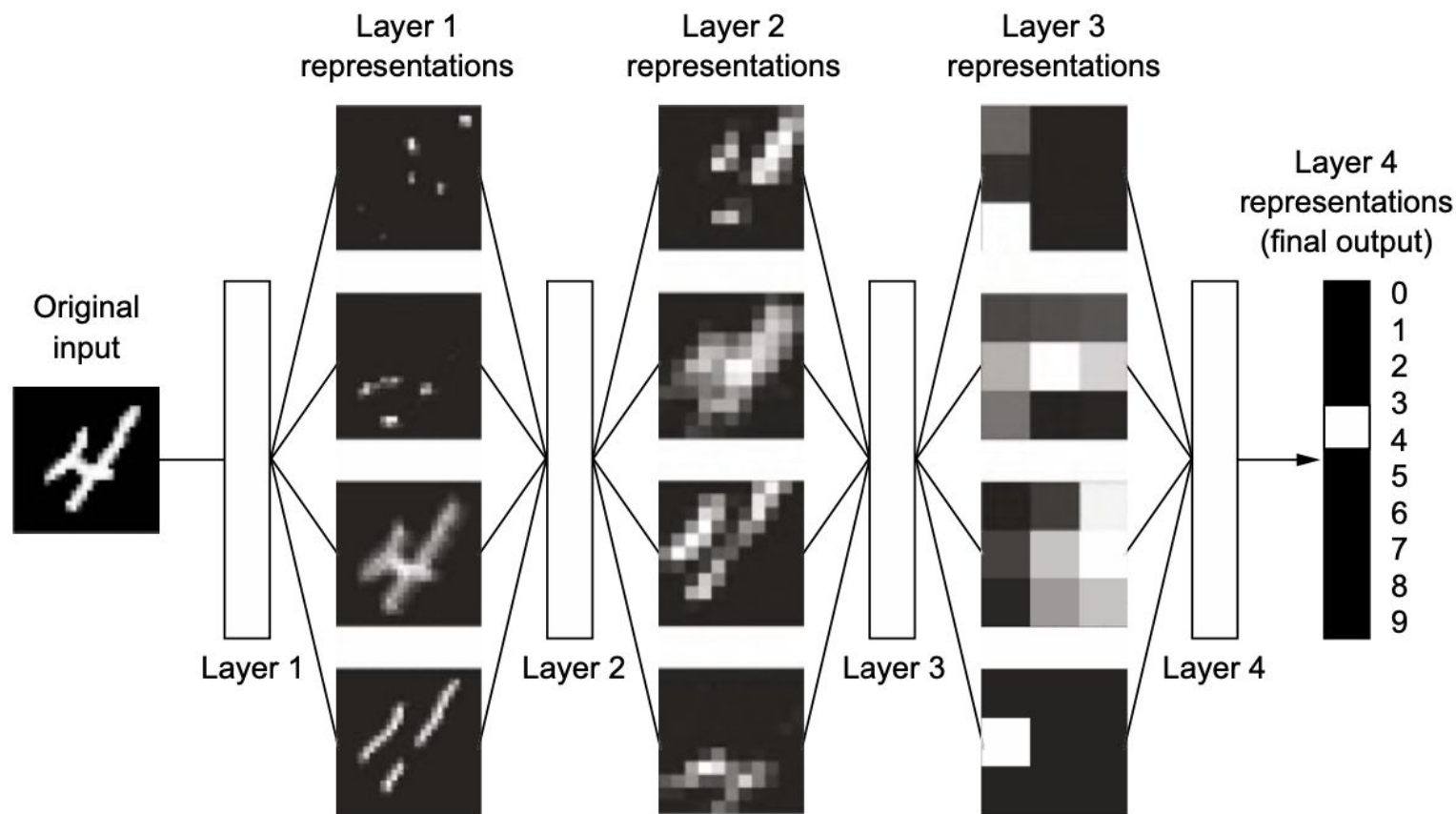


Figure 1.6 Data representations learned by a digit-classification model

Plus besoin de feature engineering

Le réseau trouve lui même la meilleur représentation / transformation des données qui minimise la fonction de coût.

Donc l'étape manuelle de transformation des données basée sur l'expertise domaine du data scientist n'est plus nécessaire

deep network as a multistage *information- distillation* process, where information goes through successive filters and comes out increasingly *purified*

Francois Chollet - Deep Learning with python

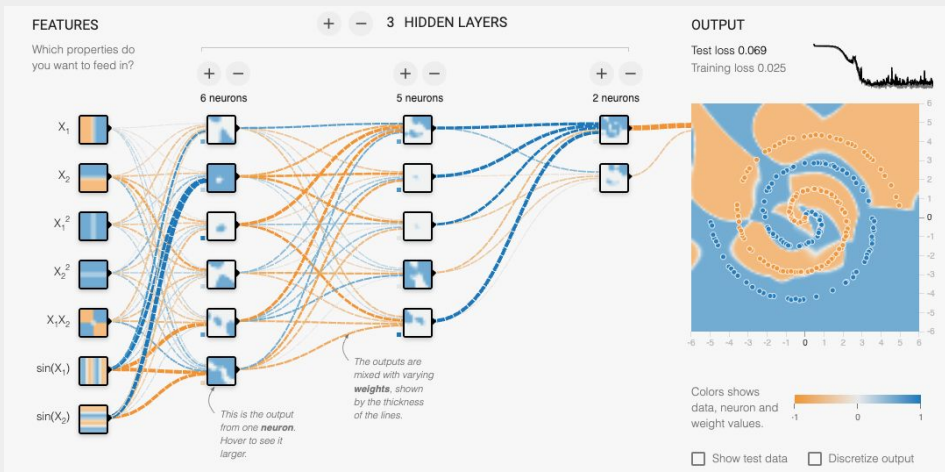
De plus

Francois Chollet - Deep Learning with python p25

- **Simplicité** : Le deep learning supprime le besoin de feature engineering et simplifie les pipelines
- **Évolutivité** : facilement parallélisable sur les GPU/TPU et peut être entraîné sur des ensembles de données de taille arbitraire (batch).
- **Versatilité et réutilisabilité** : Les modèles d'apprentissage profond peuvent être entraînés sur des données supplémentaires sans recommencer l'intégralité de l'entraînement à zéro (transfer learning)

