

Cours de reconnaissance des formes

L'entrainement de réseau profond

Adrien CHAN-HON-TONG
ONERA

Rappel : du neurone au réseau

Le neurone

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

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

$w \in \mathbb{R}^D$ 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 regroupées en 1 vecteur :

$$\begin{array}{ccc} \mathbb{R}^D & \rightarrow & \mathbb{R}^K \\ \text{couche}_{W,b} : & x & \rightarrow \begin{pmatrix} \text{neurone}_{W_1,b_1}(x) \\ \dots \\ \text{neurone}_{W_K,b_K}(x) \end{pmatrix} \end{array}$$

$W \in \mathbb{R}^{K \times D}$ 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 classiquement un seul neurone (sans activation) :

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

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

Rappel : classification supervisée

Algorithme d'apprentissage

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

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

Comment apprendre le modèle

Algorithme d'apprentissage

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

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

Classification supervisée par réseau de neurones

base d'apprentissage $(x_1, y(x_1)), \dots, (x_n, y(x_n)) \in \mathbb{R}^D \times \{-1, 1\}$

base de test $(\chi_1, y(\chi_1)), \dots, (\chi_n, y(\chi_n)) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise θ avec l'objectif que $\text{sign}(\text{reseau}_{\theta}(x_i)) \approx y(x_i)$

dans l'espoir que $\forall i \in \{1, \dots, n\}, \text{sign}(\text{reseau}_{\theta}(\chi_i)) \approx y(\chi_i)$

Classification supervisée par réseau de neurones

base d'apprentissage $(x_1, y(x_1)), \dots, (x_n, y(x_n)) \in \mathbb{R}^D \times \{-1, 1\}$

base de test $(\chi_1, y(\chi_1)), \dots, (\chi_n, y(\chi_n)) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise θ avec l'objectif que $\text{sign}(\text{reseau}_{\theta}(x_i)) \approx y(x_i)$

dans l'espoir que $\forall i \in \{1, \dots, n\}, \text{sign}(\text{reseau}_{\theta}(\chi_i)) \approx y(\chi_i)$

Comment optimise t on θ ?

ATTENTION on fait ça en TP!

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

$$\forall u, h \in \mathbb{R}^D, f(u + h) = f(u) + \nabla f_u | h + \varepsilon(h)h$$

avec $\varepsilon(h) \xrightarrow{h \rightarrow 0} 0$

Donc si $\nabla f_u \neq 0$ alors il existe $\lambda > 0$ tel que $f(u - \lambda \nabla f_u) < f(u)$

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 local (a fortiori global)

La convergence peut être lente

La descente de gradient en machine learning

u correspond à θ les poids

f ?

La descente de gradient en machine learning

u correspond à θ les poids

f est la fonction de perte, elle doit être minimale quand on a atteint le comportement espéré

exemple :

$$(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$$

$$f(\theta) = \sum_{i=1}^n \text{relu}(1 - y_i \text{reseau}_{\theta}(x_i))$$

La descente de gradient en machine learning

u correspond à θ les poids

f est la fonction de perte, elle doit être minimale quand on a atteint le comportement espéré

exemple :

$$(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$$

$$f(\theta) = \sum_{i=1}^n \text{relu}(1 - y_i \text{reseau}_{\theta}(x_i))$$

La descente de gradient en machine learning

Est ce que vous voyez le truc qui va pas marcher ?

Les limites 1/3

$$f(\theta) = \sum_{i=1}^n \text{relu}(1 - y_i \text{reseau}_{\theta}(x_i))$$

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 !

