IFT3395/6390, Devoir 2

Date affiché: Oct. 30, 2014,
À remettre: 12 nov. 2014, au *début* de la classe

---

1. (6/10) On suppose qu'on dispose d'un ensemble d'entraînement $D_n = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_n, t_n)\}$ avec $\mathbf{x}_i \in R^d$ et $t_i \in \{1, \ldots, M\}$ indiquant la classe parmi M classes.

   Soit un réseau de neurones feed-forward avec une seule couche cachée (donc 3 couches en tout si on compte la couche d'entrée et la couche de sortie). La couche cachée est constituée de d_h neurones complètement connectés à la couche d'entrée. Ils ont une non-linéarité de type tangente hyperbolique (tanh). La couche de sortie est constituée de M neurones, complètement connectés à la couche cachée. Ils ont une non-linéarité de type softmax. La sortie du j-ème neurone de la couche de sortie donnera un score pour la classe j interprété comme la probabilité que l'entrée $\mathbf{x}$ soit de cette classe j.

   Il vous est fortement conseillé de dessiner le réseau de neurones au fur et à mesure afin que vous puissiez mieux suivre les étapes (mais pas besoin de nous fournir un dessin!)

   a) Soit $\mathbf{W}^{(1)}$ la matrice $d_h \times d$ de poids et soit $\mathbf{b}^{(1)}$ le vecteur de biais caractérisant des connexions synaptiques allant de la couche d'entrée à la couche cachée. Indiquez la dimension de $\mathbf{b}^{(1)}$.

   Donnez la formule de calcul du vecteur d'activations (i.e. avant non-linéarité) des neurones de la couche cachée $\mathbf{h}^a$ à partir d'une observation d'entrée $\mathbf{x}$, d'abord sous la forme d'une expression de calcul matriciel, puis détaillez le calcul d'un éléments $\mathbf{h}^a$. Exprimez le vecteur des sorties des neurones de la couche cachée $\mathbf{h}^s$ en fonction de $\mathbf{h}^a$.

   b) Soit $\mathbf{W}^{(2)}$ la matrice de poids et soit $\mathbf{b}^{(2)}$ le vecteur de biais caractérisant les connexions synaptiques allant de la couche cachée à la couche de sortie. Indiquez les dimensions de $\mathbf{W}^{(2)}$ et $\mathbf{b}^{(2)}$. Donnez la formule de calcul du vecteur d'activations des neurones de la couche de sortie $\mathbf{o}^a$ à partir de leurs entrées $\mathbf{h}^s$ sous la forme d'une expression de calcul matriciel, puis détaillez le calcul de $\mathbf{o}^a$.

   c) L'entraînement du réseau de neurones va consister à trouver les paramètres du réseau qui minimisent la fonction de perte pour l'ensemble d'entraînement. Indiquez précisément de quoi est constitué l'ensemble $\boldsymbol{\theta}$ des paramètres du réseau. Indiquez à combien de paramètres scalaires n_θ cela correspond.

   Pour trouver la solution à ce problème d'optimisation, on va utiliser une technique de descente de gradient. Le gradient du coût L encouru pour l'i-ème exemple d'entraînement

$(\mathbf{x}_i, t_i)$ par rapport aux paramètres est:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial L(\mathbf{x}_i, t_i)}{\partial \theta_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial L(\mathbf{x}_i, t_i)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

Pour calculer le gradient on va appliquer la technique de rétropropagation du gradient.

IMPORTANT: La technique de rétropropagation des gradients correspond à une application intelligente et efficace de la règle de dérivation en chaine, qui évite de refaire inutilement des calculs coûteux. Elle suppose qu'on a déjà calculé et stocké en mémoire les activations et sorties des neurones lors de la phase de propagation avant. On peut ainsi directement s'en servir, sans les recalculer, lors du calcul des gradients. Par conséquent, contentez-vous d'exprimez le calcul des gradients en fonction de ces éléments déjà précédemment calculés, sans y resubstituer l'expression de leur calcul. En effet ceci reviendrait à les recalculer et donnerait au final des expressions extrêmement longues, complexes, et inefficaces.

