IST 597: Assignment #00100

Laxmaan Balaji
Department of Computer Science
Penn State University
State College, PA 16801
lpb5347@psu.edu
PSU ID: 917326388

1 Measuring Catastrophic Forgetting in Multi-layer perceptron

An MLP is a basic feed forward neural network. In this problem, we address the issue of MLPs forgetting previous tasks they were trained on when supplied with new tasks to be trained on. We analyse the effect of various hyperparameters when testing on multiple test sets.

1.1 Specification

The data is permuted MNIST, generated by applying 10 permutations on each image in MNIST to get 10 complete datasets.

The MLP is a **3** layer network with **relu** activations in all layers but the output layer, where it is **softmax**. **Categorical crossentropy** is the loss function used.

Early stopping was implemented to halt training when the validation accuracy didn't change by more than 10^{-5}.

1.2 Effect of various loss function on forgetting

The above expt is repeated with 4 loss functions. The loss functions are:

- **Categorical Crossentropy**
- **Mean Absolute Error**
- **Mean Squared Error**
- **MSE+MAE**

The metrics obtained are:

Metric	NLL	MSE	MAE	MSE+MAE
ACC	**0.468**	0.100	0.097	0.097
BWT	-0.561	**0.001**	-0.002	-0.001
TBWT	**-0.565**	-0.878	-0.881	-0.880
CBWT(t=1)	-0.73	**0.02**	-0.01	-0.01

The problem was broadcast as a regression problem where the output label was predicted as a number for **L1,L2,L1+L2** losses. However, that approach did not train the model well. While NLL has a forgetting problem, I was unable to get the model to train beyond 10% test accuracy for the other loss functions.

While the metrics might look impressive at first glance, they are noisy estimates due to the underlying underfitting of the model for all losses but NLL. This is very obvious from Fig. 1.

Preprint. Under review.

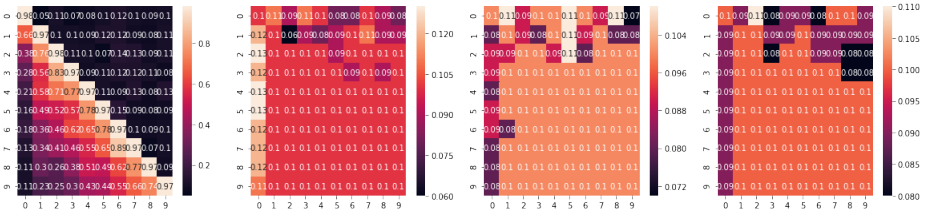

Figure 1: Rt matrices for NLL, MSE, MAE, MSE+MAE

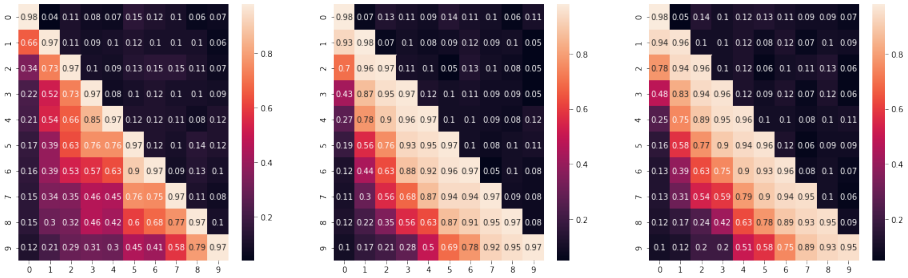

Figure 2: Rt matrices for dropout rates 0, 0.33, 0.5 respectively

Conclusion: As this is a classification problem NLL works best. Effect of forgetting due to other loss functions is too noisy to be measured.

1.3 Effect of dropout on forgetting

The experiment is repeated with dropout rates 0,0.33,0.5 The metrics obtained are:

Metric	Dropout= 0	Dropout= 0.33	Dropout= 0.5
ACC	0.445	**0.556**	0.523
BWT	-0.587	**-0.462**	-0.485
TBWT	-0.589	**-0.464**	-0.500
CBWT(t=1)	-0.73	-0.65	**-0.63**

Dropout definitely seems to help. Sparser the weights are, less the forgetting. However, in general, dropout = 0.33 seems to help more than dropout = 0.5 which in turn helps more than dropout = 0. I believe this is because addition of dropout =0.5 slows down the training. So if we were to train for more epochs with dropout 0.5, we should see it perform better than dropout 0.33.

As seen from CBWT and the Rt heatmap in Fig. 2, the effect of forgetting is slower when dropout is 0.5 .

Conclusion: Adding dropout improves the sparsity of weights which reduces forgetting.

1.4 Effect of depth on forgetting

The above expt is repeated with MLPs of depth 2,3,4 layers respectively.

