Machine Learning - 2020-2021
Universitat Pompeu Fabra

Problems 8: Deep Learning

Goal

The goal of this practice is to understand the basis of Neural Network: the hierarchical architecture and the algorithm of error back-propagation.

Notation

We assume $\mathbf{w} = (w_0, w_1 \ldots, w_D)^{\mathrm{T}}$ and $\mathbf{x}^{(n)} = (1, x_1^{(n)}, \ldots, x_D^{(n)})^{\mathrm{T}}$ when necessary. The errors will be the same that in P8:

- *Least Squares, for regression*:

Model	$h_{\mathbf{w}}(\mathbf{x})$	$= (\mathbf{w}^{\mathrm{T}}\mathbf{x})$
Least Square Error	$\mathbb{E}(\mathbf{w})$	$= \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - y^{(n)}\right)^2$
Gradient	$\frac{\partial \mathbb{E}(\mathbf{w})}{\partial w_j}$	$= \sum_{n=1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - y^{(n)}\right)x_j^{(n)}$

- *Logistic Regression*:

Model	$h_{\mathbf{w}}(\mathbf{x})$	$=P(\mathcal{C}_1\|\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x})$
		where $\sigma(a) = \frac{1}{1+e^{-a}}$ is the *Sigmoid* or *Logistic* function. $\frac{d\sigma(a)}{da} = \sigma(a)(1-\sigma(a))$
Cross-entropy error	$\mathbb{E}(\mathbf{w})$	$= -\sum_{n=1}^{N}\left(y^{(n)}\ln h_{\mathbf{w}}(\mathbf{x}^{(n)}) + (1-y^{(n)})\ln(1-h_{\mathbf{w}}(\mathbf{x}^{(n)}))\right)$
Gradient	$\frac{\partial \mathbb{E}(\mathbf{w})}{\partial w_j}$	$=\sum_{1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - y^{(n)}\right)x_j^{(n)}$

- *Softmax function*:

Model	$h_k(\mathbf{x})$	$=P(\mathcal{C}_k\|\mathbf{x}) = \frac{e^{\mathbf{w}_k^{\mathrm{T}}\mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}}}$
Cross-entropy error	$\mathbb{E}(\mathbf{w}_1, \ldots, \mathbf{w}_K)$	$= -\sum_{n=1}^{N}\left(y_1^{(n)}\ln h_1^{(n)} + \ldots, y_K^{(n)}\ln h_K^{(n)}\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_k^{(n)}\ln h_k^{(n)}$
		where $h_k^{(n)} = h_k(\mathbf{x}^{(n)})$
Gradient	$\frac{\partial \mathbb{E}(\mathbf{w}_1,\ldots,\mathbf{w}_K)}{\partial \mathbf{w}_j}$	$=\sum_{n=1}^{N}\left(h_j^{(n)} - y_j^{(n)}\right)x_j^{(n)}$

Exercises

Hierarchical structure and back-propagation

1. We have a fully connected network with the following characteristics:

   - It takes 2D vectors as input.
   - It has a hidden layer with tho neurons.
   - The output layer has one neuron.

   From this information:

   (a) Draw the network from left (input) to right (output). Remember to draw also the units that represent the bias.

   (b) In the previous figure, put the name of each element of the graph and write, as a closed formula, the output as a function of all the elements of the network.

   (c) Initialise the network using 0.1 for all the weights that arrive to the hidden layer, and 0.2 for the weights that arrive to the output layer. Take the point $\mathbf{x}^{(1)} = \left(\begin{smallmatrix}0.3\\0.5\end{smallmatrix}\right)$ with label $y^{(1)} = 1$ and calculate the output $h(\mathbf{x}^{(1)})$ of the network if the activation function of all the units is a sigmoid function, except in the output layer that we have $h(\mathbf{x}) = \mathbf{x}$.

      i. Calculate the least squares error.

ii. Draw a graph with only the elements and formulas that we will use for calculating the derivatives of the error with respect to the weights that connect with the output layer. What is the local error? How could the local error be used to calculate the derivatives of the error with respect to the weights of the output layer?

iii. Draw a graph with all the necessary to update the weights of one unit of the hidden layer. What is the local error of each hidden unit? How could it be used to calculate the weights of the hidden layer?

