Cours deep learning

L'entrainement d'un réseau profond

Adrien CHAN-HON-TONG
ONERA/DTIS département traitement de l'information et système

Rappel : du neurone au réseau

Le neurone

Dans les architectures de réseaux de neurones (profond ou pas), le neurone est un filtre linéaire couplé (sauf dernière couche) à une activation non linéaire par exemple $\max(0, x)$:

$$neurone_{w,b} : \begin{array}{ccc} \mathbb{R}^D & \rightarrow & \mathbb{R} \\ x & \rightarrow & \max(0, wx + b) \end{array}$$

$w \in \mathbb{R}^D$ et $b \in \mathbb{R}$ sont les **poids** du neurones.
$\max(0, x)$ est appelé **relu** - parfois noté $[x]_+$

Rappel : du neurone au réseau

La couche de neurone

Une couche de K neurones est une sequence de K neurones prenant la même entrée, et, dont les K sorties sont regoupées en 1 vecteur :

$$\begin{array}{rccc} & \mathbb{R}^D & \rightarrow & \mathbb{R}^K \\ couche_{W,b} : & & & \\ & x & \rightarrow & \begin{pmatrix} neurone_{W_1,b_1}(x) \\ \dots \\ neurone_{W_K,b_K}(x) \end{pmatrix} \end{array}$$

$W \in \mathbb{R}^{K \times D}$ et $b \in \mathbb{R}^K$ sont les **poids** de chacun des K neurones.

Rappel : du neurone au réseau

Le réseau de neurone

Un réseau de neurones entièrement connectées (multi layer perceptron en anglais) de profondeur Q est un empilement de Q couches de neurones - la dernière est un seul neurone sans activation en classification binaire :

$$reseau_\theta : \begin{array}{ccc} \mathbb{R}^D & \rightarrow & \mathbb{R} \\ x & \rightarrow & C_{W_Q,b_Q}(C_{W_{Q-1},b_{Q-1}}(...(C_{W_1,b_1}(x))...)) \end{array}$$

$\theta = (W_1, b_1, ..., W_Q, b_Q)$ est un Q uplets de **poids** de couches de neurones - abbrégé C

Rappel : classification supervisée

Algorithme d'apprentissage

- prend en entrée une **base d'apprentissage** : un ensemble d'observations étiquettées $x_1, y(x_1), ..., x_N, y(x_N)$ par la fonction désirée y (à valeur dans $\{-1, 1\}$).
- produit un modèle f : une fonction qui, à x, associe une classe $f(x) \in \{-1, 1\}$ (précisément, f est continue et son signe est utilisé comme prédicteur)
- ce modèle est évalué sur une **base de test** disjointe de la base d'apprentissage $\chi_1, y(\chi_1), ..., \chi_N, y(\chi_N)$ en calculant :

$$\frac{1}{N}\sum_{n=1}^{N}|f(\chi_i) - y(\chi_i)| \approx \int_{\mathbb{R}^D} |f(x) - y(x)|P(x)\mathrm{d}x$$

Comment apprendre le modèle

Algorithme d'apprentissage

- prend en entrée une **base d'apprentissage** : un ensemble d'observations étiquettées $x_1, y(x_1), ..., x_N, y(x_N)$ par la fonction désirée y (à valeur dans $\{-1, 1\}$).
- **produit un modèle f : une fonction qui, à x, associe une classe $f(x) \in \{-1, 1\}$ (précisément, f est continue et son signe est utilisé comme prédicteur)**
- ce modèle est évalué sur une **base de test** disjointe de la base d'apprentissage $\chi_1, y(\chi_1), ..., \chi_N, y(\chi_N)$ en calculant :

$$\frac{1}{N}\sum_{n=1}^{N}|f(\chi_i) - y(\chi_i)| \approx \int_{\mathbb{R}^D} |f(x) - y(x)|P(x)\mathrm{d}x$$

Comment apprendre les poids du modèle

Attention

Ici on suppose que la **structures** du réseau est prédéfini.
On ne cherche qu'à optimiser les **poids** des neurones.

