HW6: Artificial Neural Networks

Marko Ivanovski

Introduction

Neural networks reflect the behavior of the human brain, allowing the network to recognize patterns and solve different types of predictive tasks. For the following task, we were supposed to implement a Neural Network classification and regression model from scratch. Then we received two datasets, which tested both models. Finally, on the third dataset, we experimented with different architectures and regularization weights in order to return the best predictions on the test dataset.

Implementation

The implementation of such models is the trickiest part since it involves gradient computation using the chain rule and back-propagating it towards the start of the network. As an activation function, I used the sigmoid. Both classification and regression architectures are the same up to the last layer. While implementing and computing the gradient of the cost function, I noticed that both softmax and sum of squared error share the same gradient for the weights in the last layer ($A^L - Y$). This meant that the backpropagation function was shared for both models. The only difference is in the feed-forward stage where the classification applies softmax and the regression is just a linear combination of A^{L-1} and W^L. For optimizing the weights I implemented my own Gradient Descent procedure. This also meant that my algorithm has a learning rate which can be tweaked (I always set it to 0.1 or 0.2). Lastly, since I noticed that running GD on a predefined number of iterations can be inefficient I implemented two types of early stopping.

- If the loss has increased in 5 consecutive iterations on the validation dataset the algorithm stops,
- If the loss hasn't changed up to a small decimal in 5 consecutive frames on the validation dataset the algorithm stops.

Numerically verification

In order to check if my gradients are correctly calculated for the cost function I also computed them by a simple linear approximation:

$$\frac{dJ}{dw_i} \approx \lim_{h\to 0} \frac{J(w_1,\ldots,w_i+h,\ldots,w_n) - J(w_1,\ldots,w_i,\ldots,w_n)}{h} \quad (1)$$

This equation needs to be satisfied for every weight in the network. I checked the gradients for both classification and regression networks with a tolerance at the fifth decimal and they were all almost equal. Following is a very simplified version of how this was checked:

```
#Numeric gradient
h = 0.000001
pred=self.predict(X)
f_right = cross_entropy_loss(y, pred)

numerical_gradients = []
for weight wi:
    #add small h to that weight
    W[i]+=h
    pred=self.predict(X)
    f_left = cross_entropy_loss(y, pred)
    #calculate the gradient
    dW[wi]=(f_left-f_right)/h
    #revert the weight back
    W[i]-=h

#Correct gradient
#Feed forward
A = feed_forward(...)
#Backprop and obtain gradient
dW = backpropagation(A,...)

for i,j in zip(dW, numerical_gradients):
    np.testing.assert_allclose(i, j,
     ↪  atol=0.00001)
```

Housing2r and housing3 datasets

Before I evaluated how the algorithm performed on both datasets, I standardized all the columns in them. This ensures numeric stability and no possible overflow. The evaluation was performed on a 30% test split. On the housing3 dataset I compared the multinomial regression (with 2 classes from HW2) and a simple network with one hidden layer which contained 5 neurons and λ=0.1. The results were:

- LR cross-entropy loss: 0.38
- NN cross-entropy loss: 0.33

Finally on the housing2r I compared Ridge Regression with λ=0.1 (from HW3) and the same architecture as for the previous dataset. The results were:

- RidgeReg regression loss: 24.20
- NN regression loss: 9.58

Final dataset

For the final dataset I evaluated more architectures and regularization parameters. Before that, I standardized the train dataset and the test using the same mean and std from the train one. Also, since the loss during training was constantly dropping I used the second type of early stopping described

Architecture	Cross-entropy loss	Architecture	Cross-entropy loss
[5,5,5,5,5,5] lbd=0.1	1.9532	[40] lbd=0.1	0.5950
[20,20,20,20] lbd=0.1	1.9532	[30] lbd=0.1	0.5919
[20,20,20] lbd=0.1	1.9530	[40] lbd=1	**0.5695**
[20,20] lbd=0.1	0.6337	[60] lbd=0.1	0.5705
[50] lbd=0.1	0.6013	[50] lbd=0.001	0.6050
[100] lbd=0.1	0.6078	[50] lbd=1	**0.5696**
[70] lbd=0.1	0.6041	[50] lbd=5	0.6025

Table 1. Neural network loss on different architectures and different regularization values, evaluated on 30% of the train dataset.

previously. The parameters were chosen based on the performance on the validation set which was 30% of the train. Although CV would return more reliable estimates due to the time crunch this was the only feasible solution. Table 1 contains the final results. Analyzing the results in the end I decided to go with a single layer which contains 50 units and lambda=1. The whole training and writing results procedure took 2 hours and 30 minutes.