RDFIA
Sorbonne University
2023-2024

Deep Learning Practical Work 1-a and 1-b
Introduction to neural networks

Eyal COHEN, Eline POT

1 Section 1: Theoretical foundation

1.1 Supervised dataset

Question 1. ★What are the train, val and test set?

The train set is used to train the model, the validation set is used to tune and optimize the hyper-parameters, and the test set is used to evaluate the model on new data.

Question 2. What is the influence of the number of examples N?

The number of examples N influences the "generalizability" of our model. A limited set tends to underepresent the real world and leads to an underperforming model. E.g. A model trained on classifying dogs and cat cannot extract information on the classification (set the weights, ...) from 10 pictures of dogs, especially if they are not diverse enough. A model which has only been trained on dobermans on grass will not be able to classify dogs from cat accurately.

1.2 Network architecture (forward)

Question 3. Why is it important to add activation functions between linear transformations?

It is important to have activation function to characterise the mechanism of the layer (e.g. a ReLU activation allows to correct values that should not be negative, a Softmax allows to generate a probability distribution, etc.). These functions should be nonlinear because the combination of linear transformations results in a linear transformation, so using multiple linear functions in succession is redundant and useless.

Question 4. ★What are the sizes n_x, n_h, n_y in the figure 1? In practice, how are these sizes chosen?

n_x is the number of features we give to the neural network (the input size), n_y is the number of outputs we want from it (one or more if using soft-max, for instance), and n_h is the number of neurons in the hidden layer h, or more precisely, the size of the latent space represented by the h layer. In this layer, features are shared according to the weights of the connections between the input layer and h, leading to a new representation of our data.

In the example of Figure 1, we have $n_x = 2; n_h = 4; n_y = 2$.

In practice, n_x is the number of features we have on our data, n_y depends on the problem (for classification for instance, it is the number of classes), and we choose the value of n_h by experimenting with various sizes and trying to strike the right balance between speed, accuracy and spatial complexity, as well as regarding the bias-variance trade-off.

Question 5. What do vectors $\hat{y}$ and y represent? What is the difference between these two quantities?

$\hat{y}$ is the output of the neural network. It is an estimation. We represent this as $\hat{y} = f(x)$ with f our neural network. In contrast, y is the ground truth, in classification it is our label, in regression it is the value we wish to find, etc. In a classification problem: $\hat{y}$ is a vector of probabilities, whereas y is a binary vector containing only 0s and a single 1 corresponding to the right class in a single class per point problem. In a multi-class per point problem, different approaches are possible.

For instance in a 4 class classification problem, we could have: $y = (0, 1, 0, 0)$ and $\hat{y} = (0.25, 0.6, 0.10, 0.05)$ on a model performing relatively well and $\hat{y} = (0.4, 0.3, 0.2, 0.1)$ on one with performing worse.

Question 6. Why use a Soft-Max function as the output activation function?

The Soft-Max function is used at the output layer of the neural network because it enables multi-class classification. Indeed we can interpret its output as the probability, for each class, of y being part of that class, given x. In this case, we select the highest value as the classification/prediction.

Question 7. Write the mathematical equations allowing to successively produce $\tilde{h}, h, \tilde{y}, y$

There are three ways to consider this problem, hence three ways to write those equations:

- Elementwise:

$$\begin{aligned}
\tilde{h}_i &= \sum_{j=1}^{n_x} W_{ij}^h x_j + b_i^h \\
h_i &= \tanh(\tilde{h}_i) \\
\tilde{y}_i &= \sum_{j=1}^{n_x} W_{ij}^y h_i + b_i^y \\
\hat{y}_i &= SoftMax(\tilde{y}_i)
\end{aligned}$$

- Vectorwise:

$$\begin{aligned}
\tilde{h} &= W_h x + b_h \\
h &= \tanh(\tilde{h}) \\
\tilde{y} &= W_y h + b_y \\
\hat{y} &= SoftMax(\tilde{y})
\end{aligned}$$

- Matrixwise (e.g. batch of vectors):

$$\tilde{H} = XW_h^T + B_h$$
$$H = \tanh(\tilde{H})$$
$$\tilde{Y} = HW_h^T + B_y$$
$$\hat{Y} = SoftMax(\tilde{Y})$$

1.3 Loss function

Question 8. During training, we try to minimize the loss function. For Cross Entropy and squared error, how must the $\hat{y}_i$ vary to decrease the global loss function?

Using the Mean Squared Error (MSE) loss function, we aim to make each coordinate of $\hat{y}$ close to those of y, since the equation is the sum of the square of the difference of each coordinate.

With cross-entropy, the "correct" coordinate of $\hat{y}$ should be as close to 1 as possible, and we don't consider the coordinates that are 0 in y. Indeed the equation being:

$$CEL = -\frac{1}{N}\sum_i y_i log(\hat{y}_i)$$

$y_i = 0$ implies that the summand in the i^{th} coordinate is null.
For example, if $y = (0, 1, 0, 0)$ and $\hat{y} = (0.25, 0.6, 0.10, 0.05)$, we are only concerned with the 2ND coordinate of $\hat{y}$, which is 0.6. Cross-entropy is a measure of the distance between different probability distributions.

