IST 597: Assignment 00100

Jooyoung Lee
Information Sciences and Technology
Pennsylvania University
920172979

1 Problem1. Measuring catastrophic forgetting

Deep learning approaches have lead to many breakthrough in various domain, yet they suffer from credit assignment and forgetting problem.Deep learning systems have become more capable over-time, however standard multi-layer perceptron(MLP) and traditional training approaches cannot handle incrementally learning new tasks or categories without catastrophically forgetting previously learned training data. We call these problem catastrophic forgetting in neural networks. Fixing this problem is critical so that agents learn and improve incrementally when deployed in real-life setting.In simple term when you have trained model on Task A , and using same weights for learning a new Task B. Then your model forgets/loss learned information about Task A. This means it catastrophically forgot previous information.In this assignment you will be measuring and analysing catastrophic forgetting in neural networks. How will you do this?.Let's get into detail.

• You will be using permuted mnist for this assignment.Script for same is provided in the github repo

• Train a network for more than 50 epochs to reach desirable performance/accuracy.

• Now test on Task A, use the same network and then train on Task B for 20 epochs , now Test on Task A and B.Continue this for N Task(N = 10)

• Total Number of Epochs [50+20+20+20+20+20+20+20+20+20]= 230

• Create Resulting Task Matrix(R) as given in [1] and report ACC and BWT(Formulation given below)

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

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

We will be using standard MLP with various depth in this assignment You will be creating 3 models of depth 2,3,4 each with 256 hidden units.

1.1 Loss Function

We use a loss function to determine how far the predicted values deviate from the actual values in the training data. If the number is close to 0, it means the prediction is accurate. I implemented four different loss functions to compare their performances.

First, **Mean Absolute Error(MAE)**, which is also known as L1 loss function, measures the average magnitude of the errors in a set of predictions, without considering their direction.

In order to examine the effectiveness of MAE on forgetting, I retrieved the accuracy and BWT of MLP model with two hidden layers by setting minibatch size as 64, learning rate as 0.0001 and using GradientDescentOptimizer.

task	accuracy	BWT
task1	10.59	0
task2	10.31	-0.00284
task3	12.67	0.04438
task4	11.06	-0.0041
task5	10.14	-0.04067
task6	12.66	0.08544
task7	10.72	-0.03128
task8	10.96	-0.01451
task9	13.19	0.16386
task10	11.81	0.04036

Since MAE is for the linear regression task, performance for permuted mnist dataset is achieved poorly : the accuracy does not go above 13 percent and BWT is relatively low due to insignificant change in accuracy.

Second, **Mean Square Error(MSE)**, which is also known as L2 loss function, is one of basic loss functions that is easy to understand. In order to calculate it, you take the distances between target variable and predicted values, square it, and average it out across the whole dataset.

In order to examine the effectiveness of MSE on forgetting, I retrieved the accuracy and BWT of MLP model with 2 hidden layers by setting minibatch size as 64, learning rate as 0.00005 and using adamOptimizer. The reason I switched the original setting, which is learning rate = 0.0001 with GradientDescentOptimizer, is that the accuracy did not change and loss value was not changing as well with the prior hyper-parameters.

task	accuracy	BWT
task1	15	0
task2	23.23	0.08235
task3	27.55	0.16884
task4	26.32	0.13187
task5	22.26	-0.03074
task6	22.02	-0.0428
task7	20.91	-0.10893
task8	19.16	-0.23168
task9	19.81	-0.17954
task10	16.76	-0.45401

The result shows that overall accuracy using MSE loss function is better than MAE loss function. Also, the changes in BWT is more vivid as well. We can clearly see that the value of BWT is dropping to negative as the task continues, which indicates that the model is forgetting previous trained information. However, I am convinced its performance is still not robust and the changes in BWT is still weak as well.

Third, **Cross-Entropy loss**, which is also known as negative log likelihood function, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-Entropy generally is utilized in classification class.

