Deep Learning - Un cours qui dérive !

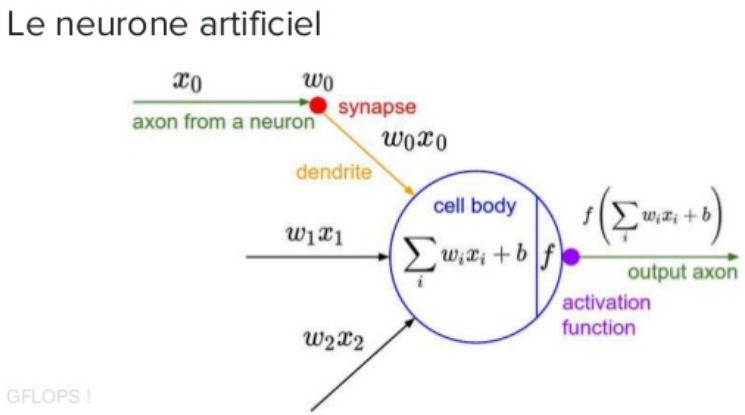
Jérôme Pasquet

December 2, 2019

Le planning

8 Oct : **Perceptron & MLP.**
9 Oct : **Dérivation des fonctions composées et fonctions de perte**
5 Déc : **Optimisation et inertie - Tensorflow**
9 Déc : **Representation de la donnée : l'autoencoder**
10 Déc : **La donnée structurée : convolution pooling et normalisation**
11 Déc : **Regularisation et réseaux convolutifs**
13 Déc : **Partiel**
27 Janv : **Les architectures modernes et *le vanishing gradient***
29 Janv : **La visualisation inversée OU Les réseaux sparses**
30 Janv : **Modèles génératifs : jusqu'au conditionnel ?**

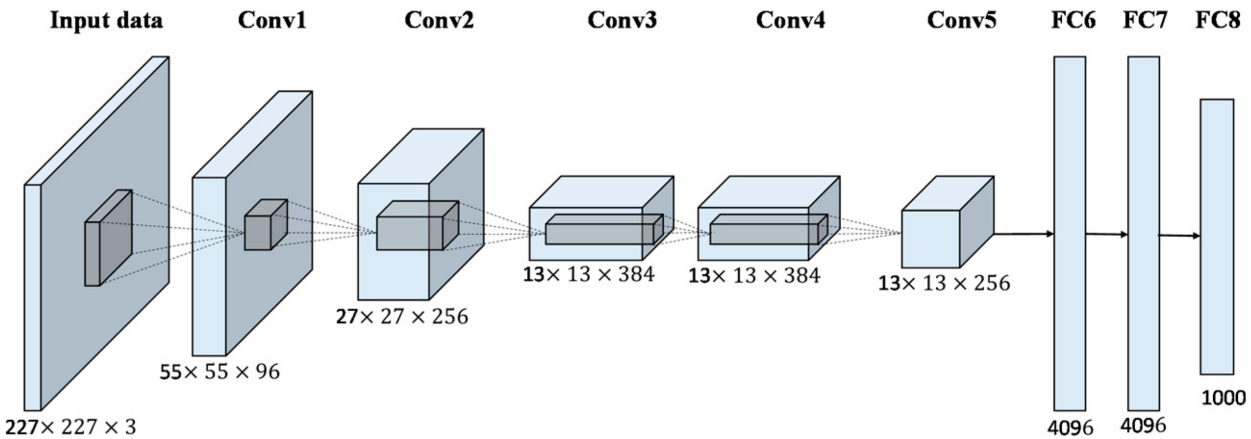
Le neurone



Peut-on classer un vecteur nul ?

Multitude de paramètres

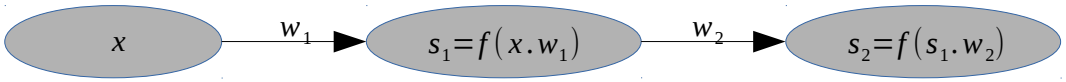
Comment optimiser les paramètres ?
Considérons des paramètre variant dans un espace restreint
$w_i \in \{-2, -1, 0, 1, 2\}$
Un réseaux à 3 paramètres a donc 125 combinaisons.