playground

<https://playground.tensorflow.org/>



deep learning au dela de classification / régression




en utilisant les représentations internes

- embeddings => word2vec, LLMs
- transfer learning : on utilise la dernière couche d'un modèle large pour entraîner un modèle spécialisé sur un dataset petit
- détection d'anomalie : autoencoder apprend à reconstruire les données d'entrée et produit des erreurs sur de nouvelles données anormales
- augmentation de données
- extraction des features : pour nourrir d'autres modèles
- style transfer
- traduction : encoder - decoder
- multi modal representation : image to caption

State of the art

<https://paperswithcode.com/sota>

Computer Vision

 Semantic Segmentation t: 307 benchmarks 5317 papers with code	 Image Classification t: 458 benchmarks 3858 papers with code	 Object Detection t: 344 benchmarks 3789 papers with code	 Representation Learning t: 16 benchmarks 3774 papers with code	 Contrastive Learning t: 1 benchmark 2260 papers with code
--	---	---	---	--






► See all 1757 tasks

Natural Language Processing

 Language Modelling t: 64 benchmarks 4691 papers with code	 Transformer 4212 papers with code	 Decoder 3563 papers with code	 Translation t: 3 benchmarks 3224 papers with code	 Question Answering t: 228 benchmarks 2962 papers with code
--	---	---	--	---






► See all 787 tasks

Medical

 Medical Image Segmentation t: 124 benchmarks 779 papers with code	 Style Transfer t: 2 benchmarks 653 papers with code	 EEG 407 papers with code	 Drug Discovery t: 30 benchmarks 389 papers with code	 Property Prediction 238 papers with code
--	--	--	---	--

► See all 309 tasks

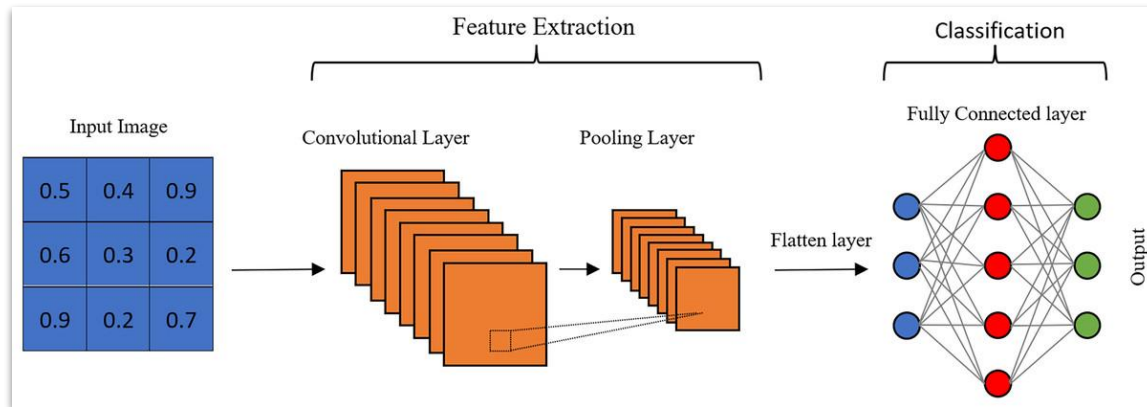
Miscellaneous

 Transfer Learning t: 20 benchmarks 2894 papers with code	 BIG-bench Machine Learning t: 1 benchmark 2320 papers with code	 Benchmarking t: 1 benchmark 1560 papers with code	 Recommendation Systems t: 73 benchmarks 1488 papers with code	 F1 Score t: 1 benchmark 1477 papers with code
---	--	--	--	--

