Fonctions de coût, gradient stochastique...

1 Exemple de fonctions de coût couramment utilisées dans les réseaux de neurônes

Nous allons étudier les fonctions de coût (ou loss) utilisées dans divers type de réseaux de neurones dans le cas supervisé. Je vous rappelle que l'apprentissage d'un réseau vise à optimiser les paramètres θ de ce réseau de façon à ce que la différence entre la réponse du réseau avec la réponse attendue (vérité terrain) soit minimale. C'est cette différence que traduit la loss. Dans ce TD on s'interessera à la loss mais pas à la structure du CNN.

De la même façon que pour approximer une droite à partir d'un ensemble de points, il existe bien sûr plusieurs fonctions de coût pour résoudre un problème.

Nous allons examiner plusiers classes de problèmes et élaborer des fonctions de coût associées. Chaque exemple sera lié à une implémentation existant en matlab pour des problèmes d'orientation, de reconnaissance ou de similarité de chiffres ou de symboles.

1.1 Regression : Mean square errors

Dans un problème de regression, on cherche à prédire des données qui ont une valeur **continue**, contrairement au problème de classification où on prédit des labels.

A titre d'exemple, on va considérer un ensemble d'images de chiffres pour lesquels on veut prédire leur orientation :

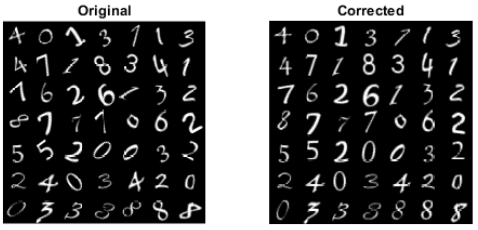


FIGURE 1 – Chiffres initiaux et chiffres attendus par le réseau après réorientation

On veut donc élaborer un modèle de régression à l'aide de réseaux de neurones convolutifs pour prédire les angles de rotation de chiffres manuscrits (Fig. 1). Dans les données d'apprentissage, on possède pour chaque image x_i l'orientation du chiffre, notée t_i. On note y_i le résultat donné par le réseau sur l'image x_i avec l'ensemble de paramètres θ ($y_i = f_\theta(x_i)$).

Questions

— Ecrire la fonction de coût permettant d'optimiser le réseau pour qu'il prédise l'orientation des chiffres.

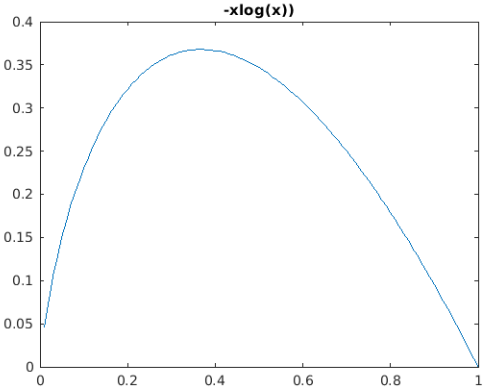
1.2 classification et Cross entropie

Dans un problème de classification en N classes, le réseau sort un vector à N composantes qu'on peut analyser comme la probabilité que la donnée d'entrée x appartienne à la classe i ($1 \leq i \leq N$ (la somme des N éléments du vecteurs de sortie vaut 1). Il faut donc comparer la distribution prédite au vecteur thérique formé uniquement de 0 avec une valeur de 1 pour la classe d'appartenance. On utilise des mesures de **comparaison de distributions** pour évaluer la similarité entre les résultats prédits et attendus. L'entropie est couramment utilisée

— L' entropie quantifie le niveau de désorganisation d'un système. Introduite par Shannon pour formaliser l'information perdue dans les signaux téléphoniques
— plus le système emet d'informations différentes, plus l'entropie est grande : un système renvoyant toujours une même valeur a une entropie nulle
— Soit une v.a. discrète X comportant n symbole avec une probabilité d'apparition p_i. L'entropie H de X est définie par

$$H(X) = -E(log(p)) = -\sum_{i=1}^{n} p_i log(p_i)$$

Propriétés de l'entropie
— $H(X) \geq 0$ (dans [0,1] $log(x) < 0$)
— $H(X) = 0$ ssi X=cst



