Réseaux de Neurones en pratique

Pascal Vincent

24 novembre 2016

1 Implémentation d'un réseau de neurones

1.1 Choix architecturaux, hyper-paramètres architecturaux

1.1.1 Quel problème le réseau doit pouvoir résoudre ?

Ceci va déterminer la non-linéarité de sortie et la fonction de perte

- classification binaire avec cible $y \in \{0, 1\}$: on a 1 neurone se sortie sigmoid calcule la probabilité de la classe 1 : $o^s = \text{sigmoid}(o^a)$
  coût : entropie-croisée sigmoide : $L(o^s, y) = -(y \ \log(o^s) + (1-y)\log(1-o^s))$
- classification multiclasse : m neurones de sortie calculant la probalilité de chacune des classes : $o^s = \text{softmax}(o^a)$
  coût : $L(o^s, y) = -\log o^s_y = -\langle(\log o^s), \text{onehot}(y)\rangle$
- régression : 1 neurone de sortie linéaire (ou m neurones de sortie linéaire pour la régression multiple), pas de non-linéarité : $o^s = o^a$
  coût d'erreur quadratique : $L(o^s, y) = \|o^s - y\|^2$
- problème multilabel (avec m différents labels) : peut être vu comme m problèmes de classification binaire (si on suppose que sachant x les labels sont indépendants). On aura donc m neurones de sortie sigmoid pour lesquels on utilisera le coût de cross-entropie (identique à classification binaire).

1.1.2 Hyper-paramètres architecturaux :

- plusieurs couches cachées : combien ?
- non-linéarité des couches cachées
- nombre de neurones par couche
- il existe d'autres types de couches (ex : couches convolutionnelles)

1.1.3 Autres hyper-paramètres de régularisation

- Force λ_1 et/ou λ_2 régularisation des poids L_2 (weight decay), et/ou L_1
- autres techniques de régularisaiton : quantité d'injection de bruit (« dropout »)

1.2 Paramètres à optimiser et initialisation des paramètres

Le nombre, la taille des paramètres θ à optimiser va dépendre des choix architecturaux. Notamment la taille des différentes couches. Ex si on a $x \in \mathbb{R}^{d_x}$ et sortie : $o \in \mathbb{R}^{d_{out}}$ une couche cachée $h \in \mathbb{R}^{d_h}$ les paramètres sont : $\theta = \{W^{(1)}, b^{(1)}, W^{(out)}, b^{(out)}\}$, ce qui correspond à combien de scalaires ? $n_\theta = d_x d_h + d_h + d_h d_{out} + d_{out}$

Il est nécessaire d'initialiser aléatoirement les paramètres du réseau (dans le but d'éviter les symétries et la saturation des neurones et idéalement pour se situer au point d'inflexion de la non-linéarité de façon à avoir un comportement non-linéaire).

Une heuristique classique :

- initialiser les *poids* d'une couche en les tirant d'une uniforme sur $\left[\frac{-1}{\sqrt{d_{in}}}, \frac{1}{\sqrt{d_{in}}}\right]$, où d_{in} 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.

Il existe de nombreuses autres heuristiques d'initialisation.

1.3 Calcul efficace de la prédiction (fprop), du coût et risque empirique régularisé, et des gradients (bprop)

Voir le devoir !

1.4 Vérification du gradient par différences finies

Le calcul du gradient par rétropropagation (bprop) est un calcul analytique efficace.

Il faut s'assurer qu'il n'est pas BUGGÉ...

Pour ce faire, on peut aussi estimer le gradient numériquement par différences finies : c'est très inefficace. On ne s'en servira donc pas pendant l'entrainement, mais ça va permettre de *vérifier* que le calcul du gradient effectué par l'appel à notre méthode bprop est correct.