Question 9. How are these functions better suited to classification of regression?

Cross-entropy focuses more on the "target" coordinate, i.e. the one that should be closest to 1. It is better suited for classification because it computes a "distance" or error between the distribution we estimated and the distribution representing the classification of the input (being a single 1 in a zeroes vector for a single class per point problem, once again we can relax and adapt this for multiple class per element problem).

MSE treats all coordinates as important. It is differentiable, stable, and convex, making it more suitable for regression. It is also more applicable to regression as it is an euclidean distance approach, more natural in a regression problem.

1.4 Optimization algorithm

Question 10. What seems to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?

	Classic Gradient Descent	**Stochastic Gradient Descent**	**Mini-Batch Gradient Descent**	**Online Stochastic Gradient Descent**
Pros	We have the guarantee of convergence (and smoothness of the convergence) to the minimum for convex functions, and it's simple to implement.	We have fast convergence. It's suitable for large data sets and helps avoid local minima.	It combines the efficiency of Batch Gradient Descent and the speed of SGD. It oscillates less and tends to converge steadily and faster.	It considers one data point at a time, making it highly efficient.
Cons	High computational time for large datasets; it can get stuck in local minima and not find the global minimum for non-convex functions.	We can have updates with noise, leading to oscillations.	Requires tuning the hyperparameter of batch size, which affects convergence.	Learning rate adjustment is required, and there is high variance in parameter updates. It also is more noisy/sensitive to noise.

A reasonable choice in the general case seems to be the mini-batch stochastic gradient since it is only one hyper-parameter to tune, but seems to have less drawbacks than the other methods.

Question 11. ★What is the influence of the learning rate η on learning?

The learning rate influences the ability of our model to converge and it convergence speed. Indeed, in the descent method, during the learning, we aim to get as close to the minimum as possible. The problem becomes: how do we control the descent and allow our model to accurately get closer to the minimum.

Controlling the learning rate is crucial as;

- too large of a learning rate could lead to a divergence, a plateau, etc. E.g.:
  - Plateau situation: A curve similar to x^2, reaching $x = 1$ would give a gradient $(2x)$ of 2 meaning we would do $x \longleftarrow x - 2$ and reach $x = -1$ which would lead to an infinite loop.
  - Divergence: A curve similar to x^4 with $a > 2$ and we reach $x = 1$. This leads to the gradient $(4x^3)$ to be 4 leading to $x \longleftarrow x - 4$ then the gradient at $x = -3$ is -36, leading to us drifting further and further away from 0 the minimum.
- too small of a learning rate is computationally expensive since reaching convergence would ask for a great number of iteration (with x^2 and $lr = 10^-10$ we would need $O(10^10)$ iterations to reach the minimum).

The control over the learning rate is often done with a scheduler to decay exponentially the learning rate when a plateau is reached of if the gradient seems to make us drift to far from a minima.

Question 12. ★Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the loss with respect to the parameters, using the naive approach and the backprop algorithm

With the naive approach, we compute the gradients of the loss with respect to each parameter in the network directly. For each parameter, we calculate its partial derivative with respect to the loss function separately. If the neural network has L layers and each layer has N parameters, we would have to compute L * N partial derivatives individually. The complexity of the naive approach is proportional then O(L * N).

However, backpropagation is more efficient because it leverages the chain rule. This way, we can compute the gradients layer by layer, starting from the output layer and propagating backward through the network. Its complexity is then O(L).

Question 13. What criteria must the network architecture meet to allow such an optimization procedure?

The architecture of the network must follow a differentiability criterion. This means that both the activation function and the loss functions must be differentiable. Secondly, we must work with a feedforward (or more precisely, no loops) network for a proper use.

Question 14. The function SoftMax and the loss of cross-entropy are often used together and their gradient is very simple. Show that the loss can be simplified by: $l = -\sum_i y_i \tilde{y}_i + \sum_i log\left(e^{\tilde{y}_i}\right)$

The loss of cross-entropy is defined by: $l(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$.

And we have: $\hat{y}_i = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}}$.

So:

$$
\begin{aligned}
l(y, \hat{y}) &= -\sum_i y_i \log(\hat{y}_i) \\
&= -\sum_i y_i \log\left(\frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}}\right) \\
&= -\sum_i y_i (\log(e^{\tilde{y}_i}) - \log(\sum_{j=1}^{n_y} e^{\tilde{y}_j})) \\
&= -\sum_i y_i \tilde{y}_i + \sum_i y_i \log(\sum_{j=1}^{n_y} e^{\tilde{y}_j}) \\
&= -\sum_i y_i \tilde{y}_i + \log(\sum_{j=1}^{n_y} e^{\tilde{y}_j}) \qquad \text{car} \sum_i y_i = 1
\end{aligned}
$$

Question 15. Write the gradient of the loss (cross-entropy) relative to the intermediate output $\tilde{y}$

We have:

$$\begin{aligned}\frac{\partial l}{\partial \tilde{y_i}} &= \frac{\partial}{\partial \tilde{y_i}}(-\sum_i y_i \tilde{y_i} + \log(\sum_{j=1}^{n_y} e^{\tilde{y}_j})) \\ &= -y_i + \frac{e^{\tilde{y_i}}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \\ &= -y_i + \hat{y}_i\end{aligned}$$

