Introduction au Deep Learning

Régularisation et sélection de modèles

J. Rynkiewicz

Université Paris 1

2022

Régularisation des réseaux de neurones (1)

- Les algorithmes d'optimisations stochastiques marchent bien en pratique. Ils convergent souvent vers un très bon minimum local de la fonction de coût.
- Comme ces modèles ont beaucoup de paramètres, il y a souvent plus de paramètres que d'observations, il faut faire attention au surapprentissage.
- Le surapprentissage correspond à un ajustement trop étroit ou exact à un ensemble particulier de données. Le modèle apprend "par coeur" les données d'apprentissage et ne peut pas généraliser sur de nouvelles données.
- Pour ne pas choisir un mauvais modèle, on découpe l'ensemble d'apprentissage en deux :
  - Une majorité des données pour estimer les paramètres du modèle (ensemble d'entrainement).
  - Une minorité de données pour estimer la généralisation du modèle (ensemble de validation).
  - En anglais, cette méthode porte le nom de "hold-out".

Régularisation des réseaux de neurones (2)

Bien qu'on choisisse le meilleur modèle à l'aide d'un ensemble de validation, Il est préférable, de controler la puissance de modélisation du MLP. Actuellement, il y a essentiellement trois méthodes utilisées en Deep Learning :

- Le weight-decay qui pénalise la fonction de coût en fonction de la taille des paramètres. L'avantage de cette technique est de stabiliser l'algorithme d'optimisation du gradient stochastique en empéchant les poids de devenir trop grands.
- Le drop out, tous les poids ne sont pas mis-à-jours à chaque mini-batch. C'est comme si devant chaque unitées cachées d'une couche, il y avait un masque qui laisse passer ou non l'information avec une probabilité p fixée, mais à choisir.
- Le mixup, qui consiste à créer des observations virtuelles qui sont un mélange des vraies observations.
- La "Batch normalization" a la réputation de régulariser le modèle mais on ne sait pas très bien quelle en est la raison. Cette technique a été introduite pour rendre la descente de gradient plus efficace.

Aucune de ces méthodes n'est exclusive, on peut très bien utiliser plusieurs en même temps.

Le weight decay

Soit $C(y_t, F_\theta(x_t))$ une fonction de coût, par exemple $C(y_t, F_\theta(x_t)) = (y_t, F_\theta(x_t))^2$, et θ le vecteur des poids du réseaux. On notera B la dimension de θ. La fonction à minimiser sera :

$$\sum_{t=1}^{n} C(y_t - F_\theta(x_t)) + \lambda \sum_{i=1}^{B} \theta_i^2$$

Les poids optimaux $\hat{\theta}$, pour le critère pénalisé, vérifierons donc :

$$\hat{\theta} = \arg\min \sum_{t=1}^{n} C(y_t - F_\theta(x_t)) + \lambda \sum_{i=1}^{B} \theta_i^2$$

Le choix de λ est un peu arbitraire, mais il est en général de l'ordre de 10^{-5} ou 10^{-6} pour des réseaux profonds avec plusieurs millions de paramètres.