► See all 263 tasks

Type de réseaux

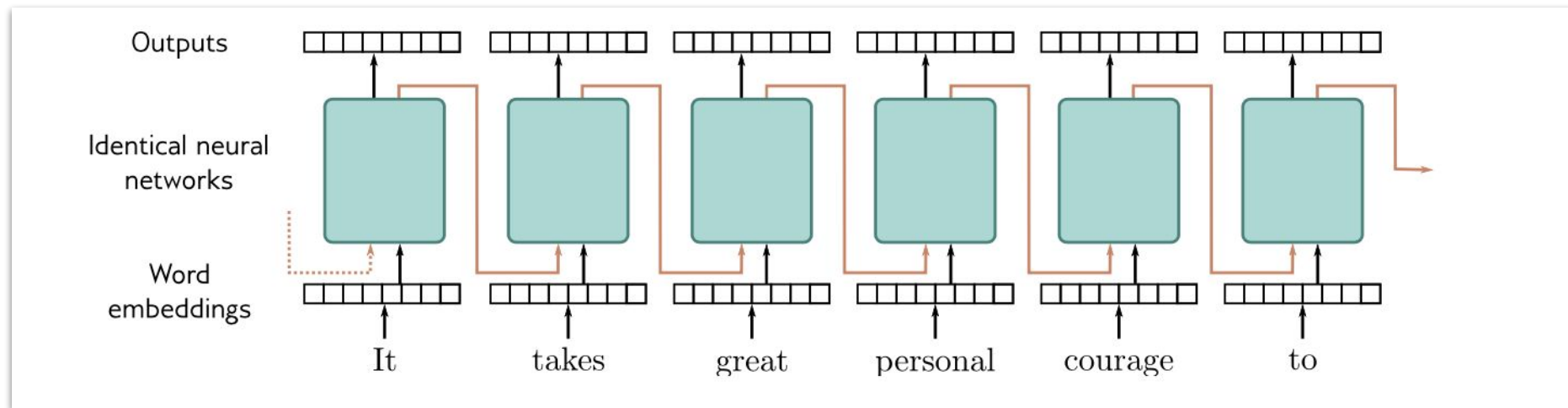
Convolution NN



feedforward networks ou chaque noeud fait une opération de filtrage appelée **convolution**

RNN : Recurrent neural networks

Le principe de base des réseaux de neurones récurrents (RNN) est de traiter des **données séquentielles** en maintenant un état de mémoire interne qui permet à l'information de **persister** et **d'influencer** le traitement des éléments suivants dans la séquence ce qui leur permet de capturer les **dépendances temporelles** dans les données.



A chaque étape, le réseau associe une représentation interne dépendante des étapes précédentes et un nouveau point de donnée

Transformers

utiliser des mécanismes **d'attention** pour traiter des séquences d'entrée et de sortie, en capturant les dépendances et les relations entre les éléments de ces séquences, sans s'appuyer sur une mémoire interne récurrente comme dans les RNN.

Le principe de base des transformers en deep learning est d'utiliser des mécanismes d'auto-attention pour pondérer dynamiquement l'influence des différents tokens d'entrée, permettant ainsi un traitement parallèle et la capture des dépendances à longue portée dans les séquences.

<https://jalammar.github.io/illustrated-transformer/>

Mécanisme d'attention

Chaque élément d'une séquence calcule une "attention", une pondération, par rapport à tous les autres éléments de la séquence,

Voir [docs/attention.md](#) dans le github du cours

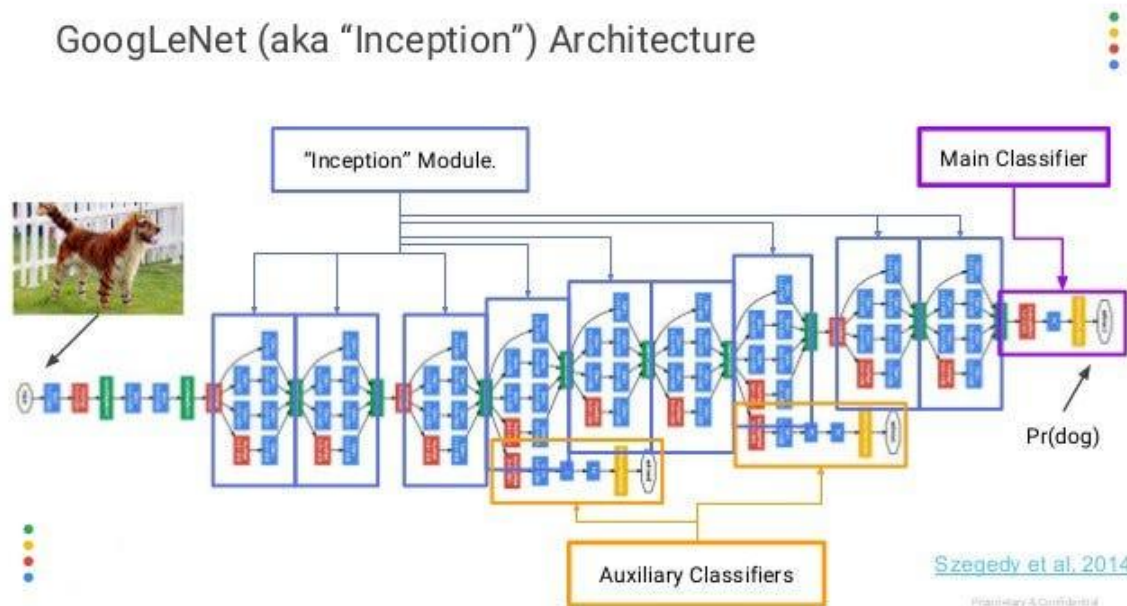
et le papier séminal ***Attention Is All You Need*** <https://arxiv.org/abs/1706.03762>

Réseaux historiques du deep learning

- AlexNet 2012 : CNN, a largement gagné image compétition ILSVRC
- VGGNet 2014 : CNN, a montré que des réseaux plus profonds capturent plus de complexité
- ResNet 2015 : réseaux très profonds, 152 couches
- Inception 2014 :
- Transformers 2017 : self attention, NLP
- GPT-3 2020 : 175 milliards de paramètres
- DALL-E 2021 : créer des images à partir de descriptions textuelles

