B(E)3M33UI — Exercise ML06: Trainning simple MLP using Gradient Descent and Backpropagation

Petr Pošík, Jiří Spilka

March 30, 2021

The goals of this exercise:

- learn about multi-layer perceptron (MLP) networks
- implement backpropagation algorithm
- use MLP from scikit-learn library

1 Backpropagation for simple MLP

Suppose that we are given the following training data, cf. Figure 1. The task is to create a simple neural network classifier that discriminates between the red and blue dots.

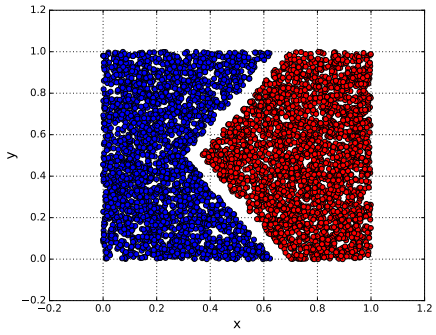

Figure 1: Trainning data

1.1 Preliminary questions

Task 1: Can one simple perceptron classify the data correctly?

Task 2: How many linear boundaries do we need here?

Task 3: What architecture of the network will be suitable?

Task 4: We need to tune the NN to the training set. What is actually tuned?

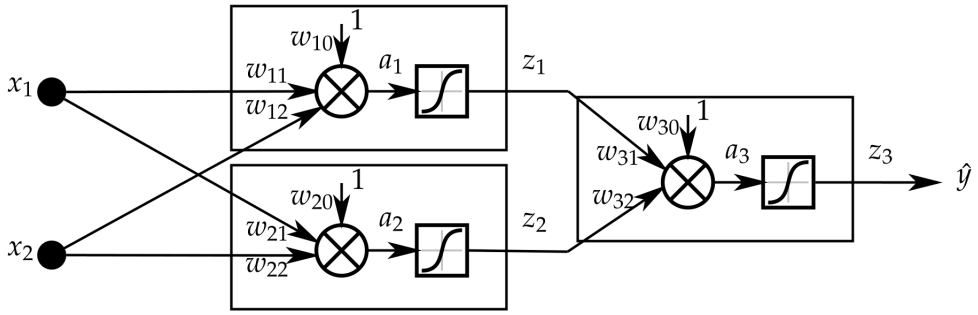

Figure 2: Architecture of simple MLP

1.2 Design of neural network

The network architecture is 2-2-1 and is shown in Figure 2

Task 5: How many weights have to be tuned in our network?

For the given multi-layer perceptron we will implement the backpropagation algorithm to estimate weights $\mathbf{w}$. MLP learning procedure (cf. lectures on NN, slide 18):

1. Starting at the input layer, we forward propagate the patterns of the training data through the network to generate an output.
2. Based on the network's output $\hat{y}$, we calculate the error that we want to minimize using a cost function.
3. We back-propagate the error, find derivative of the loss function with respect to each weight in the network, and update the weights.
4. NN is typically trained for several epochs, where one pass of all training data equals to one epoch.

For the simplicity, we will not expose the problem in vectorized form but we encourage you to implement the MLP vectorized directly.

Task 6: Given the input vector $\mathbf{x}$ compute the output z_3 (i.e. $\hat{y}$) of MLP using the forward pass. Initialize the weights $\mathbf{w}$ to some small random values.

First, let us define sigmoid function as $g(a) = \frac{1}{1+e^{-a}}$. Then, the computation of forward pass is

$$
\begin{aligned}
a_1 &= w_{10} + x_1 w_{11} + x_2 w_{12} \\
a_2 &= w_{20} + x_1 w_{21} + x_2 w_{22} \\
z_1 &= g(a_1) \\
z_2 &= g(a_2) \\
a_3 &= w_{30} + z_1 w_{31} + z_2 w_{32} \\
z_3 &= \hat{y} = g(a_3)
\end{aligned}
$$

Task 7: Having the estimated output $\hat{y}$ compute the error and back-propagate it. Find the derivative of a cost function w.r.t. each weight and update the model.

