EN.605.649 (81): Machine Learning: Assessing Model Performance: Impact of Learning Rates Across Various Models and Datasets

Kayla Ippongi
Johns Hopkins University

Editor:

Abstract

This assignment outlines the steps taken in order to build several different prediction networks - logistic regression, linear network and forward feed neural networks. These networks are trained on 3 classification datasets and 3 regression datasets to compare the differing networks and how well they can predict and generalize different types of data.

1. Introduction

In this assignment, we utilize the 6 datasets that we have been working with in the past - Breast Cancer, Car Evaluation, Congressional Vote, Abalone, Computer Hardware and Forest Fires. Additionally, we leverage the preprocessing toolkit that was built in the first assignment to handle missing data, standardize and cross validate each of the datasets before passing them to our network.

For our classification datasets we use logistic regression to build a function that best fits our data and outputs a binary prediction. This is done in an iterative manner, using cross entropy loss as our loss function, then update the weights of the function using gradient decent.

For our regression datasets, we build a multiple linear regression model, that acts similarly to logistic regression, but utilizes mean squared error as the loss function. Again, in an iterative manner using partial derivatives to update weights.

Finally, we implement a 2 layer forward feed neural network for both classification and regression datasets and another network of similar form except with an auto encoder layer as the first hidden layer.

The datasets that we are working with have numerous features, all which may or may not actually be helpful in our predictions. We hypothesize that our results for the regular forward feed neural networks with a lower learning rate will generalize the best and work well on our test data.

1.1 Logistic Regression - Classification Datasets

1.1.1 Approach & Assumptions

At each step, we take the dot product of the features by the current weights. Since this is classification we use a softmax to get our output layer. We then update our weights using the output. To do this, we calculate error (actual - predictions) then calculate the gradient by retrieving the dot product of the error and features transposed. Finally, we update our weights with our gradient multiplied by the specified learning rate. To build our logistic regression model we utilize cross entropy loss for our classification datasets, where y_i is the actual probability distribution and $y_{\hat{i}}$ is the predicted probability distribution

$$Softmax = e^{zi} / \sum e^{zi}$$

$$Error = actual - predictions$$

$$CrossEntropyLoss = -\sum_{i=1}^{n} y_i Log(y\hat{i})$$

1.1.2 Analyzation

The following graphs represent the outcomes of the 3 trained logistic regression models on the 3 classification datasets - Breast cancer, Car and Congressional Vote. Looking at the graphs in Figure 1, we see the various outcomes of the logistic regression model and the corresponding accuracy per thousand steps. Steps were set to a maximum of 7000, with differing learning rates of 0.5, 0.1, 0.05, and 0.005.

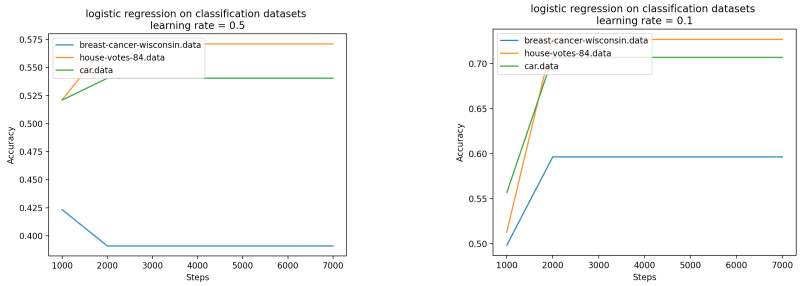
Some interesting notes to point out - most datasets seemed to plateau in accuracy after about 2000 steps. This might be a sign that any training after those steps are not necessary and 5000 steps might have been too much. Looking at the results, the house votes dataset consistently outperformed the car and breast cancer datasets, reaching a highest accuracy of around 70%. We see that the lower learning rate of 0.005 actually has overall lower accuracy than some of the larger learning rates, which is intersting to note as it contradicts our original hypothesis.

1.2 Multiple Linear Regression - Regression Datasets

1.2.1 Approach & Assumptions

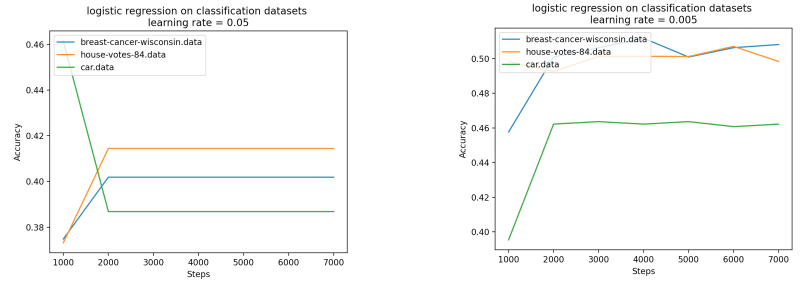
To build our multiple linear regression model we utilize mean squared error loss for our regression datasets. The weights are initially set to all zeros. The function performs similarly to logistic regression, where we take the dot product of our training set with our weights, retrieve the error between the output vs the actual labels, then calculate the partials of the weights and bias to update weights and bias.