Inception

GoogLeNet (aka "Inception") Architecture

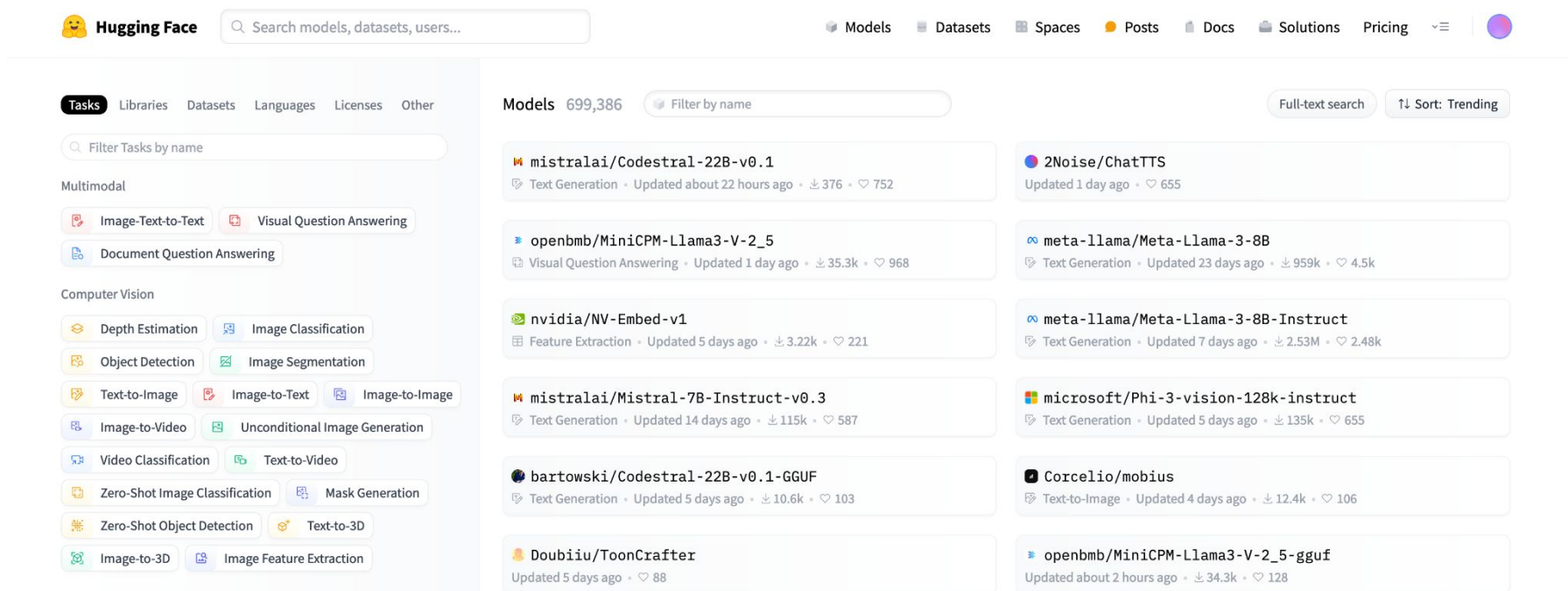


<https://medium.com/@abheerchrome/inception-v1-architecture-explained-454b2eb66baf>

Ecosystème du Deep Learning

Huggingface 700k modèles

<https://huggingface.co/models>



The screenshot shows the Hugging Face website's 'Models' page. The header includes the Hugging Face logo, a search bar, and navigation links for Models, Datasets, Spaces, Posts, Docs, Solutions, and Pricing. The left sidebar features a 'Tasks' section with filters for Multimodal (Image-Text-to-Text, Visual Question Answering, Document Question Answering) and Computer Vision (Depth Estimation, Image Classification, Object Detection, Image Segmentation, Text-to-Image, Image-to-Text, Image-to-Image, Image-to-Video, Unconditional Image Generation, Video Classification, Text-to-Video, Zero-Shot Image Classification, Mask Generation, Zero-Shot Object Detection, Text-to-3D, Image-to-3D, Image Feature Extraction). The main content area displays a list of models, sorted by trending. The first model is 'mistralai/Codestral-22B-v0.1' for Text Generation, updated 22 hours ago with 376 downloads and 752 likes. Other models include '2Noise/ChatTTS', 'openbmb/MiniCPM-Llama3-V-2_5', 'meta-llama/Meta-Llama-3-8B', 'nvidia/NV-Embed-v1', 'meta-llama/Meta-Llama-3-8B-Instruct', 'mistralai/Mistral-7B-Instruct-v0.3', 'microsoft/Phi-3-vision-128k-instruct', 'bartowski/Codestral-22B-v0.1-GGUF', 'Corcelio/mobius', and 'Doubiiu/ToonCrafter'.

Hugging Face Search models, datasets, users...