Les limites 2/3

$\text{reseau}_{\theta}(x_i)$ n'est pas vraiment dérivable

→ pseudo dérivé → on perd 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

f est une fonction dérivable de \mathbb{R}^D dans \mathbb{R}

et que $f(u) = \sum_{i=1}^n q_i(u)$

alors, en faisant comme une descente de gradient mais en prenant 1 des q_i tiré aléatoirement à la place de f 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 : $x_1, y_1, \dots, x_n, y_n, \theta_0$

1. $\theta = \theta_0$
2. $iter = 0$
3. tirer i au hasard dans $1, \dots, n$
4. $f = \text{relu}(1 - y_i \text{reseau}_\theta(x_i))$
5. calculer ∇f_θ
6. $\theta = \theta - \lambda_{iter} \nabla f_\theta$
7. $iter = iter + 1$
8. si condition d'arrêt alors sortir
9. go to 3

Descente de gradient stochastique

Qu'avons nous oublié ?

Descente de gradient stochastique

Le gradient!!!!

réseau de neurones

définitions

input : $(x_{in_i})_i$

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

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

règles du forward :

► $x_{t+1,i} = \text{relu}(\alpha_{t+1,i})$

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

► $l(w) = \text{loss}(w) = \text{cout}(w) = \text{relu}(1 - x_{10,1})$

Attention, le coût et la fonction de non linéarité sont juste des exemples

réseau de neurones

forward

$A[t][i] = 0$

for t

$A[t][0] = 1$

for i

$A[0][i] = X_{in}[i]$

for t

for i

for j

$A[t][i] += \text{relu}(A[t-1][j]) * w[t-1][i][j]$

réseau de neurones

forward

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

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

$$l(w) = \text{relu}(1 - x_{10,1})$$

objectif

On cherche à calculer $\frac{\partial l}{\partial w_{t,i,j}}$

Pas trivial

réseau de neurones

objectif

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

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

$$l(w) = \text{relu}(1 - x_{10,1})$$

On cherche à calculer $\frac{\partial l}{\partial w_{t,i,j}}$

Réduction $w - \alpha$

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

réseau de neurones

Réduction $w - \alpha$

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

Ce n'est vrai **que parce que**, si on note f fonction qui a $\alpha_{t,i,j}$ associe l , g la fonction qui a $w_{t,i,j}$ associe $\alpha_{t,i,j}$, et, h la fonction qui a $w_{t,i,j}$ associe l , on a bien $h = fog$.

réseau de neurones

objectif

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

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

$$l(w) = \text{relu}(1 - x_{10,1})$$

On cherche à calculer $\frac{\partial l}{\partial w_{t,i,j}}$

Réduction $\alpha - \alpha$

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

réseau de neurones

Attention

La somme dans $\frac{\partial l}{\partial \alpha_{t,j}} = \sum_i \frac{\partial l}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}}$ 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

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

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

$$l(w) = s(x_{10,1})$$

backward

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

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

réseau de neurones

forward backward

```
A[t][i] = 0
```

```
for t
```

```
    A[t][0] = 1
```

```
for i
```

```
    A[0][i] = xin[i]
```

```
for t
```

```
    for i
```

```
        for j
```

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

```
DA[t][i] = 0
```

```
DA[z][1] =  $\frac{\partial l}{\partial \alpha_{10,1}}$ 
```

```
for t from 9 to 1
```

```
    for j
```

```
        for i
```

```
            DA[t][j] += DA[t+1][i]*w[t][i][j]*relu'(A[t][j])
```


Conclusion

ATTENTION c'est des pseudo codes qui NE MARCHENT PAS si on fait juste un copier coller !

Je suis d'accord que c'est plutôt horrible la backpropagation + SGD mais c'est comme ça...

Conclusion

base d'apprentissage $(x_1, y(x_1)), \dots, (x_n, y(x_n)) \in \mathbb{R}^D \times \{-1, 1\}$

base de test $(\chi_1, y(\chi_1)), \dots, (\chi_n, y(\chi_n)) \in \mathbb{R}^D \times \{-1, 1\}$

on optimise θ avec l'objectif que $\text{sign}(\text{reseau}_\theta(x_i)) \approx y(x_i)$ avec SGD et backpropagation

dans l'espoir que $\forall i \in \{1, \dots, n\}, \text{sign}(\text{reseau}_\theta(\chi_i)) \approx y(\chi_i)$