Machine Learning
Università della Svizzera Italiana
Faculty of Informatics
October 18, 2019

Assignment 1: Neural Networks

Giorgia Adorni (giorgia.adorni@usi.ch)

1 The Perceptron

Before working on a neural network we will study the perceptron; a linear classifier without an activation function or a very simple single layer network as given in Figure 1. You are given a dataset of size $n : \mathcal{D} = \{(\overline{x}_1, \hat{y}_1), (\overline{x}_n, \hat{y}_2), \ldots, (\overline{x}_n, \hat{y}_n)\}$, where input is represented as k-dimensional vectors $\overline{x}_i \in \mathbb{R}^k$, and the targets $\hat{y}_i \in \mathbb{R}$ are scalar. Assume bias term $b \in \mathbb{R}$ is not part of weight matrix $\underline{W} = [w_1, w_2, \ldots, w_k]^T$.

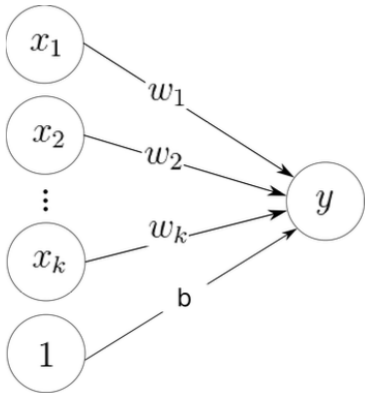

Figure 1: The linear perceptron with single output

$\overline{x}_{i,1}$	$\overline{x}_{i,2}$	$\overline{x}_{i,3}$	$\hat{y}_i$
-9	-5	5	0
4	-7	-11	0
7	6	-1	1
-9	-5	4	1
-5	-6	-1	1
-4	-4	-8	0
5	7	-9	1
2	-4	3	0
-6	1	7	0
-10	6	-7	1

Table 1: The training dataset

1. What is the dimension of the output vector $\overline{y}$ if we assume a batch of input data in form $\underline{x} \in \mathbb{R}^{d \times k}$?

Solution:
The output shape of the vector $\overline{y}$ is d. Given as input d sample for each batch, since the output is just one, we have one output for each element in the batch, so $1 \times d$ output.

2. Write down the vectorised equation for the forward pass, if input $\underline{x} \in \mathbb{R}^{d \times k}$ is batch of data, and $\overline{1}_d$ is a length d long vector of ones (watch out for dimensions to match).

Solution:

$$\overline{y} = \underline{x}\,\underline{W} + \overline{1}_d\, b \;, \tag{1}$$

where:

- $\overline{y}$ is the output vector;
- $\underline{x}$ is the input matrix;
- $\underline{W}$ is the parameters matrix (weights);
- b is the bias;
- $\overline{1}_d$ is a vector with d ones.

In this specific case, the equation does not have the multiplication by the activation function σ, and the bias b is a scalar and not a transposed vector like in the general formula $\overline{y} = \sigma(\underline{x}\,\underline{W} + \overline{1}_d\,\overline{b}^T)$

3. Write down the vectorised equation for the MSE of the perceptron.

Solution:

$$MSE = \frac{1}{2\,d}\,(\overline{y} - \hat{\overline{y}})^T(\overline{y} - \hat{\overline{y}}) \tag{2}$$

where:

- d is the number of samples in the batch;
- $\hat{\overline{y}}$ is the target vector;
- $\overline{y}$ is the predicted vector.

4. Determine the derivative of the error function w.r.t weights *Hint:* $\frac{\partial \overline{x}^T \overline{x}}{\partial \overline{x}^T} = 2\overline{x}$

Solution:

- weights:

$$\frac{\partial E}{\partial \underline{W}} = \frac{\partial \overline{y}}{\partial \underline{W}} \frac{\partial E}{\partial \overline{y}} \tag{3}$$

- bias:

$$\frac{\partial E}{\partial b} = \frac{\partial \overline{y}}{\partial b} \frac{\partial E}{\partial \overline{y}} \tag{4}$$

