Deep Learning

Practical Exercises - Week 4-6

Neurons and Neural Networks with NumPy

ICAI

FS 2024

1 Topic

In the next three lab sessions we will implement our own tiny neural network from scratch using Python and the NumPy library. The goal is to design, train, and test a classifier for each of the datasets shown in Figure 1.

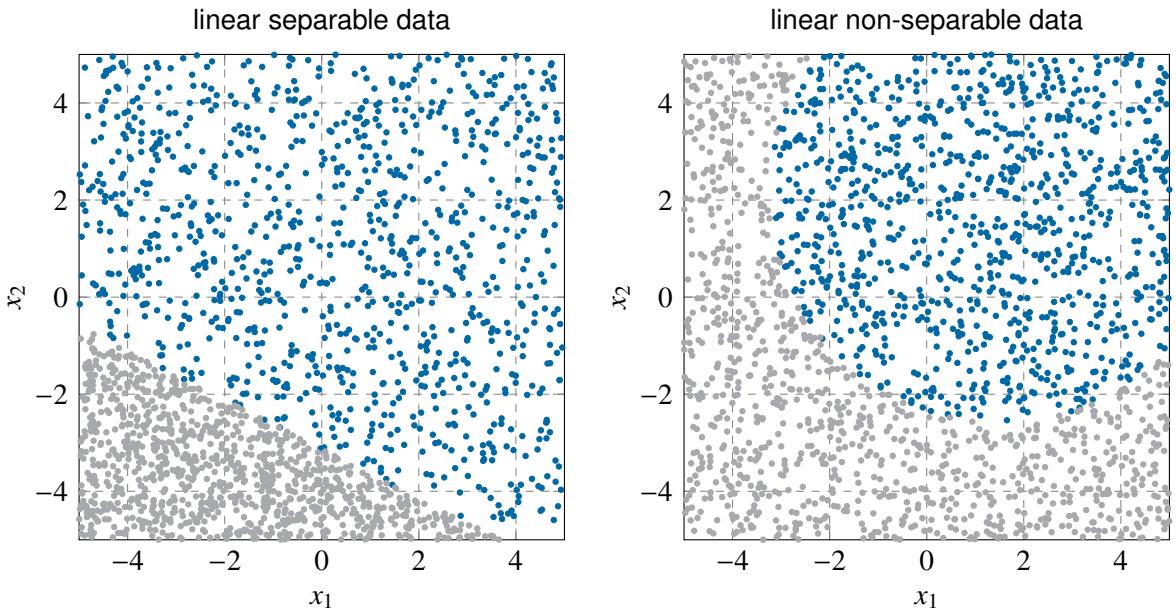

Figure 1: Two datasets that should be used to design the classifiers. One of them is linear separable (left), the other is not (right).

2 A Single Neuron

We start with the simplest possible network that consists of a single neuron with two inputs (see Figure 2).

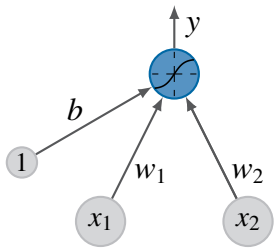

Figure 2: A single neuron with two inputs.

As activation function of this neuron the sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{2.1}$$

is used, where a is called the *pre-activation* and is given by

$$a(\boldsymbol{w}, b) = \boldsymbol{w}^T \boldsymbol{x} + b = w_1 x_1 + w_2 x_2 + b.$$

The predicted output of the neuron is calculated as

$$y(\boldsymbol{w}, b) = \sigma(a) = \sigma(x_1 w_1 + x_2 w_2 + b).$$

The mean square error (MSE) over a mini-batch of N samples is used as loss function

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - t_i)^2. \tag{2.2}$$

Here, y_i is the predicted output of the neuron for the input $\boldsymbol{x}_i$, and t_i is the corresponding (true) target value.

The training procedure as well as the evaluation of the network (neuron) have already been implemented in the file `train_neuron.py`. The training and the evaluation, however, are not working properly yet, since the functions of the module `neuron.py` are not implemented yet.