— L'entropie croisée (cross entropy) sert à comparer deux distributions P et Q. Elle est définie par

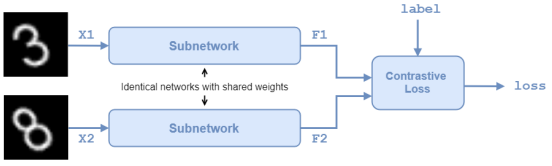
$$H(P,Q) = -\sum_{i=1}^{n} p_i log(q_i)$$

— Propriété : L'entropie $H(P)$ est la valeur minimale de la cross-entropie $H(P,Q)$
— Pour cette raison le minimum de l'entropie est utilisé pour imposer la ressemblance entre P et Q.
— C'est la fonction de coût couramment utilisée pour la classification dans les réseaux de neurones

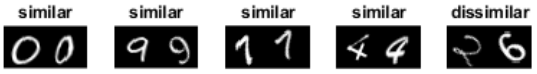
Questions
— dans le cas binaire (1 seule classe et le réseau répond par oui ou non), expliquer pourquoi la fonction de Loss est donnée par $-tlog(y) - (1-t)log(1-y)$,
où t est le label valant$O$ ou 1 et y la valeur prédite par le réseau
— Voir le réseau de neurones correspondant sous matlab sur la page *Create Simple Deep Learning Network for Classification*

1.3 Réseaux Siamois et réduction de la dimension

— objectifs : étant données des classes (par exemples de chiffres), trouver une représentation de taille réduite des images permettant de bien identifier les chiffres.
— Un réseau siamois utilise deux branches identiques (mêmes paramètres, mêmes poids). Ils aboutissent à une représentation réduite avec l'objectif que
  — deux images de la même classe ont des descripteurs semblables
  — deux images issues de classes différentes ont une différence élevée



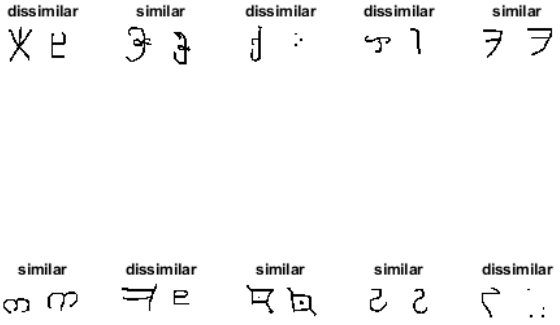
— entrainement du réseau : on donne en entrée des images similaires (même chiffre) ou dissimilaires (chiffres



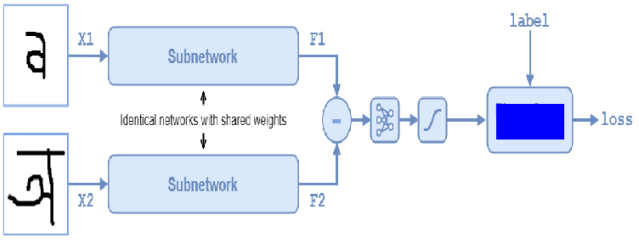
différents)
— l'étiquette (label) est y=1 si données similaires, y=0 sinon (apprentissage supervisé)
— on note $d = ||F_1 - F_2||^2$, la différence entre descripteurs. On appelle m la marge telle que deux images dissimilaires doivent avoir au moins une marge m : $d > m$
— Questions Recherche de la fonction de coût $L(y, d, m)$ qui permet d'entrainer ce réseau
  — si y=1, comment doit être d ?
  — si y=0, que devient $m - d$?
  — Montrer que la fonction $loss(y, d, m) = \sum \frac{1}{2} y d^2 + (1 - y) max(m - d, 0)^2$ satisfait nos contraintes
  — Pouvez vous proposer une autre fonction de coût ?
  — Que faire s'il existe des données mal étiquetées dans la base d'apprentissage ?