We will use cross-entropy loss function

$$E(\mathbf{x}, \mathbf{w}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

and its derivative w.r.t. to $\mathbf{w}$ to update the weights

$$\mathbf{w} \to \mathbf{w} - \alpha \nabla E(\mathbf{w})$$

where α is a learning rate (step size). Let us expose the derivative for the output layer only. For the other layers, the computation is similar. Recall that the derivative of sigmoid function is $g'(a) = g(a)(1 - g(a))$. We need to compute the derivative of loss function E w.r.t. all the weights. Using the chain rule we have

$$\frac{\partial E}{\partial w_{ji}} = \underbrace{\frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a}}_{\delta_j} \underbrace{\frac{\partial a}{\partial w_{ji}}}_{=z_i}.$$

Hence, for the last layer and weight w_{31} we need to compute the following derivatives

$$\frac{\partial E}{\partial w_{31}} = \underbrace{\frac{\partial E}{\partial z_3} \frac{\partial z_3}{\partial a_3}}_{\delta_3} \underbrace{\frac{\partial a_3}{\partial w_{31}}}_{=z_1}.$$

The individual derivatives are

$$\begin{aligned}
\frac{\partial E}{\partial z_3} &= -\frac{y}{z_3} + \frac{1 - y}{1 - z_3} = -\frac{y - z_3}{z_3(1 - z_3)} \\
\frac{\partial z_3}{\partial a_3} &= z_3(1 - z_3) \\
\delta_3 = \frac{\partial E}{\partial z_3} \frac{\partial z_3}{\partial a_3} &= -(y - z_3)
\end{aligned}$$

where δ_3 is the error at the last layer. We shall now back-propagate the error δ_3 from the last layer to the hidden layer, i.e., we shall compute δ_j for $j \in \{1, 2\}$. From the lecture:

$$\delta_j = g'(a_j) \sum_{k \in Dest(j)} w_{kj} \delta_k = z_j(1 - z_j) w_{3j} \delta_3$$

Having these δs, we can compute the gradient of the loss function w.r.t. weights

$$\nabla E(\mathbf{w}) = \begin{pmatrix} \frac{\partial E}{\partial w_{10}} = \delta_1, & \frac{\partial E}{\partial w_{11}} = \delta_1 x_1, & \frac{\partial E}{\partial w_{12}} = \delta_1 x_2 \\ \frac{\partial E}{\partial w_{20}} = \delta_2, & \frac{\partial E}{\partial w_{21}} = \delta_2 x_1, & \frac{\partial E}{\partial w_{22}} = \delta_2 x_2 \\ \frac{\partial E}{\partial w_{30}} = \delta_3, & \frac{\partial E}{\partial w_{31}} = \delta_3 z_1, & \frac{\partial E}{\partial w_{32}} = \delta_3 z_2 \end{pmatrix}$$

Now we need to update the weights for all the training data and iterate the algorithm for several epochs until the loss stops improving (i.e. is smaller than a predefined ϵ).

Task 8: Experiment with the number of epochs and plot the loss as a function of epochs.

Task 9: Visualize the decision boundary for the training set. Don't be afraid to run learning for large number of epoch and small ϵ and observe what happens with decision boundary.

Task 10: Visualize output of z_1 and z_2 as 2D plot and compare it with plot of x_1 and x_2. What can you see? For better clarity you can plot it for different epochs.

2 MLP from scikit-learn

Task 11: Use the `MLPClassifier` from scikit learn with the same architecture (2-2-1) and plot the decision boundary.

Hints:

- Read the documentation, find out how to set up the network architecture using the parameters `hidden_layer_sizes` and `activation`.
- Experiment with the `solver` parameter (values: `lbfgs`, `sgd`, `adam`).
- If the model learning does not work, try to change the default values of `max_iter`, `tol`, ...

3 Experiments

Consider another training set, the so-called XOR problem in Figure 3. These data are in the file: `data_xor.csv`.

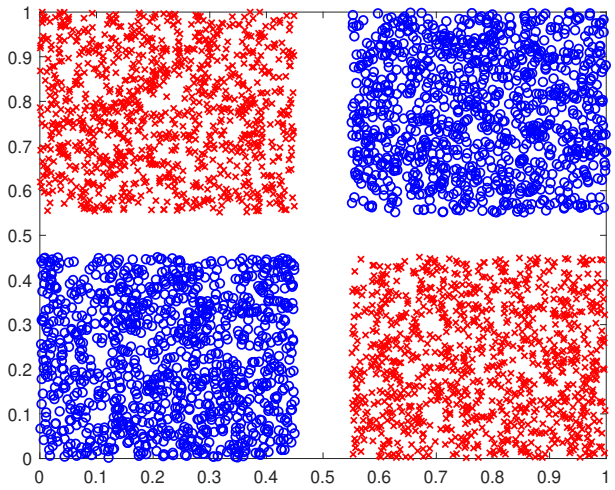

Figure 3: Training data - The XOR problem

Task 12: Use the same architecture. Can you classify the data correctly?

Task 13: Increase the complexity of the network by increasing number of hidden units.

Task 14: Visualize the decision boundaries.

Hints:

- Try to zoom out from figure, e.g. by changing limits (parameter `offset` of function `plot_xy_2D_model()`).
- Try to run the code many times and look at the learned model. Can you rely on the model outside the area covered by the training data?

4 Have fun!

Complete the exercise as homework, ask questions on the forum, and upload the solution via BRUTE!