It is your task to implement all the functions in the module `neuron.py`. To test your functions, you can use the unit test in the module `neuron.py` (code inside the `if` statement). As soon as your functions have passed the unit test, you can train and test the network with the script `train_neuron.py`. Try to understand the code of this script. Also vary the parameters (such as `linearly_separable`, `batch_size`, `learning_rate`, `n_epochs`) and try to understand the different outcomes.

Hint:
Start by calculating the gradient $\nabla J(\boldsymbol{w}, b)$ by hand / on paper.

Solution:

To calculate the weight update, the gradient

$$\nabla J(\boldsymbol{w}, b) = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \frac{\partial J}{\partial b} \end{pmatrix}$$

is required. We first consider the partial derivative with respect to w_1, which can be calculated by applying the chain rule

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial a} \frac{\partial a}{\partial w_1}.$$

The required derivatives are calculated with

$$\begin{aligned} \frac{\partial J}{\partial y_i} &= \frac{\partial}{\partial y_i} \frac{1}{2N} \sum_{i=1}^{N} (y_i - t_i)^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i), \\ \frac{\partial y_i}{\partial a} &= \sigma'(a) = \frac{\exp(-a)}{(1 + \exp(-a))^2}, \\ \frac{\partial a}{\partial w_1} &= \frac{\partial}{\partial w_1} (w_1 x_1 + w_2 x_2 + b) = x_1. \end{aligned}$$

Inserting these derivatives into the chain rule, we get

$$\frac{\partial J}{\partial w_1} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot x_{i,1}.$$

The partial derivatives with respect wo w_2 and b can be calculated similarly:

$$\frac{\partial J}{\partial w_2} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot x_{i,2}, \tag{2.3}$$

$$\frac{\partial J}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i). \tag{2.4}$$

The weight updates for the gradient descent are finally given by

$$w_1 \leftarrow w_1 - \epsilon \frac{\partial J}{\partial w_1}, \tag{2.5}$$

$$w_2 \leftarrow w_2 - \epsilon \frac{\partial J}{\partial w_2}, \tag{2.6}$$

$$b \leftarrow b - \epsilon \frac{\partial J}{\partial b}. \tag{2.7}$$

3 A Tiny Neural Network

With a single neuron, it is only possible to classify linearly separable datasets. However, the second dataset in Figure 1 is not linearly separable. Hence, a neural network with more capacity is needed. We will use a neural network as shown in Figure 3. Again, the sigmoid function (2.1) is used as the activation function and the mean squared error (MSE) (2.2) is used as the loss function.

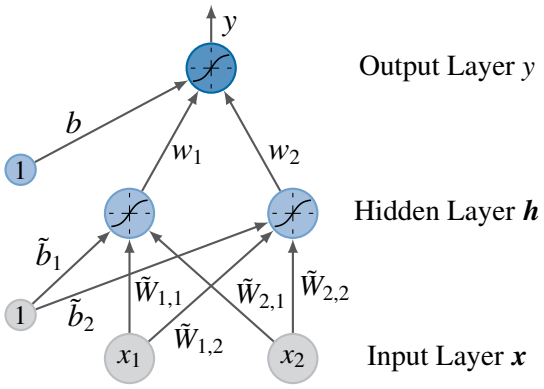

Figure 3: A neural network with two inputs and two hidden neurons.

The calculation of the output y is somewhat more complicated here:

$$y(\boldsymbol{w}, b, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{b}}) = \sigma\left(\boldsymbol{w}^T \boldsymbol{h} + b\right) = \sigma\left(\boldsymbol{w}^T \sigma(\tilde{\boldsymbol{W}}^T \boldsymbol{x} + \tilde{\boldsymbol{b}}) + b\right).$$

Again, the training procedure and the evaluation of the network have already been implemented in the file `train_network.py`. The training and the evaluation, however, are not working properly yet, since the functions of the module `network.py` are not implemented yet.

