compte rendu TP 4-5

BEROUKHIM Keyvan

1.1 Jeu de données

1)

- L'ensemble de train sert à entraîner le modèle.
- L'ensemble de validation sert à optimiser les hyper-paramètres pour prendre ceux qui donnent un meilleure score (sans toucher au test) en faisant par exemple un grid search sur l'ensemble des paramètres.
- L'ensemble de test est utilisé à la fin pour évaluer le modèle. **Aucun exemple de test ne doit être vu en train** pour ne pas biaiser les résultats; on cherche à avoir un modèle qui généralise le plus nos données.

2) Un nombre plus grand d'exemples va permettre au modèle de **mieux généraliser** et nous **évite le risque de sur-apprentissage**.

1.2 Architecture du réseau (phase forward)

3) Sans faire d'activation, accumuler des couches linéaires revient à avoir toujours un modèle linéaire qui n'est pas capable de traiter des données non linéairement séparables (dans le cas d'une classification), c'est la fonction d'activation qui nous permet d'introduire la non-linéarité et d'avoir un modèle plus complexe.

4) on a :

- n_x : Dimensions des entrées. Sur le schéma, n_x = 2.
- n_h : Nombre de neurones de la couche cachée. Sur le schéma, n_h = 4.
- n_y : Nombre de classes. Sur le schéma, n_y = 2.

n_x et n_y sont spécifiques aux données et au problème on ne les choisit pas, mais **n_h est un hyper-paramètre** qu'on peut optimiser par grid search par exemple.

5) ŷ est la prédiction du modèle et y est la valeur réelle. **La différence entre y et ŷ définit l'erreur faite par le modèle**.

6) On utilise une fonction SoftMax en sortie pour transformer la sortie de la dernière couche en nombres compris entre 0 et 1 et dont la somme est égale à 1 : cela donne ainsi une **probabilité par classe** ce qui est utile pour la classification. De plus, à la différence de la fonction ‘max’, le ‘SoftMax’ est une **fonction continue et dérivable**, cela permet d’utiliser l’**algorithme de backpropagation**.

7) $\hat{h}$= W_h*x + b_h
h = tanh($\hat{h}$)
$\tilde{y}$= W_y*h + b_y
$\hat{y}$ = SoftMax($\tilde{y}$)

1.3 Fonction de coût

8) La MSE et la cross entropy sont des **fonctions de coût convexes** par rapport à leurs entrées. En appliquant l’algorithme de descente de gradient, on atteindra alors le **minimum global**.

- Pour l’erreur quadratique, la dérivée de la loss est $\partial l/\partial \hat{y} = 2(y - \hat{y})$. Au final, pour faire diminuer l'erreur il faut que $\hat{y}$ soit proche de y.
- Pour la cross entropy, la dérivée de la loss est $\partial l/\partial \hat{y}\_i = - y\_i / \hat{y}\_i$. Pour faire diminuer l'erreur il faut aussi que $\hat{y}$ soit proche de y.

9) Le coût **MSE** est le meilleur pour les **problèmes de régression** dans l’ensemble des nombres réels: la fonction est strictement convexe, facile à calculer, et, le fait que le gradient de l’erreur soit proportionnel à l’erreur semble être un bon choix.
Pour les problèmes de classification, la MSE est quasiment impossible à faire converger. L’**entropie croisée** ou la divergence de Kullback-Leibler sont plus adaptées car elles sont **faites pour mesurer des différences de probabilités**.

1.4 Méthode d’apprentissage

10) Selon la quantité de données utilisées pour calculer le gradient on trouve :
-> Classique: Le calcul de gradient est réalisé sur tout le dataset pour faire une seule mise à jour, ainsi, cela est peut être trop lent.

->stochastic gradient descent: calcule le gradient sur un exemple à la fois avec une mise à jour à chaque fois ce qui rend l'apprentissage plus rapide et permet un apprentissage en ligne. Mais l'algorithme peut avoir du mal à converger.
->**Mini-batch gradient descent**: Le calcul de gradient est fait sur un lot d'exemples ce qui **réduit la variance** des mises à jour des paramètres pour avoir une convergence plus stable.