Retour sur le bruit blanc

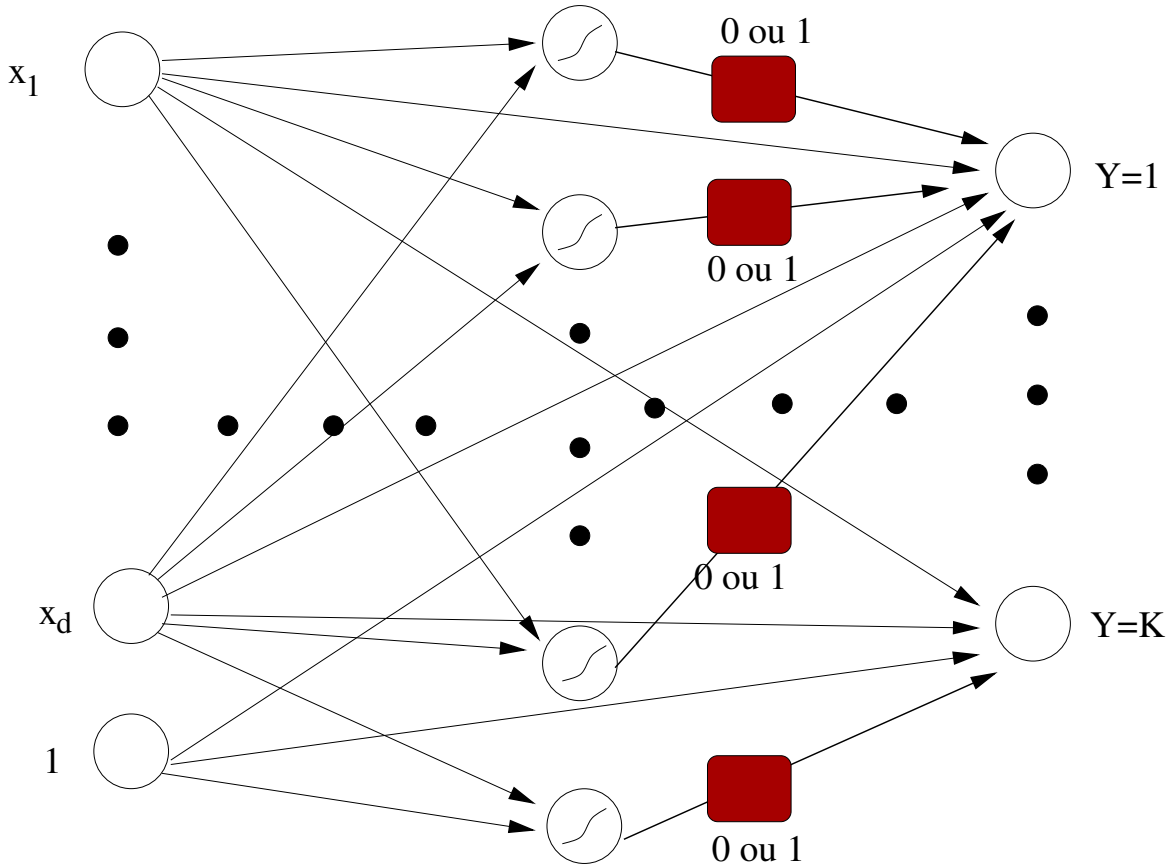
On reprend notre échantillon $((x_t(1), x_t(2), y_t)_{1 \leq t \leq 30}$, où $(X(1), X(2), Y) \sim \mathcal{N}(0, I_3)$. On pénalise maintenant le MLP avec la méthode du Weight decay et un coefficient $\lambda = 0.01$.

```
library(nnet)
library(rgl)
set.seed(1)
x <- matrix(rnorm(60),30,2)
y <- matrix(rnorm(30),30,1)
res.mod <- nnet(x,y,size=10,maxit=1000,decay=0.01,linout=T)
x1p <- runif(10000,min=min(x[,1]),max=max(x[,1]))
x2p <- runif(10000,min=min(x[,2]),max=max(x[,2]))
matp <- cbind(x1p,x2p)
mod.pred <- predict(res.mod,matp)
plot3d(c(-2,-2,2,2),c(-2,2,-2,2),c(20,20,-20,-20),type="n",
axes=F,xlab="",ylab="",zlab="")
points3d(x1p,x2p,mod.pred,col="green")
spheres3d(x[,1],x[,2],y,radius=0.5,col="red")
```

Le dropout

- Le dropout consiste à mettre un “masque aléatoire” devant les neurones d’une couche.
- Dans les logiciels de Deep Learning, l’utilisateur peut ajouter ce masque devant n’importe qu’elle couche. Les masques peuvent s’ajouter couche après couche.
- L’utilisateur doit aussi choisir la probabilité p pour laquelle le masque vaut 1 et n’inhibe pas les unités de la couche. Souvent la probabilité par défaut est $p = \frac{1}{2}$.
- Plus il y a de masques, plus p est petit, plus le réseau est régularisé. Il fera moins de surapprentissage mais il aura plus de difficulté à modéliser la bonne fonction de prédiction.
- Il n’y a pas vraiment de justification théorique de cet algorithme, mais il marche très bien en pratique.
- Référence : Srivastava et al., Dropout : A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research (2014).

Illustration du dropout



Le mixup

- Le mixup consiste à créer des observations qui mélangent les vraies observations.
- Si on note x_i et x_j les variables explicatives des exemples i et j, ainsi que y_i et y_j leurs labels associées. On crée les observations virtuelles :

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j$$

- On introduit ainsi l'information que l'interpolation linéaire des variables explicatives doit mené à l'interpolation linéaire des labels.
- Le mixup est très facile à programmer avec Pytorch ou bien tensorflow 2.4.
- Le coefficient de mélange λ est tiré au hasard suivant une loi béta dont les paramètres $\alpha > 0$ et $\beta > 0$ sont souvent égaux et entre 0.1 et 0.2.
- La loi béta a pour densité

$$f_{(\alpha,\beta)}(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}du}\mathbf{1}_{[0,1]}(x) := \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}\mathbf{1}_{[0,1]}(x)$$

La batch normalization

- Comme le dropout, la batch normalization peut s'appliquer à chaque couche.
- Si on note $z_t = (z_t(1), \cdots, z_t(N))$ le vecteur des valeurs qui entre dans une couche et B la dimension du minibatch, on calcule :

