Assignement 1

Ewen Cazuc

February 23, 2024

Abstract

You can find the code corresponding to this assignment in this GitHub repository

1

Let $X_1, X_2, \ldots, X_n$, we want to find the value of b so $J = \sum_i^n (X_i - b)^2$ is minimized.

1.1 Analytic solution for b

To minimize the sum, we have to compute its derivative :

$$J' = 2\sum_i^n (X_i - b)$$

$$<=> J' = 2((\sum_i^n X_i) - nb)$$

For J to be minimized we want J' = 0 :

$$(\sum_i^n X_i) - nb = 0$$

$$<=> \sum_i^n X_i = nb$$

$$<=> b = \frac{\sum_i^n X_i}{n}$$

For J to be minimized, b has to be equal to the mean of X_i

1.2 Relation with normal distribution

If we rewrite this sum considering noisy observations, each X_i become $x_i + eps_i$

$$K = \sum_i^n (x_i + eps_i - b)^2$$

The noise terms eps_i are independent and identically distributed (iid) according to a normal distribution with mean 0 and variance $sigma^2$

The sum of squared errors can be interpreted as the negative log-likelihood of a normal distribution. Finding the value of b that minimizes K is equivalent to finding the maximum likelihood estimation of the mean of the normal distribution.

1.3 $(X_i - b)^2$ becomes $|X_i - b|$

Here we cannot use the derivative because the absolute value function is not differentiable at the point where x = 0. However the optimal solution for b should be the median value of X_i.

2

Linear function on (x, 1): g((x, 1)) = a x + c, where a is a scalar coefficient and c is a scalar constant.

So when we apply the formula $x^TW + b$ for example to this matrix x

$$X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}$$

We obtain :

$$\begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \end{pmatrix} * \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + \begin{pmatrix} b \\ b \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + b \\ x_{21}w_1 + x_{22}w_2 + b \end{pmatrix}$$

So we have for each line a linear function with the coeficient a is w and the constant c is b.

3

In this question w try to formulate a quadratic function of x in a neural network :

$$f(x) = b + \sum_i w_i x_i + \sum_{j<=i} w_{ij} x_i x_j$$

Let's do this with an example using 3 features, here is the neural network that we need :

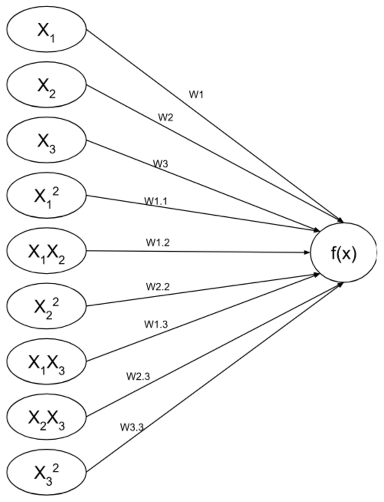


Figure 1: Neural network formulating a quadratic function for 3 features

In that case we obtain :

$$f(x) = b + w_1x_1 + w_2x_2 + w_3x_3 + w_{1.1}x_1^2 + w_{1.2}x_1x_2 + w_{2.2}x_2^2 + w_{1.3}x_1x_3 + w_{2.3}x_2x_3 + w_{3.3}x_3^2$$

This corresponds to the quadratic function for i = 3.

4

For this question I used the implementation of a multilayer perceptrons of the section 5.2 in the textbook.

4.1 Weights initialized to 0

If we initialize every weights to 0, we will have the symmetry problem which means that the output of each neurons in the hidden layer will always be the same and so will be the outputs of the neural network. The algorithm will never learn so won't work anymore.

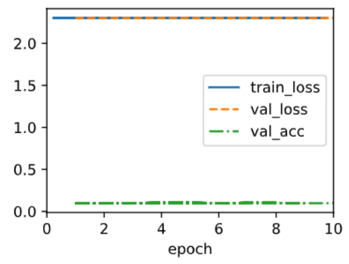


Figure 2: Loss for test and validation when weights initialized to 0

And for example, if we initialize only the weights between the input and the hidden layer, the algorithm will work but will learn only between the hidden layer and the outputs.