In order to examine the effectiveness of Cross-Entropy on forgetting, I retrieved the accuracy and BWT of MLP model with 2 hidden layers by setting minibatch size as 64, learning rate as 0.0001 and using GradientDescentOptimizer.

task	accuracy	BWT
task1	89.62	0
task2	77.27	-0.12575
task3	72.28	-0.22518
task4	63.88	-0.47563
task5	54.67	-0.83103
task6	46.41	-1.24686
task7	43.31	-1.44746
task8	36.58	-1.90771
task9	29.21	-2.48571
task10	28.54	-2.60172

The result shows that overall accuracy using cross-Entropy loss is clearly higher than both of two other functions. The initial accuracy after running 50 epochs for task1 is 89.62 percent, which seems like a robust performance, but after going through different tasks , the accuracy diminishes. Moreover, if you take a look at BWT, the values are dropping to -2. Therefore, it is concluded that the model is catastrophically forgetting the prior information.

Last but not least, we have to analyze the catastrophic forgetting by **combining L1 and L2 loss function**.

In order to do so, I retrieved the accuracy and BWT of MLP model with 2 hidden layers by setting minibatch size as 64, learning rate as 0.00005 and using adamOptimizer.

task	accuracy	BWT
task1	15.56	0
task2	22.93	0.0736
task3	25.92	0.1334
task4	24.99	0.10569
task5	23.66	0.05251
task6	20.64	-0.09855
task7	20.21	-0.1242
task8	18.33	-0.256
task9	18.93	-0.20832
task10	17.35	-0.35018

The result shows that overall accuracy using both L1 and L2 loss function is not that high compared to Cross entropy loss function. Yet again, the values of BWT are falling slowly to the negative value.

1.2 Dropout

Dropout is an incredibly popular method to combat overfitting in neural networks. The idea behind Dropout is to approximate an exponential number of models to combine them and predict the output. It can be applied by randomly disconnecting some neurons of the network.

In order to see the effect of dropout on forgetting, I implemented a dropout with p=0.5 in $compute_output$ function. Surprisingly, the effect was not that noticeable : accuracy and BWT has a slight change (± 0.003).

1.3 Effect of depth on forgetting

Multilayer perceptron can have a diffeent number of hidden layers other than input and output layer and its accuracy and BWT are highly susceptible to these numbers.

In order to investigate the effect of depth on forgetting, I retrieved the accuracy and BWT of MLP model with multiple hidden layers by setting minibatch size as 64, learning rate as 0.0001 and using GradientDescentOptimizer.

	MLP with 2 hidden layers	MLP with 3 hidden layers	MLP with 4 hidden layers
task1	89.62	92.44	37.04
task2	77.27	66.26	10.05
task3	72.28	49.21	9.89
task4	63.88	29.36	9.05
task5	54.67	21.59	10.25
task6	46.41	19.75	10.04
task7	43.31	16.62	10.14
task8	36.58	16.38	10.15
task9	29.21	14.04	10.13
task10	28.54	15.26	10.18

The chart above includes the test accuracy of each task and it is calculated by MLP model with 2, 3, and 4 hidden layers respectively. The MLP model with 3 hidden layers achieves the best performance on the initial task, but the model with 2 hidden layers surpasses it after task2. I believe the MLP model with 4 hidden layers didn't accomplish a robust performance because of its complex capacity and the overfitting problem. Still, it is obvious that overall accuracy of three models have constantly dropped after 10 different tasks, which means that they are forgetting.

	MLP with 2 hidden layers	MLP with 3 hidden layers	MLP with 4 hidden layers
task1	0	0	0
task2	-0.12575	-0.26179	-0.2699
task3	-0.22518	-0.6028	-0.27303
task4	-0.47563	-1.19844	-0.27268
task5	-0.83103	-1.50913	-0.25864
task6	-1.24686	-1.60152	-0.28161
task7	-1.44746	-1.7889	-0.26049
task8	-1.90771	-1.80602	-0.25995
task9	-2.48571	-1.99333	-0.26152
task10	-2.60172	-1.88363	-0.25744

The chart above illustrates the BWT of each task and it is calculated by MLP model with 2, 3, and 4 hidden layers respectively. All of the numbers have negativity which also indicates that three MLP models cannot handle incrementally learning new tasks regardless of their depth.