It is your task to implement all the functions in the module `network.py`. To test your functions, you can use the unit test in the module `network.py` (code inside the `if` statement). As soon as your functions have passed the unit test, you can train and test the network with the script `train_network.py`. Try to understand the code of this script. Also vary the parameters (such as `linearly_separable`, `batch_size`, `learning_rate`, `n_epochs`) and try to understand the different outcomes.

Solution:

Again, the gradient with respect to all weights and biases is needed:

$$\nabla J(\boldsymbol{w}, b, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{b}}) = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \frac{\partial J}{\partial b} \\ \frac{\partial J}{\partial \tilde{W}_{1,1}} \\ \frac{\partial J}{\partial \tilde{W}_{2,1}} \\ \frac{\partial J}{\partial \tilde{b}_1} \\ \frac{\partial J}{\partial \tilde{W}_{1,2}} \\ \frac{\partial J}{\partial \tilde{W}_{2,2}} \\ \frac{\partial J}{\partial \tilde{b}_2} \end{pmatrix}$$

To obtain the partial derivatives, we again need the chain rule. We denote the pre-activation of the output neuron by

$$a = \boldsymbol{w}^T \boldsymbol{h} + b,$$

and the pre-activation of the hidden neurons by

$$\tilde{\boldsymbol{a}} = \tilde{\boldsymbol{W}}^T \boldsymbol{x} + \tilde{b}.$$

The partial derivatives with respect to w_1, w_2 and b are calculated like for the simple neuron.

$$\frac{\partial J}{\partial w_1} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot h_{i,1}$$

$$\frac{\partial J}{\partial w_2} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot h_{i,2}$$

$$\frac{\partial J}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i)$$

For the weights $\tilde{\boldsymbol{W}}$ and $\tilde{\boldsymbol{b}}$, the chain rule has to be re-applied in the same manner, for example

$$\frac{\partial J}{\partial \tilde{W}_{1,1}} = \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial a_i} \frac{\partial a_i}{\partial h_{i,1}} \frac{\partial h_{i,1}}{\partial \tilde{a}_{i,1}} \frac{\partial \tilde{a}_{i,1}}{\partial \tilde{W}_{1,1}}.$$

This results in the following partial derivatives:

$$\frac{\partial J}{\partial \tilde{W}_{1,1}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot w_1 \cdot \sigma'(\tilde{a}_{i,1}) \cdot x_{i,1},$$

$$\frac{\partial J}{\partial \tilde{W}_{2,1}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot w_1 \cdot \sigma'(\tilde{a}_{i,1}) \cdot x_{i,2},$$

$$\frac{\partial J}{\partial \tilde{b}_1} = \frac{1}{N} \sum_{i=1}^{N} (y_i - t_i) \cdot \sigma'(a_i) \cdot w_1 \cdot \sigma'(\tilde{a}_{i,1}),$$

$$\frac{\partial J}{\partial \tilde{W}_{1,2}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)\cdot\sigma'(a_i)\cdot w_2\cdot\sigma'(\tilde{a}_{i,2})\cdot x_{i,1},$$
$$\frac{\partial J}{\partial \tilde{W}_{2,2}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)\cdot\sigma'(a_i)\cdot w_2\cdot\sigma'(\tilde{a}_{i,2})\cdot x_{i,2},$$
$$\frac{\partial J}{\partial \tilde{b}_2} = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)\cdot\sigma'(a_i)\cdot w_2\cdot\sigma'(\tilde{a}_{i,2}).$$

The weight updates are, again, of the form

$$\theta \leftarrow \theta - \frac{\partial J}{\partial \theta}$$

for all trainable parameters $\theta \in \left\{w_1, w_2, b, \tilde{W}_{1,1}, \tilde{W}_{2,1}, \tilde{W}_{1,2}, \tilde{W}_{2,2}, \tilde{b}_1, \tilde{b}_2\right\}$.