Alexnet possède : $5^{60000000}$ combinaisons.... contre 10^{82}atomes dans l'univers.

Multitude de paramètres

Comprendre la descente de gradient



Si la vérité connue sur x est y alors on peut définir une fonction de coût comme étant : $J = \frac{1}{2}(s_2 - y)^2$

Objectif : minimiser la fonction de coût

Il y a deux variables : les données et les poids.
→ nous ajustons les poids

$$J = \frac{1}{2}(f(f(w_1 . x) . w_2) - y)^2$$

Minimiser itérativement la fonction de coût via ses dérivées partielles $\frac{\delta J}{\delta w_i}$.

$$w_i = w_i - \alpha . \frac{\delta J}{\delta w_i}$$

N'échappent pas au problème du sur-apprentissage.

Théorème de dérivation des fonctions composées

Voir le cours de Michel Fournié.
http://www.math.univ-toulouse.fr/~fournie/WEB-IUT/tm212.pdf

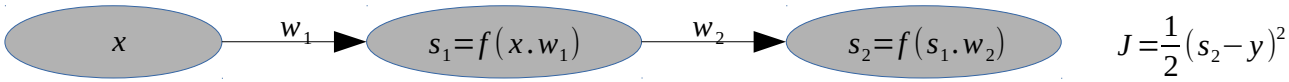
$$\frac{\delta x}{\delta y} = \frac{\delta x}{\delta k} \cdot \frac{\delta k}{\delta y}$$

$$(g \circ f)' = f'.(g' \circ f)$$

Apprentissage par descente de gradient

Au tableau !

Exercice 1 : rétropropagation en milieu simple

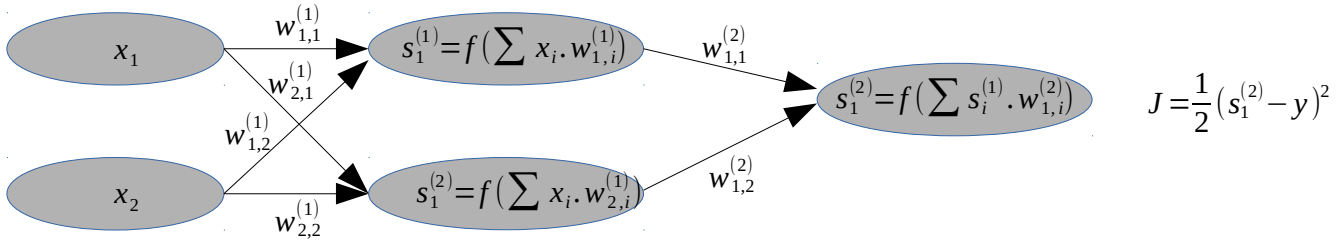


Nous considérerons une fonction f(x) = tanh(x)
- Calculez la dérivée de tanh
- Quel avantage celle-ci présente pour notre application ?

Minimiser itérativement le modèle avec les valeurs de x et de y suivantes :

y	x
0	*-1.2*
1	*0.3*
0	*-0.1*
1	*0.0*
0	*-0.4*
1	*0.7*

Exercice 2 : Maîtrise du gradient



Minimiser itérativement le modèle avec les valeurs de x_1, x_2 et de y suivantes :

y	x_1	x_2
0	*-1*	*-1*
1	*-1*	*1*
0	*1*	*1*
1	*1*	*-1*

Le modèle sera initialisé avec les valeurs suivantes :

$$w^{(1)}_{1,1} = 0.5 \qquad w^{(2)}_{1,1} = 0.2$$
$$w^{(1)}_{1,2} = -0.3 \qquad w^{(2)}_{1,2} = -0.1$$
$$w^{(1)}_{2,1} = 0.7$$
$$w^{(1)}_{2,2} = -0.1$$

