1 Intro to Neural Networks

The purpose of this practical exercise is to understand how a typical neural network is constructed. To achieve this, we will work on the established mathematical equations that govern its operation. We will implement it to obtain a functional model and verify that it converges towards the desired solution. In a second step, we will work with the PyTorch tool to understand how to implement a model more easily and thus be able to increase complexity for future practical work.

1.1 Theoretical foundation

1. What are the train, val and test sets used for ? Train set is designed to enable the model to learn the relationships between the inputs X and the outputs Y.
Val set is made for adjusting the model's hyperparameters and evaluating its performance. With this dataset, we can compare different model configurations before selecting the most effective one.
Test set is designed to evaluate the performance of the final model after it has been fully trained. It should not be used for adjusting or modifying the model. This evaluation is impartial regarding the capabilities of our model.

2. What is the influence of the number of examples N ? The more data we have, with a larger number of N, the more information we have to ensure that the model converges to the true relationships between inputs and outputs.

1.2 Network architecture (forward)

3. Why is it important to add activation functions between linear transformations ? Activation functions introduce a non-linear interpretation of data and improve stability. It allows representing the excitation and activation thresholds of synapses.

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 input data. n_y is the number of output data. n_h is the number of neurons in a layer. Their sizes are arbitrary and chosen by the data scientist depending on the dataset and the complexity of the task.

5. What do the vectors $\hat{y}$ and y represent ? What is the difference between these two quantities ? $\hat{y}$ is the model's prediction. y is the truth. To quantify the model's performance, we compare $\hat{y}$ and y.

6. Why use a SoftMax function as the output activation function ? We often use SoftMax for cases of multiple classification.

7. Write the mathematical equations allowing to perform the forward pass of the neural network, i.e. allowing to successively produce , h, and starting at x.

$$\tilde{h} = W_{h1} \times x + b_{h1}$$
$$h = tanh(\tilde{h})$$
$$\tilde{y} = W_{h2} \times h + b_{h2}$$
$$\hat{y} = SoftMax(\tilde{y})$$

1.3 Loss function

8. During training, we try to minimize the loss function. For cross entropy and squared error, how must the vary to decrease the global loss function L ? As we can deduce by the cross entropy equation, to minimize the global loss function, $\hat{y}_i$ should be closer to 1.

$$CE = -\sum_{i}^{N} y_i log(\hat{y}_i)$$

For minimizing the square error, $y_i - \hat{y}_i$ should be close to 0. To do so, $\hat{y}_i$ should be close or equal to the value of y_i.

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2$$

9. How are these functions better suited to classification or regression tasks ? For classification, it is better to use a cross entropy loss function. For regression, it is better to use squared error.

1.4 Optimization algorithm

10. What seem 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 ? The classical gradient descent method, updates the weights after calculating the average gradient over the entire training dataset. This method requires significant and large storage capacity. It is very time-consuming.

Stochastic gradient descent updates the weights once the average gradient is calculated over a small, random subset of the training data. This method requires less memory since only the batch size is stored in memory. It also leads to faster convergence as the weights are updated more frequently.

Online stochastic gradient descent updates the weights after each individual training instance. This method only requires keeping a single data instance in memory. It is a method that updates its weights at a high frequency, which can potentially degrade the stability of the model.
The most efficient method appears to be stochastic gradient descent in a general case.

11. What is the influence of the learning rate η on learning ? The learning rate η influences the learning rate, stability, and convergence of the model. If η is too large, the model loses stability and can lead to divergence. If η is too small, the model converges very slowly.

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. A naive application would involve computing gradients multiple times and repeatedly. This method is inefficient. The backpropagation method allows traversing the network only once. In fact, the gradient computed with respect to the output layer is used to calculate the gradients of the preceding layers. The backpropagation method becomes more efficient as the network exceeds two layers in size.

13. What criteria must the network architecture meet to allow such an optimization procedure ? To apply backpropagation, you need to have a sequential network, meaning that the output depends on the input.

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. We have the cross entropy loss function :

$$l = -\sum_{i} y_i log(\hat{y})$$

where y is the thruth and is the prediction.
And the softmax function egual to :

$$\hat{y} = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{k} e^{\tilde{y}_i}}$$

where k is the number of classes.

When we combine these two equations :

$$l = -\sum_i y_i log(\hat{y})$$
$$l = -\sum_i y_i log(\frac{e^{\tilde{y_i}}}{\sum_{j=1}^{k} e^{\tilde{y_i}}})$$
$$l = -\sum_i y_i (log(e^{\tilde{y_i}}) - log(\sum_j e^{\tilde{y_j}}))$$
$$l = -\sum_i y_i (\tilde{y_i} - log(\sum_j e^{\tilde{y_j}}))$$
$$l = -\sum_i y_i \tilde{y_i} + \sum_i y_i (log(\sum_j e^{\tilde{y_j}}))$$

y_i is 1 for the correct class i and 0 for all other classes

$$l = -\sum_i y_i \tilde{y_i} + log(\sum_i e^{\tilde{y_i}})$$

15. Write the gradient of the loss (cross-entropy) relative to the intermediate output The product $y_i \times y_j$ is not equal to 0 when $i = j$.

