Introduction to Deep Learning

Convolutional networks

J. Rynkiewicz

Université Paris 1

2022

Supervised classification : example of CIFAR10

We want to predict a class (or category) Y according to a variable X : $Y = f(X)$. Y has value in E, a finite set of cardinal K and X has value in $\mathbb{R}^d$. To illustrate this problem we will consider the image set CIFAR10, where E is a set of ten categories :

1. Planes
2. Cars
3. Birds
4. Cats
5. Deers
6. Dogs
7. Frogs
8. Horses
9. Boats
10. Trucks

X is a 32×32 image on 3 color channels (RGB), so $d \simeq 3000$.

Estimation (learning) of a model (1)

- We want to estimate f_θ using observations $\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix}\right)$, (learning sample).
- We want this estimation to be accurate on new observations (which are not in the training set). This is the "generalization" capacity of the model, it is estimated on a test set : $\left(\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix}, \cdots, \begin{pmatrix} x_{n+T} \\ y_{n+T} \end{pmatrix}\right)$.

In the CIFAR10 example, $n = 50000$ et $T = 10000$. If we note $f_{\hat{\theta}}$ the estimate of the classification function a measure of the performance could be :

- $\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{y_i\}}(f_{\hat{\theta}}(x_i))$ (correct classification rate for the learning set)
- $\frac{1}{T}\sum_{i=1}^{T} \mathbf{1}_{\{y_{n+i}\}}((f_{\hat{\theta}} x_{n+i}))$ (correct classification rate for the test set)

Estimation (learning) of a model (2)

- The correct classification rate is not easy to manipulate directly for learning.
- Instead, we use the conditional likelihood of the observations to estimate f_θ.
- We note $g_\theta(k, x_i)$, the conditional probability of $Y = k$, if we observe x_i : $g_\theta(k, x_i) = P_\theta(Y_i = k \,|X_i = x_i\,)$.
- The conditional log-likelihood is written :

$$\ln\left(L_\theta\left(\left(\begin{array}{c} x_1 \\ y_1 \end{array}\right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array}\right)\right)\right) = \sum_{i=1}^n \sum_{k=1}^K \mathbf{1}_{\{k\}}(y_i) \ln\left(g_\theta(k, x_i)\right).$$

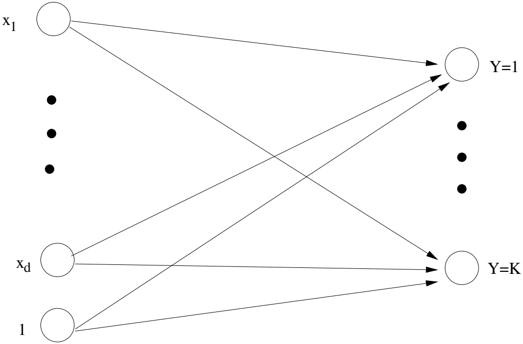
- We minimize the opposite of the conditional log-likelihood using a gradient descent.

The generalized linear model

The simplest model for the g function is the generalized linear model :

$$P(Y=k\,|X=x_i\,)=g_\theta(k,x_i)=\frac{\exp\left(\alpha_k+\beta_k^T x_i\right)}{\sum_{l=1}^K \exp\left(\alpha_l+\beta_l^T x_i\right)}$$

with $\theta=(\alpha_1,\beta_1,\cdots,\alpha_K,\cdots,\beta_K)$. We can schematize this model by :

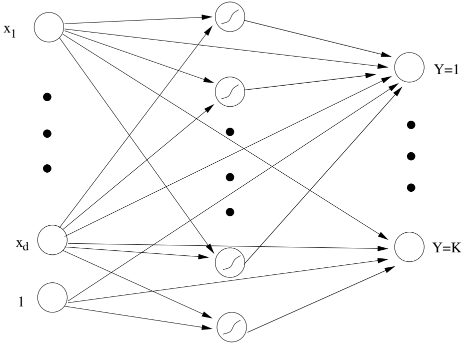


The one hidden layer MLP

We can add non-linearities in the g function :

$$P(Y=k\,|X=x_i\,)=g_\theta(k,x_i)=\frac{\exp\left(F_{\theta_k}(x_i)\right)}{\sum_{l=1}^{K}\exp\left(F_{\theta_l}(x_i)\right)}$$

with $F_{\theta_k}(x)=\alpha_k+\beta_k^T x+\sum_{h=1}^{H}a_h\sigma(b_k+W_k^T x)$ a perceptron function with a hidden layer and a skip layer. We can schematize this model by :



Application of these two models to CIFAR10

We estimate these models on the CIFAR10 database, thanks to the R library "nnet". For both models, the conditional maximum likelihood estimator is computed thanks to a gradient batch algorithm (BFGS).