1.4 Optimizer

Optimizer is one of the strategies for initializing parameters. It helps the model to reduce the loss and optimize the weight and bias values. Among various optimization functions, I implemented GradientDescentOptimizer, Adamoptimizer and RMSPropoptimizer to test their effect on catastrophic forgetting.

task	accuracy	BWT
task1	89.62	0
task2	77.27	-0.12575
task3	72.28	-0.22518
task4	63.88	-0.47563
task5	54.67	-0.83103
task6	46.41	-1.24686
task7	43.31	-1.44746
task8	36.58	-1.90771
task9	29.21	-2.48571
task10	28.54	-2.60172

In terms of GradientDescentOptimizer, the accuracy and BWT gradually reduced from 89.62 percent and 0. Even if it achieved a robust performance on task1, it is evident that the model forgot the prior training information since accuracy does not increase and BWT value is dropping to the negative values.

task	accuracy	BWT
task1	89.74	0
task2	77.17	-0.12575
task3	72.2	-0.22518
task4	63.38	-0.47563
task5	54.97	-0.83103
task6	46.65	-1.24686
task7	43.31	-1.44746
task8	36.58	-1.90771
task9	29.51	-2.48571
task10	28.22	-2.60172

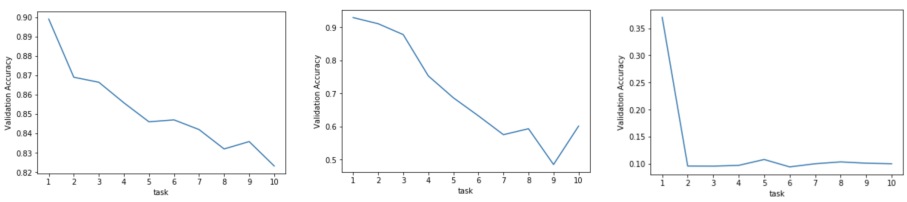
In terms of AdamOptimizer, the result is similar to that of GradientDescentOptimizer: catastrophic forgetting phenomenon is also witnessed regarding the overall accuracy and BWT.

task	accuracy	BWT
task1	92.84	0
task2	84.6	-0.0824
task3	80.05	-0.17335
task4	74.67	-0.33482
task5	67.36	-0.62744
task6	60.75	-0.9575
task7	52.34	-1.46257
task8	53.18	-1.40336
task9	49.73	-1.6795
task10	44.59	-2.14263

In terms of RMSPropoptimizer, the outcome has the best performance for 1o tasks compared to other optimizers. The model kicked off with 92.84 percent accuracy on the first task and ended up with 44.59 percent accuracy on the last task which is relatively not that bad. The overall drop in BWT is also proportionally smaller than other two optimizers. Nevertheless, the forgetting phenomenon is notable.

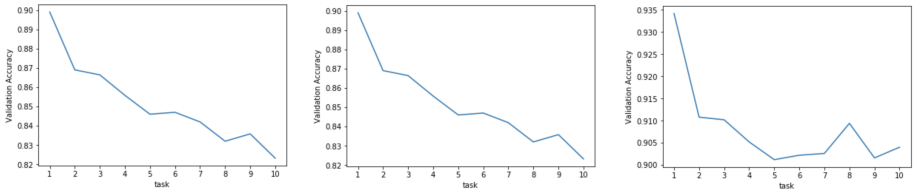
1.5 Plot your validation results

Plotting a validation outcome visually helps you understand the model's performance much easier than manually comparing the result one by one. I retrieved the accuracy BWT of validation results for decrease in model prediction after finishing the training process on all given 10 tasks. To be more specific, I implemented the model with a different number of hidden layers by setting minibatch size as 64, learning rate as 0.0001 and using GradientDescentOptimizer. As you can tell from the plotted graphs, a two hidden layered MLP model has the best validation accuracy which ranges from 90 percent to 80 percent. One of the interesting common traits witnessed from the graph is that accuracy have a tendency to abate.



[Fig1] validation output graphs using MLP model with 2 hidden layers, 3 hidden layers, and 4 hidden layers from left to right

I also wanted to study the model with minibatch size as 64, learning rate as 0.0001 and using three different optimizers. Th plotted graphs illustrate that a two hidden layered MLP model with RMSPropoptimizer has the best validation accuracy which ranges from 93.5 percent to 90 percent.



[Fig2] validation output graphs using a two hidden layered MLP model with GradientDescentOptimizer, AdamOptimizer, and RMSPropoptimizer from left to right