Thus:

$$\boxed{\frac{\partial l}{\partial \tilde{y_i}} = \hat{y}_i - y_i} \qquad \text{et:} \quad \boxed{\nabla_{\tilde{y}} l = \hat{y} - y}$$

Question 16. Using the backpropagation, write the gradient of the loss with respect to the weights of the output layer $\nabla_{W_y} l$. Note that writing this gradient uses $\nabla_{\tilde{y}} l$. Do the same for $\nabla_{b_y} l$

Computation of $\nabla_{\tilde{y}} l$:

$$\frac{\partial l}{\partial W^y_{i,j}} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial W^y_{i,j}}$$

We already have: $\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y}_k$
And:

$$\begin{aligned}\frac{\tilde{y_k}}{\partial W^y_{i,j}} &= \frac{\partial}{\partial W^y_{i,j}} \left(\sum_{a=1}^{n_h} W^y_{k,a} h_a + b^y_k \right) \\ &= \begin{cases} 0 & \text{if} k \neq i \\ \frac{\partial}{\partial W^y_{i,j}} \left(\sum_{a=1}^{n_h} W^y_{i,a} h_a + b^y_i \right) = h_j & \text{else.} \end{cases}\end{aligned}$$

Thus:

$$\frac{\partial l}{\partial W^y_{i,j}} = (\hat{y}_i - y_i) h_j \qquad \text{and:} \quad \boxed{\nabla_{\tilde{y}} l = (\hat{y} - y)^{\mathrm{T}} h = \nabla_{\tilde{y}} l^{\mathrm{T}} h}$$

Computation of $\nabla_{b^y} l$:

$$\frac{\partial l}{\partial b^y_i} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial b^y_i}$$

We already have: $\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y}_k$
And:

$$\begin{aligned}\frac{\tilde{y_k}}{\partial b^y_i} &= \frac{\partial}{\partial b^y_i} \left(\sum_{a=1}^{n_h} W^y_{k,a} h_a + b^y_k \right) \\ &= \begin{cases} 0 & \text{if} k \neq i \\ 1 & \text{else.} \end{cases}\end{aligned}$$

Thus:

$$\frac{\partial l}{\partial b_i^y} = (\hat{y}_i - y_i) \qquad \text{and:} \quad \boxed{\nabla_{b^y} l = (\hat{y} - y)^{\mathrm{T}} = \nabla_{\tilde{y}} l^{\mathrm{T}}}$$

Question 17. ★Compute the other gradients: $\nabla_{\tilde{h}} l$, $\nabla_{W_h} l$, $\nabla_{b_h} l$

Computation of $\nabla_{\tilde{h}} l$:

$$\frac{\partial l}{\partial h_i^y} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial h_i^y} \frac{\partial h_i^y}{\partial \tilde{h_i^y}}$$

We already have: $\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y}_k$
And:

$$\frac{\tilde{y_k}}{\partial h_i^y} = \frac{\partial}{\partial h_i^y}\left(\sum_{j=1}^{n_h} W_{k,j}^y h_j + b_k^y\right) = W_{k,i}^y$$
$$\frac{h_i^y}{\partial \tilde{h_i^y}} = 1 - \tanh^2(\tilde{h_i}) = 1 - h_i^2$$

Thus:

$$\frac{\partial l}{\partial h_i^y} = (1 - h_i^2) \sum_k (\hat{y}_k - y_k) W_{k,i}^y$$

Therefore:

$$\boxed{\nabla_{\tilde{h}} l = (1 - h^2) \odot (\hat{y} - y) W^y = (1 - h^2) \odot \nabla_{\tilde{y}} l W^y}$$

Computation of $\nabla_{W^h} l$:

$$\begin{aligned}\frac{\partial l}{\partial W_{i,j}^h} &= \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial W_{i,j}^h} \\ &= \sum_{k,a} \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial \tilde{h_a}} \frac{\partial \tilde{h_a}}{\partial W_{i,j}^h}\end{aligned}$$

We already have: $\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y}_k$
And:

$$\begin{aligned}\frac{\partial \tilde{y_k}}{\partial \tilde{h_a}} &= \frac{\partial \tilde{y_k}}{\partial h_a} \frac{\partial h_a}{\partial \tilde{h_a}} \\ &= W_{k,a}^y (1 - \tanh^2 \tilde{h_a}) \\ &= W_{k,a}^y (1 - h_a^2)\end{aligned}$$

$$\frac{\partial \tilde{h_a}}{\partial W_{i,j}^h} = \frac{\partial}{\partial W_{i,j}^h}\left(\sum_{b=1}^{n_h} W_{a,b}^h x_b + b_a^h\right) = \begin{cases} 0 & \text{if} a \neq i \\ x_j & \text{else.} \end{cases}$$

Thus:

$$\frac{\partial l}{\partial W_{i,j}^h} = \sum_k (\hat{y}_k - y_k) W_{k,i}^y (1 - h_i) x_j$$

Therefore:

$$\boxed{\nabla_{W^h} l = (1 - h^2) \odot ((\hat{y} - y) W^y) x = \nabla_{\tilde{h}} l^{\mathrm{T}} x}$$