Les fonctions de transfert

La fonction d'activation doit être non linéaire [*sinon équivalent à une seule couche*] et différentiable [*sinon le gradient n'est pas calculable*].
Preuve :

$$s = (x.w_1 + b_1).w_2 + b_2$$
$$s = x.\underbrace{w_1 w_2}_{w'} + \underbrace{b_1 w_2 + b_2}_{b'}$$

De nombreuses fonctions d'activation existent comme la **ReLU** **(**$max(0, x)$**)**, **la sigmoid** **(**$\frac{1}{1+e^{-x}}$**)** ou la fonction **PReLU** **(**$\begin{cases} x & si\ x > 0 \\ \alpha.x & sinon \end{cases}$**)**.
Pour chacune de ces fonctions, calculons leurs dérivées et déduisons quelques propriétés.

Théorème d'approximation universelle

Theorem 2. *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \qquad \text{for all} \quad x \in I_n.$$

DM : Lisez le papier suivant : Approximation by Superpositions of a Sigmoidal Function - G. Cybenkot

$\Rightarrow$ D'après vous, pourquoi faire plusieurs couches ?

Théorème d'approximation universelle

Theorem 2. *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \qquad \text{for all} \quad x \in I_n.$$

DM : Lisez le papier suivant : Approximation by Superpositions of a Sigmoidal Function - G. Cybenkot

$\Rightarrow$ D'après vous, pourquoi faire plusieurs couches ?

Fonctions de perte

La fonction de perte (*loss function*) permet de calculer la différence entre une connaissance de notre modèle et la sortie prédite par le modèle appris.

Le choix de la fonction est-il motivé ou arbitraire?

Considérons un problème de regression avec des données d'entrées indépendantes et identiquement distribuées et des labels répartis selon une loi gaussienne. Quelle fonction de perte vous semble la plus adaptée ?

Considérons un problème de classification

La fonction $\sigma_i^{(Softmax)} = \frac{e^{a_i}}{\sum_z^Z e^{a_z}}$ permet de représenter une loi de probabilité sur Z entrées.

- Calcul de ladérivée : $\frac{\delta \sigma_i^{(softmax)}}{\delta a_j}$
- Insérer la dans le calcul d'une entropie croisée : $L = -\sum_i y_i log(\sigma_i)$ avec $\sigma_i \in [0..1] \ \forall i$.
- Considérez maintenant une fonction de *perte softmax cross entropy*.

Apprentissage par paquet

Pour une base contenant n éléments (x_k, t_k)

Rétropropagation online

$$w_i = w_i - \alpha . \frac{\delta J(x_k, t_k)}{\delta w_i}$$

Rétropropagation stochastique

$$w_i = w_i - \alpha . \frac{\sum_k^n \delta J(x_k, t_k)}{\delta w_i}$$

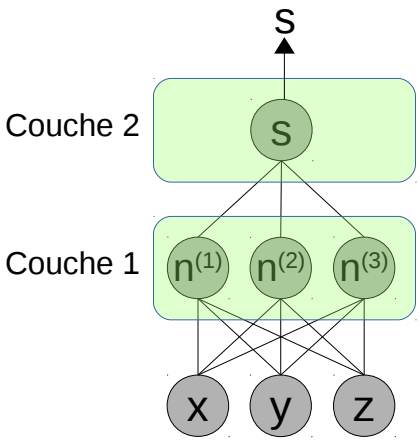
Rétropropagation stochastique par paquets

