PROJECT

Your first neural network

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

Excellent work in setting up your first neural network and applying it to real world data.

I noticed that you left out the analysis on prediction result. Being an optional part of the project, it's ok to do that. At any rate, If you look closely at the last chart, you will see that the model's predictions are not really good during the Christmas period. This is mainly because Christmas is a bit special in that the number of riders is relatively smaller during that period, and our model has only seen such periods once during the training (we have only data for two years). In another words, not enough data for the model to learn well to predict periods like Christmas.

Your project passed review. I also left some comments/suggestions for your reference. Happy Learning!

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 work here. One alternative implementation you might be interested in is to do it with one line: self.activation_function = lambda x: 1/(1+np.exp(-x))

✓ **All unit tests must be passing**

Forward Pass

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

First of all, nice job in reusing self.run method to remove duplicates codes. I like it!

Great work in using matrix multiplication to implement the affine transformation (https://en.wikipedia.org/wiki/Affine_transformation), which is a key component in neural network model. Vectorization like this can greatly speed up computations.
If you feel adventurous, you could even try adding a bias term. Of course, this is not mandatory requirement for this project.

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

Perfect!

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

Great work here! Yes, in this model, the output layer should take the output of hidden layer as input.

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

Good catch here. In neural network, the role of the activation function is to introduce non-linearity and allow the model to produce a non-linear decision boundary via non-linear combinations of the weighted inputs. (https://www.quora.com/What-is-the-role-of-the-activation-function-in-a-neural-network)
Here the network is being used for regression, the output of the network should be the raw input to the output unit. In another words, there is not activation function for the output unit, or we would say f(x) = x.

Backward Pass

✓ **The network output error is implemented correctly**

Correct!

✓ **The error propagated back to the hidden layer is implemented correctly**

Well done. The output error is properly propagated back to the hidden layer with the hidden-to-output weights.

✓ **Updates to both the weights are implemented correctly.**

Great job in correctly updating the weights via their corresponding derivatives.

✓ **Hidden layer gradient(hidden_grad) is calculated correctly.**

Hidden layer gradient(hidden_grad) is actually the gradient for sigmoid function. Well done.

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.**

Well done. looking at your plot of training and testing loss, I can see that you chose an appropriate number of epochs such that the loss on the training set is low and the loss on the testing set isn't increasing.

✓ **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.**

Hi, saw you notes.

I think 24 is certainly a valid pick. considering that the model's performance "Training loss: 0.082 ... Validation loss: 0.155" is pretty much what we can get with current data/features/model architectures. Guess previous reviewer tries to have you experiment a bit more.

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

Again, I think 0.085 is a valid pick. considering that the training loss consistently decrease and the model converges within a reasonable time frame.

DOWNLOAD PROJECT

RETURN TO PATH