Soit $J(\theta; D_{train})$ l'objectif qu'on veut minimiser par descente de gradient pour trouver la valeur optimale des paramètres $\theta = \{\theta_1, \ldots, \theta_{n_\theta}\}$ pour D_{train}. Il peut s'agir d'un risque empirique régularisé par ex :

$$J(\theta; D_{train}) = \frac{1}{|D_{train}|} \sum_{x,y \in D_{train}} \left(L(f_\theta(x), y) + \lambda \|\theta\|^2 \right)$$

On notera $J(\theta; D_{mini})$ ce même objectif, mais sur un minibatch D_{mini} (possiblement restreint à un unique exemple). Rappelons que le gradient $\frac{\partial J}{\partial \theta}(\theta; D_{mini})$ d'un tel objectif est l'ensemble des dérivées partielles de J par rapport à chacun des paramètres scalaires θ_k. Par exemple soit un des paramètres scalaires

$\theta_k = W^{(1)}_{3,5}$ correspondant à une entrée de la matrice de poids d'une couche d'un réseau de neurones. Le gradient de J par rapport à ce paramètre est :

$$\frac{\partial J}{\partial \theta_k}(\theta; D_{mini}) = \lim_{\epsilon \to 0} \frac{J(\theta_1, \ldots, \theta_k + \epsilon, \ldots, \theta_{n_\theta}; D_{mini}) - J(\theta_1, \ldots, \theta_k, \ldots, \theta_{n_\theta}; D_{mini})}{\epsilon}$$

Ceci suggère donc l'algorithme suivant pour vérifier le calcul du gradient par différence finie :

- Initialiser les paramètres θ selon l'heuristique d'initialisation aléatoire choisie.
- Prenez un minibatch D_{mini}.
- Calculez $J_{ref} = J(\theta; D_{mini})$
- Prévoir pour stocker le gradient estimé par différence finie, un ensemble de vecteurs/matrices/tenseurs de mêmes dimensions que ce que vos paramètres θ. Nous nommerons cet ensemble $\nabla^{diff-finie}$.
- Prendre ϵ petit (ex : 1e-5)
- Pour chaque vecteur ou matrice ou tenseur de paramètre dans θ et pour chaque composante scalaire θ_k de ce vecteur ou matrice ou tenseur :
  - se souvenir de la valeur de θ_k : $\theta_k^{ref} \leftarrow \theta_k$
  - changer (perturber) $\theta_k \leftarrow \theta_k + \epsilon$
  - Recalculer $J_{preturb} \leftarrow J(\theta; D_{mini})$ où θ contient le θ_k modifié
  - ramener θ_k à sa valeur initiale : $\theta_k \leftarrow \theta_k^{ref}$
  - La dérivée partielle par rapport à ce paramètre θ_k est calculée comme : $\nabla^{diff-finie}_{\theta_k} \approx \frac{J_{perturb} - J_{ref}}{\epsilon}$. Copiez cette valeur dans $\nabla^{diff-finie}$.
- Calculer le gradient analytique par rétropropagation ∇^{bprop} en appelant vos fonctions fprop puis bprop
- Comparer $\nabla^{diff-finie}$ à ∇^{bprop} : vous pouvez afficher les deux pour chacune des vecteurs/matrices/tenseurs constituant vos paramètres θ. Pour chacune repérez les éléments qui différent le plus (en valeur absolue de leur différence : $|\nabla^{diff-finie} - \nabla^{bprop}|_k$), affichez les, ainsi que leur différence.

Évidemment, pour que le calcul de gradient par différence finie ne soit pas trop long, et pour que l'affichage ne soit pas trop volumineux, il vaut mieux faire cette vérification sur de tout petits réseaux (restreignez la dimension d'entrée au besoin).

1.5 Entrainement : apprentissage des paramètres par descente de gradient

Le principe de la descente de gradient par mini-lot (minibatch) est très simple. On itère sur :

- extraire le prochain minibatch D_{min} des données d'entrainement
- calculer les prédictions $f_\theta(D_{mini})$ avec fprop
- calculer $J(\theta; D_{mini})$ et l'accumuler afin d'en faire la moyenne sur l'ensemble d'entrainement

— calculer le gradient $\nabla_\theta = \frac{\partial J}{\partial \theta}(\theta; D_{mini})$ avec bprop (plus précisément, on va calculer $\nabla_{param} = \frac{\partial J}{\partial param}$ pour chaque vecteur,matrice,tenseur de paramètre qui constitue θ)
— mettre-à-jour les paramètres en effectuant un pas de gradient $\theta \leftarrow \theta - \eta\nabla_\theta$ (plus précisément, on va pour chaque vecteur,matrice,tenseur de paramètre qui constitue θ faire $param \leftarrow param - \eta\nabla_{param}$). Le taux d'apprentissage η est un scalaire positif.

