Introduction to Deep Learning

Regularization and model selection

J. Rynkiewicz

Université Paris 1

2022

Regularization of neural networks (1)

- Stochastic optimization algorithms work well in practice. They often converge to a very good local minimum of the cost function.
- As these models have many parameters, there are often more parameters than observations, so we must be careful about overfitting.
- Overfitting corresponds to a too close or exact fit to a particular data set. The model learns "by heart" the training data and cannot generalize on new data.
- To avoid choosing the wrong model, we split the training set in two :
  - A majority of the data to estimate the model parameters (training set).
  - A minority of data to estimate the generalization of the model (validation set).
  - This method is called “hold-out”.

Regularization of neural networks (2)

Although the best model is chosen using a validation set, it is preferable to control the modeling power of neural networks.
Currently, there are mainly three methods used in Deep Learning :

- The weight-decay which penalizes the cost function according to the size of the parameters. The advantage of this technique is to stabilize the stochastic gradient optimization algorithm by preventing the weights from becoming too large.
- The drop out, not all weights are updated at each mini-batch. It is as if in front of each hidden unit of a layer, there was a mask that lets the information pass or not with a fixed probability p, but to be chosen.
- The mixup, which consists in creating virtual observations that are a mix of real observations.
- Batch normalization has the reputation of regularizing the model but it is not clear why. This technique was introduced to make gradient descent more efficient.
- This is not an exhaustive list !

None of these methods is exclusive, it is possible to use several at the same time.

The weight decay

Let $C(y_t, F_\theta(x_t))$ be a cost function, for example $C(y_t, F_\theta(x_t)) = (y_t, F_\theta(x_t))^2$, and θ the weights vector of the net. Let B be the dimension of θ. The function to minimize will be :

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

The optimal weights $\hat{\theta}$, for the penalized criterion, will check :

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

The choice of λ is somewhat arbitrary, but it is usually in the order of 10^{-5} or 10^{-6} for deep networks with several million parameters.

Back to white noise

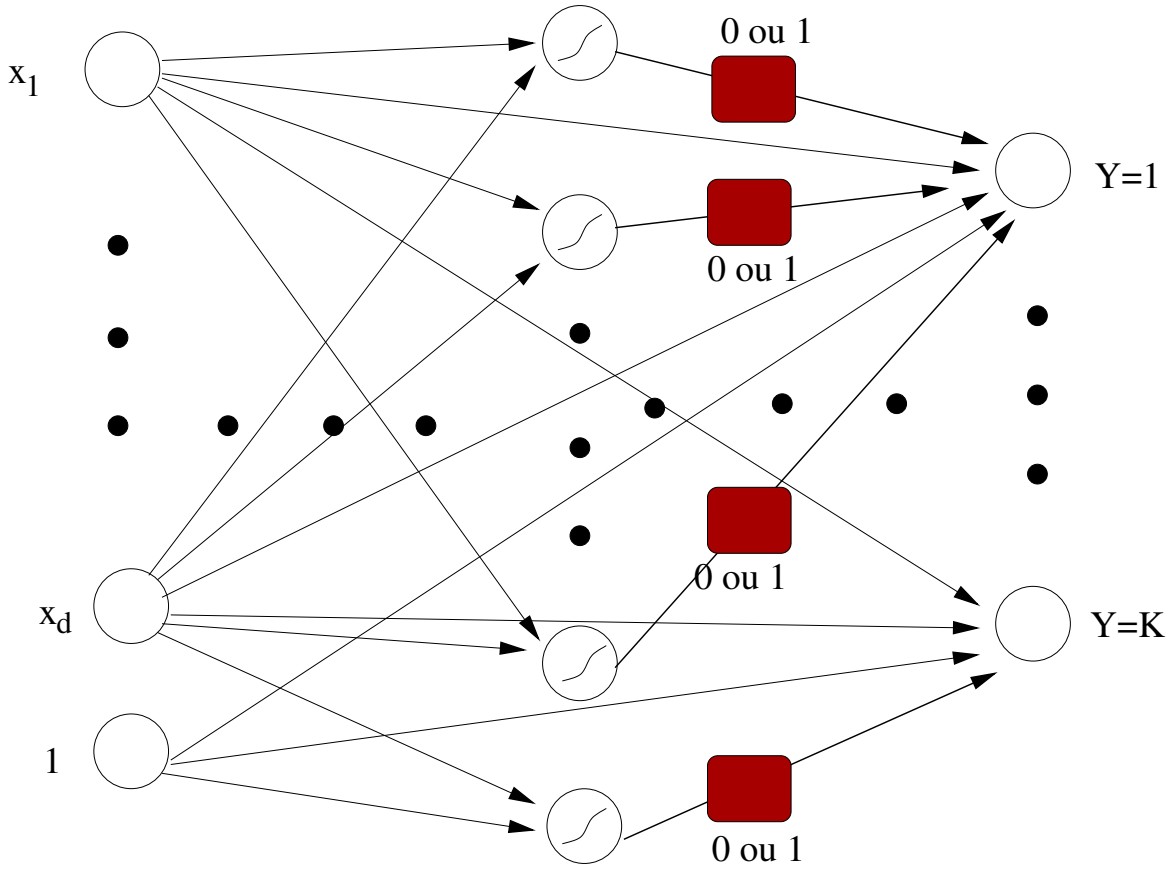
Let's go back to our sample $((x_t(1), x_t(2), y_t)_{1 \leq t \leq 30}$, where $(X(1), X(2), Y) \sim \mathcal{N}(0, I_3)$. We now penalize the MLP with the weight decay method and a coefficient $\lambda = 0.01$.

```
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")
```