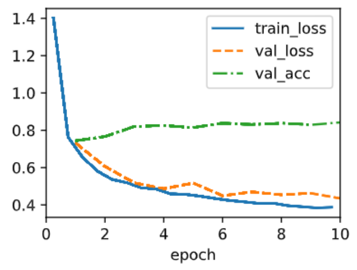


Figure 3: Loss for test and validation when weights for only first layer initialized to 0

4.2 weights initialized with a normal distribution, variance = 1000

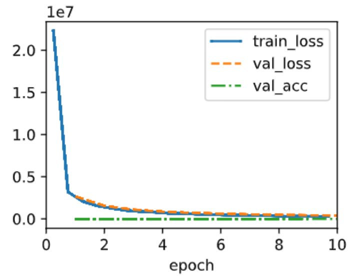


Figure 4: Loss for test and validation when weights between inputs and hidden layer initialized to 0

At the beginning the loss is really high, probably due to the great difference between the weights as we use a variance of 1000.

5

For this question I also used the implementation of a multilayer perceptrons of the section 5.2 in the textbook using the same value of parameters for each value of learning rate.

5.1 Experimenting different values of learning rate

Let's experiment different values of learning rate to see how fast the loss decreases for each value.

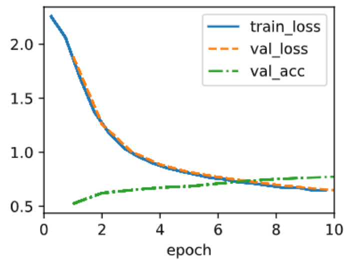


Figure 5: Learning rate = 0.01

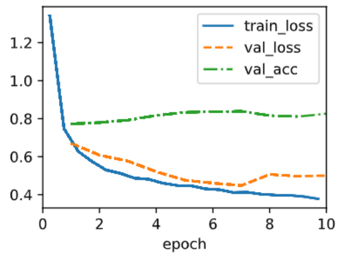


Figure 6: Learning rate = 0.1

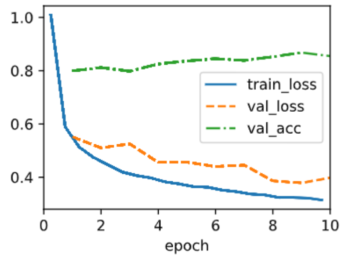


Figure 7: Learning rate = 0.3

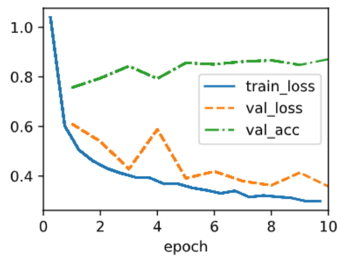


Figure 8: Learning rate = 0.5

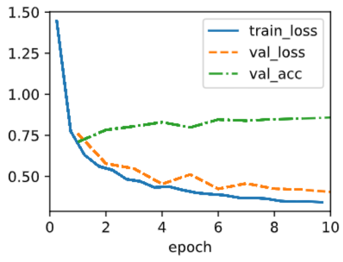


Figure 9: Learning rate = 0.7

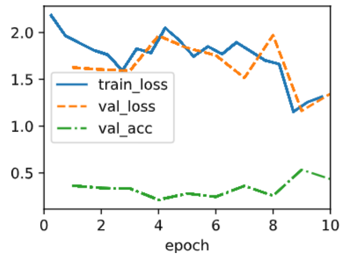


Figure 10: Learning rate = 1

We can see that with the very low learning rate lr = 0.01, there is almost no gap between the test loss and the validation loss but both are quite high because it decreases too slowly between two epochs. It could have decreased more if we had more epochs.

When we increase the learning rate, the loss decreases and the accuracy increases as we always have the same number of epochs. There is here a gap between the test loss and the validation loss as the algorithm doesn't learn as well as when we have a really small learning rate.

When we have a learning rate lr = 1, it is a to great value and the algorithm don't have time to learn so the loss sometimes increases and sometimes decreases between to epochs.