- The generalized linear model :

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\left(\begin{array}{c} x_1 \\ y_1 \end{array} \right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array} \right) \right) \right).$$

  The number of parameters is about 30000, after several days of computation we obtain :

- The MLP has a hidden layer with a skip layer and ten hidden units. The conditional log-likelihood is penalized by the squared norm of the parameter vector to limit a bit the possible overfitting.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\left(\begin{array}{c} x_1 \\ y_1 \end{array} \right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array} \right) \right) \right) + \mu \|\theta\|^2, \text{ où } \mu = 10^{-9}.$$

  The number of parameters is about 60000, after more than a week of calculation with the BFGS we obtain :

Application of these two models to CIFAR10

We estimate these models on the CIFAR10 database, thanks to the R library "nnet". For both models, the conditional maximum likelihood estimator is computed thanks to a gradient batch algorithm (BFGS).

- The generalized linear model :

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\left(\begin{array}{c} x_1 \\ y_1 \end{array} \right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array} \right) \right) \right).$$

  The number of parameters is about 30000, after several days of computation we obtain :

  - Correct classification rate for the learning set : 54.19%.
  - Correct classification rate for the test set : 34.30%.

- The MLP has a hidden layer with a skip layer and ten hidden units. The conditional log-likelihood is penalized by the squared norm of the parameter vector to limit a bit the possible overfitting.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\left(\begin{array}{c} x_1 \\ y_1 \end{array} \right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array} \right) \right) \right) + \mu \|\theta\|^2, \text{ où } \mu = 10^{-9}.$$

  The number of parameters is about 60000, after more than a week of calculation with the BFGS we obtain :

Application of these two models to CIFAR10

We estimate these models on the CIFAR10 database, thanks to the R library "nnet". For both models, the conditional maximum likelihood estimator is computed thanks to a gradient batch algorithm (BFGS).

- The generalized linear model :

$$\hat{\theta} = \arg\min_{\theta} - \ln\left(L_\theta\left(\left(\begin{array}{c} x_1 \\ y_1 \end{array}\right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array}\right)\right)\right).$$

  The number of parameters is about 30000, after several days of computation we obtain :
  - Correct classification rate for the learning set : 54.19%.
  - Correct classification rate for the test set : 34.30%.
- The MLP has a hidden layer with a skip layer and ten hidden units. The conditional log-likelihood is penalized by the squared norm of the parameter vector to limit a bit the possible overfitting.

$$\hat{\theta} = \arg\min_{\theta} - \ln\left(L_\theta\left(\left(\begin{array}{c} x_1 \\ y_1 \end{array}\right), \cdots, \left(\begin{array}{c} x_n \\ y_n \end{array}\right)\right)\right) + \mu\|\theta\|^2, \text{ où } \mu = 10^{-9}.$$

  The number of parameters is about 60000, after more than a week of calculation with the BFGS we obtain :
  - Correct classification rate for the learning set : 58.80%.
  - Correct classification rate for the test set : 32.47%.

Classification of CIFAR10 with a deep network

- The network learned using the stochastic gradient.
- As the updates of θ are not done after the passage on all the data, it is easy to transform the data in a random way to enrich the training set. These transformations are :
  - Reversal of the left and right of the image ("h-flip").
  - Small random cropping of the image ("Random-crop").
- The conditional log-likelihood is penalized by $10^{-5} \times \|\theta\|^2$.
- We split the learning set : 40000 examples for training and 10000 examples for the validation.
  - After each run on the training set, the average classification error on the validation base is evaluated.
  - In the end, the model with the best validation error will be chosen (hold-out method).
- After the training (about 1h30 with a graphic card) we obtain the following results :

Classification of CIFAR10 with a deep network

- The network learned using the stochastic gradient.
- As the updates of θ are not done after the passage on all the data, it is easy to transform the data in a random way to enrich the training set. These transformations are :
  - Reversal of the left and right of the image (“h-flip”).
  - Small random cropping of the image (“Random-crop”).
- The conditional log-likelihood is penalized by $10^{-5} \times \|\theta\|^2$.
- We split the learning set : 40000 examples for training and 10000 examples for the validation.
  - After each run on the training set, the average classification error on the validation base is evaluated.
  - In the end, the model with the best validation error will be chosen (hold-out method).
- After the training (about 1h30 with a graphic card) we obtain the following results :
  - Correct classification rate for the learning set : 96.93%

Classification of CIFAR10 with a deep network

- The network learned using the stochastic gradient.
- As the updates of θ are not done after the passage on all the data, it is easy to transform the data in a random way to enrich the training set. These transformations are :
  - Reversal of the left and right of the image ("h-flip").
  - Small random cropping of the image ("Random-crop").
- The conditional log-likelihood is penalized by $10^{-5} \times \|\theta\|^2$.
- We split the learning set : 40000 examples for training and 10000 examples for the validation.
  - After each run on the training set, the average classification error on the validation base is evaluated.
  - In the end, the model with the best validation error will be chosen (hold-out method).
- After the training (about 1h30 with a graphic card) we obtain the following results :
  - Correct classification rate for the learning set : 96.93%
  - Correct classification rate for the validation set : 95.44%