Plan

- Descente de gradient stochastique
- Descente de gradient stochastique en machine learning
- Backpropagation

La descente de gradient

F est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$ alors
$\forall u, h \in \mathbb{R}^D$, $F(u+h) = F(u) + \nabla F_u | h + o(h)$
avec $ho(h) \underset{h \to 0}{\to} 0$ (notation petit o classique)
Donc si $\nabla F_u \neq 0$ alors il existe $\lambda > 0$ tel que $F\left(u - \lambda \nabla F_u\right) < F(u)$

La descente de gradient

pseudo code

input : F, u_0

1. $u = u_0$
2. calculer ∇F_u
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F(u - \lambda \nabla F_u) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

cet algorithme converge vers un point u^* tel que $\nabla F_u = 0$

La descente de gradient

pseudo code

input : F, u_0

1. $u = u_0$
2. calculer ∇F_u
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F(u - \lambda \nabla F_u) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

Attention

u^* peut **ne pas être un minimum global**

u^* peut **ne pas être un minimum local** (point de selle)

La convergence peut être très lente

La descente de gradient en machine learning

u correspond à θ les poids

La descente de gradient en machine learning

u correspond à θ les poids

F s'appelle la fonction de perte (*loss*), elle doit être minimale quand on a atteint le comportement espéré

La descente de gradient en machine learning

u correspond à θ les poids

F s'appelle la fonction de perte ($loss$), elle doit être minimale quand on a atteint le comportement espéré

idealement on voudrait :
$(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

$$loss\,(\theta) = \sum_{i=1}^{n} sign(-y_i\, reseau_\theta(x_i))$$

mais il nous faut une fonction dérivable

La descente de gradient en machine learning

u correspond à θ les poids

F s'appelle la fonction de perte (*loss*), elle doit être minimale quand on a atteint le comportement espéré

on peut par exemple prendre
$(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

$$loss\,(\theta) = \sum_{i=1}^{n} relu(1 - y_i\, reseau_\theta(x_i))$$

c'est la hinge loss - elle est conceptuellement simple

La descente de gradient en machine learning

u correspond à θ les poids

F s'appelle la fonction de perte (*loss*), elle doit être minimale quand on a atteint le comportement espéré

en pratique on prend généralement
$(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

$$loss(\theta) = \sum_{i=1}^{n} y_i \log(reseau_\theta(x_i)) + (1 - y_i) \log(1 - reseau_\theta(x_i))$$

c'est la binary cross entropy (suppose que $reseau_\theta(x_i) \in]0, 1[$ ce qu'on peut obtenir en utilisant une sigmoide sur le dernier neurone)

SGD

Les limites 1/3

$$loss\,(\theta) = \sum_{i=1}^{n} relu(1 - y_i\, reseau_\theta(x_i))$$

si $n = 1000000$ ça veut dire qu'à chaque fois que je veux calculer ∇f_θ je dois faire 1000000 calculs - chacun avec plusieurs étages de plusieurs filtres ! et pour trouver λ n'en parlons pas !

SGD

Les limites 2/3

$reseau_\theta(x_i)$ n'est pas vraiment dérivable (*relu* non plus)
$\rightarrow$ pseudo dérivé $\rightarrow$ on pert la certitude de converger
est ce que c'est grave sur un problème déjà non convexe ?

Les limites 3/3

NIPS 2017 (Gradient Descent Can Take Exponential Time to Escape Saddle Points) : on peut mettre un temps exponentiel à sortir d'un point de selle
est ce que c'est grave vu qu'on va faire du early stopping ?

Descente de gradient stochastique

Les limites 2 et 3 sont marginales.

La SGD (Descente de gradient stochastique) est une solution à la limite 1 et dans une moindre mesure à la limite 3 (NIPS 2017 (How to Escape Saddle Points Efficiently) : la descente de gradient perturbée s'échappe des points de selle)

Descente de gradient stochastique

loss est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$
et que $\mathit{loss}(u) = \sum_{i=1}^{n} q_i(u)$
alors, en faisant comme une descente de gradient mais en prenant 1 des q_i tiré aléatoirement à la place de *loss* et avec un pas (λ) fixe sélectionné a priori
on converge en espérance vers le minimum dans le cas convexe

