Deep Learning

Jérôme Pasquet

February 3, 2020

Play !

https://playground.tensorflow.org/

Les fonctions de transfert

	f	f'	Avantages
ReLU	$max(0, x)$	$\begin{cases} 1 & x > 0 \\ 0 & sinon \end{cases}$	
Sigmoid	$\frac{1}{1+e^{-x}}$	$f(1-f)$	
PReLU	$\begin{cases} x & x > 0 \\ \alpha x & sinon \end{cases}$	$\begin{cases} 1 & x > 0 \\ \alpha & sinon \end{cases}$	

Comment est-mis à jour le paramètre α du PReLU?

Lire : Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification - Kaiming He et al.

Considérons un problème de classification

La fonction $\sigma_i^{(Softmax)} = \frac{e^{a_i}}{\sum_{z=0}^{Z} e^{a_z}}$ permet de représenter une loi de probabilité sur Z entrées.

- Calcul de la dérivée : $\frac{\delta\sigma_i^{(softmax)}}{\delta a_j}$
- 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*.

Finissons la démonstration

- Calcul de la dérivée : $\frac{\delta \sigma_i^{(softmax)}}{\delta a_j}$

Finissons la démonstration

- Calcul de la dérivée : $\frac{\delta \sigma_i^{(softmax)}}{\delta a_j}$

$$\frac{\delta \sigma_i^{(softmax)}}{\delta a_j} = \frac{\delta \frac{e^{a_i}}{\sum e^{a_k}}}{\delta a_j} = \begin{cases} \sigma_i^{(softmax)}(1 - \sigma_i^{(softmax)}) & i = j \\ -\sigma_i^{(softmax)} \sigma_j^{(softmax)} & sinon \end{cases}$$

Finissons la démonstration

- Calcul de la dérivée : $\frac{\delta \sigma_i^{(softmax)}}{\delta a_j}$

$$\frac{\delta \sigma_i^{(softmax)}}{\delta a_j} = \frac{\delta \frac{e^{a_i}}{\sum e^{a_k}}}{\delta a_j} = \begin{cases} \sigma_i^{(softmax)}(1 - \sigma_i^{(softmax)}) & i = j \\ -\sigma_i^{(softmax)} \sigma_j^{(softmax)} & sinon \end{cases}$$

- Calcul de la dérivée : $\mathscr{L} = -\sum y_i log(\sigma_i)$

Finissons la démonstration

- Calcul de la dérivée : $\frac{\delta \sigma_i^{(softmax)}}{\delta a_j}$

$$\frac{\delta \sigma_i^{(softmax)}}{\delta a_j} = \frac{\delta \frac{e^{a_i}}{\sum e^{a_k}}}{\delta a_j} = \begin{cases} \sigma_i^{(softmax)}(1 - \sigma_i^{(softmax)}) & i = j \\ -\sigma_i^{(softmax)} \sigma_j^{(softmax)} & sinon \end{cases}$$

- Calcul de la dérivée : $\mathscr{L} = -\sum y_i log(\sigma_i)$

$$\frac{\delta \mathscr{L}}{\delta a_i} = \frac{-\delta(\sum_k y_k log(\sigma_k))}{\delta a_i} = -\sum_k y_k \frac{\delta log(\sigma_k)}{\delta \sigma_k} \frac{\delta \sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \frac{\delta \sigma_k}{\delta a_i}$$

Finissons la démonstration

- Considérons la fonction σ comme étant une fonction softmax.

$$\frac{\delta \mathscr{L}}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \frac{\delta \sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \begin{cases} \sigma_i^{(softmax)}(1 - \sigma_i^{(softmax)}) & i = k \\ -\sigma_i^{(softmax)} \sigma_k^{(softmax)} & sinon \end{cases}$$

Finissons la démonstration

- Considérons la fonction σ comme étant une fonction softmax.