Classification of CIFAR10 with a deep network

- The network learned using the stochastic gradient.
- As the updates of θ are not done after the passage on all the data, it is easy to transform the data in a random way to enrich the training set. These transformations are :
  - Reversal of the left and right of the image ("h-flip").
  - Small random cropping of the image ("Random-crop").
- The conditional log-likelihood is penalized by $10^{-5} \times \|\theta\|^2$.
- We split the learning set : 40000 examples for training and 10000 examples for the validation.
  - After each run on the training set, the average classification error on the validation base is evaluated.
  - In the end, the model with the best validation error will be chosen (hold-out method).
- After the training (about 1h30 with a graphic card) we obtain the following results :
  - Correct classification rate for the learning set : 96.93%
  - Correct classification rate for the validation set : 95.44%
  - Correct classification rate for the test set : 92.26%

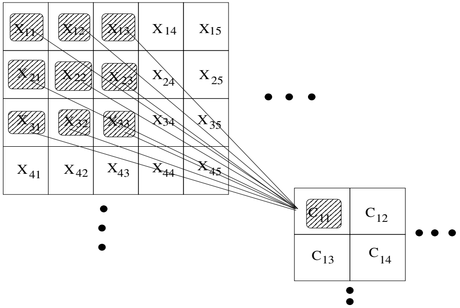
Convolutional Networks

- These impressive results, compared to the classical models of the 1990s, are obtained using a convolutional network.
- The network used in this example is called a VGG-16.
- This network links convolutional layers and max-pooling functions.
- It ends with a layer without constraint (dense layer).
- The original network was used on much more detailed images, it has been adapted to CIFAR10 images.
- There are even better networks (such as the Resnet) but the VGG is particularly easy to study.

Convolutional layer

A convolutional layer means introducing equality constraints between many weights :

Convolution 3×3

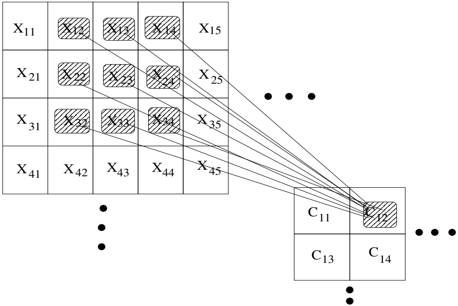


- $C_{11} = \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ij} X_{i,j}$

Convolutional layer

A convolutional layer means introducing equality constraints between many weights :

Convolution 3×3



- $C_{11} = \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ij} X_{i,j}$
- $C_{12} = \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ij} X_{i,j+1}$

Channels of convolutional layers

- The convolutional layers are grouped in parallel channels.
- For the n-th layer, there are $C(n)$ parallel channels.
- The general equations for the $C_{ijk}(n)$ neuron of layer n and channel k will therefore be :

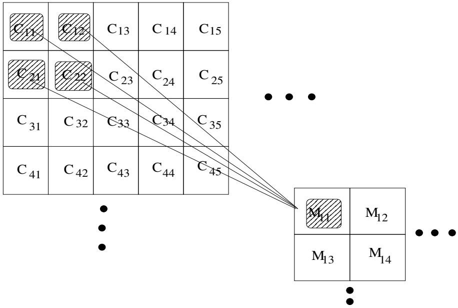
$$C_{ijk}(n) = \sum_{k=1}^{C(n-1)} \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ijk} C_{i+i',j+j',k}(n-1)$$

- For VGGs, the number of channels increases after each pass through the max-pooling function because it reduces the surface of these layers.
- In the end, there are millions of weights in a VGG, but they are extremely constrained in the convolutional layers.

Layer of “max-pooling”

A max-pooling layer computes the maximum of several units on small areas :

Max-pooling 2×2

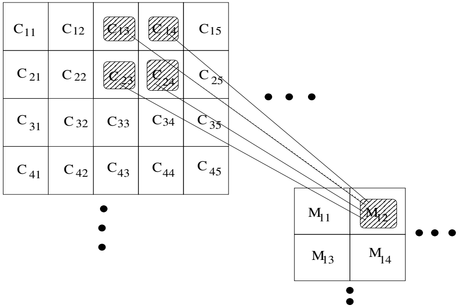


- $M_{11} = \max_{i,j=1}^{i,j=2} C_{ij}$

Layer of “max-pooling”

A max-pooling layer computes the maximum of several units on small areas :