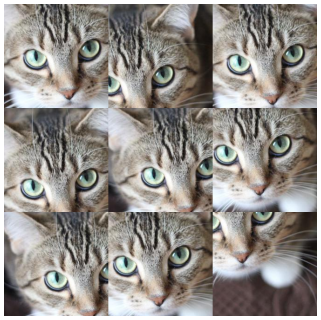
$$m(k) = \frac{1}{B}\sum_{t=1}^{B} z_t(k)$$
$$\sigma(k) = \frac{1}{B}\sum_{t=1}^{B}(z_t(k) - m(k))^2$$

- On centre et normalise les données : $\tilde{z}_t(k) = \frac{z_t(k)-m(k)}{\sigma(k)}$.
- Comme on ne peut pas être sûr que cette normalisation soit bénéfique, on introduit des nouveaux paramètres $(\alpha(k), \beta(k))$ qui permettent de rectifier la transformation (qui peuvent même l'inverser) : $\beta(k)\tilde{z}_t(k) + \alpha(k)$.
- $(\alpha(k), \beta(k))$ sont optimisés comme tous les autres paramètres du réseaux par descente de gradient.
- Cet algorithme a été introduit pour améliorer l'optimisation du réseau et on s'est rendu compte que cela permettait de limiter aussi le surapprentissage ! On ne sait pas vraiment pourquoi.

Data augmentation

Dans certain cas, comme la reconnaissance d'images, on peut créer facilement de nouveaux exemples sans changer la classe de l'image, l'augmentation du nombre d'exemples limite le surapprentissage.
hflip : en renverse la gauche et la droite.

RandomCrop : on recadre l'image de façon aléatoire.



Le hold-out

- Pendant l'apprentissage, on utilise les techniques présentées pour régulariser le réseaux :
  - weight decay
  - drop out
  - batch normalization
  - Mixup
  - data augmentation
  - Il est possible d'utiliser plusieurs de ces méthodes en même temps !
- Toutes ces techniques dépendent d'hyperparamètres, qu'il faut fixer de façon plutôt arbitraire.
- On choisit donc le meilleur modèle grâce à ses performances sur un ensemble de validation : méthode du "hold-out".
- Le découpage des données en ensemble d'entrainement, ensemble de validation est arbitraire et dépend du nombre de données disponibles.
- Souvent, on prend 10 à 20 pourcents des données pour l'ensemble de validation.

Théorie du hold-out

- On peut se demander si le modèle sélectionné par la procédure du hold-out est un bon modèle.
- Des bornes stastitiques permettent de donner une réponse à cette question.
- Cette réponse est exprimée sous la forme d'une inégalité oracle.
- C'est une inégalité qui majore la différence entre le modèle choisi par la procédure et le meilleur choix possible si on avait eu connaissance de celui-ci (l'oracle).
- On va montrer que, pour la classification, la procédure du hold-out est quasiment optimale.
- Si on a suffisamment de données, par exemple si on est dans un contexte "Big data", le hold-out est la procédure de choix pour trouver un bon modèle.

Les différentes erreurs

- Soit g_θ une fonction de classification de $\mathbb{R}^d$ dans $\{1, \cdots, K\}$.
- On dispose d'un échantillon i.i.d. $((X_1, Y_1), \cdots, (X_n, Y_n))$ où tous les couples on la même loi μ qu'un couple générique (X, Y).
- L'erreur de généralisation de la fonction g_θ sera $L(g_\theta) = E_\mu(\mathbf{1}_{g_\theta(X)\neq Y})$, où $\mathbf{1}_{g_\theta(X)\neq Y}$ vaut 1 si $g_\theta(X)$ est différent de Y et 0 sinon.
- L'erreur d'apprentissage de la fonction g_θ sera

$$\hat{L}_n(g_\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{g_\theta(X_i)\neq Y_i}$$

- Si on suppose qu'on dispose d'un ensemble de valitation $((X_1, Y_1), \cdots, (X_m, Y_m))$ indépendant de l'ensemble d'apprentissage $((X_1, Y_1), \cdots, (X_n, Y_n))$, l'erreur de validation sera :

$$\hat{L}_m(g_\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{g_\theta(X_i)\neq Y_i}$$

Compromis biais-variance

- On estime les poids du réseau en minimisant l'opposée de la log-vraisemblance, on obtient alors le réseau $g_{\hat{\theta}_n}$.
- $g_{\hat{\theta}_n}$ est implicitement une fonction des données d'apprentissage $((X_1, Y_1), \cdots, (X_n, Y_n))$. On note $\mathbb{E}L(g_{\hat{\theta}_n})$ l'espérance de l'erreur d'apprentissage par rapport à la loi de $((X_1, Y_1), \cdots, (X_n, Y_n))$.
- On note L^* l'erreur optimale de l'oracle g^*, si les modèles possibles peuvent être très complexes alors ils peuvent se rapprocher très près de l'oracle g^* (le biais est faible).
- Si les modèles possibles peuvent être très complexes, ils risquent d'avoir un surapprentissage élevé sur la base d'apprentissage (la variance est grande).
- On cherche la famille de modèle qui réalise le meilleur compromis biais-variance : elle est suffisament riche pour approcher g^* et suffisamment petite pour ne pas trop surapprendre.
- Cela revient trouver le modèle qui va minimiser $\mathbb{E}L(g_{\hat{\theta}_n}) - L^*$.