$$\frac{\delta \mathscr{L}}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \frac{\delta \sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \begin{cases} \sigma_i^{(softmax)}(1 - \sigma_i^{(softmax)}) & i = k \\ -\sigma_i^{(softmax)} \sigma_k^{(softmax)} & sinon \end{cases}$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1 - \sigma_i^{(softmax)}) - \sum_{k \neq i} y_k \frac{1}{\sigma_k} (-\sigma_i^{(softmax)} \sigma_k^{(softmax)})$$

Finissons la démonstration

- Considérons la fonction σ comme étant une fonction softmax.

$$\frac{\delta \mathscr{L}}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \frac{\delta \sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \begin{cases} \sigma_i^{(softmax)}(1-\sigma_i^{(softmax)}) & i = k \\ -\sigma_i^{(softmax)} \sigma_k^{(softmax)} & sinon \end{cases}$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1-\sigma_i^{(softmax)}) - \sum_{k \neq i} y_k \frac{1}{\sigma_k} (-\sigma_i^{(softmax)} \sigma_k^{(softmax)})$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1-\sigma_i^{(softmax)}) + \sum_{k \neq i} y_k \sigma_i$$

Finissons la démonstration

- Considérons la fonction σ comme étant une fonction softmax.

$$\frac{\delta\mathscr{L}}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k}\frac{\delta\sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k}\begin{cases}\sigma_i^{(softmax)}(1-\sigma_i^{(softmax)}) & i=k\\ -\sigma_i^{(softmax)}\sigma_k^{(softmax)} & sinon\end{cases}$$

$$\frac{\delta\mathscr{L}}{\delta a_i} = -y_i(1-\sigma_i^{(softmax)}) - \sum_{k\neq i} y_k \frac{1}{\sigma_k}(-\sigma_i^{(softmax)}\sigma_k^{(softmax)})$$

$$\frac{\delta\mathscr{L}}{\delta a_i} = -y_i(1-\sigma_i^{(softmax)}) + \sum_{k\neq i} y_k\sigma_i$$

$$\frac{\delta\mathscr{L}}{\delta a_i} = -y_i + \sigma_i(y_i + \sum_{k\neq i} y_k)$$

Finissons la démonstration

- Considérons la fonction σ comme étant une fonction softmax.

$$\frac{\delta \mathscr{L}}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \frac{\delta \sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k} \begin{cases} \sigma_i^{(softmax)}(1 - \sigma_i^{(softmax)}) & i = k \\ -\sigma_i^{(softmax)} \sigma_k^{(softmax)} & sinon \end{cases}$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1 - \sigma_i^{(softmax)}) - \sum_{k \neq i} y_k \frac{1}{\sigma_k} (-\sigma_i^{(softmax)} \sigma_k^{(softmax)})$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1 - \sigma_i^{(softmax)}) + \sum_{k \neq i} y_k \sigma_i$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i + \sigma_i(y_i + \sum_{k \neq i} y_k)$$

Conclusion ?

Finissons la démonstration

- Considérons la fonction σ comme étant une fonction softmax.

$$\frac{\delta \mathscr{L}}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k}\frac{\delta \sigma_k}{\delta a_i} = -\sum_k y_k \frac{1}{\sigma_k}\begin{cases} \sigma_i^{(softmax)}(1-\sigma_i^{(softmax)}) & i=k \\ -\sigma_i^{(softmax)}\sigma_k^{(softmax)} & sinon \end{cases}$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1-\sigma_i^{(softmax)}) - \sum_{k\neq i} y_k \frac{1}{\sigma_k}(-\sigma_i^{(softmax)}\sigma_k^{(softmax)})$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i(1-\sigma_i^{(softmax)}) + \sum_{k\neq i} y_k \sigma_i$$

$$\frac{\delta \mathscr{L}}{\delta a_i} = -y_i + \sigma_i(y_i + \sum_{k\neq i} y_k)$$

Conclusion ?
Multiclasses ?