Computation of $\nabla_{b^h} l$:

$$\frac{\partial l}{\partial b_i^h} = \sum_{k,a} \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial \tilde{h_a}} \frac{\partial \tilde{h_a}}{\partial b_i^h}$$

We already have: $\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y_k}$, and: $\frac{\partial \tilde{y_k}}{\partial \tilde{h_a}} = W^y_{k,a}(1 - h_a^2)$
Then:

$$\begin{aligned} \frac{\partial \tilde{h_a}}{\partial b_i^h} &= \frac{\partial}{\partial b_i^h} \left(\sum_{j=1}^{n_h} W^h_{i,j} x_j + b_i^h \right) \\ &= \begin{cases} 0 & \text{if } a \neq i \\ 1 & \text{else.} \end{cases} \end{aligned}$$

Thus:

$$\frac{\partial l}{\partial b_i^h} = \sum_k (\hat{y_k} - y_k) W^y_{k,i} (1 - h_i)$$

Therefore:

$$\boxed{\nabla_{b^h} l = (1 - h^2) \odot ((\hat{y} - y) W^y) = \nabla_{\tilde{h}} l^{\mathrm{T}}}$$

2 Section 2: Implementation

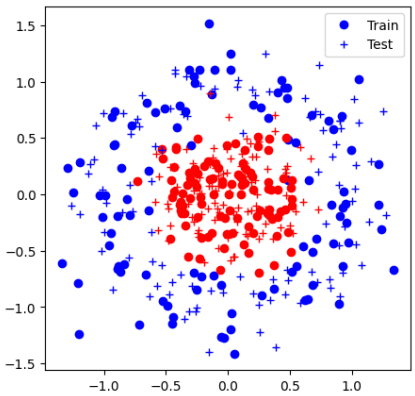

Figure 1: CirclesData

2.1 Forward and backward manuals

