1003243509 Agia 1
March 13, 2019

ROB313 Intro to Learning from Data - Assignment 4

Objectives

In this assignment, we will gain experience using *autograd* to implement neural networks capable of making classifications on the MNIST_small dataset. The neural network architecture used for this task has 784 inputs, with two hidden layers of size M, and a log-softmax output layer with 10 outputs representing the class-conditional log probability for each of the MNIST_small classes. The objective is to evaluate the effects of the model's performance when increasing and decreasing the number of neurons in each hidden layer under various learning rates.

Code Structure and Strategies

The code structure is designed to be modular and easily understandable. Each question has a corresponding code section in the script and in the main block, which is boldly outlined with comments. To run the code for each question, the only modifications needed to be made to the file is to set the booleans corresponding to each question to True in the main block. The results of each question are printed neatly on the terminal when run, and relevant plots will be generated and saved in the current directory. The remaining code is structured as follows:

Q1: **forward_pass(**weights, biases, inputs**)**

This function computes a forward pass of the inputs through the neural network with the provided weights and biases in vectorized form. The output is a log-softmax class-conditional probability for each input data point.

Q2: **negative_log_likelihood(**weights, biases, inputs, labels**)**

This function computes the negative log-likelihood of the inputs from a categorical probability distribution. To do this, a forward pass of the inputs through the neural network with the provided weights and biases is computed. An inner product of the output from the neural network and the input data labels is then computed and summed. The negative of this sum, representing the negative log-likelihood is then returned.

Q3-Q6: **nn_sgd(**data, size, mini_batch, iterations**)**, **gen_plot(**x), **plot_digit_mod**(x)

The neural network models used for questions 3-6 were constructed and trained with the nn_sgd() function based on the provided parameters. The required plots were created using the gen_plot() function, and plot_digit_mod() was used to visualize the weights of the neural network model once trained. The plot_digit_mod() function was also used to visualize the input data points where the class-conditional probability was significantly low.

The code is well commented organized to make the code easy to read and to reduce the time spent searching for critical/error prone sections. Utility methods for frequently used functions (e.g. update_weights()) were written. In addition, the neural network function **nn_sgd()** was written to be quite generalizable, as it was used to create models of various forms for questions 3-6.

Q3 – SGD of a Neural Network with 100 Neurons per Hidden Layer

A neural network with 100 neurons per hidden layer was created and trained using stochastic gradient descent with a mini-batch size of 250. The optimization of categorical likelihood distribution was used to compute the gradients of the neural network with the help of the *autograd* library. Two versions of the neural network were trained with learning rates $\eta_1 = 0.0001$ and $\eta_2 = 0.001$ for 4000 iterations and 1000 iterations respectively. For each model, the full-batch validation neg. log-likelihood and the mini-batch training neg. log-likelihood were computed at each training iteration. The normalized trends for each model, comparing the validation log-likelihood and the training log-likelihood, are shown in *Figures 1-2*.