(d) Repeat the previous exercises but with a sigmoid function in the output layer and the Cross-Entropy error of the logistic regression.

2. We have a fully connected network with the following characteristics:

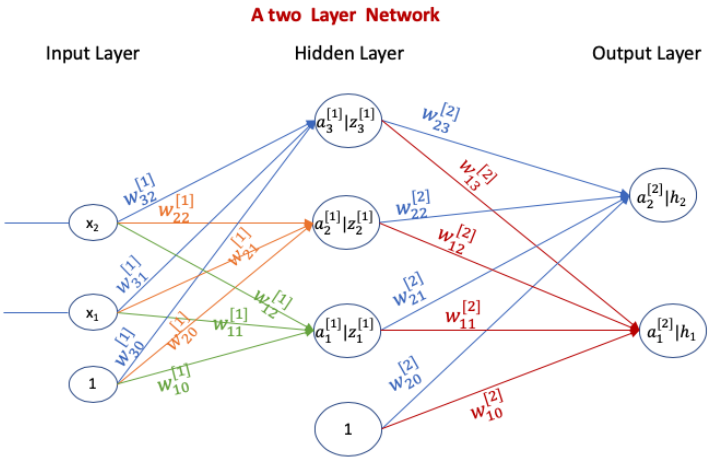
- It takes 2D vectors as input.
- It has a hidden layer with three neurons.
- The output layer has two neuron.

From these information:

(a) Draw the network from left (input) to right (output). Remember to draw also the units that represent the bias.

(b) In the previous figure, put the name of each element of the graph and write, as a closed formula, the two outputs as a function of all the elements of the network.

Solution:



$$h_1(\mathbf{x}) = \varphi\left(w_{10}^{[2]}1 + w_{11}^{[2]} z_1^{[1]} + w_{12}^{[2]} z_2^{[1]} + w_{13}^{[2]} z_3^{[1]}\right) = \varphi\left(w_{10}^{[2]}1 + w_{11}^{[2]} \varphi(a_1^{[1]}) + w_{12}^{[2]} \varphi(a_2^{[1]}) + w_{13}^{[2]} \varphi(a_3^{[1]})\right)$$

$$= \varphi\left(w_{10}^{[2]}1 + w_{11}^{[2]} \varphi(w_{10}^{[1]}1 + w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2) + w_{12}^{[2]} \varphi\left(w_{20}^{[1]}1 + w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2\right) + w_{13}^{[2]} \varphi\left(w_{30}^{[1]}1 + w_{31}^{[1]} x_1 + w_{32}^{[1]} x_2\right)\right)$$

$$h_2(\mathbf{x}) = \varphi\left(w_{20}^{[2]}1 + w_{21}^{[2]} \varphi(w_{10}^{[1]}1 + w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2) + w_{22}^{[2]} \varphi\left(w_{20}^{[1]}1 + w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2\right) + w_{23}^{[2]} \varphi(w_{30}^{[1]}1 + w_{31}^{[1]} x_1 + w_{32}^{[1]} x_2)\right)$$

(c) Initialise the network using 0.1 for all the weights that arrive to the hidden layer, and 0.2 for the weights that arrive to the output layer. Take the point $\mathbf{x}^{(1)} = \binom{0.3}{0.5}$ with label $y^{(1)} = \binom{1}{0.5}$ and compute the outputs $h_k(\mathbf{x}^{(1)})$ of the network if the activation function of all the units is a sigmoid function.

i. Calculate the least squares error.

Solution:

Hidden layer

$$a_1^{[1]} = 0.1(1 + 0.3 + 0.5) = 0.18 \implies z_1^{[1]} = \sigma(a_1^{[1]}) = \frac{1}{1 + e^{-0.18}} = 0.55$$

Similarly, $z_2^{[1]} = z_3^{[1]} = 0.55$.