Since in both the cases, the second factor become

$$\frac{\partial E}{\partial \overline{y}} = \frac{\partial}{\partial \overline{y}} \frac{1}{2\,d} (\overline{y} - \hat{\overline{y}})^T(\overline{y} - \hat{\overline{y}}) = \frac{1}{2\,d}\,2\,(\overline{y} - \hat{\overline{y}}) = \frac{1}{d}\,(\overline{y} - \hat{\overline{y}})$$

and

$$\frac{\partial \overline{y}}{\partial \underline{W}} = \frac{\partial}{\partial \underline{W}} \underline{x}\,\underline{W} + \overline{1}_d\,b = \underline{x}^T \ ,$$

$$\frac{\partial \overline{y}}{\partial b} = \overline{1}_d^T \ ,$$

we obtain

- weights:

$$\frac{\partial E}{\partial \underline{W}} = \frac{\partial \overline{y}}{\partial \underline{W}} \frac{\partial E}{\partial \overline{y}} = \frac{1}{d} \underline{x}^T (\overline{y} - \hat{\overline{y}}) \tag{5}$$

- bias:

$$\frac{\partial E}{\partial b} = \frac{\partial \overline{y}}{\partial b} \frac{\partial E}{\partial \overline{y}} = \frac{1}{d} \overline{1}_d^T (\overline{y} - \hat{\overline{y}}) \tag{6}$$

5. Write down the equation of the weight update by gradient descent.

Solution:
From the derivatives of the error function defined in the Equations 5 and 6, we can compute the parameters update using the following formulas:

- weights:

$$\underline{W}^{\text{NEW}} = \underline{W}^{\text{OLD}} - \eta \frac{\partial E}{\partial \underline{W}} = \underline{W}^{\text{OLD}} - \eta \frac{1}{d} \underline{x}^T (\overline{y} - \hat{\overline{y}}) \tag{7}$$

- bias:

$$b^{\text{NEW}} = b^{\text{OLD}} - \eta \frac{\partial E}{\partial b} = b^{\text{OLD}} - \eta \frac{1}{d} \overline{1}_d^T (\overline{y} - \hat{\overline{y}}) \tag{8}$$

6. Suppose $k = 3$, $d = 10$, with a batch of data given in Table 1. The initial weights are $w_1 = -0.1$, $w_2 = -0.3$, $w_3 = 0.2$ and the bias weight is $b = 2$. Compute the weights after one step of gradient descent with learning rate of $\eta = 0.02$ (report their value up to 2 decimal points). *Hint: MATLAB/Octave/Python might come in handy here.*

Solution:
The updated weights are $[0.17, -0.03, 0.03]$ and the updated bias is 1.97.

```python
import numpy as np

x = np.array([[ -9, -5,  5],
              [  4, -7, -11],
              [  7,  6,  -1],
              [ -9, -5,  4],
              [ -5, -6,  -1],
              [ -4, -4,  -8],
              [  5,  7,  -9],
              [  2, -4,  3],
              [ -6,  1,  7],
              [-10,  6,  -7]], dtype = np.float32)

y_hat = np.array([0, 0, 1, 1, 1, 0, 1, 0, 0, 1], dtype = np.float32)

W = np.array([-0.1, -0.3, 0.2], dtype = np.float32)
b = 2
eta = 0.02
ones_d = np.ones(10)
y = np.matmul(x, W) + b
delta = y - y_hat

W_new = W - eta * np.matmul(x.T, delta) * (1 / d)
b_new = b - eta * np.matmul(ones_d.T, delta) * (1 / d)

print("New weights = ", W_new.round(2), " \nNew bias = %.2f" % b_new)
```

7. Learning in multi-layer neural networks is usually done with help of two methods: backpropagation and gradient descent. Describe briefly what role in the learning process each of the two has (Should not be longer than 6 lines).

Solution:
Backpropagation is a procedure for computing the gradient of complex functions in an efficient way. In particular it consists in calculating the derivative of the error function with respect to the parameters, applying repeatedly the chain rule.

Gradient descent is an iterative optimization algorithm that tries to improve the model performance finding a local minimum of a given loss function, updating the network parameters (weights and bias) by using the gradient of this function.