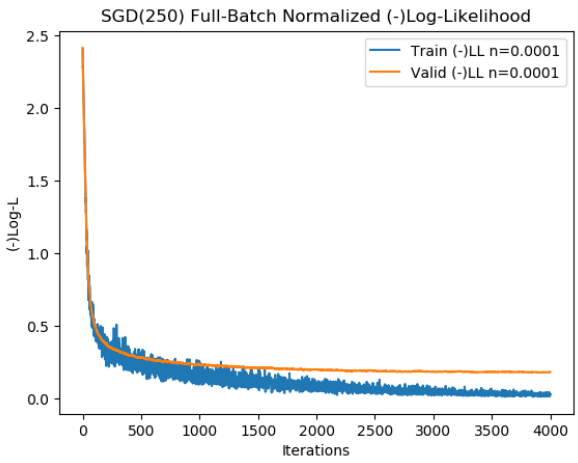

Figure 1 – (-) Log-likelihood for $\eta = 0.0001$

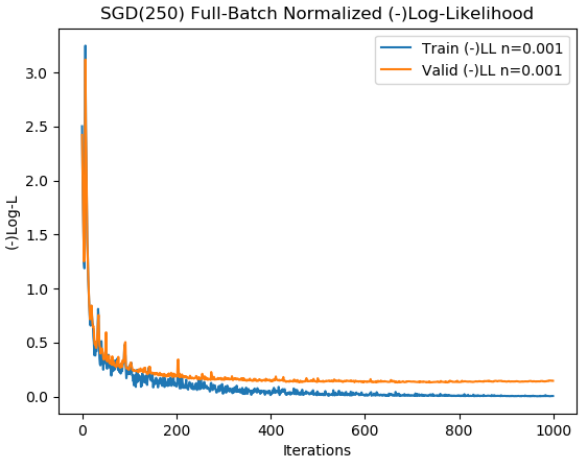

Figure 2 – (-) Log-likelihood for $\eta = 0.001$

From both *Figure 1* and *Figure 2*, we can see that the normalized trends for the validation log-likelihood and the mini-batch training log-likelihood follow the same initial trend. We also see that as the training log-likelihood continues to decrease at large iterations, the validation log-likelihood seems to plateau. In fact, the validation log-likelihood tends to increase for very large training iterations, which implies that the neural network is overfitting to the training set. *Table 1* compares the optimal validation log-likelihood to the final validation and final mini-batch training log-likelihood (i.e. after all training iterations). Note that the mini-batch training log-likelihood was scaled to estimate the log-likelihood over the entire training set.

Table 1 – Final Validation, Final Mini-batch, Optimal Validation Log-likelihood for Neural Network Models

Learning Rate	**Final Train (-) LL**	**Final Valid (-) LL**	**Optimal Valid (-) LL**
0.0001	312.158	184.862	181.110
0.001	52.688	145.659	130.374

For each learning rate, the optimal validation log-likelihood occurs at a training iteration before the total number training iterations. For $\eta_1 = 0.0001$, it occurs at $iteration_{opt,1} = 3192$ (4000 total). And for $\eta_2 = 0.001$, it occurs at $iteration_{opt,2} = 739$ (1000 total). These results are expected, and confirm that at large training iterations the model overfits to the training set, compromising its performance on the validation set.

Q4 – Training Neural Networks using SGD with various Hidden Layer sizes

Three neural network models with hidden layers of size 10, 90 and 180 were trained using SGD with a mini-batch size of 250. Each model was trained for a total of 2000 iterations. Learning rates of $\eta_1 = 0.0001$, $\eta_2 = 0.001$ and $\eta_3 = 0.001$ were used for 10, 90 and 180 neurons per hidden layer, respectively. The training iteration number and associated weights/biases that minimized the validation log-likelihood were tracked for each model. These optimal weights/biases were used to compute the optimal validation log-likelihood, validation accuracy ratio, test log-likelihood, and test accuracy ratio. These values are reported in *Table 2*.

Table 2 – Validation and Test Log-likelihood and Accuracy Ratios for varying Neural Net sizes

Neurons per Hidden Layer	**Optimal Training Iteration**	**Validation Log-likelihood**	**Validation Accuracy Ratio**	**Test Log-likelihood**	**Test Accuracy Ratio**
10	1833	279.434	0.913	293.843	0.909
90	632	178.259	0.953	149.706	0.959
180	658	166.479	0.950	152.558	0.957

Analyzing these results, we see a clear distinction in the performance of the 10-neuron model compared to the 90 and 180 neuron model. Although all three models show convergence, it is evident the larger models are capable of better generalizing to data points that have never been seen. This is shown by the higher accuracy ratios, and lower log-likelihoods of the larger models.

Selecting the learning rate to match the number of neurons per hidden layer was found to be important in the context of convergence and generalization. At high learning rates (i.e. $\eta_{high} = 0.01$), the neural networks became less effective in convergence and generalization with fewer neurons per hidden layer. This is expected, as models with fewer neurons have fewer weights and are more susceptible to being skewed by a "bad batch" of gradients from a randomly generated mini-batch. Neural networks with more weights (i.e. the 180-neuron model) are not as susceptible to this issue. In the case that a high learning rate shows strong convergence, we should expect generalization to be negatively affected since the model is likely to overfit to the training set.