Apprentissage par paquets

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

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

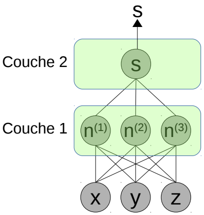
Rétropropagation stochastique par paquets

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

Avec n_b la taille des paquets

Representation sous tensorflow

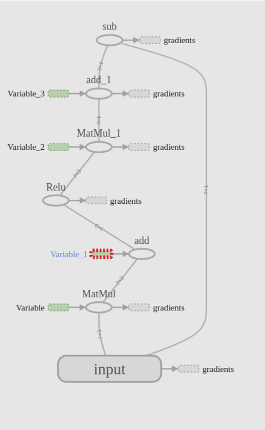
Représentation usuelle



Représentation formelle

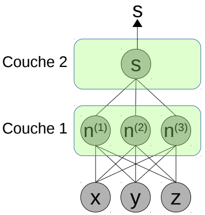
$$n^{(1)}(x,y,z)=w_1^{(1)}.x+w_2^{(1)}.y+w_3^{(1)}.z$$
$$n^{(2)}(x,y,z)=w_1^{(2)}.x+w_2^{(2)}.y+w_3^{(2)}.z$$
$$n^{(3)}(x,y,z)=w_1^{(3)}.x+w_2^{(3)}.y+w_3^{(3)}.z$$
$$s(x,y,z)=w_1^{(s)}.n^{(1)}+w_2^{(s)}.n^{(2)}+w_3^{(s)}.n^{(3)}$$

Représentation tensorflow (simplifiée)



Representation sous tensorflow

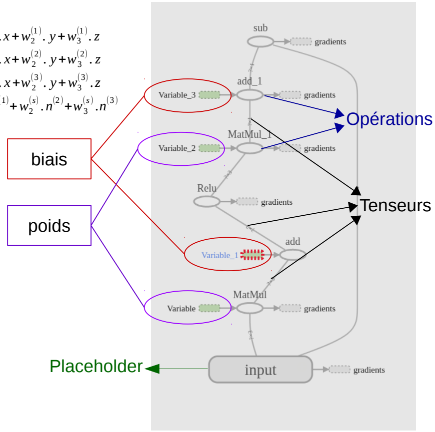
Représentation usuelle



Représentation formelle

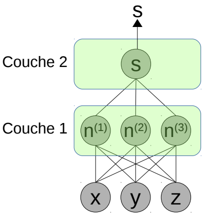
$$n^{(1)}(x,y,z)=w_1^{(1)}.x+w_2^{(1)}.y+w_3^{(1)}.z$$
$$n^{(2)}(x,y,z)=w_1^{(2)}.x+w_2^{(2)}.y+w_3^{(2)}.z$$
$$n^{(3)}(x,y,z)=w_1^{(3)}.x+w_2^{(3)}.y+w_3^{(3)}.z$$
$$s(x,y,z)=w_1^{(s)}.n^{(1)}+w_2^{(s)}.n^{(2)}+w_3^{(s)}.n^{(3)}$$

Représentation tensorflow (simplifiée)



Representation sous tensorflow

Représentation usuelle

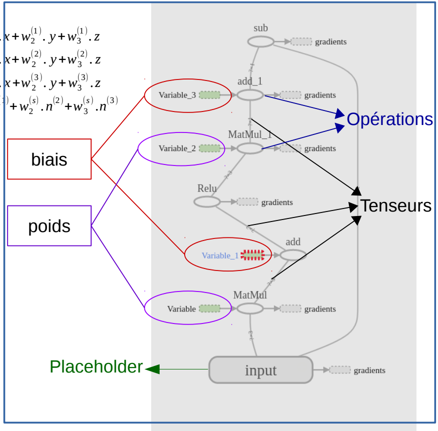


Représentation formelle

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

Session

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.

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)

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)
Quelle 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.