Output layer

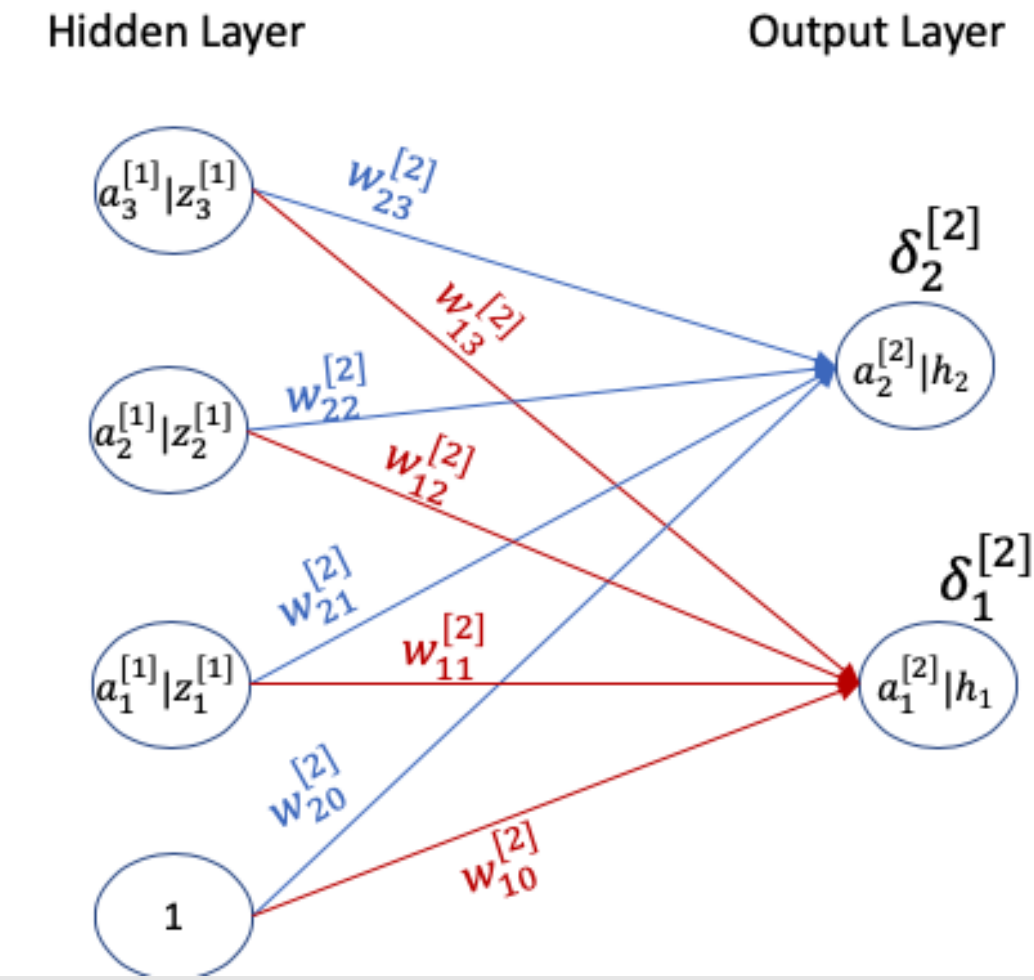
$$a_i^{[2]} = 0.2(1 + 0.55 + 0.55 + 0.55) = 0.53 \implies h_i(a_i^{[2]}) = \sigma(0.53) = 0.63 \quad \forall i = 1, 2.$$

Least square error

$$E_{LS}(\mathbf{w}) = \frac{1}{2}\left(\left(h_1 - y_1^{(1)}\right)^2 + \left(h_2 - y_2^{(1)}\right)^2\right) = \frac{1}{2}\left((0.63 - 1)^2 + (0.63 - 0.5)^2\right) = 0.077$$

ii. Draw a graph with only the elements that will be used to compute the derivatives of the error with respect to the weights that connect with the output layer. What is the local error? How could it be used to update the weights of the output layer?