We then get our predictions via the linear equation that our weights and bias make up of.



(a) Learning rate 0.5

(b) Learning rate 0.1



(c) Learning rate 0.05

(d) Learning rate 0.005

Figure 1: Logistic Regression on Classification Datasets, 7000 steps

$$\frac{\partial w}{\partial x} = \frac{1}{n} * 2(X^T \cdot error) \tag{1}$$

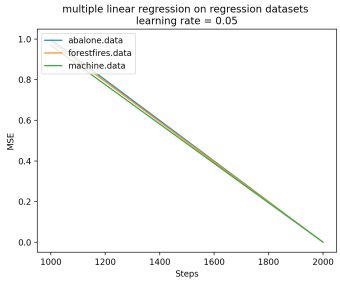
$$yHat = train \cdot weights + bias$$
$$error = yHat - actuallabels$$
$$\frac{\partial w}{\partial x} = \frac{1}{n} * 2(X^T \cdot error)$$
$$MeanSquaredError = \sqrt{\frac{1}{n} * \sum_{n=1}^{n} (actual - predicted)^2}$$

1.2.2 Analyzation

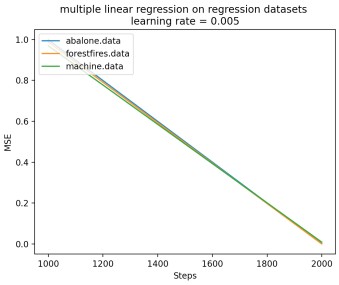
For linear regression, the steps were set to a maximum of 2000 steps, this was chosen as the models converged faster than the logistic regression model which was set at 7000 steps. The differing learning rates were set at 0.1, 0.05 and 0.005. However, despite the differing learning rates, the regression models performed in similar manners, each converging close to 2000 steps.

For future approaches and to improve this experiment, we should also compare the learning rate against differing amount of steps.

(a) Learning rate 0.1



(b) Learning rate 0.05



(c) Learning rate 0.005

Figure 2: Multiple Linear Regression on Regression Datasets, 2000 steps

1.3 Neural Network - 2 Layer

1.3.1 Approach & Assumptions

We train the forward feed 2 layer neural network on 300 steps, with 3 differing learning rates - 0.1, 0.05, and 0.005 for both the classification and regression datasets. Like subsequent runs, each dataset is preprocessed to handle missing values via imputations and categorical data, 5 fold cross validated, and standardized.

The input layer is the size of the input dataset, while the size of the hidden layer was simply set at 10. Initial weights and biases are randomized. However, to get a more optimal result, the size of the hidden layer should be played around with more and tuned in order to get higher precision - this is something that I would like to incorporate for the future. The size of the output layer was set to the number of unique features in the feature we are attempting to predict.

In forward propagation we simply make a forward pass through the network using dot product. See the diagram in Figure 3 to see how the weights are forward propagated through the network. Additionally, we use the rectified linear unit activation function at each output layer.

In backpropagation we use partial derivatives to calculate the rate of change for each weight (w1 and w2) in respect to the overall cost of the network. Thus the rate to change each weight are as follows follow the following partial derivatives, where m is the size of the input. These partials are then saved to update the weights accordingly.