Hence, a learning rate of $\eta_1 = 0.0001$ was found to give good convergence and generalization for the 10 neuron per hidden layer model. And for the 90 and 180 neuron per hidden layer models, a slightly larger learning rate of $\eta_{2,3} = 0.001$ was shown to converge appropriately with impressive generalization. Overall, the architecture that yielded the best results was the 90 neurons per hidden layer model, with a learning rate $\eta_2 = 0.001$. This model scored an impressive 152.558 test log-likelihood corresponding to a test accuracy ratio of 95.9%.

Q5 – Visualizing the Weights of 16 Neurons in Hidden Layer 1

After training a model with 100 neurons per hidden layer for 1000 iterations at a learning rate of $\eta = 0.001$, the weights of 16 randomly selected neurons in the first hidden layer were plotted. The plots are shown in *Figures 3-18* below. The plots generated by transforming 784 weights from each neuron onto a 28x28 grayscale grid, where whiter cells represent larger weights. A

linear combination of the digit pixels and weights from each neuron passed through an activation function is used to predict a class for a given input digit. Hence, the weights of each neuron are responsible for extracting or activating specific features of a number, or perhaps a set of numbers. This can be seen in the images below, as we see formation of the weights into interesting shapes that are useful in extracting features from input digits, enabling the model to make accurate predictions.