Models Datasets Spaces Posts Docs Solutions Pricing

Tasks Libraries Datasets Languages Licenses Other

Filter Tasks by name

Multimodal

- Image-Text-to-Text
- Visual Question Answering
- Document Question Answering

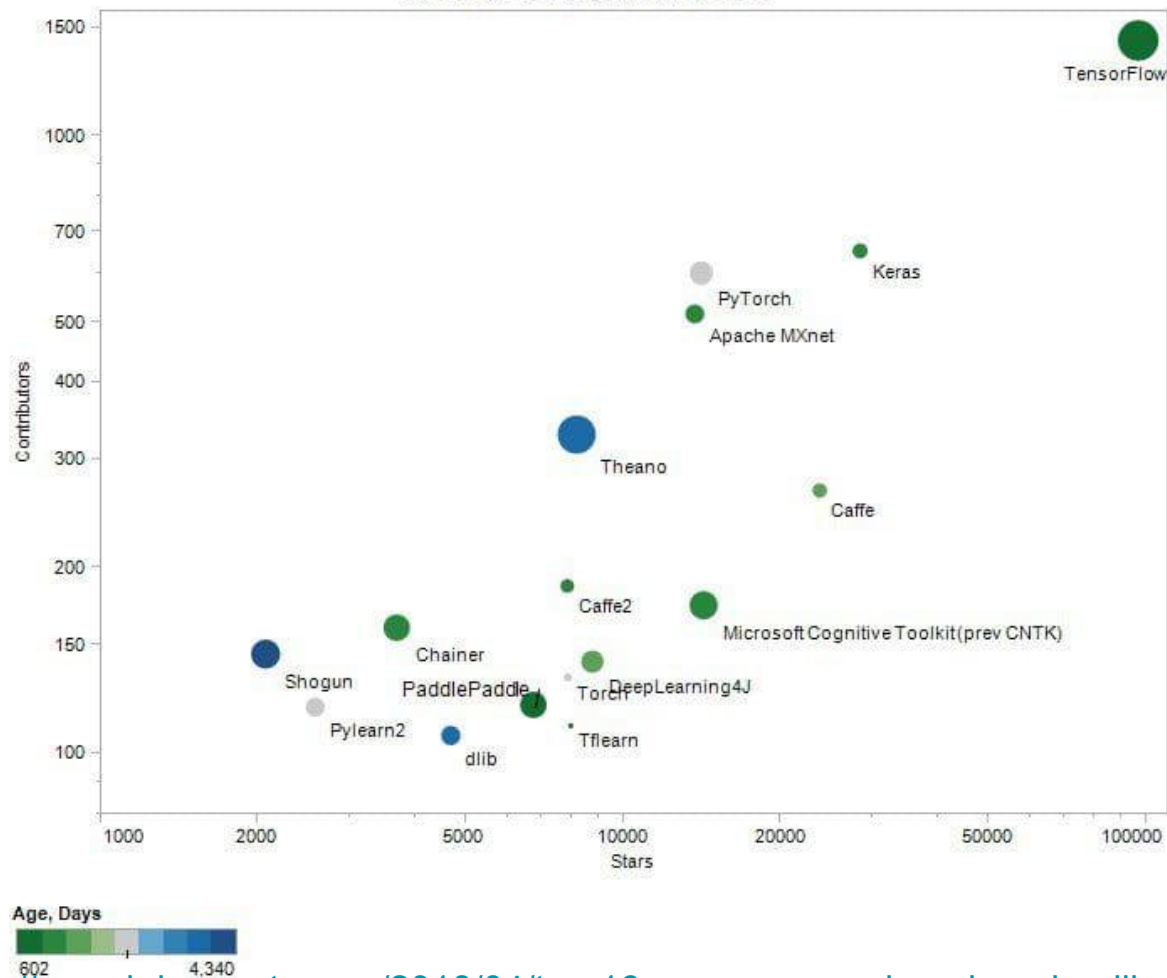
Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Text-to-Image
- Image-to-Text
- Image-to-Image
- Image-to-Video
- Unconditional Image Generation
- Video Classification
- Text-to-Video
- Zero-Shot Image Classification
- Mask Generation
- Zero-Shot Object Detection
- Text-to-3D
- Image-to-3D
- Image Feature Extraction

Models 699,386 Filter by name Full-text search Sort: Trending

- mistralai/Codestral-22B-v0.1**
Text Generation • Updated about 22 hours ago • 376 • 752
- 2Noise/ChatTTS**
Updated 1 day ago • 655
- openbmb/MiniCPM-Llama3-V-2_5**
Visual Question Answering • Updated 1 day ago • 35.3k • 968
- meta-llama/Meta-Llama-3-8B**
Text Generation • Updated 23 days ago • 959k • 4.5k
- meta-llama/Meta-Llama-3-8B-Instruct**
Text Generation • Updated 7 days ago • 2.53M • 2.48k
- mistralai/Mistral-7B-Instruct-v0.3**
Text Generation • Updated 14 days ago • 115k • 587
- microsoft/Phi-3-vision-128k-instruct**
Text Generation • Updated 5 days ago • 135k • 655
- bartowski/Codestral-22B-v0.1-GGUF**
Text Generation • Updated 5 days ago • 10.6k • 103
- Corcelio/mobius**
Text-to-Image • Updated 4 days ago • 12.4k • 106
- Doubiiu/ToonCrafter**
Updated 5 days ago • 88
- openbmb/MiniCPM-Llama3-V-2_5-gguf**
Updated about 2 hours ago • 34.3k • 128

Top Deep Learning Libraries, 2018



TensorFlow
Software

PyTorch
Computer application

Theano
Software

Google JAX
Computer program



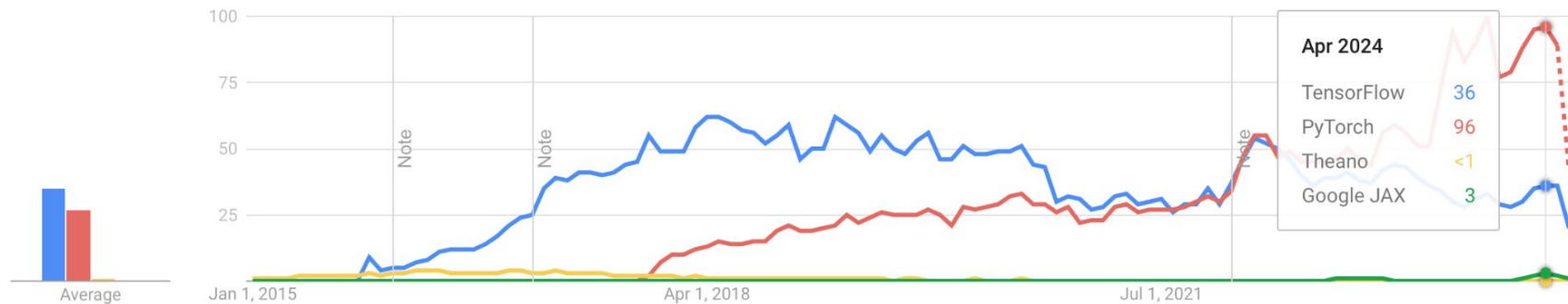
Worldwide ▼

1/1/15 - 6/2/24 ▼

Computers & Electronics ▼

Web Search ▼

Interest over time ?