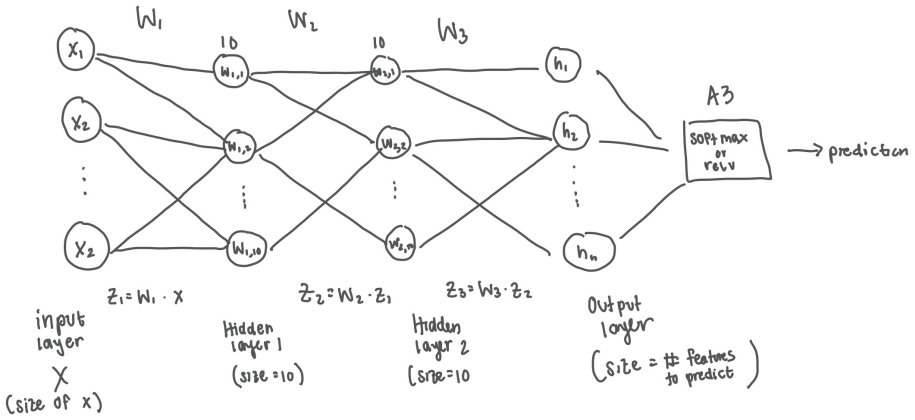


Figure 3: Neural Network Outline

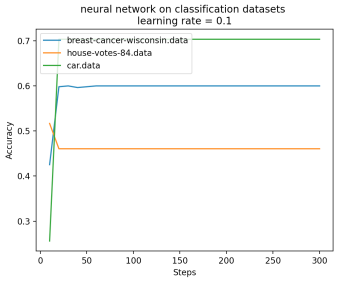
$$
\begin{aligned}
\frac{\partial C}{\partial z3} &= A3 - actualLabels \\
\frac{\partial C}{\partial W3} &= \frac{1}{m} * \frac{\partial C}{\partial z3} \cdot A2^T \\
\frac{\partial C}{\partial W2} &= \frac{1}{m} * \frac{\partial C}{\partial z2} \cdot A1^T \\
\frac{\partial C}{\partial W1} &= \frac{1}{m} * \frac{\partial C}{\partial z1} \cdot X^T \\
\frac{\partial C}{\partial z2} &= (W3^T \cdot \frac{\partial C}{\partial z3}) * (Z3 > 0) \\
\frac{\partial C}{\partial z1} &= (W2^T \cdot \frac{\partial C}{\partial z2}) * (Z2 > 0)
\end{aligned}
$$

1.3.2 Analyzation

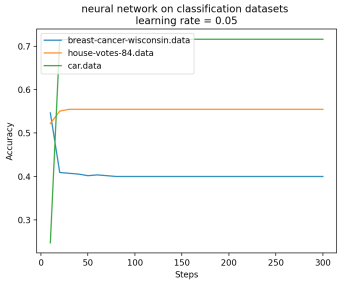
In Figure 4 we have the accuracies for the neural network for our 3 classification datasets and regression datasets in Figure 4. Both were run on 300 steps, where in each step we forward propagate the network, calculate error, then use backpropgation to update the weights. Learning rates of 0.1, 0.005 and 0.0005 were utilized.

The car dataset performed well with higher learning rates of 0.1 and 0.05 with accuracy plateauing around 0.7, however, performed poorly when learning rate was reduced to 0.005. On the other hand, the house votes and breast cancer datasets performed better with lower learning rates, hitting accuracy's of 65%

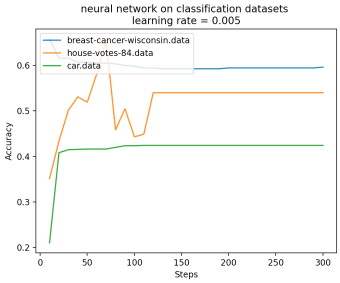
With the regression datasets, we see how the lower learning rates cause our function to converge slowly, for example the Abalone dataset had a higher accuracy of 60% with a 0.0005 learning rate compared to a 50% accuracy with the 0.005 learning rate. Since learning rates affect how much our weights and bias terms are being updated, slower convergence with smaller rates intuitively make sense.



(a) Learning rate 0.1

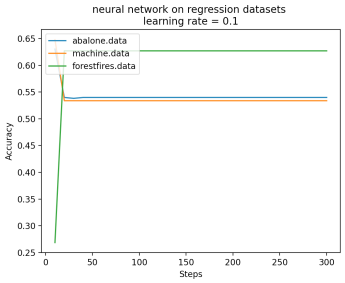


(b) Learning rate 0.05

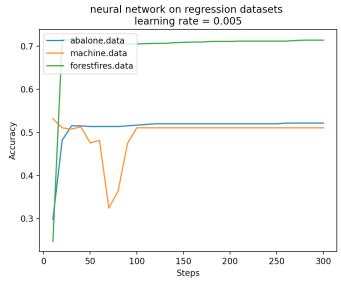


(c) Learning rate 0.005

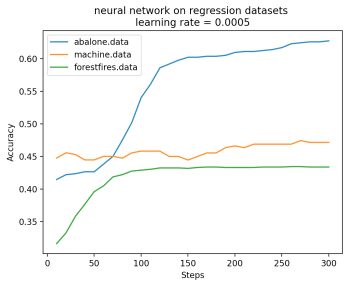
Figure 4: Neural Network on Classification Datasets, 300 steps