5.2 Increasing the number of epoch

Let's see if by increasing the number of epoch we can lower the value of the loss :

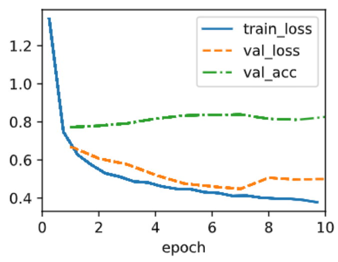


Figure 11: 10 epochs

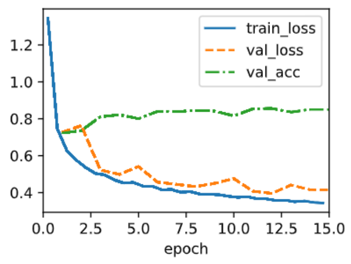


Figure 12: 15 epochs

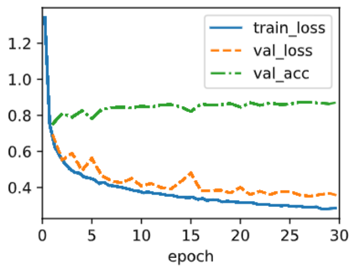


Figure 13: 30 epochs

We can see that between 10 and 15 epochs the loss value has decreased but no so much between 15 and 30 epochs (considering that there is a gap of 15 epochs). Indeed, the value of the loss reach a limit

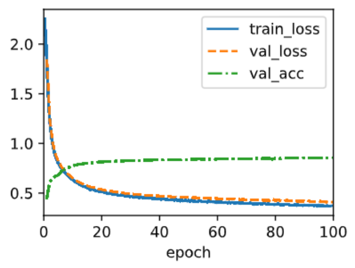


Figure 14: Learning rate = 0.01 and 100 epochs

6

In PyTorch, if the number of examples in your dataset is not divisible by the batch size, the data iterator handles the remaining examples in the last batch of the epoch.

7

K-fold cross-validation is very expensive to compute because the training is done on (K - 1) subset of the dataset and the test on one subset K times !

8

Let's experiment with the value of the weight decay in the estimation problem in Section 3.7.

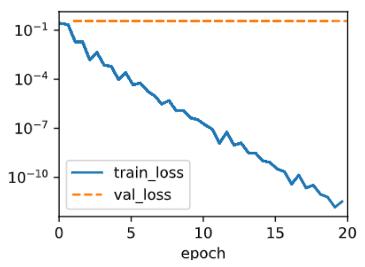


Figure 15: Lambda = 0

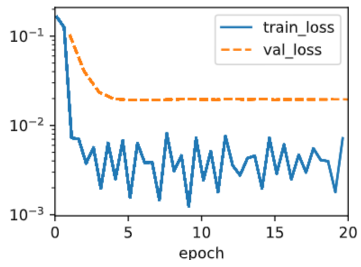


Figure 16: Lambda = 15

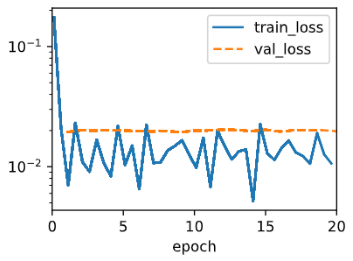


Figure 17: Lambda = 60

9

The Frobenius norm is defined as the square root of the sum of the squares of a matrix's elements :

$$\|\mathbf{X}\|_F = \sqrt{\sum_i^n \sum_i^n x_{ij}^2}$$

10

After performing the house price prediction problem in Section 5.7 using the implementation of the notebook I obtained a log mse = 0.03004748551174998, using lr = 0.33, max epoch = 100 and k = 15.

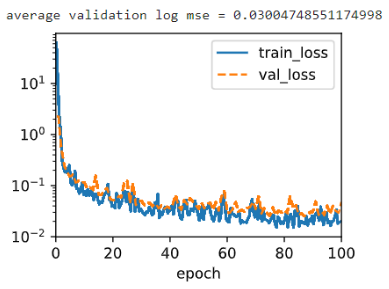
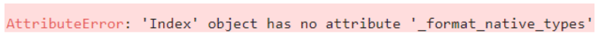


Figure 18: Training and validation losses per epoch - house price prediction

I coudn't submit my work due to this error that I coudn't fix :



```
AttributeError: 'Index' object has no attribute '_format_native_types'
```

Figure 19: Error on House price problem