Borne de généralisation

- Soit une famille finie $\{g_{\hat{\theta}_1}, \cdots, g_{\hat{\theta}_N}\}$ de fonctions estimées sur l'ensemble d'apprentissage. On note

$$\tilde{k} = \arg \min_{k \in \{1,\cdots,N\}} L\left(g_{\hat{\theta}_k}\right) \text{ et } \hat{k} = \arg \min_{k \in \{1,\cdots,N\}} \hat{L}_m\left(g_{\hat{\theta}_k}\right)$$

- On aura

$$P\left(L\left(g_{\hat{\theta}_{\hat{k}}}\right) - L\left(g_{\hat{\theta}_{\tilde{k}}}\right) > \varepsilon\right) \leq 2Ne^{-2m\varepsilon^2}.$$

- On peut aussi montrer l'inégalité oracle :

$$E\left(L\left(g_{\hat{\theta}_{\hat{k}}}\right) - L(g^*)\right) \leq L\left(g_{\hat{\theta}_{\tilde{k}}}\right) - L(g^*) + 2\sqrt{\frac{\log N}{2m}}$$

- Ainsi, l'erreur du modèle sélectionné est proche de celle de l'oracle si il y a une fonction dans la famille $\{g_{\hat{\theta}_1}, \cdots, g_{\hat{\theta}_N}\}$ qui a une erreur proche de l'oracle.

Une inégalité oracle sous condition de bruit

Théorème

Soit $\left(g_{\hat{\theta}_k}\right)_{1\leq k\leq N}$ *une famille de classifieurs obtenus sur l'ensemble d'apprentissage. Soit* $\hat{k}$ *l'indice qui minimise le risque empirique sur l'ensemble de validation :*

$$\hat{k} = \arg \min_{k\in\{1,\cdots,N\}} \hat{L}_m(g_{\hat{\theta}_k}) = \arg \min_{k\in\{1,\cdots,N\}} \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{g_{\hat{\theta}_k}(X_i)\neq Y_i}$$

Soit une fonction $w(.)$ *tel que, pour tout classifieur* g*,*

$$\sqrt{Var\left[\mathbf{1}_{g\neq g^*}\right]} \leq w(L(g) - L(g^*)) \text{ et telle que } \frac{w(x)}{\sqrt{x}} \text{ soit non-croissante.}$$

Soit τ^* *la plus petite solution positive de* $w(\epsilon) = \sqrt{m}\epsilon$*, si* $\theta \in]0;1[$*, alors :*

$$E\left(L(g_{\hat{\theta}_{\hat{k}}}) - L(g^*)\right) \leq (1+\theta)\inf_{k\in\{1,\cdots,N\}}\left[\mathbb{E}L(g_{\hat{\theta}_k}) - L^* + \left(\frac{8}{3m} + \frac{4\tau^*}{\theta}\right)(\log(N)+1)\right].$$

Condition de Mammem-Tsybakov

- Soit $\alpha \in [0,1]$, la condition de Mammem Tsybakov peut s'écrire ainsi : $\exists \beta > 0,\ \forall g \in \{0,1\}^{\mathcal{X}},\ E\left(\mathbf{1}_{g(X) \neq g^*(X)}\right) \leq \beta \left(L(g) - L(g*)\right)^{\alpha}$.
- De plus, si $\eta(X) = E(Y = 1|X)$, et il existe $s > 0$ tel que $|2\eta(X) - 1| > s$, presque sûrement, alors le cas $\alpha = 1$est réalisé.
- Si une telle condition est vraie avec exposant α, alors on peut choisir $w(r) = \left(\frac{r}{h}\right)^{\frac{\alpha}{2}}$, pour un h positif et $\tau^* = \left(\frac{1}{mh^{\alpha}}\right)^{-\frac{1}{2-\alpha}}$.
- L'inégalité du théorème précédent devient :

$$E\left(L\left(g^*_{\theta_{\hat{k}}}\right) - L(g^*)\right) \leq$$
$$(1+\theta)\left(\left(L\left(g^*_{\theta_{\tilde{k}}}\right) - L(g^*)\right) + \left(\frac{8}{3m} + \frac{4}{\theta(mh^{\alpha})^{\frac{1}{2-\alpha}}}\right)(\log(N)+1)\right)$$

- Une convergence rapide de vitesse $\frac{1}{m}$ est atteinte si $\alpha = 1$.