UDACITY

PROJECT

# Your first neural network

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

## Meets Specifications

Good job implementing your first neural network! This is the essential and foundational building block of Deep Learning.

With respect to the final question, the main region of error in the model is an overestimate in late December. This is the USA Holiday Season and we don't have enough training examples of this, to model it accurately.

### Code Functionality

**All the code in the notebook runs in Python 3 without failing, and all unit tests pass.**

**The sigmoid activation function is implemented correctly**

Nice use of lambda function for implementation of sigmoid activation. An alternative implementation is:

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

### Forward Pass

**The forward pass is correctly implemented for the network's training.**

**The run method correctly produces the desired regression output for the neural network.**

Yes, we must activate the output to the hidden layer via the sigmoid function implemented earlier in the code. This squeezes our inputs into probability space.

**The input to the output layer is implemented correctly in both the train and run methods.**

**The output of the network is implemented correctly in both the train and run methods.**

This is an excellent implementation of a regression network output. When building regression model the output should be the raw input to the output layer, that is final_inputs.

### Backward Pass

**The network correctly implements the backward pass for each batch, correctly updating the weight change.**

**Updates to both the input-to-hidden and hidden-to-output weights are implemented correctly.**

## Hyperparameters

**The number of epochs is chosen such the network is trained well enough to accurately make predictions but is not overfitting to the training data.**

Good job tuning! Manual tuning is mostly science but still some art. Overall a good approach is to use the learning curve as a guide. The learning curve is an essential guide to tuning and can really tell us if we have the correct settings.

Tuning, especially manual tuning, is challenging, but it is a great experience. Once you have successfully tuned an NN or any other ML, manually, you have great insight into what is going on with automated methods (they are no longer black box).

The key is to use the plot to find the right combination of parameters. A good approach is to set the hidden nodes and find a good number of epochs and learning rate for that number of hidden nodes. Nodes can then be adjusted up or down depending on the fit (determined by looking at the final plot comparing the model to the actual data).

To optimize epochs, it's good to keep going until you see the validation set start to increase while the training set continues to decrease. The optimized epoch setting is just below this (just below overfitting). It may be required to adjust the number of epochs after making revisions to other settings.

One way we can know when to stop is by looking at descent of the training curve over a certain number of epochs (say 10% of total epochs). If the slope is not changing from one epoch window to another, we can stop. This is known as "early stopping."

**The number of hidden units is chosen such that the network is able to accurately predict the number of bike riders, is able to generalize, and is not overfitting.**

15 hidden units is reasonable for capturing the complexity of the data.

The number of hidden units must be large enough to avoid high bias (underfitting) but not so large as to create a high variance (overfit) model. Essentially, to avoid overfitting the more hidden units we have the more data we need.

Generally, the number of hidden units should not exceed twice the number of input units, but must be enough that the network can generalize. A good guideline is halfway between the number of input and output units. This resource may help develop further understanding.

https://www.quora.com/How-do-I-decide-the-number-of-nodes-in-a-hidden-layer-of-a-neural-network

Based on the above we should try at least 8 units and an effective range is 10 - 20. Per previous comments, something fun and informative for building models is to continue to play with parameters until we see the validation error start to increase while the training error continues to decrease. This is a sign of overfitting. Once that is observed we can dial back our parameters slightly to achieve a high performing model.

**The learning rate is chosen such that the network successfully converges, but is still time efficient.**

The learning rate should be in a range where the our ratio of self.lr to number of records is ~0.01. Too high of a learning rate causes learning curves that jump down in the beginning and then flatten quickly. By lowering the rate we have a more gradual and sustained descent.

It's tempting to use a low learning rate such as a quotient of 0.001 since this slows descent and increase overall sampling. The downside of this is that it may take a long time to converge or even worse, we could get stuck in a local minimum and never reach the global minimum.

The current quotient is 0.1/128 is a bit low, but also appears to have worked. Try playing with a quotient of 0.1 - 0.01 (I would try moving up from the lower bound). As the learning rate changes the number of epochs should be adjusted in inverse proportion (fancy way of saying that as one increases the other should decrease.)

As an aside, learning rates as low as 0.1 (not the quotient, the actual rate), can work. Using a higher learning rate just helps us save on computational load, since we user fewer epochs and every epoch represents a pass through the network (forward and backward).

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