La sortie des neurones de sortie est donnée par

$$\mathbf{o}^s = \text{softmax}(\mathbf{o}^a)$$

$$o_k^s = \frac{\exp\left(o_k^a\right)}{\sum_{j=1}^{M} \exp\left(o_j^a\right)}$$

La fonction de perte est la log-vraisemblance négative:

$$L(\mathbf{x}, t) = -\log \mathbf{o}_t^s(\mathbf{x})$$

Comme vous l'avez montré dans le devoir 1, les dérivées partielles du coût L par rapport aux activations des neurones de la couche de sortie sont

$$\frac{\partial L(\mathbf{x}, t)}{\partial o_k^a} = \begin{cases} o_k^s - 1 & \text{si} \quad k = t \\ o_k^s & \text{si} \quad k \neq t \end{cases}$$

ou, sous forme d'une expression matricielle, en définissant la dimension de chacune des matrices ou vecteurs manipulées:

$$\frac{\partial L(\mathbf{x}, t)}{\partial \mathbf{o}^a} = \mathbf{o}^s - \text{onehot}_m(t)$$

d) Montrez que les gradients par rapport aux paramètres $\mathbf{W}^{(2)}$ et $\mathbf{b}^{(2)}$ sont:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{o}^a}\left(\mathbf{h}^s\right)^{\mathrm{T}} \quad \text{et} \quad \frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{o}^a}$$

où $\frac{\partial L}{\partial \mathbf{o}^a}$ et $\mathbf{h}^s$ sont des vecteurs colonnes. Précisez les dimensions.

e) En utilisant la règle de dérivation en chaîne

$$\frac{\partial L}{\partial h_j^s} = \sum_{k=1}^{M} \frac{\partial L}{\partial o_k^a} \frac{\partial o_k^a}{\partial h_j^s}$$

montrez que les dérivées partielles du coût L par rapport aux sorties des neurones de la couche cachée sont:

$$\frac{\partial L}{\partial \mathbf{h}^s} = \left(\mathbf{W}^{(2)}\right)^{\mathrm{T}} \frac{\partial L}{\partial \mathbf{o}^a}$$

où $\frac{\partial L}{\partial \mathbf{o}^a}$ est un vecteur colonne. Précisez les dimensions.

f) Calculez les dérivées partielles par rapport aux activations des neurones de la couche cachée. Comme L ne dépend de l'activation h_j^a d'un neurone de la couche cachée qu'au travers de la sortie h_j^s de ce neurone, la règle de dérivation en chaîne donne:

$$\frac{\partial L}{\partial h_j^a} = \frac{\partial L}{\partial h_j^s} \frac{\partial h_j^s}{\partial h_j^a}$$

Notez que $\mathbf{h}^s = \tanh(\mathbf{h}^a)$, où la tangente hyperbolique s'applique élément par élément. La formule de la tangente hyperbolique est: $\tanh(z) = \frac{\sinh z}{\cosh z} = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}-1}{e^{2z}+1}$ Comme étape intermédiaire, commencez par démontrer que $\frac{\partial \tanh z}{\partial z} = 1 - \tanh^2(z)$. Exprimez le calcul sous forme d'une expression matricielle, en définissant la dimension de chacune des matrices ou vecteurs manipulés.

g) Calculez les gradients par rapport aux éléments des paramètres $\mathbf{W}^{(1)}$ et $\mathbf{b}^{(1)}$ de la couche cachée. Exprimez ce calcul du gradient sous forme d'une expression matricielle.

h) Considérez une régularisation de type "weight decay" quadratique qui pénalise la norme au carré (norme $L2$) des poids (mais pas des biais). Comment cela change-t-il le gradient par rapport aux différents paramètres?

i) Expliquez en détail tout ce qui va changer en ce qui concerne les dérivées si on utilise la non-linéarité *rectifieur RELU*:

$$\mathrm{RELU}(h_j^a) = \begin{cases} h_j^a & \text{si} \quad h_j^a \geq 0 \\ 0 & \text{si} \quad h_j^a < 0 \end{cases}$$

à la place de la non-linéarité tanh.

2. (4/10) On vous demande d'implémenter le réseau de neurones, et de l'appliquer aux données utilisées pour le devoir 1 (regardez le devoir 1 pour les informations sur ces données si vous en avez oublié les détails):

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/train_images.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/test_images.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/train_labels.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/test_labels.txt`

Vous pouvez soit (i) utiliser le calcul du gradient pas à pas tel que vous l'avez dérivé dans la question précédente (incluant le weight decay), soit (ii) utiliser une implémentation existante (comme la librairie theano).