$$w_i = w_i - \alpha . \frac{\sum_k^{k+n_b} \delta J(x_k, t_k)}{\delta w_i}$$

Avec n_b la taille des paquets

Representation sous tensorflow

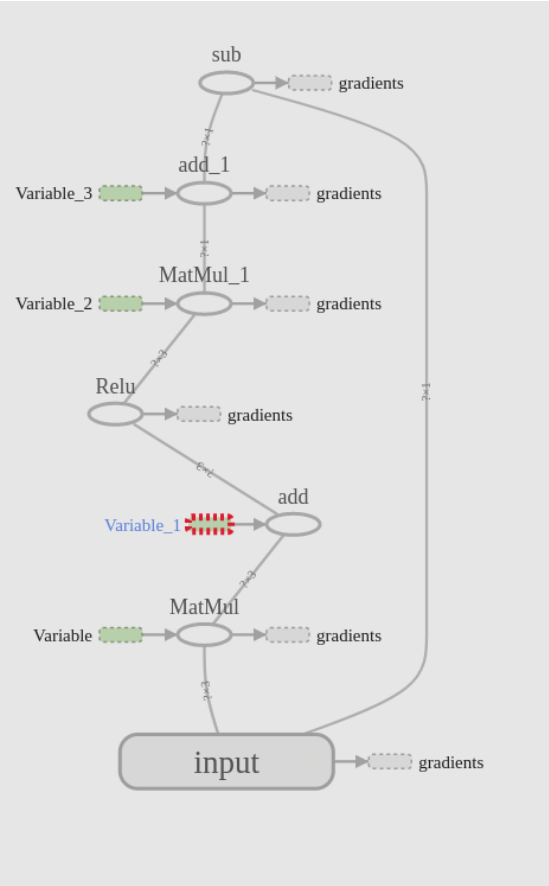
Représentation usuelle



Représentation formelle

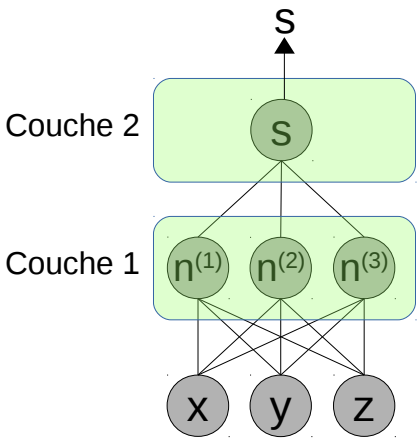
$$n^{(1)}(x,y,z) = w_1^{(1)}.x + w_2^{(1)}.y + w_3^{(1)}.z$$
$$n^{(2)}(x,y,z) = w_1^{(2)}.x + w_2^{(2)}.y + w_3^{(2)}.z$$
$$n^{(3)}(x,y,z) = w_1^{(3)}.x + w_2^{(3)}.y + w_3^{(3)}.z$$
$$s(x,y,z) = w_1^{(s)}.n^{(1)} + w_2^{(s)}.n^{(2)} + w_3^{(s)}.n^{(3)}$$

Représentation tensorflow (simplifiée)