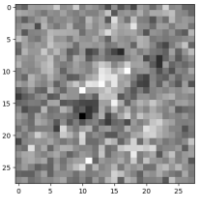

Figure 3 – Neuron 27

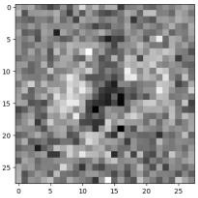

Figure 4 – Neuron 30

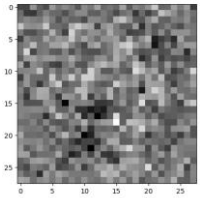

Figure 5 – Neuron 31

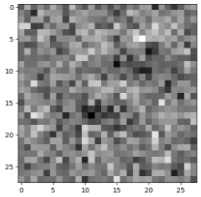

Figure 6 – Neuron 37

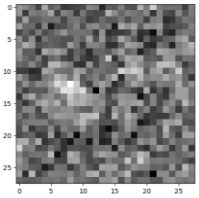

Figure 7 – Neuron 41

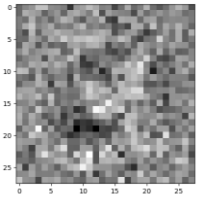

Figure 8 – Neuron 42

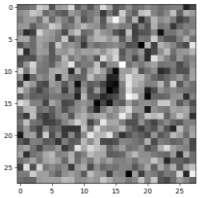

Figure 9 – Neuron 47

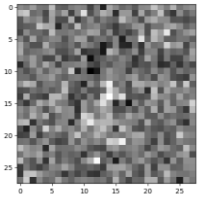

Figure 10 – Neuron 49

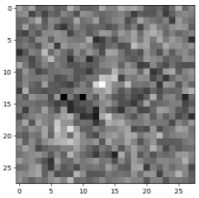

Figure 11 – Neuron 53

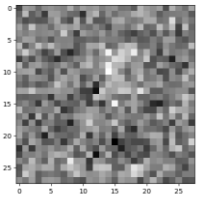

Figure 12 – Neuron 69

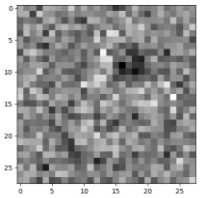

Figure 13 – Neuron 72

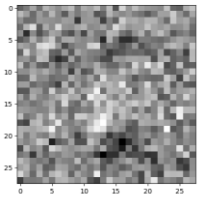

Figure 14 – Neuron 82

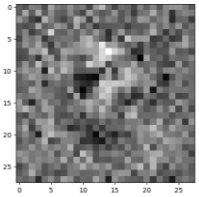

Figure 15 – Neuron 83

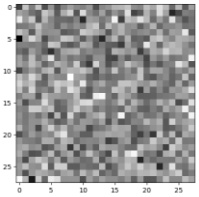

Figure 16 – Neuron 92

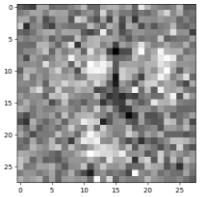

Figure 17 – Neuron 93

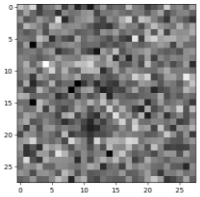

Figure 18 – Neuron 95

Q6 – Visualizing Test Points with Low Class Condition Probabilities

Using the same neural network from Q5, we made predictions on the test set. These predictions were then sorted in increasing order in terms the largest class conditional probability of each data point. *Figures 19-30* show the 12 test inputs with the lowest class conditional probabilities (R=Rank, P=Point).

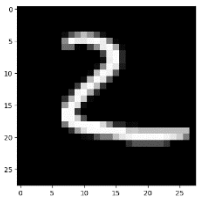

Figure 19 – (R1, P305)

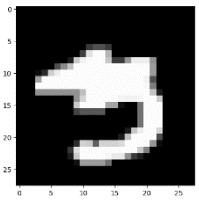

Figure 20 – (R2, P917)

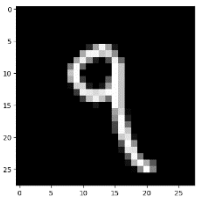

Figure 21 – (R3, P361)

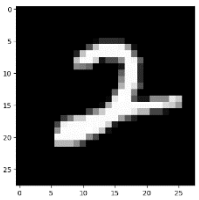

Figure 22 – (R4, P647)

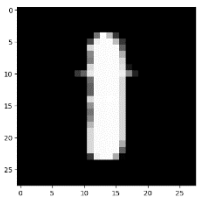

Figure 23 – (R5, P555)

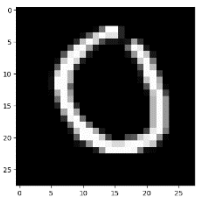

Figure 24 – (R6, P762)

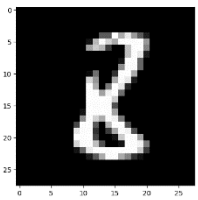

Figure 25 – (R7, P730)

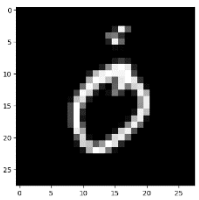

Figure 26 – (R8, P962)

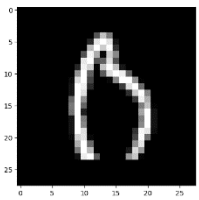

Figure 27 – (R9, P487)

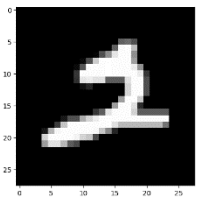

Figure 28 – (R10, P399)

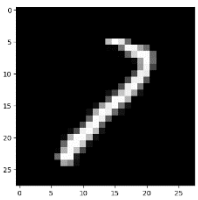

Figure 29 – (R11, P488)

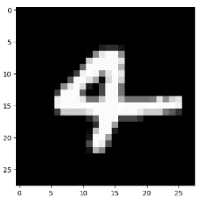

Figure 30 – (R12, P262)

Most of these test set digits clearly have some irregularities. For instance, *Figures 20, 26, 27* are each heavily deformed, making it quite difficult for the model to make a classification with high probability. The neural network also seems to predict low class conditional probability for the number 2 (i.e. *Figures 19, 22, 28)*, which implies that this number (or perhaps these specific examples of the number 2) has similar features to several other numbers. Lastly, the model seems to have issues with classifying numbers with subtle, but key defining features of specific digits. For example, with *Figure 24*, it is quite easy to recognize that this is the number 0. However, a defining characteristic of 0 from the perspective of the neural network may be a fully connected loop, making it unable to make a classification with high probability. Similar arguments can be made for *Figure 23* and *Figure 25*.