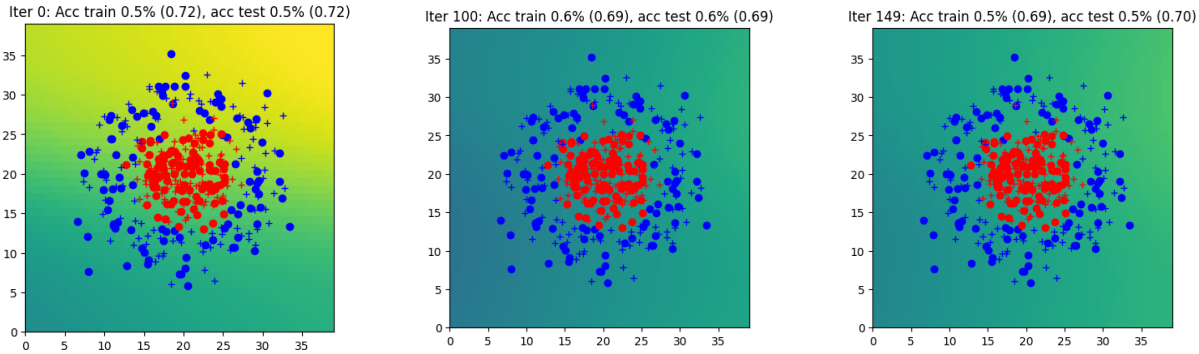

Figure 2: iterations

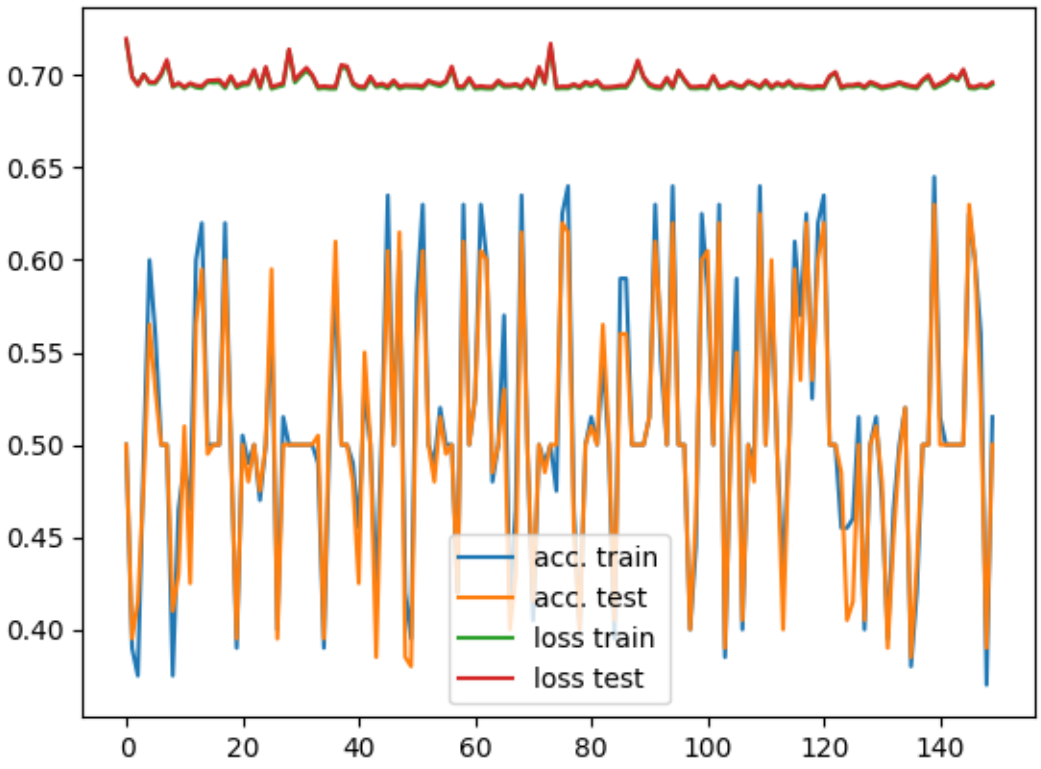

Figure 3: accuracy and losses curves

2.2 Simplification of the backward pass with torch.autograd

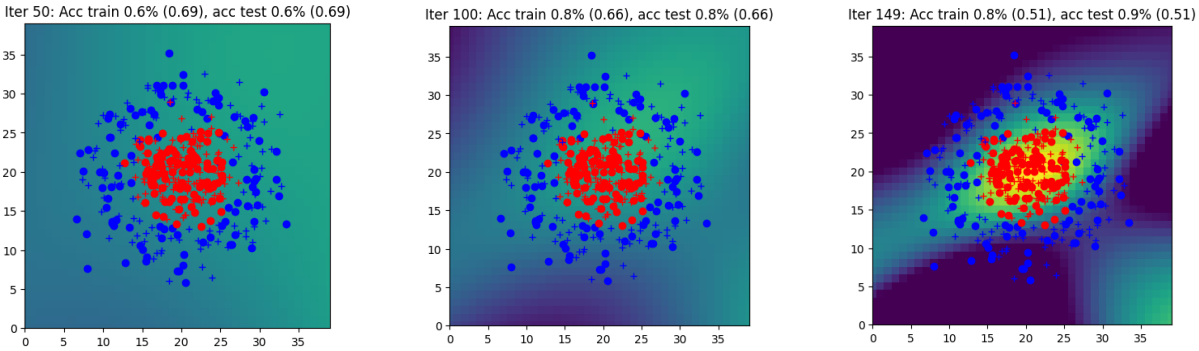

Figure 4: iterations

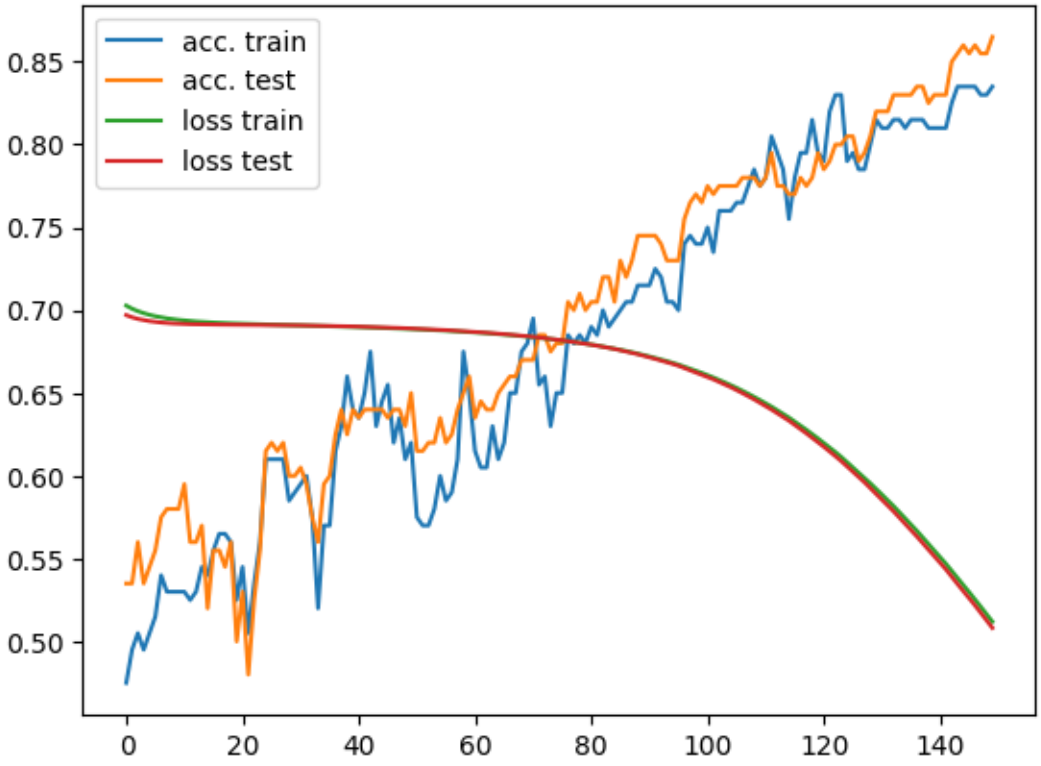

Figure 5: accuracy and losses curves

2.3 Simplification of the forward pass with torch.nn layers

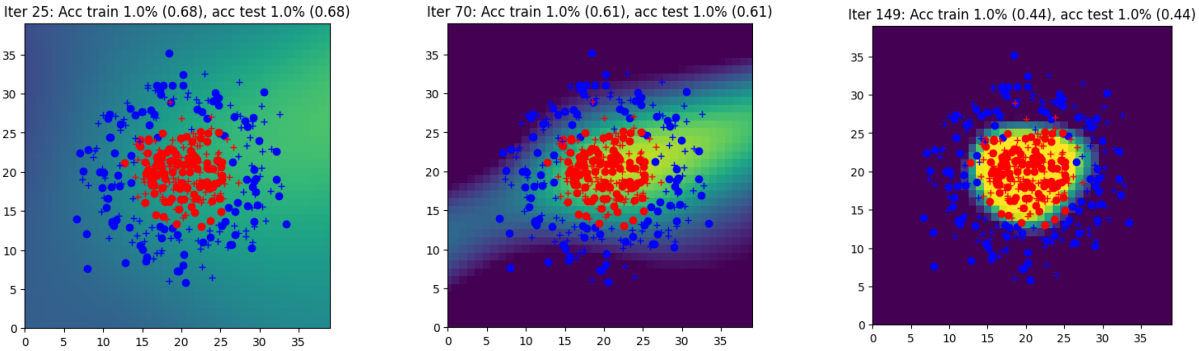

Figure 6: iterations

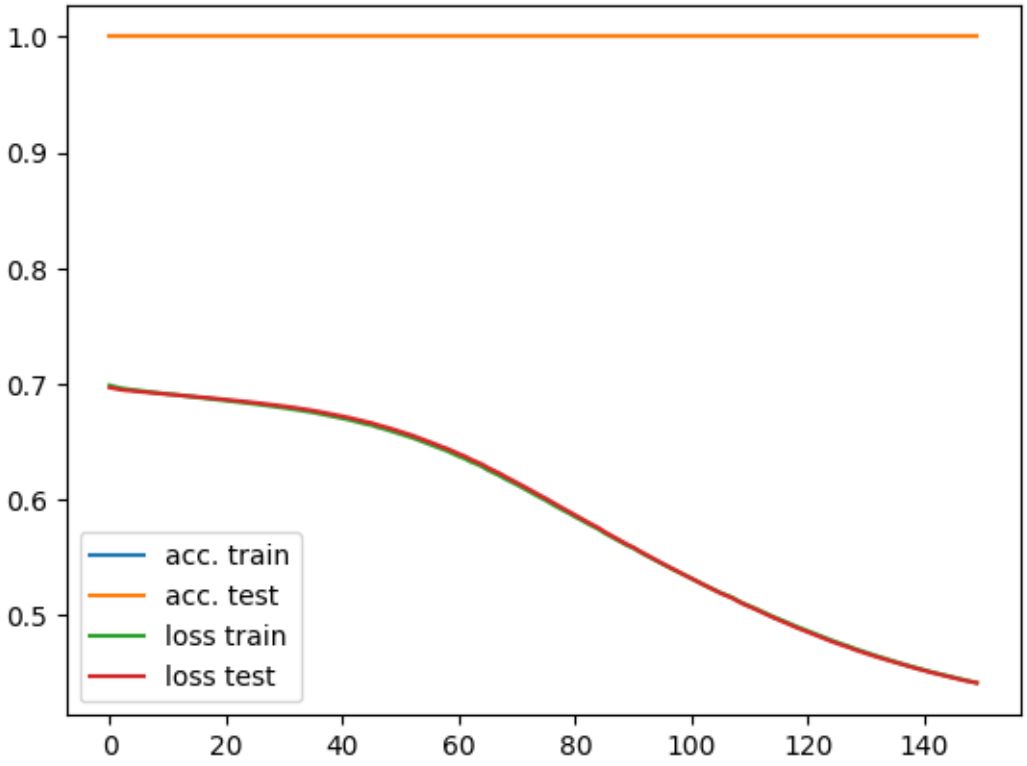