tensorflow, pytorch, JAX

TensorFlow: 65% (Google 2015) Large-scale production deployment, graph optimization, and a comprehensive ecosystem. production, scalability and optimization

PyTorch: 25% (Meta 2016) Dynamic computation graphs, ease of use, and strong integration with the Python ecosystem. plus académique et recherche

JAX: 10% (Google 2018) High-performance numerical computing, automatic differentiation, and efficient compilation through XLA. research, computation speed

Caffe (Berkeley), Paddle (Baidu)



TensorFlow



PyTorch



Ecosysteme tensorflow



TensorFlow

- Lite : mobile, micro controllers
 - <https://github.com/tensorflow/tflite-micro>
- JS : browser
 - <https://github.com/tensorflow/tfjs>
- Extended (TFX) : déploiement des modèles en production
- model garden : repository de modèles
 - <https://github.com/tensorflow/models/tree/master/official>
 - <https://github.com/tensorflow/tfjs-models>
 - <https://github.com/tensorflow/hub>
- datasets : tf.data
- tensorboard : dashboard de visualisation, debugging
- et beaucoup d'autres <https://www.tensorflow.org/resources/libraries-extensions>

Keras vs Tensorflow Keras

François Chollet, <https://keras.io/> , Keras 3.0

- API simple autour de JAX, pytorch et tensorflow
- `tf.keras` : high level API de tensorflow

Nombreux tutoriaux <https://keras.io/examples/>

Tenseurs

tenseurs

En machine learning classique, on travaille sur des dataframe pandas adaptées aux données tabulaires

En deep learning, on utilise des tenseurs

Structure de représentation de données adapté à tous les types de données

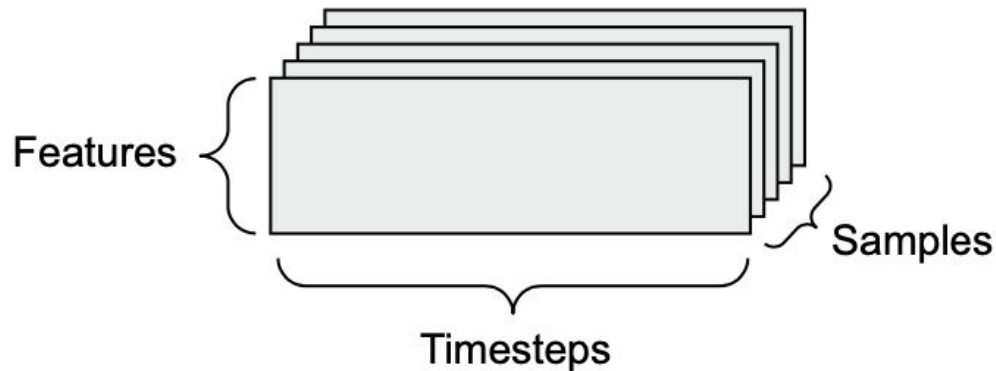
- scalaire - rang 0
- vectoriel - rang 1
- matriciel - rang 2
- mais aussi - rang 3, 4, 5, ...
- 1
- [1,2,3, ...]
- [[1,2], [3,4], [5,6], ...]
- [[[1,2], [3,4], [5,6]], [[7,8], [9,10], [11]], ...]

tenseurs

Pour N échantillons :

- rang 2 :
 - données tabulaires : $N * P$ features samples, dataframe, csv, excel
- rang 3 :
 - images noir et blanc : $N \text{ images} * \text{hauteur} * \text{largeur}$
 - series temporelles : samples, time, value
- rang 4 :
 - images couleurs : $N * \text{hauteur} * \text{largeur} * \text{channel (rgb)}$
- rang 5
 - video : samples, frames, height, width, channels

tenseurs de rang 3 pour série temporelle

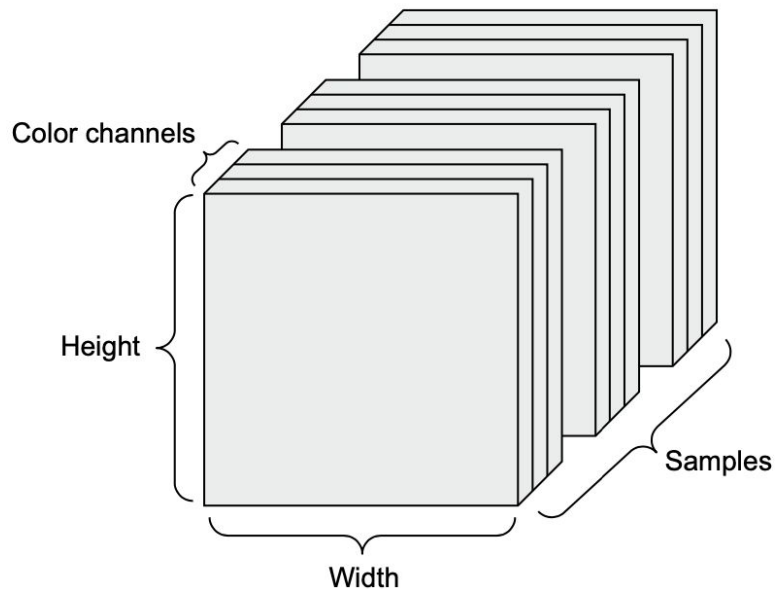


stock prices: 3 indicateurs par minute : $N \text{ jours} * M \text{ minute} / \text{jour} * 3$

tweets : chaque tweet est une séquence de 280 caracteres obtenue à partir d'un alphabet de 128 characters.