Indices:

- Initialisation des paramètres: Comme vous le savez, il est nécessaire d'initialiser aléatoirement les paramètres du réseau (dans le but d'éviter les symétries et la saturation des neurones). Initialisez les poids d'une couche en les tirant d'une uniforme sur $\left[\frac{-1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}\right]$ où n_c est le nombre d'entrées de cette couche (le nombre de neurones d'entrée auxquels chaque neurone de cette couche est connecté, donc ça change typiquement d'une couche à l'autre). Les biais peuvent quant à eux être initialisés à 0. Justifiez votre choix de toute autre initialisation.
- Si vous utilisez le calcul du gradient pas à pas on vous suggère d'écrire des méthodes *fprop*, *bprop* et *grad* comme discuté en classe.
- Vérification du gradient par différence finie: On peut estimer le gradient numériquement par différences finies. Vous devez implémenter cette estimation de façon à vérifier votre calcul (ou le calcul effectué par la librairie qui vous utilisez) du gradient. Pour ce faire, calculez d'abord la valeur de la perte pour la valeur courante des paramètres (sur un exemple d'entraînement). Ensuite, pour chaque paramètre θ_k (un scalaire), modifiez ce paramètre par une petite valeur ($10^-6 < \epsilon < 10^-4$) et recalculez la perte (même exemple d'entraînement) puis ramenez le paramètre à sa valeur de départ. La dérivée partielle par rapport à chaque paramètre est alors estimé en divisant la variation de la perte par ϵ. Le ratio de votre gradient calculé par rétropropagation du gradient estimé par différence finie devrait se situer entre 0.99 et 1.01.
- Taille des lots: On demande que votre calcul et descente de gradient opère sur des mini-lots (minibatch, par opposition à batch) de taille 100. Dans le cas de mini-lots, on ne manipule pas un unique vecteur d'entrée, mais plutôt un lot de vecteurs d'entrée, groupés dans une matrice (qui donneront de même une matrice au niveau de la couche cachée, et de la sortie). (Dans le cas d'une taille de lot de un, on obtient l'équivalent d'un gradient stochastique.)

À remettre:

- Vérification du gradient: produisez un graphique montrant le gradient estimé par différence finie en utilisant *le premier example d'entraînement*. Dans le même graphique, montrez aussi le gradient calculé par rétropropagation d'erreur. Affichez ceci même si vous utilisez une librairie comme theano qui calcule les gradients automatiquement.
- Entraînement du réseau sur les données d'entraînement:

  Produisez les courbes d'entraînement et de test (courbes d'erreur de classification et du coût en fonction du nombre d'époques d'entraînement). Joignez à votre rapport les courbes obtenues avec vos meilleurs valeurs d'hyper-paramètres, c.a.d.

pour lesquels vous avez atteint la plus basse erreur de classification sur l'ensemble de test. Produisez deux graphiques: un pour les courbes de taux d'erreurs de classification (train et test, bien précisées dans la légende) et l'autre pour la perte moyenne (le L moyen sur train et test). Indiquez dans votre rapport la valeur des hyper-paramètres retenues correspondant aux courbes que vous joignez.

- Non-linéarité *rectifieur*: Répétez l'expérience précédente en utilisant cette non-linéarité. Vous pouvez vérifier le gradient comme auparavant (mais pas besoin de nous fournir le resultat de vérification).
- (Bonus, 1 point) Répétez l'expérience précédente en utilisant dropout (proposé récemment par http://arxiv.org/pdf/1207.0580.pdf) à la place de "weight decay" pour la régularisation pendant l'entraînement: Multipliez les sorties h_j^s dans le mini-batch par 0 avec probabilité $\frac{1}{2}$. (Cela doit être fait indépendamment pour chaque exemple d'entraînement et indépendamment pour chaque unité cachée.) Pour appliquer le modéle après l'entraînement sur des données de test, il ne faut pas faire cette corruption d'unités. À la place, multipliez les poids de la couche cachée vers la couche de sortie par 0.5.