CENG-463 Assignment #02 Report
Neuronal Network

Student: Joel Amarou Heuer, 202102201

0. Table of Content

---

1. Code Dependencies

Programming language

The code is written in Python-3 programming language.

Used libraries

- numpy
- pandas
- nltk (for tokenization of words; used with permission of Prof. Selma Tekir)
  → install with: `pip install --user -U nltk`

2. Questions

[Q.01] “What were the trade-offs you faced?”

Trade-Off in Pre-Processing

The most significant tradeoff I had to make was encoding the sentences/words. Before textual data can serve as input to the network, it has to be in numeric form.
I initially tried "one-hot" encoding as encoding, but my computer doesn't have enough processing power to implement this. So I switched to "Bag-of-Words" encoding. The disadvantage of this encoding is that the order of the words in the sentence is lost. Missing order may affect the performance of the prediction.

Input data (used feature)

Only the "commentsReview" feature was used as input data. Again, the trade-off is a possibly lower performance for lower complexity.

[Q.02] “What was the reason that your network learned?”

Input

Training input data (here: comments in text form) always have an associated result, the Y variable (here: ratings). One after the other, input data, including their Y-variable, are fed into the network's input layer to estimate the y-variable for the input data without knowing it. This procedure is the training phase of the model.

Forward Propagation

Every sample passes through all network layers moving forward (Forward Propagation). In this step, all neurons on each layer are calculated to estimate the sample's y-value in the output layer. Thereby the output of the neurons depends on the result of neurons of previous layers. In two steps, the calculation for one neuron at a time is done:

1. Calculation of the z-value: z = data of the previous layer * weights * bias (linear combination)
2. Calculation of the a-value: activation function(z)

From step 1, you can see that the weights and biases play an enormous role in the calculation. The goal is to adjust this in such a way that the estimate becomes very good.

An activation function applied to each neuron makes some neurons more important than others in the overall prediction calculation. This process terminates in the output layer, where the final prediction (regression) is calculated.

Backward Propagation

A loss is calculated since this estimation can differ from the truth (y-value). Based on this, the weights and biases of all neurons are adjusted (backward propagation). The loss is now propagated backward through the network. The formula used to calculate the partial derivative is derived from the chain rule.

Learning
The more samples pass through the network, i.e., forward & backpropagation are performed, the weights and biases for each neuron improve. The estimation gets closer to the truth (y-value), resulting in a minor loss. Overall this is why it is said that the network learns.

[Q.03] “How would you improve the accuracy more?”

The problem to be solved is a regression problem (estimation of valuations). Accordingly, no accuracy can be measured, only the loss (deviation between prediction and actual value). Accuracy can be measured in classification problems.
However, the loss can be reduced, e.g., by

1. hyperparameter tuning
2. feeding even more training data into the network.
3. changing the way of encoding words, e.g. to one-hot encoding
4. using more or different input data (in my case, I only use “commentsReview”)

[Q.04] “Reasoning of why did you design your network in that way”

To answer this question, the choice of hyperparameters is interesting. In the following, I will refer to the different hyperparameters one after the other.

learning_rate = 0.01
I tried different values that became exponentially smaller and took the best one. Values tested were: 1, 0.1, 0.01, 0.001, 0.0001.
Especially good were 0.01 and 0.001

output_size = 1
Since this is a regression problem, only one neuron is necessary for the output layer, which computes the estimation for the rating.

number_of_epochs = 1
The number of Epochs specifies how often each sample is used as input to the network. In my choice, each sample is used exactly once. Repeated testing showed that this choice for Epoch results in a low loss.

For **[hidden_layer_sizes]** I chose **num_neurons_in_h1 = 33** & **num_neurons_in_h2 = 3**.
The task specifies that our network has two hidden layers. Accordingly, only two values have to be chosen. Repeated testing revealed that the first hidden layer should have about 10-45 neurons and the second hidden layer significantly fewer neurons to ensure a low loss.

weights & biases = random
Biases are initially set to 1.
Weights are initially randomized and have corresponding dimensions since weights must exist as a matrix. In total, there are three weight-bias pairs: **(a)** Between input-layer and hidden-layer-1 (b) Between hidden-layer-1 and hidden-layer-2 **(c)** Between hidden-layer-2 and output-layer.

3. Reasoning about my choices

Activation Function (Sigmoid & ReLU)

As an activation function for the hidden layer, I used the sigmoid function since this is a default from the assignment.

As a linear activation function for the output layer I used ReLU because the function maps negative values to 0 and returns x otherwise. This property is helpful for this assignment because a rating to be estimated should never be negative (the rating scale ranges from 1 to 10).

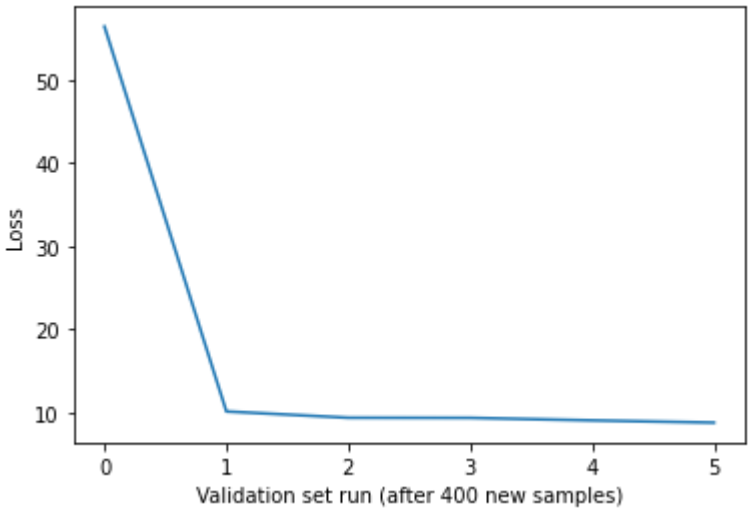
Loss Function (RSS)

Since this task is a regression problem, sum-of-squared error (RSS) is used as a loss function. RSS calculates the loss as the distance between the estimate and the ground truth. Since this task is not a classification problem, other loss functions such as Cross-Entropy-Error cannot be applied.

4. Plot

Loss (plot)

The plot on the right visualizes the loss of the model during the training phase of the neural network. Loss is shown on the y-axis. The x-axis indicates the run of the validation set, which is always used after 400 new samples. The validation set runs through the model 6 times during the training phase (after 0, 400, 800, 1200, 1600, and 2000 samples). Between each run, the model has time to learn (improve its weights and biases).



The graph clearly shows that the loss at the beginning (i=0) is very high (loss=56). This high loss is because the network has not yet had a chance to learn and makes a prediction based on the randomized weights. After 400 training data or after the second run of the validation set, the loss has been minimized to about 10. Subsequent training reduces the loss by only one unit to about 9.

Overall, the loss falls monotonically with an increasing number of training data.