Solution:



To use the gradient descent algorithm, we need to compute:

$$\frac{\partial E_{LS}(\mathbf{w})}{\partial w_{ij}^{[l]}}$$

with

$$E_{LS}(\mathbf{w}) = \frac{1}{2}\left(\left(h_1 - y_1^{(1)}\right)^2 + \left(h_2 - y_2^{(1)}\right)^2\right)$$

For the output layer we have $\frac{\partial E_{LS}}{\partial w_{ij}^{[2]}} = \frac{\partial E_{LS}}{\partial a_i^{[2]}} \frac{\partial a_i^{[2]}}{\partial w_{ij}^{[2]}}$. The first term is the local error:

$$\delta_i^{[2]} = \frac{\partial E_{LS}}{\partial a_i^{[2]}} = \frac{\partial}{a_i^{[2]}} \frac{1}{2} \left(\left(h_1 - y_1^{(1)} \right)^2 + \left(h_2 - y_2^{(1)} \right)^2 \right) = \left(h_i(a_i^{[2]}) - y^{(1)} \right) h'(a_i^{[2]})$$
$$= \left(h_i(a_i^{[2]}) - y_i^{(1)} \right) h(a_i^{[2]}) \left(1 - h(a_i^{[2]}) \right)$$

Therefore,

$$\delta_1^{[2]} = -0.37 \cdot 0.63 \cdot (1 - 0.63) = -0.09,$$
$$\delta_2^{[2]} = 0.13 \cdot 0.63 \cdot (1 - 0.63) = 0.03.$$

Then, $\frac{\partial E_{LS}}{\partial w_{ij}^{[2]}} = \delta_i^{[2]} \frac{\partial a_i^{[2]}}{\partial w_{ij}^{[2]}} = \delta_i^{[2]} z_j^{[1]}$

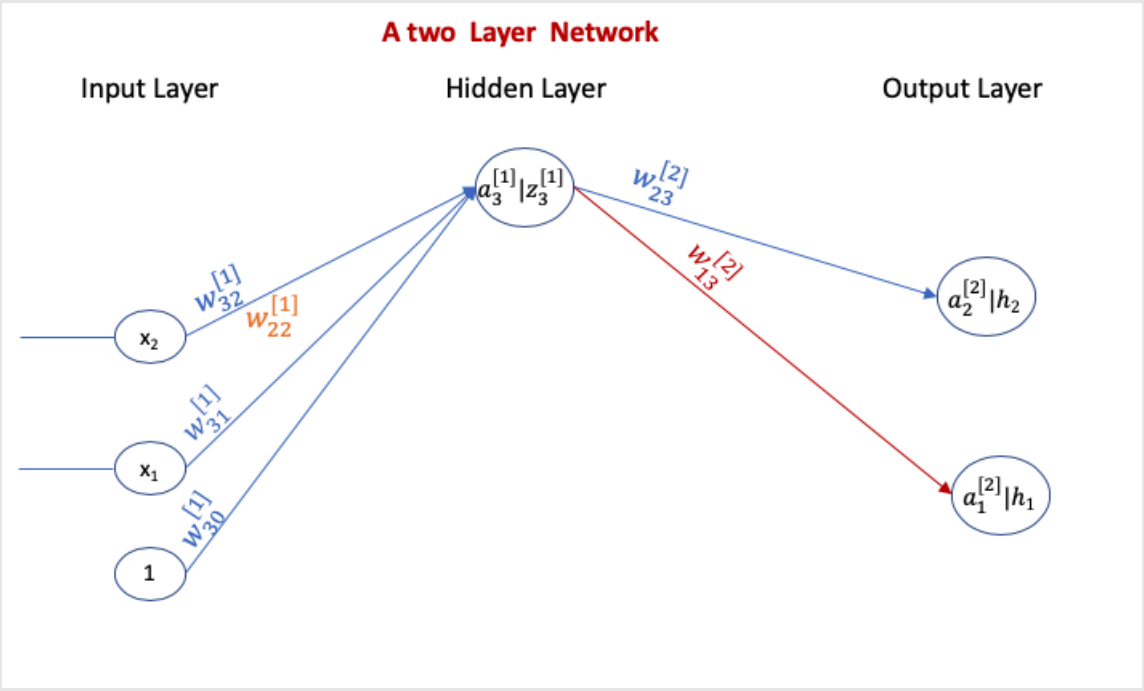
For example $\frac{\partial E_{LS}}{\partial w_{11}^{[2]}} = \delta_1^{[2]} \frac{\partial a_1^{[2]}}{\partial w_{11}^{[2]}} = \delta_1^{[2]} z_1^{[1]} = -0.09 \cdot 0.55 = -0.05.$

Then, we obtain new weights using the gradient descent method as

$$w_{11}^{[2]t+1} = w_{11}^{[2]t} - \alpha(-0.05) = 0.2 + \alpha(0.05).$$

iii. Draw a graph with all the necessary to update the weights of one unit of the hidden layer. What is the local error of each hidden unit? How could it be used to update the weights of the output layer?

