Réseaux de neurones
(2 couches, 1 couche cachée)

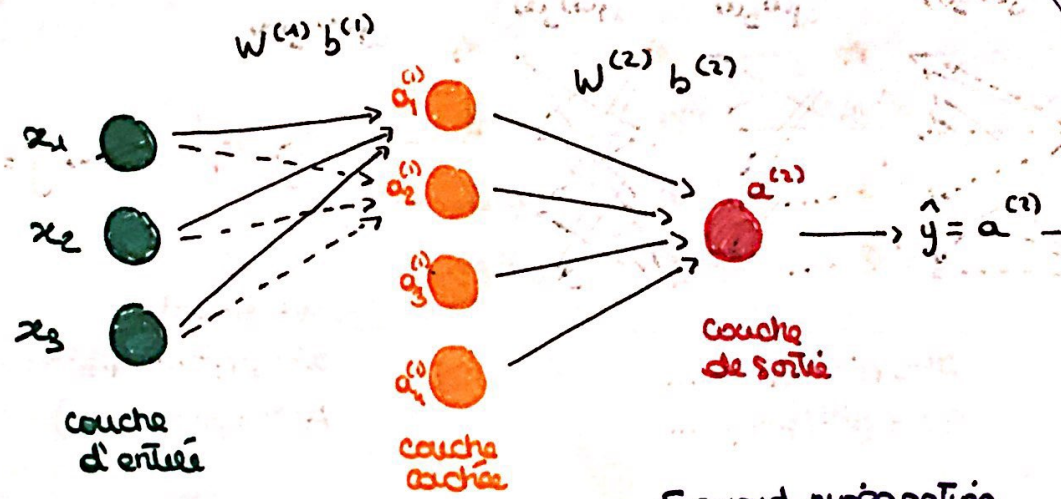
Classification binaire

$y \sim \mathcal{B}(p)$ on a $\begin{cases} P(Y=1)=p \\ P(Y=0)=1-p \end{cases}$

on cherche Θ qui maximise la log Likelihood $\max \log P(Y|x)$

$\Leftrightarrow$ minimiser la BCE

$\Leftrightarrow$ minimiser le coût global.



↳ pour tous les exemples on calcule le coût global.

forward propagation →

$A^{(0)} = X$

$Z^{(1)} = A^{(0)} W^{(1)} + b^{(1)}$

$A^{(1)} = g^{(1)}(Z^{(1)})$

$Z^{(2)} = A^{(1)} W^{(2)} + b^{(2)}$

$A^{(2)} = g^{(2)}(Z^{(2)})$

$\hat{y} = A^{(2)}$

Pour tous les exemples.

Backward propagation ←

$dZ^{(1)} = dA^{(1)} \odot g^{(1)\prime}(Z^{(1)})$

$dW^{(1)} = \frac{1}{m} A^{(0)T} dZ^{(1)}$

$db^{(1)} = \frac{1}{m} \sum_{i=1}^{m} dZ^{(1)}$

$dZ^{(2)} = A^{(2)} - Y$

$dW^{(2)} = \frac{1}{m} A^{(1)T} dZ^{(2)}$

$db^{(2)} = \frac{1}{m} \sum_{i=1}^{m} dZ^{(2)}$

$\boxed{dA^{(1)} = dZ^{(2)} W^{(2)T}}$

update

$W^{(1)} = W^{(1)} - \alpha dW^{(1)}$

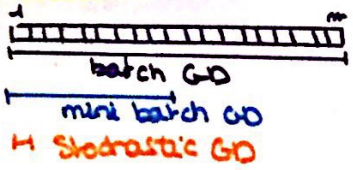
$b^{(1)} = b^{(1)} - \alpha db^{(1)}$

$W^{(2)} = W^{(2)} - \alpha dW^{(2)}$

$b^{(2)} = b^{(2)} - \alpha db^{(2)}$

Réseaux de neurones profonds (plus de 2 couches cachées)