Representation sous tensorflow

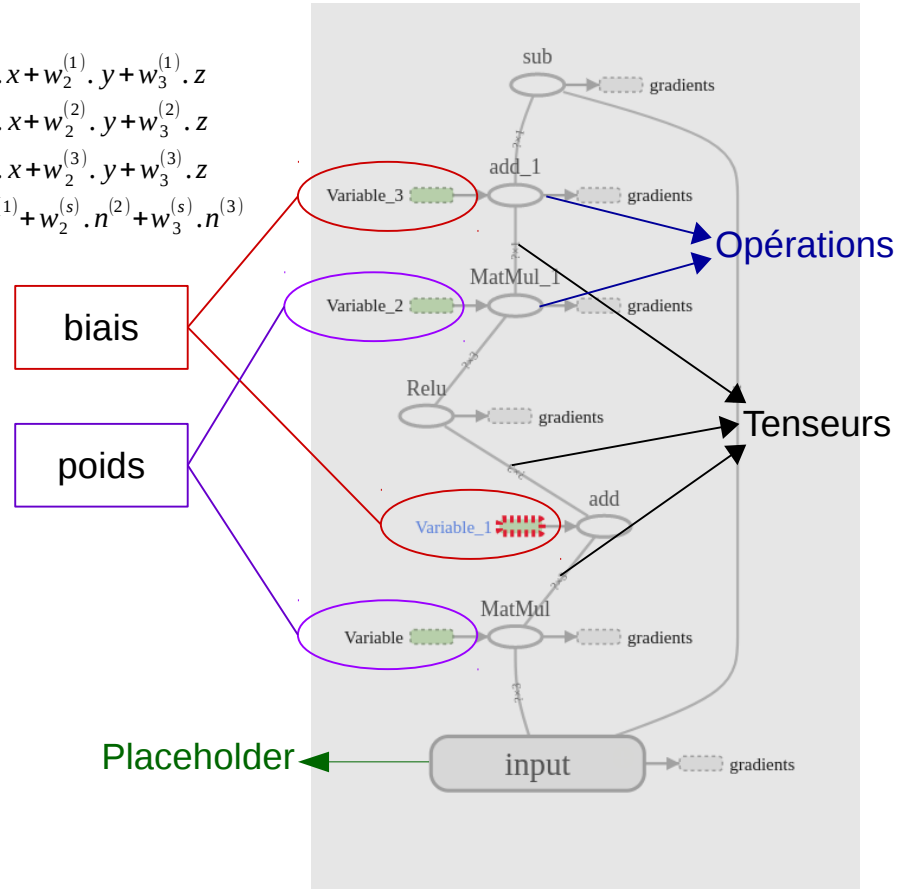
Représentation usuelle



Représentation formelle

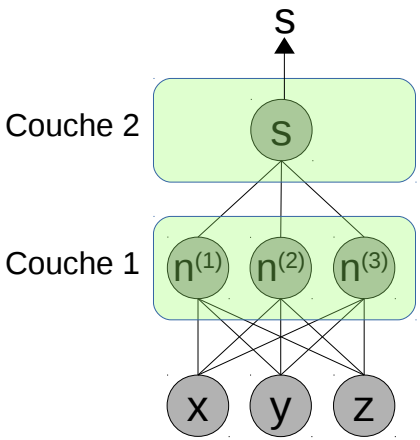
$$n^{(1)}(x,y,z) = w_1^{(1)}.x + w_2^{(1)}.y + w_3^{(1)}.z$$
$$n^{(2)}(x,y,z) = w_1^{(2)}.x + w_2^{(2)}.y + w_3^{(2)}.z$$
$$n^{(3)}(x,y,z) = w_1^{(3)}.x + w_2^{(3)}.y + w_3^{(3)}.z$$
$$s(x,y,z) = w_1^{(s)}.n^{(1)} + w_2^{(s)}.n^{(2)} + w_3^{(s)}.n^{(3)}$$

Représentation tensorflow (simplifiée)