The metrics obtained are:

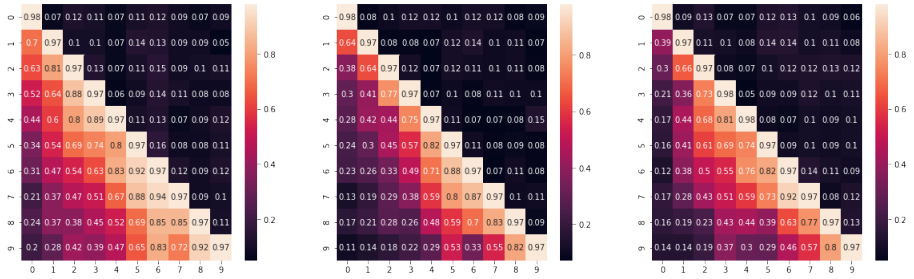

Figure 3: Rt matrices for depths 2,3,4 respectively

Metric	Depth 2	Depth 3	Depth 4
ACC	**0.585**	0.414	0.421
BWT	**-0.430**	-0.621	-0.615
TBWT	**-0.435**	-0.625	-0.618
CBWT(t=1)	**-0.58**	-0.71	-0.78

As seen in Fig. 3, we see the effect of forgetting is slowest in the MLP of depth 2 and rapid in the MLP of depth 4. One way to reason out the same would be that, with more expressive power, the deeper MLPs tend to overfit the training data and as a result, when they are exposed to a new task, the alteration of weights leads to a stronger decrease in test accuracy on previous tasks.

However it is worthwhile to note that while all metrics favor the depth 2 MLP, it is the depth 4 MLP that comes in second, very slightly better than the depth 3 MLP. Why it follows that trend remains to be seen.

Conclusion: The simplest model with depth 2 saw the least forgetting. However after depth=3, increasing the depth reduced forgetting.

1.5 Effect of Optimizers on forgetting

The above expt is repeated with Adam, SGD with **Nesterov momentum = 0.5** and RMSprop. Each of their learning rates was set to 0.001.

The metrics obtained are:

Metric	Adam	SGD	RMSprop
ACC	0.411	**0.686**	0.456
BWT	-0.624	**-0.133**	-0.577
TBWT	-0.626	**-0.306**	-0.576
CBWT(t=1)	-0.63	**-0.306**	-0.51

As seen in Fig. 4, we see that forgetting is least for SGD. RMSprop comes in at second and Adam is most forgetful. One thing to note however, is that SGD is converging much slower when compared to RMSprop and Adam.

While SGD is hardly forgetting, the fact that it trained up to only $75 - 80\%$ test accuracy might be a factor. To get more concrete results, the experiment can be repeated by training SGD for more epochs.

An interesting observation when training was that **SGD achieved only 75% validation acuracy on Task 1, but quickly increased to a maximum of 80% accuracy on subsequent tasks in *lesser training epochs***. I believe that this boost in performance on unseen data is due the lack of forgetting seen in SGD.

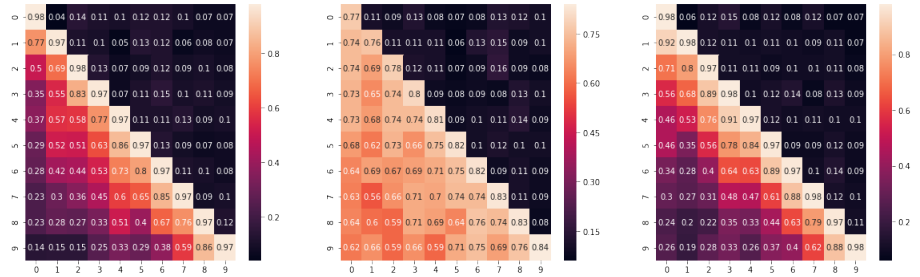

Figure 4: Rt matrices for Adam, SGD, RMSprop respectively

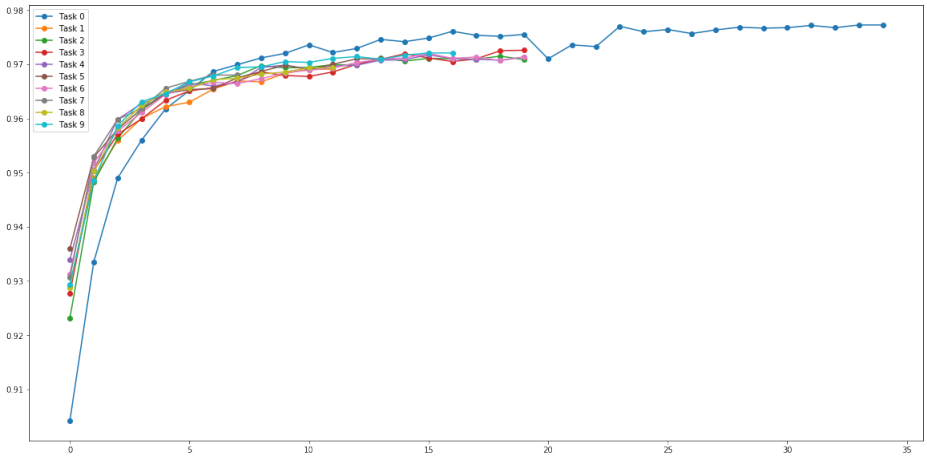

Figure 5: validation accuracies for a model

Conclusion: SGD trains the slowest but however the decrease in test performance across tasks is much smaller. It is the optimizer that provides least forgetting.

1.6 Validation accuracies

A model of depth 3 is trained and the validation accuracy at the end of each epoch for every task is stored. Plotting them yields the graph in Fig. 5. We see that after task 1, there is a small increase in the initial validation accuracy for all subsequent tasks. This is due to the effect of the previous task in slightly boosting initial performance.

1.7 Bonus

The TBWT and CBWT(t=1) metrics have been calculated and reported in all the above experiments.