Figure 7: accuracy and losses curves

2.4 Simplification of the SGD with torch.optim

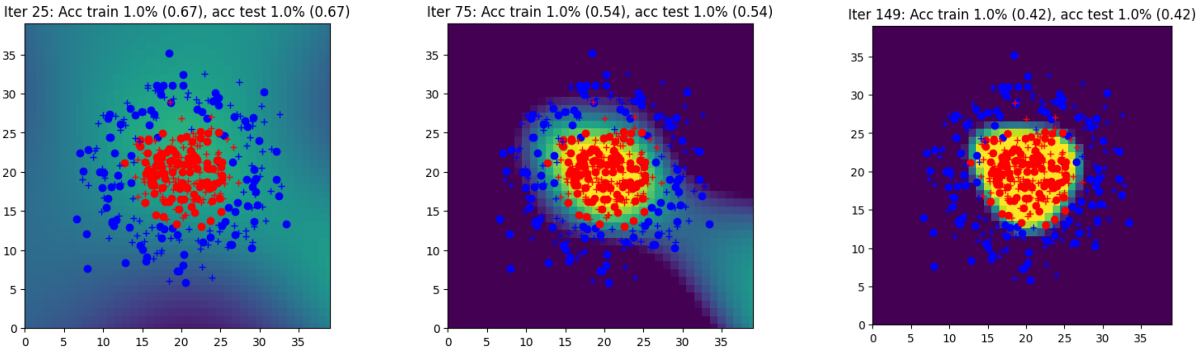

Figure 8: iterations

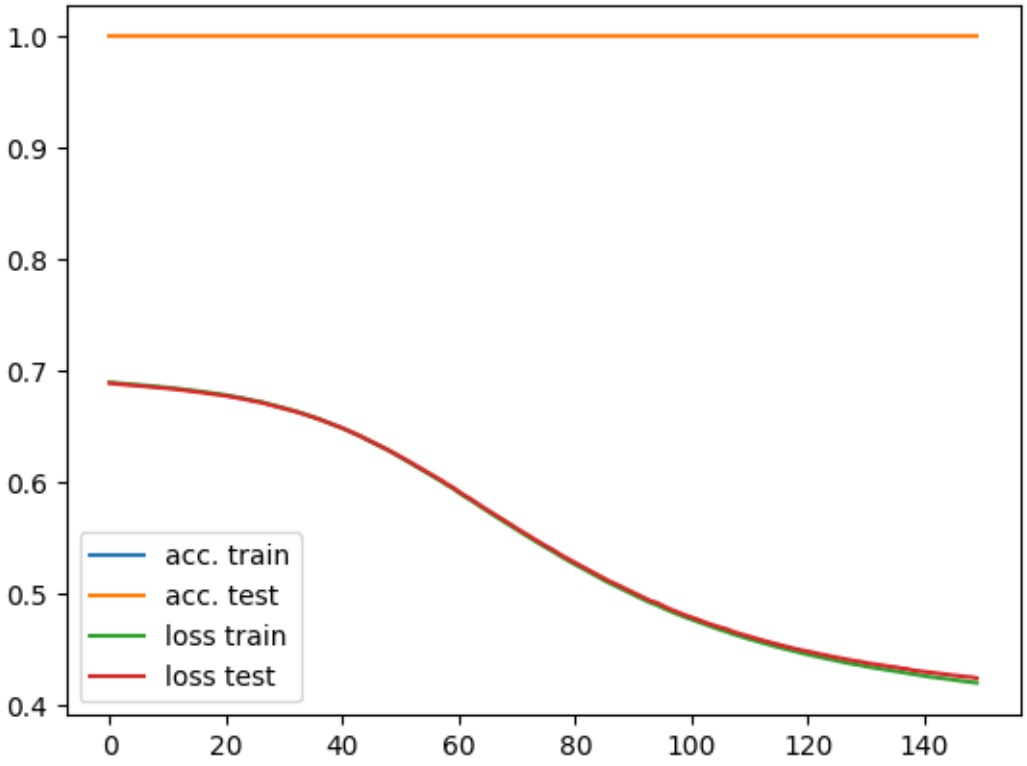

Figure 9: accuracy and losses curves

2.5 MNIST application

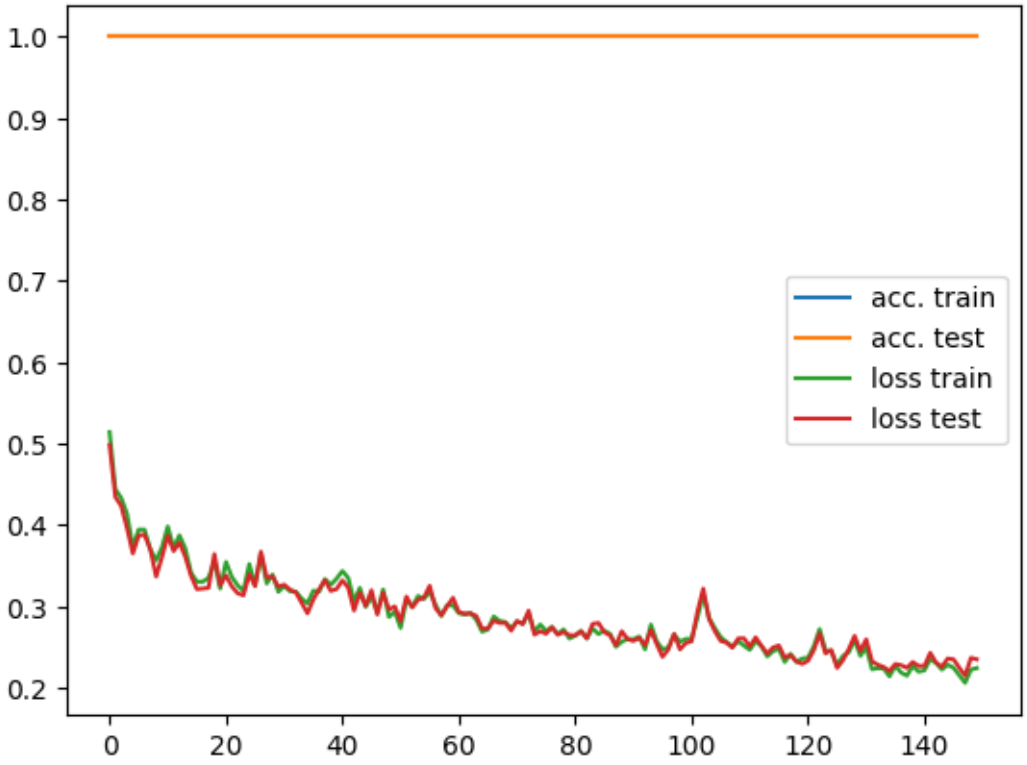

Figure 10: accuracy and losses curves

2.6 Bonus: SVM

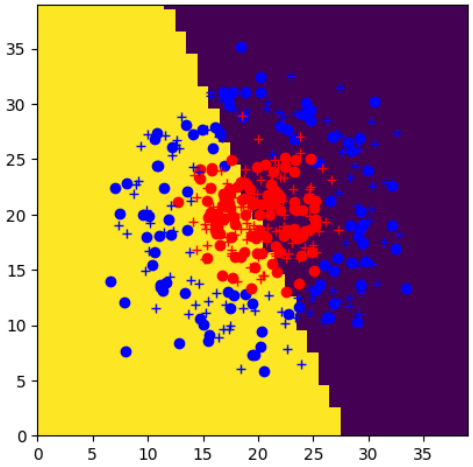

Figure 11: illustration of linear SVM

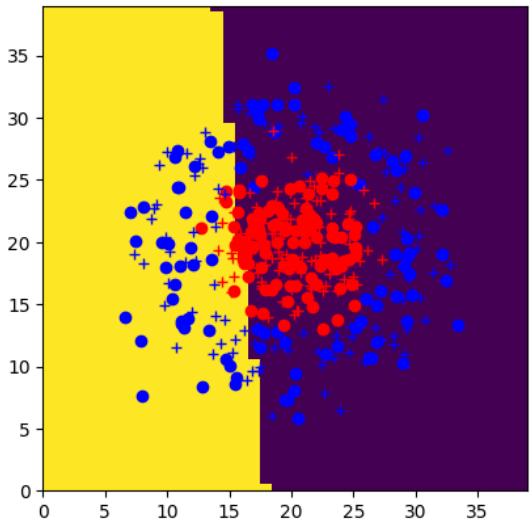

