UDACITY

PROJECT

Your first neural network

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

3 SPECIFICATIONS REQUIRE CHANGES

Good first attempt! 👏

The main problem with your network is that **the number of hidden units is too high**!!! This can make the network not good at generalizing for new data.

Just try to set a structure, choose a number: 8 hidden units will be easier to train and will generalize better and around 20 will take more time to train and will predict better.

I have also tried to guide you through the iterative process of selecting the parameters. This process will guarantee an optimal performance. You'll find further details in the review.

I really hope I was able to help you out and gave you resources and insights to go a little beyond!

> Please, be sure to rate my work! Hope you enjoy this project and the rest of the course! Cheers! 🍻

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

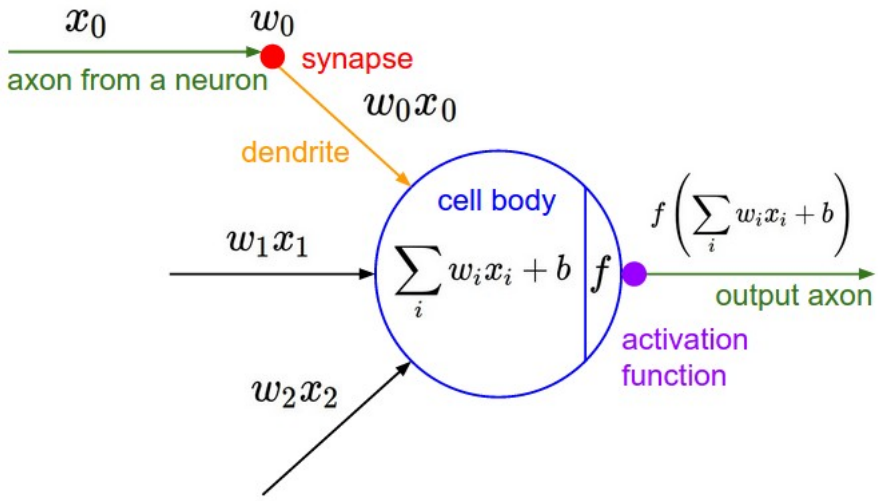
The sigmoid is well defined and implemented as a python `lambda` function, which helps to get shorter and more generic code. Well done here! 😎

Forward Pass

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

Nicely done! Student implemented the first hidden layer using numpy function `dot()` for matrix multiplication! The bias term could also be added to have a comprehensive a scheme.

Remember a layer of a neural network consists of a number of neurons connecting one layer to the next one, each one like this:



There you see how a simple bias term can be added before passing the input to the activation function.

Reference: Stanford CS231 Convolutional Neural Networks

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

Good! 👏 Both in `train` and `run` methods the output of the layers is obtained from the activation function, in the form `self.activation_function(hidden_inputs)`, which calls the sigmoid function defined in the constructor of the `NeuralNetwork` class.ac

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

The input to the output layer is correctly implemented, as done with the input to the hidden layer 👍. Same comments as in that part, bias could be added in this layer too.

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

Good work!! 👍 Since we are dealing with a regression problem and not classification, the output does not need to be limited to [0,1] interval. The output is the raw input to the output layer, with no function applied.

Backward Pass

The network output error is implemented correctly

Output errors calculated correctly!! 🎉

The output error is calculated as the target minus the output of the network.

Updates to both the weights are implemented correctly.

Everything is perfectly implemented 👏

- `hidden_error` term is well implemented, as the backpropagation of the output error.
- `hidden_error_term` is correct, taking into account the derivative of the sigmoid is sigmoid(x)(1-sigmoid(x)).
- `output_error_term` is correct, it is just the output `error`.
- The weights increments for the batch `delta_weights_i_h` and `delta_weights_h_o` are correct, taking into account the dimensions.
- Also, both `self.weights_hidden_to_output` and `self.weights_inputs_to_hidden` weights are updated properly and the number of records is considered. Well done in this part 👍

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 work so far! 👏🏻

However, as I will tell you below the number of hidden units is too high.

Also, the validation loss achieved is around 0.43, and for this project we are aiming fo validation error lower than 0.2.

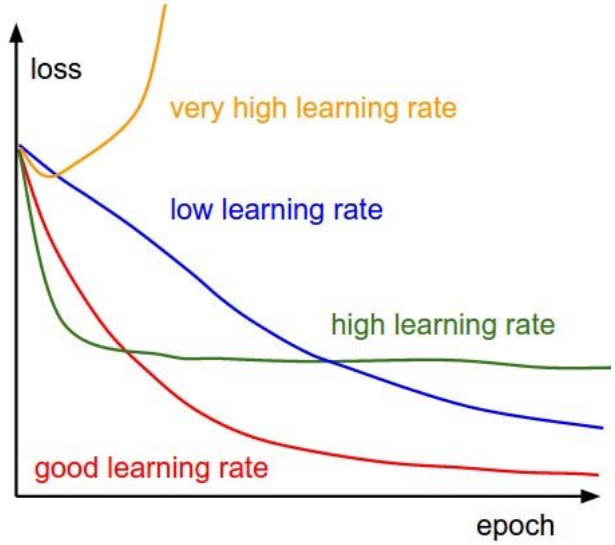
Another point is that the network is not stable yet, you can see how the curves are still decreasing. We should've added more epochs till the loss curves were constant.

I will tell you as few tweaks so your tuning procedure can be improved. I will comment all on this point:

- First, in order to be able to judge better, set the vertical limits **manually** to a reasonable range (i.e. `plt.ylim(0,2)`) to focus on the important part and be able to compare between curves with detail.
- Your network has too many **hidden units** and it is starting to incur in overfitting: decrease the number of hidden units a little bit. A good value is the mean between inputs and output, but it is fairly open. I have seen networks with good performance from 5 to 25 hidden nodes. It is up to you, but **120 is way too much**. My personal suggestion will be **8-20**.

Once the network structure is set, tune the other parameters:

- Choose a **learning rate** not too big, so we can reach the optimal point, and not too low so we can predict accurately. Use this image as criteria:



Reference Image 1

A good way of choosing the learning rate is to keep the quotient of the learning rate / the number of records should end up around 0.01 (i.e. self.lr / n_records ~ 0.01). This depends also on the structure of the network since more complicated networks usually require higher learning rate. Simpler networks may miss a lot with high learning rates. Usually, values used in this kind of networks range from 0.5-0.01, depending on the number of hidden nodes.

- Choose a number of **epochs** enough so we achieve a stationary state with low losses and validation loss not increasing. This depends on the learning rate: higher means fewer epochs, lower means more epochs. Do not worry if it is necessary to add more epochs: neural nets require training and it is perfectly normal to spend some minutes if we can guarantee an optimal convergence.

> 💡 My honest recommendation would be first to choose a moderate number of hidden units (from 8-20) and then, just try to achieve the red curve in the image I provided and then expand the number of epochs to see what is the validation loss achieved when the curve is stationary, flat.
>
> - If it is more than 0.2: then we should change the learning rate so we can go further and start the process again.
>
> Sometimes the learning rate has to be decreased and other times increased. One thing you can do to know that, is to try first to drecrease the learning rate and see if the minimum loss achieved is lower or not than before. If it is higher, then you should have increased the learning rate instead.
>
> - Once you improve the validation loss, see what is the best value achieved in stable state, and if it is less than 0.2, we're finished! 🙌🏻

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.

See "epochs section" ☝🏻.

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

See "epochs section" ☝ .

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Student FAQ Reviewer Agreement