Ce taux d'apprentissage η est un **hyper-paramètre critique de l'optimiseur**. Il sera **crucial de bien l'ajuster spécifiquement pour chaque problème et pour chaque configuration de valeurs des autres hyper-paramètres** (architecturaux ou régularisation), sinon l'optimisation se fera mal, trop lentement, ou pas du tout.

1.5.1 Hyper-paramètres de l'optimiseur

En plus des hyper-paramètres architecturaux et de régularisation, chaque algorithme d'optimisation (variantes de descente de gradient ou autre) a aussi des hyper-paramètres, qui vont influer sur l'efficacité de l'optimisation.

Un hyper-paramètre crucial de la descente de gradient est le taux d'apprentissage η.

1.6 Évaluation de mesures de performance d'intérêt

L'objectif optimisé durant l'entrainement est souvent différent du coût qui nous intéresse vraiment (par ex. le taux d'erreur de classification, l'argent gagné, ou le nombre de vies sauvées). On peut donc distinguer, parmi les mesures d'erreurs / de perfromance :

— l'objectif que l'entrainement optimise (un risque empirique régularisé) notée ici J
— la perte moyenne L sans les termes de régularisation (ex : la log vraisemblance moyenne ou entropie croisée)
— le ou les coûts qui nous intéressent vraiment (ex : erreur de classification, nombre de vies sauvées, etc...)

Il faut donc également **implémenter** le calcul de ces *autres* mesures de performance obtenues avec les prédiction du modèle, sur un ensemble de données quelconque (qui pourra être l'ensemble de validation ou de test).

La sélection de modèle et d'hyper-paramètres contrôlant la capacité se fera généralement sur la base du coût / de la mesure de performance qui nous intéresse vraiment, évalué hors-échantillon-d'entrainement, c.a.d. sur un ensemble de validation.

1.7 Prévoir de tester sur un ensemble de validation au cours de l'entrainement pour arrêt prématuré

Un algorithme d'apprentissage qui utilise une méthode d'optimisation itérative (tel qu'une descente de gradient) modifie progressivement ses paramètres θ pour

optimiser un objectif $J(\theta)$.

Une technique importante de régularisation est l'arrêt prématuré, qui consiste à ne pas optimiser jusqu'au bout : on s'arrête avant de tomber dans un régime de sur-apprentissage !

Voici comment on peut procéder :

- on définit une **époque** d'entrainement comme un passage complet au travers de l'ensemble d'entrainement
- initialiser les paramètres selon l'heuristique d'initialisation aléatoire
- Pour un nombre N d'époques :
  - effectuer la mise-à-jour des paramètres par descente de gradient par minibatch, en couvrant l'ensemble d'entrainement au complet 1 fois, tout en accumulant la valeur de l'objectif d'optimisation J calculé au fur et à mesure sur les minibatch
  - Calculer et afficher la valeur moyenne de l'objectif d'optimisation J au cours de cette époque. **Si l'entrainement fonctionne, la valeur de J devrait s'améliorer à chaque époque !** Sauvegarder cette valeur dans un fichier.
  - Évaluer les autres mesures de performance d'intérêt sur l'ensemble de validation (et optionnellement sur l'ensemble de test et/ou d'entrainement), les afficher et les sauvegarder.
  - Si la principale mesure de performance d'intérêt sur l'ensemble de validation est la meilleure rencontrée à ce jour, conserver une copie θ^* de la valeur des paramètres appris.

La sauvegarde des évaluations de performance (et de l'objectif d'optimisation J) permet d'**afficher les courbes d'apprentissage.**

Quand on voit que la performance qui nous intéresse empire sur l'ensemble de validaiton, tout en continuant de s'améliorer (ou stagner) sur l'ensemble d'entrainement, alors on sait qu'on est en sur-apprentissage. On peut alors interrompre l'entrainement, et se servir des paramètres optimaux θ^* dont on s'est souvenu qui donnaient lieu à la meilleure performance de validation.