Figure 12: linear SVM

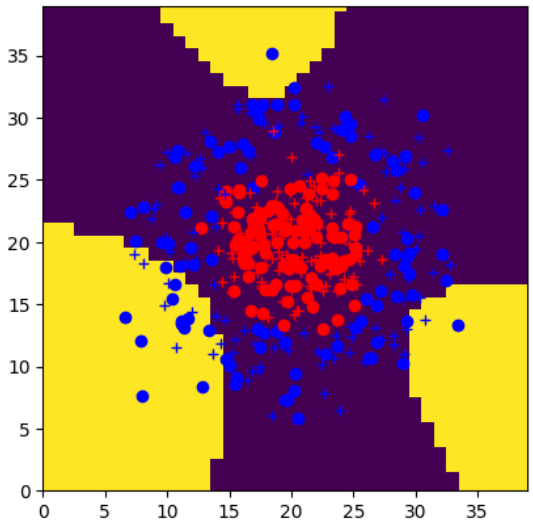

Figure 13: polynomial SVM

Celui avec la meilleur accuracy est le rbf (Radial Basis Function). La fonction RBF est efficace pour des tâches de classification complexes et non linéaires. Ici, c'est bien adapté car on a affaire à des données qui doivent être séparées par un cercle.

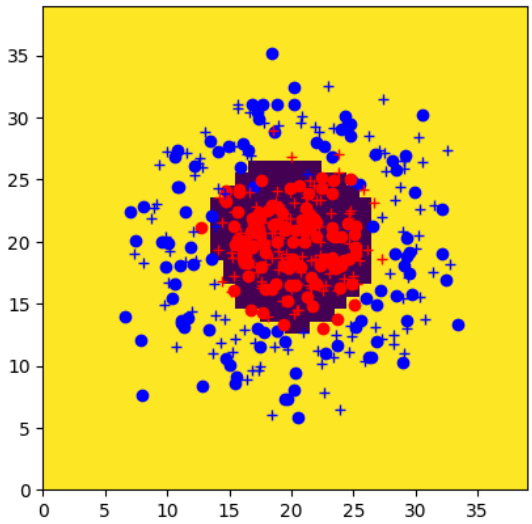

Figure 14: RBF SVM

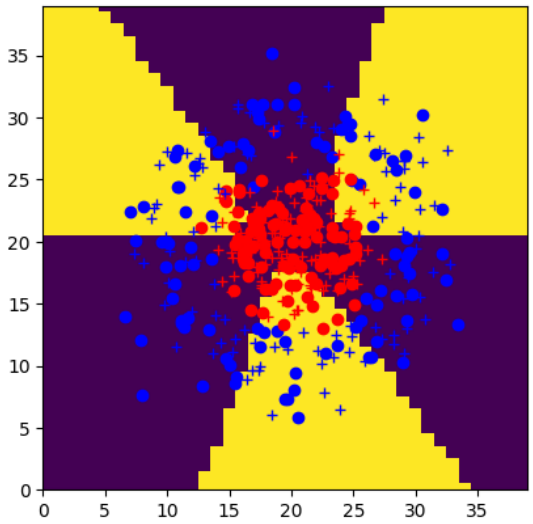

Figure 15: sigmoid SVM