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

Re your question and your approach: Do not rely too much on the result of a single run. NN is a randomized algorithm, and the current implementation with constant learning rate you are guaranteed to have big variability even in last epochs. The way to account for this is to introduce decay factor (say 0.99) by which you would multiply your learning rate after each epoch to allow the network to converge somewhere, and make the model in the later epochs more stable. Even so, in order to compare the parameters, do not compare them on a result of single run, but rather on average of a series of runs with each combination of parameters (say 1000). This is not what is expected of you in this project, but if you decide to do it professionally in the future, this is a good start.
On training error still decreasing and validation error staying the same. Since you goal is to build a model to predict the correct values of data points which network didn't see, you have to worry about validation set in the first place. If validation set does not seem to decrease any more, you should probably stop the learning, and if you are unhappy with the results, try different parameters/model/approach.

You made great job on this project. Your network produces the correct results, and your implementation is great!
Good luck with the nanodegree!

Code Functionality

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

Your code passed the automatic unit tests. Great job!

The sigmoid activation function is implemented correctly

I am glad to see you use lambda function here

Forward Pass

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

The output of the hidden layer is implemented correctly in both the `train` and `run` methods.

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.

Yay, you got forward pass right!

Backward Pass

The network output error is implemented correctly

Updates to both the weights are implemented correctly.

Your implementation of back-propagation algorithm is perfect!

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. The way to judge if the number of epochs is too high is by taking a look at validation loss graph. If it start to increase again, then it means the neural network starts to overfit.
On the other side, if the validation loss is still decreasing right until the end, it means that you have ended the training too early, and the number of epochs is too low.
Most students used 1000-10000 epochs.

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 adequate for this project. In general, as a rule of thumb you can start to experiment with the number of units equal to the mean of the number of units in the neighboring layers. So in this project with 56 input units and 1 output unit, you could have started with something around 25.
For more information you can read this resource: https://www.quora.com/How-do-I-decide-the-number-of-nodes-in-a-hidden-layer-of-a-neural-network

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

0.01 is perfect!

DOWNLOAD PROJECT

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Rate this review

Student FAQ