Machine Learning for Data Science HW4 Neural Networks

Nikolay Kormushev

Introduction

In this report, I implement two versions of a fully connected regularized neural network from scratch. One version for classification and one version for regression.

Afterwards I use one housing dataset and one bigger and more complicated dataset to compare my results to simpler models like ridge and multinomial regression by optimizing my neural network.

Implementation Details

Both

For both models I add an intercept and use the $fmin\_l\_bfgs\_b$ optimizer. My architecture was a feed forward layer followed by a ReLU layer, initialized using he initialization which when for ReLU layers preserves the variance between layers and avoids vanishing gradients. This pattern is followed for all except the first and final layers. The latter I will explain in the next sections but the first I randomly initialized because I found they were sensitive and if I used xavier or he initialization probably due to the small numbers my gradient calculations failed. For both models I use L2 regularization but it is possible to swap it out easily for L1.

Regression NN

The final layer for regression is just a feed forward layer without an activation initialiazed using xavier to preserve variance in the activations and avoid vanishing or exploding gradients. For loss I used Mean squared error to motivate my network to predict values close to the mean.

Classification NN

The final layer here is again fully connected layer but with a softmax activation. The initialisation is xavier for the same reasons as before, but this time the bias is not set to 0 as normally but $\frac{1}{n}$ where n is the number of output classes so at the start we have an equal chance of predicting each class which can lead to the network converging faster.

For a loss I use log loss because we have categorical values and it preserves the distribution of the classes in the dataset.

Testing

For testing I used the python unittest library. To validate my methods I compared my gradients for the NN with the analytical gradients computed by adding and removing a small offset to each weight separately and taking the partial derivatives using the derivative formula.

Most of my other tests were component tests for my regression model classes which looked at my methods in isolation to test for edge cases.

Results

Evaluation

I compared the NN models with simpler models like linear regression and multinomial regression. For accurate results I used k-fold cross validation to test on different train test splits and compared the final results. To account for the variance in the initialization of the weights I also ran these k-folds 100 times.

For the comparison I also used neural networks with 1 hidden layer. 1 layer is needed to introduce a non-linearity through the activation function because with 0 layers we revert to the original algorithms for linear and multinomial regression. We also know that theoretically one layer is sufficient to approximate any function. Adding more layers is more or less necessary due to memory constraints so if we show one layer is better for sure more will be as well. Also if we look it another way adding more weights or layers will just help us learn more patterns so as long as we make sure not to overfit we should perform better.

Housing regression

I used scikit learns ridge regression with a regularization of 0.001. On Table 2 we can see how much was the avg of the avg of the k-fold errors through the 100 iterations. We can also see the mean of the standard error of the k folds for the 100 iterations.

The results show that the ANN is performing much better than the Ridge regression as we expected on average. The stderr based on the distribution is quite similar although better for ANN. I believe this is probably due to using the same splits in the k-folds for each of the 100 iterations.

Table 1. Error comparison regression

Model	Mean squared error	k-fold stderr
ANN	26.985 ± 0.042	± 7.001
Ridge Regression	37.432 ± 0	± 7.804

Housing classification

I used the multinomial regression I implemented for my last homework for comparison.

Table 2. Error comparison classification

Model	Log loss	k-fold stderr
ANN	0.254 ± 0.0003	± 0.014
Multinomial Regression	0.278 ± 0	± 0.025

Again the results show the classification ANN performing better than the multinomial regression with a smaller average loss. The k-fold standard error are somewhat similar which is probably again due to using the same folds. And after 100 iterations our standard error of the mean of the means is pretty much 0.

Choosing parameters on large dataset

For the last part first I drop the id column. After I normalized the training data saving the means and standard deviation so I can use them on the test data later.

Then I choose the model parameters based on a grid search cross validation which I implemented to choose the best lambda for my implementations and the dataset. I also used the GridSearchCV method from scikit-learn to see how it works which required me to implement some extra methods in my models like setParams and score.

In Figure 1 we can see some of the combinations I tried with the grid search. The resulting best parameters were $units = [160]$ and $lambda = 18$. I trained a model using them and saved the predictions and also measured execution time by training 100 times the results of which of which can be seen in Table 3. I tried multilayer networks as well which are not on the plot but I did not get any improvement from them.

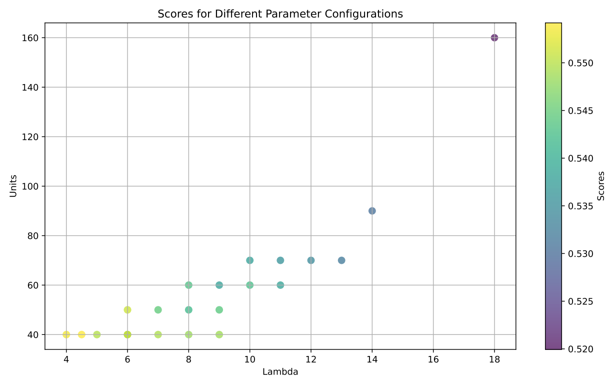

Figure 1. Tested combinations of units and lambda

While running this I also limited maxiter of the optimizer to 100 and tolerance to 0.001. I do this to avoid overfitting. The number 100 was chosen with the interest of time. The tolerance seemed good for me since if we try to optimize with steps smaller then the 3rd decimal I believe the improvement in performance on average datasets shouldn't be big.

Table 3. Execution time

Model	Execution time
ANN 1 layer 160 unit and lambda 18	134.21 ± 1.98

Conclusion

As a conclusion I delved into neural networks and initialization methods and how to better avoid vanishing gradients and how sensitive the network can be to the initial weight values. Often gradients would not be calculated correctly if values were too large or small.

From the tests I ran we can see that neural network do better than their more simpler counterparts as expected since they have more weights and they can learn more although we should be wary of ovefitting which we can avoid somewhat with regularization. I also played around with grid search for optimizing a network and choosing parameters.