MULTI-LAYER PERCEPTRON FORGETTING (LAB 2)

IRIS ROBEDEAUX

1. Introduction

The concepts tackled in Lab 2 represent the forgetting within Multi-Layer Perceptrons (MLPs). As mentioned in the lab assignment, when MLPs are trained on different tasks, they forget the tasks trained earlier. In this lab, I explore different tunings for this model that impact its performance. The base code for this lab is sourced from a GeeksForGeeks article [1] which lays out the basic code for making a MLP in TensorFlow for Python. Several of the graphs were produced after-the-fact in R from the exported Task Matrix for each run. This was done to streamline creating the models since the graphs might compile incorrectly, and the models can take a great deal of time to train.

2. Metrics for Forgetting

Two performance metrics are used to compare the forgetting of MLPs. Average Accuracy (ACC) is given by

$$\text{ACC} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i},$$

where T is the total number of tasks trained on, and R is the Task Matrix. Backward Transfer (BWT) is given by

$$\text{BWT} = \frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - \overline{b}_i,$$

where $\overline{b}_i$ is the vector of test accuracies for each task at random initialization. Both of these metrics, as defined in [2], have different scales. ACC is measured on a 0 to 1 scale, with higher values indicating that the model is performing well on previous tasks from the one trained. BWT measures forgetting, where negative values indicate forgetting and positive values indicate learning from previous tasks being maintained. Within my tests, I only experienced negative BWT values, which makes sense given we only measure forgetting instead of accounting for it with advanced learning methods.

3. The Problem

To begin tackling this problem, I first created a MLP that was easily customizable. I created a simple mutable command that would allow me to change the optimizer, number of layers, etc as parameters to a function, instead of having to rewrite the code for each experiment.

Date: October 2025.

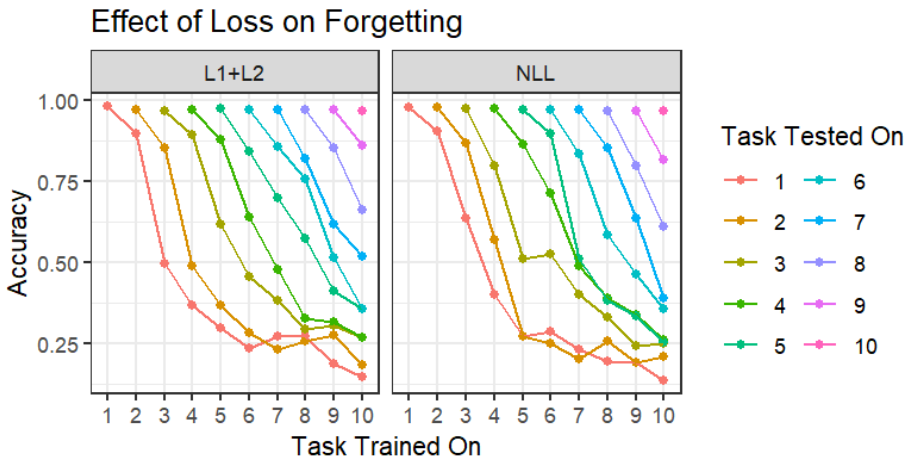

FIGURE 1. Forgetting of MLPs based on loss function.

	Hidden Layers		
	2	3	4
ACC	0.463	0.458	0.449
BWT	-0.568	-0.57	-0.582

TABLE 1. The ACC and BWT for MLPs with 2, 3, and 4 hidden layers

The data is the MNIST dataset, with each task being a permutation of the data (the pixels being shuffled randomly). This creates new patterns for the model to learn without needing a completely new data set.

3.1. **Loss Function.** The first parameters I focused on was the loss function. In general, the accuracy for previous tasks always decreases after being trained on a new task. L1 and L2 loss produced the worst results, with L1 having ACC 0.425 and BWT -0.609. L2 is worse, with an ACC of 0.399 and BWT of -0.639. L1+L2 loss improves on both with an ACC of 0.46 and a BWT of -0.57. Negative Log Likelihood (NLL) Loss has a similar performance, with an ACC of 0.463 and a BWT of -0.568. Comparisons between NLL and L1+L2 loss are given in Figure 1. The performance between the two is very close. Though the BWT for NLL is a bit less than for L1+L2, the gain is accuracy is more valuable, so while the two are comparable, I chose to continue with NLL Loss for the duration of the lab.