11) La convergence dépend du choix de η : si il est trop petit, le modèle va être **long à entraîner**, si il est trop grand, le modèle **peut ne pas arriver à converger**. En pratique, on fait souvent **décroitre η au fil des itérations**.

12) Dans l'algorithme de backpropagation, en partant de la fin du réseau, chaque couche calcule sa dérivée en fonction de son entrée et de ses poids et transmet à la couche précédente l'erreur à corriger. On obtient une **complexité proportionnelle au nombre de couches** du réseau.
Avec l'approche naïve, on calcule les dérivées de toutes les couches de manière indépendante, on calcule ainsi n fois la dérivée de la dernière couche ce qui est extrêmement inefficace. Au final, on obtient une **complexité proportionnelle au carré du nombre de couches** du réseau.

13) L'algorithme de backpropagation **nécessite que de tout le réseau soit dérivable**. On ne peut par exemple pas utiliser de SVM dedans.

14) $$\ell(y,\hat{y}_t) = -\sum_i y_i \log(\hat{y}_i) = -\sum_i y_i \log\left(\frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}}\right)$$

$$= -\sum_i y_i \tilde{y}_i + \sum_i y_i \log\left(\sum_j e^{\tilde{y}_j}\right)$$ puisque on a un seul y_i non nul

$$\ell(y_t,\hat{y}_t) = -\sum_i y_i \tilde{y}_i + \log\left(\sum_i e^{\tilde{y}_i}\right) =$$

15) $$\frac{\partial \ell}{\partial \tilde{y}_i} = \frac{\partial\left(-\sum_i y_i \tilde{y}_i + \log\left(\sum_i e^{\tilde{y}_i}\right)\right)}{\partial \tilde{y}_i} = -y_i + \frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}} = -y_i + \hat{y}_i$$

16) 1) $\dfrac{\partial \tilde{y}_k}{\partial W_{y_{ij}}} = \dfrac{\partial \sum_m W_{y_{km}} \cdot h_m + b_{y_k}}{\partial W_{y_{ij}}}$

$= \begin{cases} 0 \text{ si } k \neq i \\ h_j \text{ sinon} \end{cases}$

2) $\dfrac{\partial \tilde{y}_k}{\partial b_{y_j}} = \begin{cases} 0 \text{ si } k \neq j \\ 1 \text{ sinon} \end{cases}$ 3) $\dfrac{\partial \tilde{y}_k}{\partial h_j} = W_{y_{kj}}$

1) $\Rightarrow$ $\dfrac{\partial l}{\partial W_{y_{ij}}} = \sum_k \dfrac{\partial l}{\partial \tilde{y}_k} \dfrac{\partial \tilde{y}_k}{\partial W_{y_{ij}}} = \dfrac{\partial l}{\partial \tilde{y}_i} h_j$

2) $\Rightarrow$ $\dfrac{\partial l}{\partial b_{y_j}} = \dfrac{\partial l}{\tilde{y}_j}$

17)

3) $\Rightarrow$ $$\frac{\partial l}{\partial h_j} = \sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial h_j} = \sum_k \frac{\partial l}{\tilde{y}_k} W_{ykj}$$

$$\frac{\partial l}{\partial \tilde{h}_j} = \frac{\partial l}{\partial h_j} \frac{\partial h_j}{\partial \tilde{h}_j} = \frac{\partial l}{h_j} \left(1 - \tanh^2(\tilde{h}_j)\right)$$

De même :

$$\frac{\partial l}{\partial W_{h\,ij}} = \sum_k \frac{\partial l}{\partial \tilde{h}_k} \frac{\partial \tilde{h}_k}{\partial W_{hij}} = \frac{\partial l}{\partial \tilde{h}_i} x_j$$

$$\frac{\partial l}{\partial b_{h\,i}} = \frac{\partial l}{\tilde{h}_j}$$