Descente de gradient stochastique

pseudo code

input : $x_1, y_1, ..., x_n, y_n, \theta_0$

1. $\theta = \theta_0$
2. $iter = 0$
3. tirer i au hasard dans 1,...,n
4. $partial\_loss = relu(1 - y_i reseau_\theta(x_i))$
5. calculer $\nabla partial\_loss_\theta$
6. $\theta = \theta - \lambda_{iter} \nabla partial\_loss_\theta$
7. $iter = iter + 1$
8. si condition d'arrêt alors sortir
9. go to 3

Descente de gradient stochastique

Mais ça suppose qu'on sache calculer le gradient ! ! ! !

réseau de neurones

définitions

input : $(xin_i)_i$

variables : $(w_{t,i,j})_{t,i,j}$

convention : $x_{0,i} = xin_i, x_{t,0} = 1$

règles du forward :

- $x_{t+1,i} = relu(\alpha_{t+1,i})$
- $\alpha_{t+1,i} = \sum_j x_{t,j} w_{t,i,j}$
- $loss(w)$ peut se calculer à partir de la dernière couche

réseau de neurones

forward

```
for t
    for i
        for j
            A[t][i] += relu(A[t-1][j])*w[t-1][i][j]
```

réseau de neurones

forward

$x_{t+1,i} = relu\left(\alpha_{t+1,i}\right)$

$\alpha_{t+1,i} = \sum_j x_{t,j} w_{t,i,j}$

$loss\left(w\right)$ se calcule à partir de la dernière couche

objectif

On chercher à calculer $\frac{\partial loss}{\partial w_{t,i,j}}$

Pas trivial

réseau de neurones

Réduction w - α

$$\frac{\partial loss}{\partial w_{t,i,j}} = \frac{\partial loss}{\partial \alpha_{t,i,j}} \frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} = \frac{\partial loss}{\partial \alpha_{t,i}} x_{t,j}$$

Réduction α - α

$$\frac{\partial loss}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} w_{t,i,j} relu'(\alpha_{t,j})$$

relu est une fonction linéaire par morçeau, sa *dérivé* est donc une constante par morçeau

Attention

La somme dans $\frac{\partial loss}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}}$ ne vient **pas** de la somme dans $\alpha_{t+1,i} = \sum_j x_{t,j} w_{t,i,j}$.

Elle vient de $f(u) = a(b(u), c(u))$ implique $\frac{\partial f}{\partial u} = \frac{\partial a}{\partial b}\frac{\partial b}{\partial u} + \frac{\partial a}{\partial c}\frac{\partial c}{\partial u}$.

Lui même vient de $f(u+h) = f(u) + f'(u)h$

réseau de neurones

forward

$x_{t+1,i} = relu\left(\alpha_{t+1,i}\right)$

$\alpha_{t+1,i} = \sum_j x_{t,j} w_{t,i,j}$

$loss(w)$ se calcule à partir de la dernière couche

backward

$$\frac{\partial l}{\partial w_{t,i,j}} = \frac{\partial l}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial w_{t,i,j}} = \frac{\partial l}{\partial \alpha_{t+1,i}} x_{t,j}$$

$$\frac{\partial l}{\partial \alpha_{t,j}} = \sum_i \frac{\partial l}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}} = \sum_i \frac{\partial l}{\partial \alpha_{t+1,i}} w_{t,i,j} h'\left(\alpha_{t,j}\right)$$

réseau de neurones

forward backward

```
for t
    for i
        for j
            A[t][i] += relu(A[t-1][j])*w[t-1][i][j]
DA[z][1] = se calcule à partir de la dernière couche
for t from z to 1
    for j
        for i
            DA[t][j] += DA[t+1][i]*w[t][i][j]*relu'(A[t][j])
```

ATTENTION c'est juste un pseudo code qui NE MARCHE PAS si on fait juste un copier coller !

Conclusion

base d'apprentissage $(x_1, y(x_1)), ..., (x_n, y(x_n)) \in \mathbb{R}^D \times \{-1, 1\}$
base de test $(\chi_1, y(\chi_1)), ..., (\chi_n, y(\chi_n)) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise θ avec l'objectif que $sign(reseau_\theta(x_i)) \approx y(x_i)$ avec SGD et backpropagation

dans l'espoir que $\forall i \in \{1, ..., n\}, sign(reseau_\theta(\chi_i)) \approx y(\chi_i)$