1.4 Réseaux siamois : définir la similarité entre deux images

objectifs : comparer des images de caractères, dans des alphabets différents, et déterminer si elles sont semblables ou pas



— Entrée : des couples d'images similaires ou non. Label $\{0, 1\}$
— Sortie du réseau : une valeur dans $[0, 1]$: proche de 0 si images éloignées, proches de 1 si images similaires
— Structure du réseau : F1 et F2 de taille 4096, puis couche FC et sigmoide pour ramener les scores dans [0,1]



Question :
— Quelle est la différence par rapport au cas précedent ?

— Proposer une fonction de coût pour réaliser cet apprentissage

2 Mise en peuvre du gradient stochastique

2.1 Présentation de la méthode

L'algorithme du gradient stochastique est une méthode de descente de gradient utilisée pour la minimisation d'une fonction objectif qui est écrite comme une somme de fonctions différentiables.

C'est un cas très fréquent dans l'estimation aux moindres carrés par exemple ($min_{a,b} \sum (y(i) - ax_i - b)^2$) ainsi que dans le domaine de la classification supervisée où on minimise la somme des distances entre valeur attendue et valeur prédite par le système sur l'ensemble des données disponibles.

La plupart du temps on cherche à trouver la valeur de w minimisant le risque empirique :

$$Q(w) = \frac{1}{n} \sum_{1}^{n} Q_i(w)$$

Une méthode de descente de gradient est le plus souvent employée. Il s'agit d'une méthode itérative construisant une suite w_n définie par

$$w_{n+1} = w_n - \eta \nabla Q(w) = w_n - \eta \frac{1}{n} \sum_{1}^{n} \nabla Q_i(w)$$

η est le pas d'itération. On l'appelle fréquemment *Taux d'apprentissage*.

En apprentissage, le nombre n de données utilisées peut être très grand (des millions de données...). Dans ce cas le calcul de la somme des gradients peut prendre un temps prohibitif.

Dans la méthode de descente du gradient stochastique (SGD), on procède à une simplification drastique. Au lieu de calculer le gradient du risque empirique ∇Q exactement, chaque itération estime ce gradient sur la base d'une seule (ou d'un petit nombre de) mesure. L'algorithme réalise ainsi une mise à jour après chaque exemple.

— choix de w_0 et de η
  — for i=1 :n do
    choisir aléatoirement un exemple noté $l(i)$
    $w_{n+1} = w_n - \mu \nabla Q_{l(i)}(w)$
  — fin do

2.2 Mise en oeuvre sur un exemple jouet

Nous allons tester cette méthode sur un exemple très simple du calcul du minimum d'une fonction constituée de deux termes $f = \frac{1}{2}(f_1 + f_2)$. Au lieu de calculer le gradient de f et d'appliquer directement une descente de gradient, nous allons utiliser SGD et ainsi choisir aléatoirement une des deux fonctions et appliquer la descente de gradient soit sur f_1 soit sur f_2 en fonction du tirage effectué. La structure grossière de l'algo est écrite dans Algorithme 1).

Algorithm 1 Minimisation par gradient stochastique

$niter = 1000$
choisir aléatoirement une valeur initiale x_0
for $i = 1 : niter$ **do**
  choisir aléatoirement un nombre u entre 0 et 1
  $numfonc = (u > .5)$
  appliquer une itération de descente de gradient à partir de x_i avec $f1$ ou f_2 en utilisant $numfonc * f_1 + (1 - numfonc) * f_2$. Choisir $\mu = 1/(20 + i)$.
end for

Appliquer cet algorithme avec $f_1(x) = (x + 1)^2$ et $f_2(x) = (x - 1)^2$. Visualiser sur un graphique le tableau des x_i générés.

Répéter cette étape avec 5 initialisations différentes et afficher les résultats sur un même graphique.

Ajouter sur ce graphique le résultat obtenu par une descente de gradient *classique*.