Solution:



Similar to the previous exercise:

$$\delta_j^{[1]} = \frac{\partial E_{LSn}}{\partial a_j^{[1]}} = \frac{\partial E_{LSn1}}{\partial a_1^{[2]}}\frac{\partial a_1^{[2]}}{\partial z_j^{[1]}}\frac{\partial z_j^{[1]}}{\partial a_j^{[1]}} + \frac{\partial E_{LSn2}}{\partial a_2^{[2]}}\frac{\partial a_2^{[2]}}{\partial z_j^{[1]}}\frac{\partial z_j^{[1]}}{\partial a_j^{[1]}}$$
$$= \delta_1^{[2]} w_{1j}^{[2]} h'(a_j^{[1]}) + \delta_2^{[2]} w_{2j}^{[2]} h'(a_j^{[1]})$$

Then, for example

$$\delta_3^{[1]} = (\delta_1^{[2]} w_{13}^{[2]} + \delta_2^{[2]} w_{23}^{[2]}) h'(a_3^{[1]})$$
$$= (-0.09^{\mathrm{T}} 0.2 + 0.03^{\mathrm{T}} 0.2) \cdot 0.55 \cdot (1 - 0.55) = -0.003$$

Using the Gradient descent as before we can update the weights.

We need $\frac{\partial E_n}{\partial w_{ij}^{[1]}} = \delta_i^{[1]} \frac{\partial a_i^{[1]}}{\partial z_j^{[0]}}$

In the case of $\frac{\partial E_n}{w_{30}^{[1]}}$

$$\frac{\partial E_n}{w_{30}^{[1]}} = -0.003 \cdot 1 = -0.003$$

$$w_{30}^{[1]t+1} = w_{30}^{[1]t} - \alpha \frac{\partial E_n}{\partial w_{ij}^{[1]}} \qquad w_{30}^{[1]t+1} = 0.1 - \alpha(-0.003)$$

iv. For each weight write from which units the error comes from in the back-propagation algorithm. Draw the network and put the local error of each unit $\delta_i^{[l]}$ (same formalism layer-unit than before). Use these values to explain the rule used by the back-propagation algorithm.

Solution:
In the hidden layer:
$\delta_i^{[1]} \Leftarrow \delta_1^{[2]} \delta_2^{[2]}$, so for $w_{ij}^{[1]}$ the error comes across all the weights $w_{ki}^{[2]}$ with $k = 1, 2$ and the units $h_1, h_2, z_i^{[2]}$.
In the output layer:
for $w_{ij}^{[2]}$ the error comes from the output neuron i.

The back propagation of the error. Describe: 1) the node father of the weight we are calculating the derivative, 2) all the connected nodes with this father (the "ancestors"), 3) the weights that connect the ancestors multiplied by the local error of the ancestor, 4) The addition of the multiplications of the last step, multiplied by the origin of the weight we are calculating the derivative.

(d) Repeat the previous exercise but in the output layer the error cost is the Cross-Entropy error of the logistic regression.