The dropout

- The dropout consists in putting a "random mask" in front of the neurons of a layer.
- In Deep Learning software, the user can add this mask in front of any layer. Masks can be added layer by layer.
- The user must also choose the probability p for which the mask is 1 and does not inhibit the layer units. Often the default probability is $p = \frac{1}{2}$.
- The more masks there are, the smaller p is, the more regularized the network is. It will do less overfitting but it will have more difficulty to model the right prediction function.
- There is no real theoretical justification for this algorithm, but it works very well in practice.
- Reference : Srivastava et al., Dropout : A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research (2014).

Illustration of the dropout



The mixup

- Mixup is about creating observations that mix real observations.
- If we denote x_i and x_j the explanatory variables of examples i and j, and y_i and y_j their associated labels. We create the virtual observations :

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

- We thus introduce the information that the linear interpolation of the explanatory variables must lead to the linear interpolation of the labels.
- The mixup is very easy to program with Pytorch or tensorflow 2.4.
- The mixing coefficient λ is drawn randomly according to a beta law whose parameters $\alpha > 0$ and $\beta > 0$ are often equal and between 0.1 and 0.2.
- The beta law has density function :

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

The batch normalization

- Like dropout, batch normalization can be applied to each layer.
- If we denote $z_t = (z_t(1), \cdots, z_t(N))$ the vector of values that enters a layer and B the dimension of the minibatch, we compute :

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

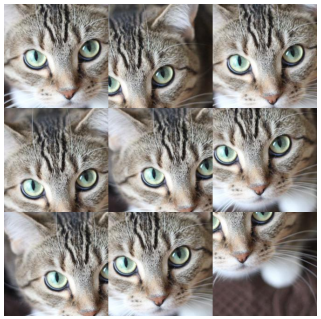
- We center and normalize the data : $\tilde{z}_t(k) = \frac{z_t(k)-m(k)}{\sigma(k)}$.
- Since we cannot be sure that this normalization is beneficial, we introduce new parameters $(\alpha(k), \beta(k))$ which allow us to rectify the transformation (which can even reverse it) : $\beta(k)\tilde{z}_t(k) + \alpha(k)$.
- $(\alpha(k), \beta(k))$ are optimized like all the other parameters of the network by gradient descent.
- This algorithm was introduced to improve the optimization of the network and it was found that it also limited overfitting ! We don't really know why.

Data augmentation

In some cases, such as image recognition, one can easily create new examples without changing the class of the image, increasing the number of examples limits overlearning.

hflip : We reverse the left and the right.

RandomCrop : The image is cropped randomly.



The hold-out

- During the training, the techniques presented are used for models regularization :
  - weight decay
  - drop out
  - batch normalization
  - data augmentation
  - It is possible to use several of these methods at the same time !
- All these techniques depend on hyperparameters, which must be set in a rather arbitrary way.
- We choose the best model thanks to its performances on a validation set : The hold-out method.
- The division of the data into training set and validation set is arbitrary and depends on the number of available data.
- Often 10 to 20 percent of the data is taken for the validation set.

