apprentissage profond - deep learning

partie 3 : l'apprentissage d'un réseau

Adrien CHAN-HON-TONG
ONERA

rappel

MLP

- X l'espace de **points**
- P, y, f, w
- $e_t = \frac{1}{M} \sum_m \mathbf{1}_-(y(\chi_m) f(\chi_m, w)) \approx \int_X \mathbf{1}_-(y(x) f(x, w)) P(x) \mathrm{d}x$
- l'erreur d'apprentissage $e_a = \frac{1}{N} \sum_n \mathbf{1}_-(y(x_n) f(x_n, w))$
- deep learning = choisir
  $f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$ et
  optimiser w pour avoir notamment e_a faible

Rappel

ConvNet

- X l'espace des **images**
- P, y, f, w
- $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m, w)) \approx \int_X \mathbf{1}_-(y(x)f(x, w))P(x)\mathrm{d}x$
- l'erreur d'apprentissage $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$
- deep learning = choisir
  $f(x, w) = w_Q \times poolrelu(w_{Q-1} \star poolrelu(...(poolrelu(w_1 \star x))))$
  ou plus généralement un truc qui mélange convolutions, pooling et relu
  et optimiser w pour avoir notamment e_a faible

Comment est ce qu'on optimise w tel que e_a soit petit ?

$$f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$
$$e_a = \frac{1}{N} \sum_n \mathbf{1}_-(y(x_n) f(x_n, w))$$
$x_1, ..., x_N$ une base d'apprentissage

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 $F\left(u - \lambda \nabla F_u\right) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

La descente de gradient pour nous

u correspond à w les poids et $F = e_a$?

La descente de gradient pour nous

u correspond à w les poids et $F \approx e_a$ il nous faut une fonction lisse

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

on va donc lisser :
$(x_1, y_1), ..., (x_N, y_N) \in \mathbb{R}^D \times \{-1, 1\}$

$$loss\,(w) = \sum_{n=1}^{N} \mathbf{1}_{-}(y(x_n) f(x_n, w))$$

La descente de gradient en pour nous

u **correspond à** w **les poids et** $F \approx e_a$ il nous faut une fonction lisse

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

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

$$loss\,(w) = \sum_n relu(1 - y_n f\,(x_n, w))$$

c'est la hinge loss - elle est conceptuellement simple et historiquement c'est la loss des SVM - aujourd'hui on utilise plutôt la binary cross entropy

SGD

oui mais...

$$loss\,(w) = \sum_n relu(1 - y_n f\,(x_n, w))$$

Si $N = 1000000$ ça veut dire que pour calculer $loss(w)$ je dois appliquer f (plusieurs couches) à 1000000 points ! Pire, j'ai besoin de $\nabla_w loss$

Descente de gradient stochastique

loss est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$
et que $\mathit{loss}(u) = \sum_{i=1} q_i(u)$
alors **dans le cas convexe**, en faisant comme une descente de gradient mais en prenant une sous sommes des q_i tirée aléatoirement et cela même avec une politique $\lambda(t)$ fixée a priori (qui doit quand même vérifier certaines conditions).

Descente de gradient stochastique

pseudo code

input : $x_1, y_1, ..., x_n, y_n,\ w_0$

1. $w = w_0$
2. $iter = 0$
3. tirer n au hasard dans 1,...,N
4. $partial\_loss = relu(1 - y_n f(x_n, w))$
5. calculer $\nabla_w partial\_loss$
6. $\theta = \theta - \lambda_{iter} \nabla_w partial\_loss$
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])
```

Bilan

Deep learning

- X, P, y
- $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m, w)) \approx \int_X \mathbf{1}_-(y(x)f(x, w))P(x)\mathrm{d}x$
- ce qui *marche le mieux aujourd'hui* pour avoir e_t petit c'est d'utiliser du deep learning :
- $\Rightarrow$ de prendre f comme un réseau de neurones et de minimiser une approximation lisse de $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$ via une descente de gradient stochastique

Monteriez vous dans un avion ou taxi autonome sans pilote ?

Pour le projet ATTOL (Autonomous Taxi, Take-Off and Landing), le 18 décembre dernier, une première mondiale a été accomplie : un décollage entièrement autonome basé vision, sans utilisation de l'ILS, ni du GPS, a été réalisé à plusieurs reprises sur l'aéroport de Toulouse-Blagnac. L'ONERA a contribué au développement et à la mise au point l'algorithme de fusion de données, qui élabore le signal de déviation à l'axe de piste, nécessaire au contrôle de l'avion, à partir des informations visuelles issues de la caméra et inertielles provenant des centrales à bord de l'appareil. Ainsi, contrairement aux systèmes actuels dont les signaux se trouvent hors de l'appareil, ceux-ci se trouvaient embarqués, d'où la qualification d'un *système autonome*.

https ://www.onera.fr/fr/actualites/attol-des-pilotes-dans-lavion-mais-un-systeme-autonome

Bilan

- X, P, y
- $e_t = \frac{1}{M}\sum_m \mathbf{1}_-(y(\chi_m)f(\chi_m, w)) \approx \int_X \mathbf{1}_-(y(x)f(x, w))P(x)\mathrm{d}x$
- ce qui *marche le mieux aujourd'hui* pour avoir e_t petit c'est d'utiliser du deep learning :
- $\Rightarrow$ de prendre f comme un réseau de neurones et de minimiser une approximation lisse de $e_a = \frac{1}{N}\sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$ via une descente de gradient stochastique

Le plus important c'est de comprendre le formalisme global (mon problème se met-il sous la forme X, P, y) et de rester lucide sur ce qui peut marcher et comment on doit l'évaluer (e_a vs e_t vs e_r ...)