Solution:
The Error: $E_{CS}(\mathbf{w}) = -\sum_{n=1}^{N} \left(y^{(n)} \ln h_{\mathbf{w}}(\mathbf{x}^{(n)}) + (1 - y^{(n)}) \ln(1 - h_{\mathbf{w}}(\mathbf{x}^{(n)})) \right)$
In our case, we assume each output is compared with the expected value with a cross-entropy:
$E_{CE}(\mathbf{w}) = -\left(y_1^{(1)} \ln h_1(\mathbf{x}^{(1)}) + (1 - y_1^{(n)}) \ln(1 - h_1(\mathbf{x}_1^{(n)})) \right)$
$-\left(y_2^{(1)} \ln h_2(\mathbf{x}^{(1)}) + (1 - y_2^{(n)}) \ln(1 - h_2(\mathbf{x}_1^{(n)})) \right)$
If we consider the $a_i^{[2]}$, the cross-entropy error for each output should be written:
$E_{CEi}(\mathbf{w}) = -\left(y_i^{(1)} \ln h_i(a_i^{[2]}) + (1 - y_i^{(n)}) \ln(1 - h_1(a_i^{[2]})) \right)$
For the point $\mathbf{x}^{(1)}$:
$E_{CE}(\mathbf{w}) = E_{CE1}(\mathbf{w}) + E_{CE2}(\mathbf{w}) = -\left(1ln(0.63) + (1-1)ln(1-0.63) \right)$
$-\left(0.5ln(0.63) + (1-0.5)ln(1-0.63) \right) = 1.2$
In our case, $h_i(z) = \sigma(z)$ with derivative $h_i'(z) = \sigma(z)(1 - \sigma(z))$, we take only the output layer:

$\delta_i^{[2]} = \frac{\partial \mathbb{E}_{\text{CE}}(\mathbf{w})}{\partial a_i^{[2]}} = \frac{\partial \mathbb{E}_{\text{CE}}(\mathbf{w})}{\partial a_i^{[2]}} = -\big(y_i^{(1)}(1-\sigma(a_i^{[2]}) - (1-y_i^{(n)})\sigma(a_i^{[2]})\big) = \big(\sigma(a_i^{[2]}) - y_i^{(n)}\big)$

Then:

$\delta_1^{[2]} = (0.63 - 1) = -0.37 \qquad \delta_2^{[2]} = (0.63 - 0.5) = 0.13$

Taking this local errors in the output layer, the backpropagation algorithm continues as before.

3. (a) Repeat the exercise 2c) using $h_k(\mathbf{x}) = \text{softmax}(\mathbf{x})$, and the cross-entropy error.
   (b) Demonstrate that in the case of 2 classes, the cross-entropy of the softmax is identical to the cross-entropy of the logistic.

4. *(Jupyter Notebook)* In this exercise, we will create our own Neural Network based on a Multilayer Perceptron (MLP) architecture and modify it to add NN's hyperparameters as learning rate or size of hidden layer. The MLP is formed by a hidden layer with 4 neurons for a binary classification. The input data is a 2D dataset.
   (a) Execute the following code for epochs=1500.
      i. Which is the initialised coefficients for layer 1? And the last coefficients for layer 1?
      ii. Which is the initialised coefficients for layer 2? And the last coefficients for layer 2?
      iii. Which is the initial error? And the last error?
      iv. Plot the error vs number of epochs.
   (b) Modify the class Neural Network to incorporate the following variables in the code:
      i. `n nn hidden layer` that represents the number of neurons at the hidden layer.
      ii. `learning rate` to adjust the gradient descent of both layers.
   (c) Execute the previous code for a learning rate=1 and the following size of the hidden layer: [1, 2, 4, 6, 8, 10].
      i. Which is the initialised coefficients for layer 1 for size=10? And the last coefficients for layer 1 for size=10?
      ii. Which is the initialised coefficients for layer 2 for size=10? And the last coefficients for layer 2 for size=10?
      iii. Which is the initial error for every size? And the last error for every size?
      iv. Plot the error vs number of epochs for every size. Which is the size with the largest error after the training?
   (d) Execute the previous code for a size of the hidden layer=4 and the following values for the learning rate: [0.01, 0.1, 0.5, 1, 10, 50].
      i. Which is the final error for learning rate=0.01? And for learning rate=50?
      ii. With learning rate= 10, does the error converge to a minimum value? If it does, how many epochs are needed?
      iii. Which learning rate converges before: learning rate=1 or learning rate=0.5?
      iv. Which learning rate achieves a lowest error: learning rate=1 or learning rate=0.01?
      v. From your point of view, which is your recommended learning rate? Why?
      vi. Plot the error vs number of epochs for every learning rate. Which is the learning rate with the largest error after the training?