(a) Learning rate 0.1



(b) Learning rate 0.005



(c) Learning rate 0.0005

Figure 5: Neural Network on Regression Datasets, 300 steps

1.4 Neural Network - 2 Layer with Autoencoder layer

1.4.1 Approach & Assumptions

To build a neural network with an auto encoder layer, I expanded our original 2 layer neural network to a total of 4 hidden layers. Our first layer, suddenly becomes 2 layers - an encoding layer and a decoding layer. The first layer would encode a compressed representation of its input (aka dataset X), and our output layer "decodes" to the original size of dataset X then feeds back into our regular neural network hidden layer 2.

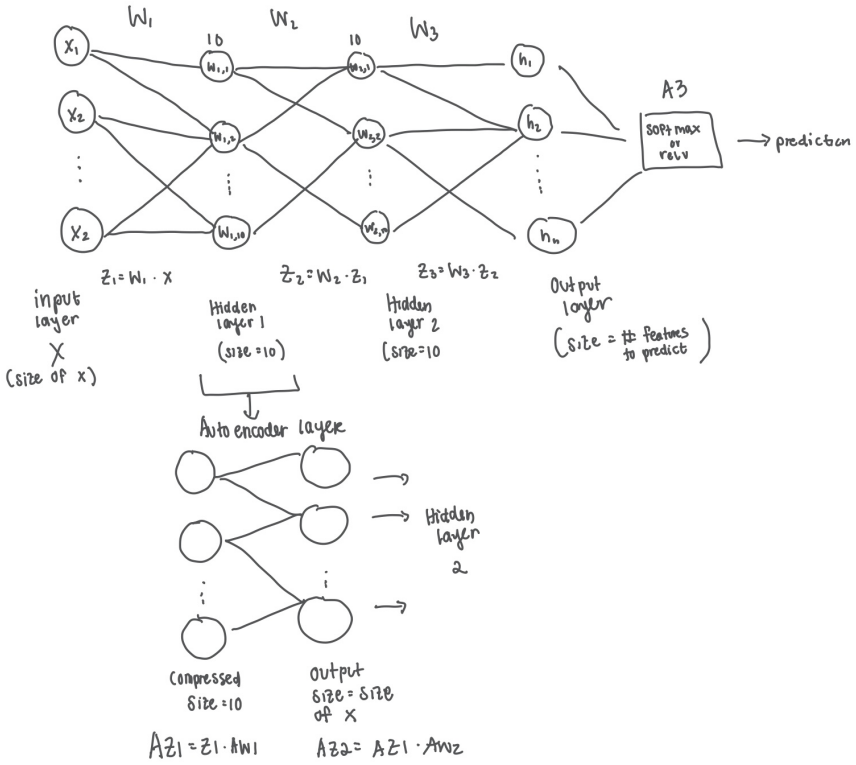


Figure 6: Neural Network with Autoencoder Outline

1.4.2 Analyzation

I ran out of time to fully implement the experiments bug free - but the general code for the auto encoder neural network lives in Auotencoder.py.

2. Conclusion

In this assignment we have built 4 different types of models, logistic regression, linear regression, 2 layer neural network and a neural network with an auto encoder layer. Our original hypothesis, that our regular 2 layer network with lower learning rates would perform better is not always true. We saw from the results that a lower learning rate does not always improve the accuracy of the model, and in fact can sometimes hurt the performance. There are some datasets that performed better under lower learning rates, such as the abalone dataset, while there are other sets like the forest fires dataset that did better with higher learning rates.

When building a neural net it's important to consider the input dataset its parameters in order to build an optimal network. There's also a lot of variables to consider that can affect the training of the model - activation functions, number of steps, and learning rates etc. We don't want to use to many epochs to overfit or a miniscule learning rate that slows down the convergence. Thus to train a network we can use forward propagation to get the output, then use the output and the desired output into a cost function and use partial derivatives of the cost func to every weight in the network to figure out what rate to change each weight in order to minimize the cost.

There are still a lot of improvements to be made in these experiments that I conducted, but there are several outcomes and conclusions we can make based on the results. We see that the learning rate effects how fast our model converges or not, and can help with either improving or reducing accuracy based on whether the network gets overfit/underfit.

For future work, I would also like to focus on how changing and tuning the number of hidden nodes within the hidden layer effects the overall loss and accuracy for the network. To create an optimal network involves optimizing the number of hidden nodes, which requires more experiments.