2 Application pratique des réseaux de neurones

2.1 Décider à quel problème on a affaire

Classification binaire ? multiclasse ? régression simple ? régression multiple ? multilabel ?

Problème non-supervisé ? (un autoencodeur est-il approprié ?)

2.2 Penser à ce que devraient être nos entrées (inputs) et cible

Pour les problèmes supervisés :

- cible = ce que sur le terrain on va vouloir prédire

— entrées = l'ensemble des informations potentiellement pertinentes dont on disposera sur l'exemple, *au moment de faire la prédiction.*

Prévoir constituer ainsi sous forme de table, l'ensemble d'exemples dont on dispose avec leurs étiquettes cible (peut impliquer extraction d'une base de donnée, et travail de formattage).

2.3 Pré-traitement et encodage des données

Examiner les données dont on dispose :

- combien d'exemples étiquetés ? (n)
- (combien d'exemples non-étiquetés, si on pense vouloir s'en servir)
- Proportion des classes si classification. (Si fortement débalancées, envisager d'utiliser une stratégie de rebalancement).
- Combien de traits caractéristiques au départ ?
- Lister chaque trait caractéristique et pour chacun :
  - préciser sa nature : catégorique/discret, continu (numérique), ordinal
  - ses statistiques de base. Pour les catégoriques : la proportion de chaque catégorie dans l'ensemble de données. Pour les autres : valeur moyenne, minimale, maximale, écart-type
  - relever les bizarreries, comprendre ce qu'elles signifient, et les corriger ! (ex : age=999, taille=-1).
- Effectuer un pré-traitement des entrées :
  - pour les traits catégoriques : les encoder en one-hot
  - standardiser les autres (surtout si ils présentent de gros écarts d'échelle) : standardiser = soustraite leur moyenne et diviser par leur écart type (ajouter un petit epsilon à l'écart-type si certaines dimensions présentent des écarts-types très faibles).
- Adaptez la représentation de la cible y à l'algorithme que vous allez utiliser. Ex : s'attend-t-il à une classe représentée par {0,1}, par {-1,1} par {1,2} par un vecteur onehot, ... ?

2.4 Décider de la méthode de validation appropriée

- Si n suffisamment grand : découper en train/valid/test
- Sinon, prévoir utiliser la validation croisée en k blocs (ou leave-one-out dans les cas extrèmes) pour la validation.

2.5 Recherche des hyper-paramètres et entrainement

Utilisez la procédure de sélection de modèle appropriée pour trouver les hyper-paramètres optimaux donnant la meilleure performance de généralisation telle qu'estimée sur les données de validation.

On va distinguer :

- les hyper-paramètres de contrôle de capacité (hyper-params architecturaux et de régularisation qui ont un effet direct sur la capacité effective du modèle, ex : nombre de neurones cachés)

— les hyper-paramètres de l'optimiseur (ex : taux d'apprentissage η)

Il y a un troisième « hyper-paramètre » particulier : le nombre d'époques d'optimisation car c'est un hyper-param de l'optimisation mais qui exerce un contrôle direct sur la capacité (par la technique d'arrêt prématuré).

2.5.1 La procédure haut niveau est la suivante :

- Pour chaque combinaison de valeur d'**hyper-paramètre de contrôle de capacité** (architecturaux et régularisation) considérés
  - Faire une recherche pour trouver les valeurs d'**hyper-paramètres de l'optimiseur** (taux d'apprentissage ....) qui permettent le mieux d'optimiser l'objectif d'entrainement J sur l'ensemble **d'entrainement** (tel que visible sur la courbe d'entrainement affichant $J(D_{train})$ en fonction des époques) .
  - Par arrêt prématuré, retenir le modèle (donc la valeur des paramètres) ayant obtenu la meilleure performance d'intérêt sur l'ensemble de **validation** : on peut pour cela se baser sur la courbe d'apprentissage donnant la performance d'intérêt sur l'ensemble de validation.