3.2. **Layers.** Next, I looked into how layers impact forgetting of a MLP model. Keeping the loss as MLP, I tested with 2, 3, and 4 hidden layers. The ACC and BWT for each is given in Table 1. The ACC and BWT get worse as the layers increase, creating an inverse relationship between the number of layers and the accuracy of a MLP on previous tasks. For this reason, I used 2 layers for the continuation of this lab.

3.3. **Dropout.** Dropout was tested next. I applied a 0.5 drop rate, which achieved better performance. Figure 2 shows the difference between the models on performance throughout the tasks. While forgetting is inevitable, there is an improvement. Especially on the later

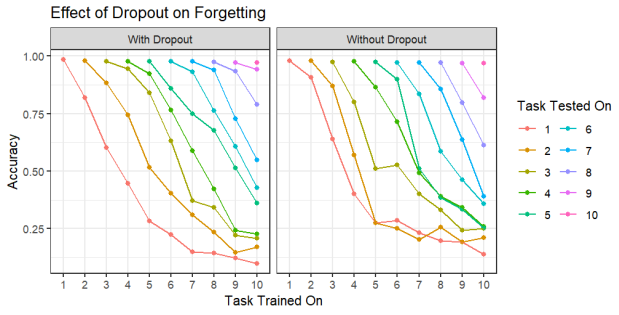

FIGURE 2. A graph comparing the MLP forgetting between a model without dropout and a model with dropout.

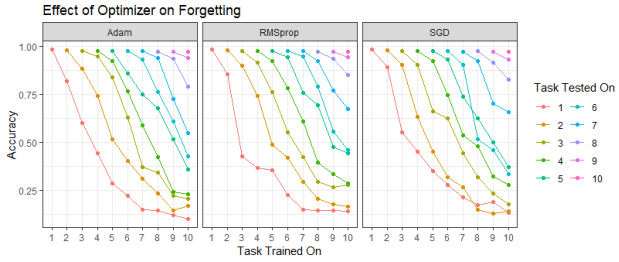

FIGURE 3. The effect of different optimizers (Adam, SGD, and RMSProp) on MLP forgetting.

tasks, the graph shows that the model with dropout has a slower rate of forgetting that the one without. The final ACC with dropout was 0.474 with a BWT of -0.557. This shows quite the improvement on the model without dropout, so I continued to use it.

3.4. **Optimizers.** The final parameters I tried were optimizers. The three tested were Adam, Stochastic Gradient Descent (SGD), and Root Mean Square Propagation (RMSProp). Adam was the optimizer used for the rest of this lab, ending with a previously given ACC of 0.474 and a BWT of -0.557. SGD gave an ACC of 0.482 and a BWT of -0.548. Finally, RMSProp gave an ACC of 0.521 and a BWT of -0.504. Out of these optimizers, RMS outperformed the others, giving the best ACC and BWT seen so far. As shown in Figure 3, RMSProp maintains the most information about previous tasks. The slopes for each of the tasks is much smaller, keeping what's been learned before. Though the top of the graph is similar to Adam, RMSProp keeps the most of Task 1 especially, and keeps more of Tasks 2 and 3 as training continues. These tails are what makes RMSProp outperform the others.

4. CONCLUSION

After many tests. The MLP that forgets the least out of these parameters has 2 hidden layers, NLL loss, 0.5 dropout, and uses the RMSProp optimizer. The overarching theme is that MLPs are generally not flexible to new tasks. Though they can train on new tasks and perform well on them, they cannot maintain previously learned information for very long.

5. GITHUB LINK

The GitHub repository for this lab is found here:

- https://github.com/iris-robedeaux/CS-599-Labs/tree/master/Assignment3
It contains all Python and R code used in this assignment.

References

[1] GeeksforGeeks. Multilayer perceptron learning in tensorflow, Nov 2021.
[2] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning, 2022.

Email address: ir496@nau.edu