Hold-out theory

- It is questionable whether the model selected by the hold-out procedure is a good model.
- Statistical benchmarks provide an answer to this question.
- This answer is expressed as an oracle inequality.
- It is an inequality that increases the difference between the model chosen by the procedure and the best possible choice if we had known about it (the oracle).
- We will show that, for classification, the hold-out procedure is almost optimal.
- If you have enough data, for example if you are in a big data context, the hold-out is the procedure of choice to find a good model.

The different errors

- Let g_θ be a classification function from $\mathbb{R}^d$ into $\{1, \cdots, K\}$.
- We have an i.i.d. sample. $((X_1, Y_1), \cdots, (X_n, Y_n))$ where all couples have the same law μ as a generic couple (X, Y).
- The generalization error of the function g_θ sera $L(g_\theta) = E_\mu(\mathbf{1}_{g_\theta(X) \neq Y})$, where $\mathbf{1}_{g_\theta(X) \neq Y}$ is 1 if $g_\theta(X)$ is not equal to Y and 0 otherwise.
- The learning error of the g_θ function will be

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

- If we assume that we have a validation set $((X_1, Y_1), \cdots, (X_m, Y_m))$ independent of the training set $((X_1, Y_1), \cdots, (X_n, Y_n))$, the validation error will be :

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

Bias-variance trade-off

- We estimate the weights of the network by minimizing the opposite of the log-likelihood, we obtain the network $g_{\hat{\theta}_n}$.
- $g_{hat\theta_n}$ is implicitly a function of the training data $((X_1, Y_1), \cdots, (X_n, Y_n))$. We denote $\mathbb{E}L(g_{hat\theta_n})$ the expectation of the learning error with respect to the distribution of $((X_1, Y_1), \cdots, (X_n, Y_n))$.
- We note L^* the optimal error of the oracle g^*, if the possible models can be very complex then they can be very close to the oracle g^* (the bias is low).
- If the possible models can be very complex, they are likely to have a high overlearning on the learning base (the variance is large).
- We look for the family of models that achieves the best bias-variance compromise : it is rich enough to approximate g^* and small enough not to overfit.
- This means finding the model that will minimize $\mathbb{E}L(g_{\hat{\theta}_n}) - L^*$.

Generalization bound

- Let be a finite family $\{g_{hat\theta_1}, \cdots, g_{hat\theta_N}\}$ of functions estimated on the training set. Let us denote

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

- We will have

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

- We can also show the oracle inequality :

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

- Thus, the error of the selected model is close to that of the oracle if there is a function in the family $\{g_{hat\theta_1}, \cdots, g_{hat\theta_N}\}$ which has an error close to the oracle.

An oracle inequality under noise condition

Théorème

Let $\left(g_{\hat{\theta}_k}\right)_{1\leq k\leq N}$ *be a finite family of classification functions estimated on the training set. Let* $\hat{k}$ *the index that minimizes the empirical risk on the validation set :*

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

Let be a function $w(.)$ *such that, for any classifier* g*,*

$$\sqrt{Var\left[\mathbf{1}_{g\neq g^*}\right]} \leq w(L(g) - L(g^*)) \text{ and such that } \frac{w(x)}{\sqrt{x}} \text{ is non-increasing.}$$

Let τ^* *be the smallest positive solution of* $w(\epsilon) = \sqrt{m}\epsilon$*, if* $\theta \in]0;1[$*, then :*

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

Mammem-Tsybakov condition

- Let $\alpha \in [0,1]$, the condition of Mammem Tsybakov can be written as follows : $\exists \beta > 0$, $\forall g \in \{0,1\}^{\mathcal{X}}$, $E\left(\mathbf{1}_{g(X) \neq g^*(X)}\right) \leq \beta \left(L(g) - L(g*)\right)^{\alpha}$.
- Moreover, if $\eta(X) = E(Y = 1|X)$, and $s > 0$ exists such that $|2\eta(X) - 1| > s$, almost surely, then the case $\alpha = 1$ is realized.
- If such a condition is true with exponent α, then we can choose $w(r) = \left(\frac{r}{h}\right)^{\frac{\alpha}{2}}$, for a h positive and $\tau^* = \left(\frac{1}{mh^{\alpha}}\right)^{-\frac{1}{2-\alpha}}$.
- The inequality of the previous theorem becomes :

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

- A fast convergence of speed $\frac{1}{m}$ is reached if $\alpha = 1$.