Différents types d'entraînement



Différents types de descentes de gdt

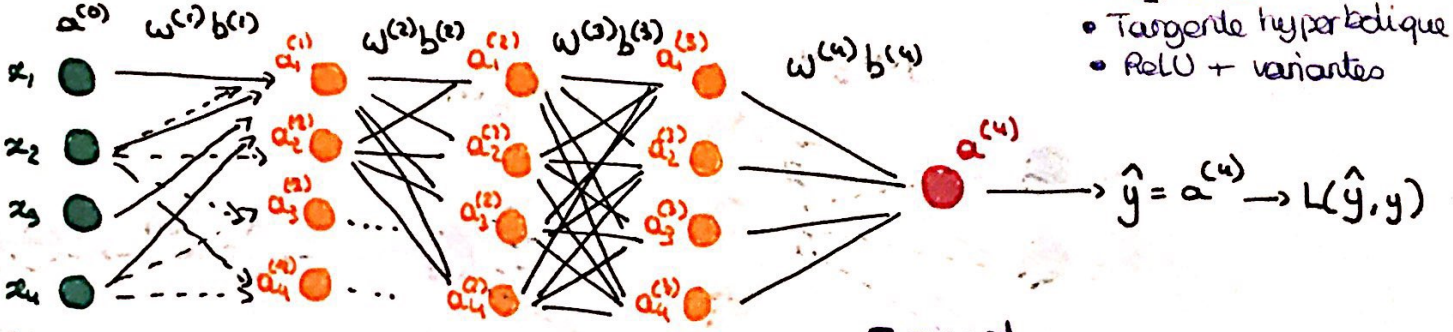
- mini batch GD
- Momentum
- NAG
- AdaGrad
- AdaDelta
- Adam
- learning rate.

Fonctions d'activation $a = g(z)$ Sortie.

- Sigmoïde (binaire)
- Softmax (multiclasse)

Fonctions d'activation couches => non linéaire

- Sigmoïde
- Tangente hyperbolique } Vanishing gradient.
- ReLU + variantes



Forward

$A^{(0)} = X$ $\quad Z^{(1)} = A^{(0)} w^{(1)} + b^{(1)}$ ... $\quad Z^{(4)} = A^{(3)} w^{(4)} + b^{(4)}$ $\quad \hat{y} = A^{(4)}$

$A^{(1)} = g^{(1)}(Z^{(1)})$ ... $\quad A^{(4)} = g^{(4)}(Z^{(4)})$

Pour chaque couche, on met en cache certains éléments pour la backpropagation.

Backward

$A^{(0)}, Z^{(1)}, w^{(1)}$ .... $A^{(2)}, Z^{(3)}, w^{(3)}$ $\quad A^{(3)}, Z^{(4)}, w^{(4)}$

$dZ^{(3)} = dA^{(3)} \odot g^{(3)\prime}(Z^{(3)})$ $\quad dZ^{(4)} = dA^{(4)} \odot g^{(4)\prime}(Z^{(4)})$ ← $dA^{(4)}$

$dw^{(1)}$ $\quad dw^{(4)} = \frac{1}{m} A^{(3)T} dZ^{(4)}$

$db^{(1)}$ $\quad db^{(4)} = \frac{1}{m} \sum_{i=1}^{m} dZ^{(4)}$

$dA^{(1)} = \ldots$ $\quad dA^{(3)} = dZ^{(4)} w^{(4)T}$

La back propagation est un algorithme efficace qui permet de calculer les dérivées en chaîne des composées en stockant les résultats intermédiaires.

La dérivée de la sigmoïde est comprise entre 0 et $\frac{1}{4}$ donc plus le réseau est profond, plus db ou dw est petit et disparaît.

Update $\quad w^{(1)} = w^{(1)} - \alpha dw^{(1)}$ ...

$b^{(1)} = b^{(1)} - \alpha db^{(1)}$

Initialisation des poids

He init = $\text{random}(\text{size}\,\ell, \text{size}\,(\ell-1)) \times \sqrt{\frac{2}{\text{size}\,\ell-1}}$

Xavier = = $\times \sqrt{\frac{1}{\text{size}\,\ell-1}}$

Bengio = = $\times \sqrt{\frac{2}{\text{size}\,\ell-1 \times \text{size}\,\ell}}$

Normalisation = on peut normaliser les entrées par le min et max pour avoir des valeurs similaires de poids, éviter l'explosion des activations et l'instabilité numérique.

Dropout = on enlève aléatoirement certains neurones ce qui retire des connections.
=> joue le rôle de régularisation

Régularisation du réseau réduit l'overfitting.