Representation sous tensorflow

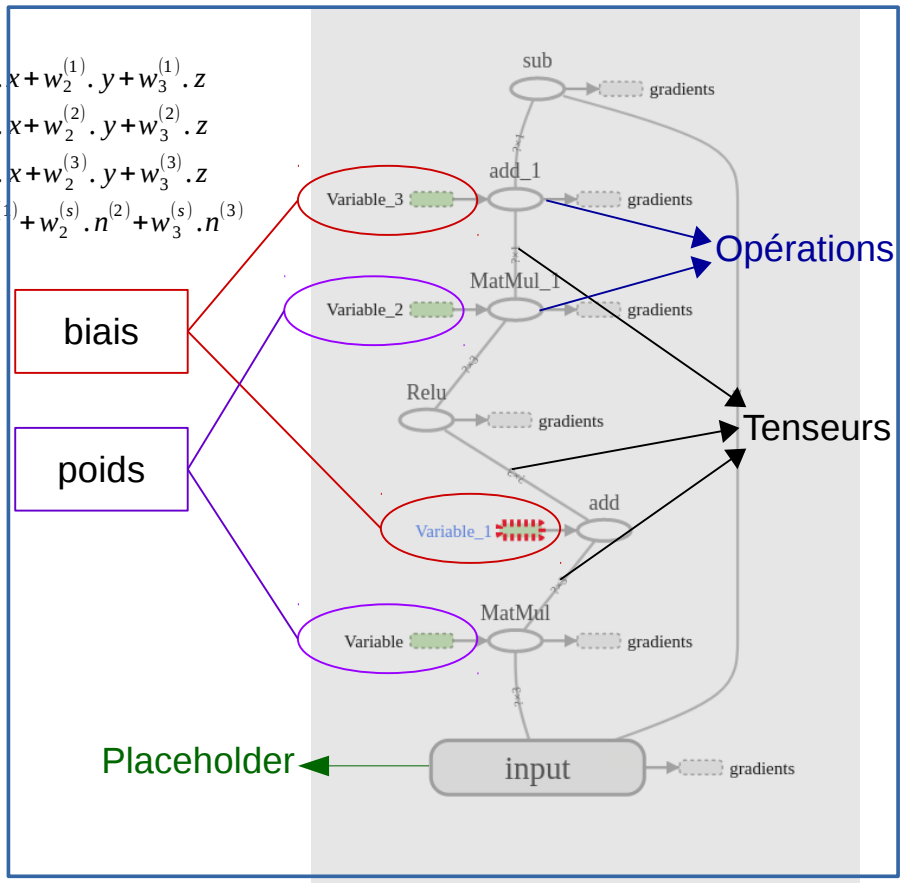
Représentation usuelle



Représentation formelle

$$n^{(1)}(x,y,z) = w_1^{(1)}.x + w_2^{(1)}.y + w_3^{(1)}.z$$
$$n^{(2)}(x,y,z) = w_1^{(2)}.x + w_2^{(2)}.y + w_3^{(2)}.z$$
$$n^{(3)}(x,y,z) = w_1^{(3)}.x + w_2^{(3)}.y + w_3^{(3)}.z$$
$$s(x,y,z) = w_1^{(s)}.n^{(1)} + w_2^{(s)}.n^{(2)} + w_3^{(s)}.n^{(3)}$$

Représentation tensorflow (simplifiée)



Installation de tensorflow

https://www.tensorflow.org/install/source_windows
https://www.tensorflow.org/install/source

Trop lent ?
===> https://colab.research.google.com

Quelques objets importants

data = tf.placeholder(type=type, shape=shape, name=name)

- **type** : tf.int32, tf.float32, tf.uint16, tf.float64, tf.string ...
- **shape** : La taille du tenseur
- **name** : Le nom du tensor dans le graph de tensorflow.

Vous devez comprendre ce qu'est un placeholder ! Si ce n'est pas le cas posez une question maintenant !

Quelques objets importants

data = tf.placeholder(type=type, shape=shape, name=name)

- **type** : tf.int32, tf.float32, tf.uint16, tf.float64, tf.string ...
- **shape** : La taille du tenseur
- **name** : Le nom du tensor dans le graph de tensorflow.

Vous devez comprendre ce qu'est un placeholder ! Si ce n'est pas le cas posez une question maintenant !

Quelques objets importants

v = tf.Variable(initial_value=initial_value, trainable=trainable, name=name)

- **initial_value**: définit l'initialisation de la variable
- **trainable**: Définit si la variable est optimisable
- **name** : Le nom du tensor dans le graph de tensorflow.

- **tf.truncated_normal(shape, mean, stddev, dtype, name)**
- **tf.constant(value, dtype, shape, name)**

Quel est la différence entre une variable et un placeholder ?

Quelques objets importants

v = tf.Variable(initial_value=initial_value, trainable=trainable, name=name)

- **initial_value**: définit l'initialisation de la variable
- **trainable**: Définit si la variable est optimisable
- **name** : Le nom du tensor dans le graph de tensorflow.

- tf.truncated_normal(shape, mean, stddev, dtype, name)
- tf.constant(value, dtype, shape, name)

Quel est la différence entre une variable et un placeholder ?

Quelques objets importants

- **tf.matmul(A, B)** : multiplication de la matrice A par B
- **tf.nn.tanh** : fonction tangente hyperbolique
- **tf.nn.relu** : fonction ReLU
- **optimizer= tf.train.GradientDescentOptimizer(0.01)** : Construit un objet (**optimizer**) de type SGD avec un **learning rate** de 0.01
- **train = optimizer.minimize(cout)** : Applique l'optimiseur à la fonction de coût (**cout**)
- **sess = tf.Session()** : Construit un objet de type session
- **sess.run(tf.initialize_all_variables())** : Initialise toutes les variables dans le graphe tensorflow.