UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Predicting Bike-Sharing Patterns

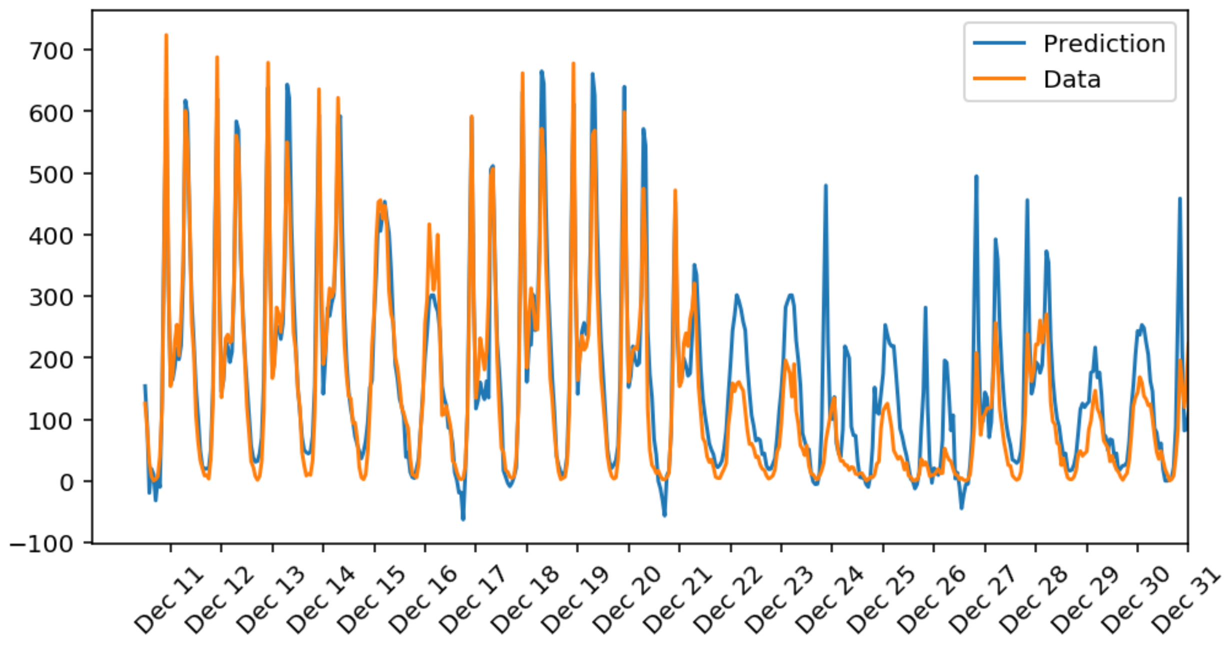
REVIEW

CODE REVIEW 2

HISTORY

Meets Specifications

Congratulations on finishing up the project successfully. I have added all the pros and cons of your project in the given specification below. Overall it was Awesome Project.



Your model actually fails to perform after 21st December because of the Christmas Holidays in which bicycle usage dropped due to the festive. Neural net failed to capture this pattern during the Christmas festival. You need to keep in mind that the neural networks need a lot of data to train on so that it can make accurate predictions.

Good luck with the nano degree! You're going to have fun in learning more about the neural networks.

Code Functionality

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

Your code works well! The model has converged to a loss minima! Also, all the unit test passes

```
<unittest.runner.TextTestResult run=5 errors=0 failures=0>
```

The sigmoid activation function is implemented correctly

Nice!
Use of lambda function shows you have a good background of python programming language.

- A sigmoid function maps any value to a value between 0 and 1. We use it to convert numbers to cumulative probabilities.
- Sigmoid activation units have a problem of vanishing gradient descent. Check out this forum post for more details about it https://discussions.udacity.com/t/relu-vs-softmax/220732/7
- Check out this forum post to know about other kinds of non-linear activation units https://discussions.udacity.com/t/choice-of-activation-functions/450992/2

Forward Pass

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

Nice implementation of the matrix multiplication of the input-to-hidden weights. You can also introduce bias term in your work. Bias is a value that allows you to to shift the activation function to the left or right. Here is one of the many resources that you can look up to know more about the bias term.
https://www.quora.com/What-is-bias-in-artiÒcial-neural-network

Check out the implementation along with adding bias in the below link;
https://databoys.github.io/Feedforward/

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

Correct implementation of the activation function applied to the hidden layer inputs in both train and run.

Backward Pass

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

Correct way to multiply the hidden-to-output weights by the output of the hidden layer.

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

Correct implementation of weights being updated.

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.

Udacity project assistant API can operate for certain duration time for each run i.e. there's timeout limit. Having 5000 as number of iterations may exceed the timeout limit. The number of iterations of 2500-3000 is reasonable for this project to get the desired results provided the rest of the hyper-parameters are tuned optimally.

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.

Since the number of input units features .shape[1] = 56, the number of hidden units should not be more than twice of 56. It is also a good idea to keep the number of hidden units in between the number of input units and the output units. Therefore it will be better to start within the range of 30-50 number of hidden units and then see how the network does if you increase or decrease the number of hidden units further.

There’s also a good answer here for how to decide the number of nodes in the hidden layer.
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.

The value of the learning rate scales the size of weight updates. If this is too big, the weights tend to explode and the network fails to the data. Normally a good choice to start the learning rate value is 0.1; however, if you effectively divide the learning rate by n_records, try starting out with a learning rate of 1. In either case, if the network has problems fitting the data, try reducing the learning rate. So try the learning rate with more than 0.5 and less than 1.0

The number of output nodes is properly selected to solve the desired problem.

Check out this link for the neural net to be used as regression https://deeplearning4j.org/logistic-regression

The training loss is below 0.09 and the validation loss is below 0.18.

```
Progress: 100.0% ... Training loss: 0.060 ... Validation loss: 0.139
```

The above results meet the requirement of loss values.

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

RETURN TO PATH