$$\frac{\partial l}{\partial \tilde{y_i}} = -y_i + \frac{e^{\tilde{y_i}}}{\sum_j e^{\tilde{y_j}}}$$
$$\frac{\partial l}{\partial \tilde{y_i}} = -y_i + \hat{y_i}$$
$$\nabla_{\tilde{y}} l = \hat{y} - y$$

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

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

Reminder :

$$\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y_k}$$

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

Therefore :

$$\frac{\partial l}{\partial W^y_{i,j}} = (\hat{y_i} - y_i) h - j$$
$$\nabla_{\hat{y}l} = (\hat{y}_y)^T h = \nabla_{\hat{y}} l^T h$$

For $\nabla_{b^y} l$: $\frac{\partial l}{\partial b_i^y} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial b_i^y}$

$$\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y_k}$$

$$\begin{aligned}\frac{\partial \tilde{y_k}}{\partial b_i^y} &= \frac{\partial}{\partial b_i^y}\left(\sum_{a=1}^{n_h} W_{k,a}^y h_a + b_k^y\right) \\ &= \begin{cases} 1 & \text{if } k = i \\ 0 & \text{else} \end{cases}\end{aligned}$$

Therefore :

$$\frac{\partial l}{\partial b_i^y} = (\hat{y_i} - y_i)$$

$$\begin{aligned}\nabla_{b^y} l &= (\hat{y} - y)^T \\ &= \nabla_y l^T\end{aligned}$$

17. Compute the other gradients :
For $\nabla_{\tilde{h}} l$

$$\frac{\partial l}{\partial W_{i,j}^y} = \sum_k \frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial W_{i,j}^y}$$

Reminder :

$$\begin{aligned}\frac{\partial l}{\partial \tilde{y_k}} &= -y_k + \hat{y_k} \\ \partial \tilde{y_k} \partial h_i^y &= \frac{\partial}{\partial h_i^y}(\sum_{j=1}^{n_h} W_{k,j}^y h_j + b_k^y) \\ &= W_{k,i}^y \\ \frac{h_i^y}{\partial \tilde{h_i^y}} &= 1 - tanh^2(\tilde{h_i}) \\ &= 1 - h_i^2 \\ \frac{\partial l}{h_i^y} &= (1 - h_i^2) \sum_k (\hat{y_k} - y_k) W_{k,i}^y\end{aligned}$$

Therefore :

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

For $\nabla_{\tilde{W^h}} l$

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

Reminder :

$$\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y_k}$$

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

$$\begin{aligned}\frac{\partial \tilde{h_a}}{\partial W^h_{i,j}} &= \frac{\partial}{\partial W^h_{i,j}}(\sum_{b=1}^{n_h} W^h_{a,b}x_b + b^h_a) \\ &= \begin{cases} x_l & \text{if } a = i \\ 0 & \text{else} \end{cases} \\ \frac{\partial l}{\partial W^h_{i,j}} &= \sum_k (\hat{y_k} - y_k)W^y_{k,i}(1 - h_i)x_j\end{aligned}$$

Therefore :

$$\begin{aligned}\nabla_{W^h} l &= (1 - h^2) \odot ((\hat{y} - y)W^y)x \\ &= \nabla_{\tilde{h}} l^T x\end{aligned}$$

Therefore :

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

For $\nabla_{b^{\tilde{h}}} l$

$$\begin{aligned}\frac{\partial l}{\partial b^h_i} &= \sum_k \frac{\partial l}{\partial \tilde{y_k}}\frac{\partial \tilde{y_k}}{\partial b^h_i} \\ &= \sum_{k,a} \frac{\partial l}{\partial \tilde{y_k}}\frac{\partial \tilde{y_k}}{\partial \tilde{h_a}}\frac{\partial \tilde{h_a}}{\partial b^h_i}\end{aligned}$$

Reminder :

$$\frac{\partial l}{\partial \tilde{y_k}} = -y_k + \hat{y_k}$$

$$\frac{\partial \tilde{y_k}}{\partial h_a} = W^y_{k,a}(1 - h_a^2)$$

$$\begin{aligned}\frac{\partial}{\partial b^h_i} &= \frac{\partial}{\partial b^h_i}\left(\sum_{j=1}^{n_h} W^h_{i,j}x_j + b^h_j\right) \\ &= \begin{cases} 1 & \text{if } a = i \\ 0 & \text{else} \end{cases} \\ \frac{\partial l}{\partial b^h_i} &= \sum_k (\hat{y_k} - y_k)W^y_{k,i}(1 - h_i)\end{aligned}$$

Therefore :

$$\begin{aligned}\nabla_{b^h} l &= (1 - h^2) \odot ((\hat{y} - y)W^y) \\ &= \nabla_{\tilde{h}} l^T\end{aligned}$$