tenseurs de rank 4 : images en couleur

samples, height, width, color_depth (RGB)



Example

```
import tensorflow as tf

tensor = tf.constant([[[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]])

print("Shape:", tensor.shape) # Output: (2, 2, 3)
print("Rank:", tensor.ndim)   # Output: 3
print("Dimensions:", tensor.shape.as_list()) # Output: [2, 2, 3]
print("Size:", tf.size(tensor).numpy()) # Output: 12
```

Operations

On peut faire les opérations classiques sur les tenseurs de la même façon que sur les vecteurs ou matrices

<https://www.tensorflow.org/guide/tensor#basics>

pas besoin de tf on peut définir un tenseur avec numpy

```
batch_images = np.random.randint(0, 256, size=(batch_size, height, width), dtype=np.uint8)
```

exemple : Opérations sur 2 tenseurs

```
tf.math.add(tensor1, tensor2),  
tf.math.subtract(tensor1, tensor2),  
tf.math.multiply(tensor1, tensor2),  
tf.math.divide(tensor1, tensor2)  
  
tf.math.square(tensor),  
tf.math.sqrt(tensor),  
tf.math.exp(tensor)
```


exemple : créer un tenseur d'images

```
import tensorflow as tf

# Load grayscale image files

image_files = ['image1.jpg', 'image2.jpg', 'image3.jpg', 'image4.jpg']

batch_images = tf.stack(
    [tf.io.decode_jpeg(tf.io.read_file(file), channels=1) for file in image_files]
)

# Print the shape of the tensor

print("Tensor shape:", batch_images.shape)
```

exemple : Couche *Flatten* de Keras

Dans le MLP, le couche Flatten aplatit le tenseur des données en entrée

```
import tensorflow as tf

# Create a tensor
tensor = tf.constant([[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]])

# Print the original shape
print("Original shape:", tensor.shape)

# Flatten the tensor using tf.keras.layers.Flatten
flatten_layer = tf.keras.layers.Flatten()

flattened_tensor = flatten_layer(tensor)

# Print the flattened shape

print("Flattened shape:", flattened_tensor.shape)
```

- quelques opérations sur les tenseurs
- charger une dataframe
- charger des images et les transformer

https://github.com/SkatAI/deeplearning/blob/master/notebooks/les_tenseurs.ipynb

Pour aller plus loin, le notebook de Tensorflow :

<https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/guide/tensor.ipynb>

Tensorflow - Keras

cycle de vie d'un modèle

Il y a 5 étapes

- **Définir** le modèle : architecture
- **Compiler** le modèle : fonction de coût, optimiseur, métrique d'évaluation
- **Entraîner** le modèle : définir le nombre d'epoch et le batch size
- **Évaluer** le modèle : calculer le score sur dataset de validation
- **Inférence** : faire des prédictions

Définir l'architecture du modèle

```
model = keras.Sequential([  
    keras.layers.Flatten(),  
    keras.layers.Dense(10, activation='relu'),  
    keras.layers.Dense(10, activation='softmax')  
])
```

Compiler le modele

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=[ 'accuracy' ]  
)
```

Keras :

- [Metrics](#)
- [Optimizers](#)

Entraîner le modèle

```
model.fit( x_train, y_train,  
          epochs=5,  
          batch_size=32,  
          validation_data=(x_test, y_test)  
          )
```


évaluer le modèle

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
```

prédiction

```
classifications = model.predict(x_test)
```

voir le modèle

```
model.summary()
```

notebook : Fashion-MNIST avec Keras

le modèle : Sequential

- flatten
- dense 128, activation relu
- dense 10, activation softmax

Observer l'influence :

- de l'architecture
- des paramètres
- du solver
- de la fonction d'activation
- performance, overfit ?

https://github.com/SkatAI/deeplearning/blob/master/notebooks/keras_sequential_fmnist.ipynb

Backpropagation

backprop

nécessaire pour adapter les coefficients des noeuds en fonction de l'erreur et du résultat de la fonction de coût

- pass forward
 - input data est passé de couche en couche de gauche à droite,
 - chaque couche calcule une prédiction, une valeur à travers la fonction d'activation
 - en sortie du réseau, calcul de l'écart entre **prédiction** et la **valeur réelle** avec la fonction de coût
- backward
 - l'erreur est propagée dans chaque noeud de droite à gauche et le gradient de la fonction de coût est calculé pour chaque noeud
 - les coeffs sont maj en utilisant le gradient

math du back prop

Summary

- Error term for output layer:

$$\delta_j = \frac{\partial L}{\partial a_j} \cdot \sigma'(z_j)$$

- Error term for hidden layers:

$$\delta_j = \left(\sum_k \delta_k w_{jk} \right) \cdot \sigma'(z_j)$$

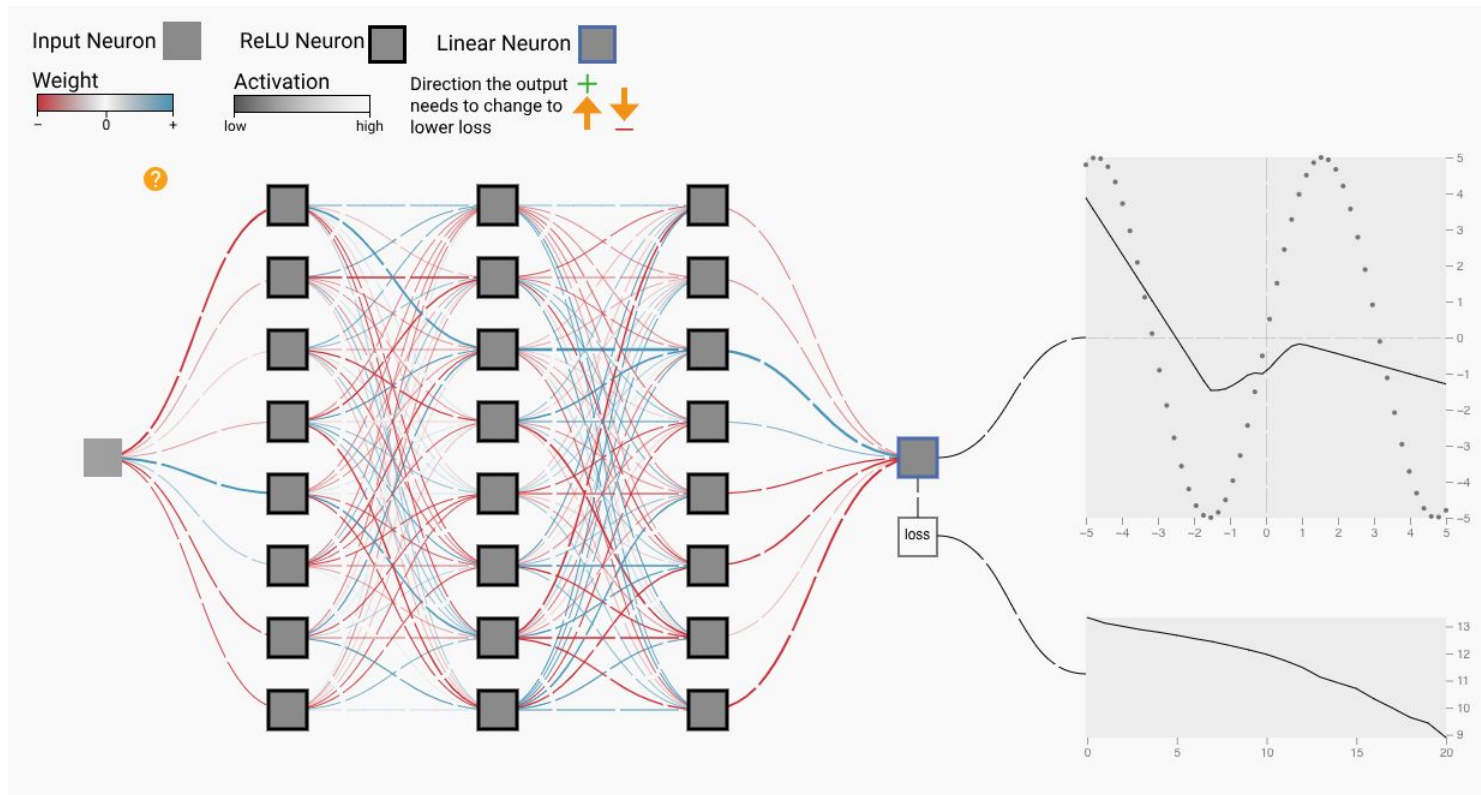
- Gradient with respect to weights:

$$\frac{\partial L}{\partial w_{ij}} = a_i \cdot \delta_j$$

- Gradient with respect to biases:

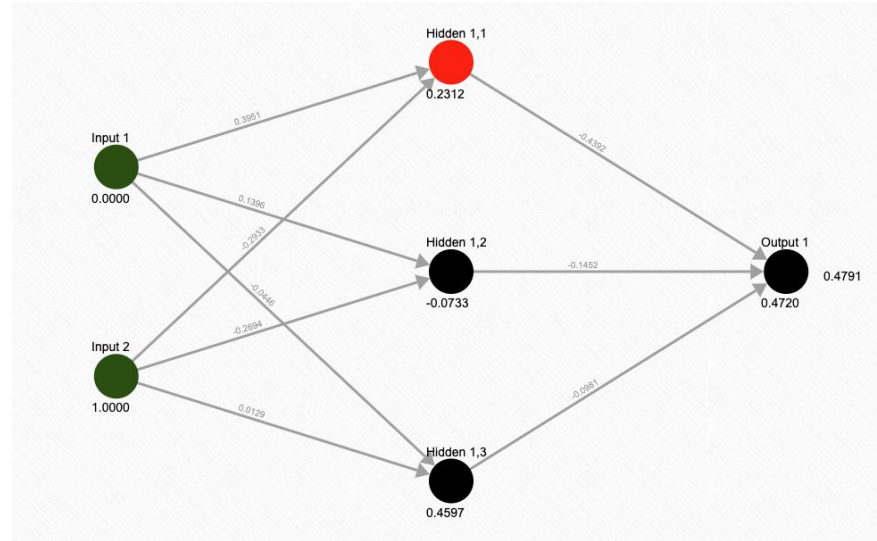
$$\frac{\partial L}{\partial b_j} = \delta_j$$

Visualiser le backprop



Explication interactive du backprop <https://xnought.github.io/backprop-explainer/>

Visualiser le backprop



<http://experiments.mostafa.io/public/ffbpnn/>