Max-pooling 2 $\times$ 2

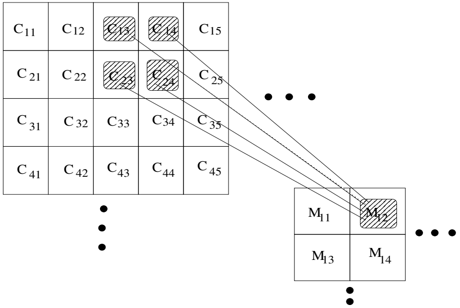


- $M_{11} = \max_{i,j=1}^{i,j=2} C_{ij}$
- $M_{12} = \max_{i,j=1}^{i,j=2} C_{i,j+2}$

Layer of “max-pooling”

A max-pooling layer computes the maximum of several units on small areas :

Max-pooling 2 × 2



- $M_{11} = \max_{i,j=1}^{i,j=2} C_{ij}$
- $M_{12} = \max_{i,j=1}^{i,j=2} C_{i,j+2}$
- The output of such a layer will have a height and width divided by two !

VGG-16

The "VGG-16" deep network links convolution layers 3×3 (followed by a non-linearity ReLU) : $\sigma(x) = \max(0, x)$) and layers of "max-pooling" 2×2. In order not to reduce the size of the image layer by the convolutions we add a 0 frame around the input layer (padding).
If we write :

- CV3 the convolution layers 3×3.
- ReLU, the application of the ReLU non-linearity on each of the units.
- MP2 the max-pooling layers 2×2.
- $NC \times W \times H$ the dimension of the layers (number of channels NC, width W, height H)
- $LIN(NI, NO)$ a dense linear layer of dimensions NI for the inputs and NO for the output.

the architecture of a VGG-16 will be :

$$
\begin{aligned}
&(NC = 3, W = 32, H = 32) \overset{ReLU(CV3)}{\rightarrow} (NC = 64, W = 32, H = 32) \overset{ReLU(CV3)}{\rightarrow}\\
&(NC = 64, W = 32, H = 32) \overset{MP2}{\rightarrow} (NC = 64, W = 16, H = 16) \overset{ReLU(CV3)}{\rightarrow}\\
&(NC = 128, W = 16, H = 16) \overset{ReLU(CV3)}{\rightarrow} (NC = 128, W = 16, H = 16) \overset{MP2}{\rightarrow}\\
&(NC = 128, W = 8, H = 8) \overset{ReLU(CV3)}{\rightarrow} (NC = 256, W = 8, H = 8) \overset{ReLU(CV3)}{\rightarrow}\\
&(NC = 256, W = 8, H = 8) \overset{ReLU(CV3)}{\rightarrow} (NC = 256, W = 8, H = 8) \overset{MP2}{\rightarrow}\\
&(NC = 256, W = 4, H = 4) \overset{ReLU(CV3)}{\rightarrow} (NC = 512, W = 4, H = 4) \overset{ReLU(CV3)}{\rightarrow}\\
&(NC = 512, W = 4, H = 4) \overset{ReLU(CV3)}{\rightarrow} (NC = 512, W = 4, H = 4) \overset{MP2}{\rightarrow}\\
&(NC = 512, W = 2, H = 2) \overset{ReLU(CV3)}{\rightarrow} (NC = 512, W = 2, H = 2) \overset{ReLU(CV3)}{\rightarrow}\\
&(NC = 512, W = 2, H = 2) \overset{MP2}{\rightarrow} (NC = 512, W = 1, H = 1) \overset{LIN(512,10)}{\rightarrow} (NC = 1, W = 1, H = 10)
\end{aligned}
$$

Estimation of VGG-16 parameters

The update of the parameters is done by mini-batch of 128 observations. It is a variant of the stochastic gradient algorithm :

- $\theta_{n+1} = \theta_n - \gamma \frac{1}{128} \sum_{i=1}^{128} \frac{\partial \ln(L_\theta(x_{t+i}, y_{t+i}))}{\partial \theta}$. During the algorithm, γ decreases from 0.1, to 0.01.
- To accelerate the descent of the gradient, a momentum of 0.9 (see the 2nd course).
- The conditional log-likelihood is penalized by $10^{-5} \times \|\theta\|^2$.
- We select the model with the best validation error (hold-out method).
- We recall that the correct classification rate for the test set of this model is : 92.26%

Other convolutional networks

- The main progress in the classification performance of convolutional networks is related to modifications in the architecture of these models.
- Although they differ in architecture, all these networks combine convolutional and pooling layers.
- One of the most famous is the Resnet, which introduces "skip layers" connections that allow for better gradient calculations.
- With the Resnet model, we can increase the number of hidden layers (up to 150 !).
- There are also models to reduce the number of parameters while keeping the performance as possible. This allows them to be used in "small" computers (smartphones).