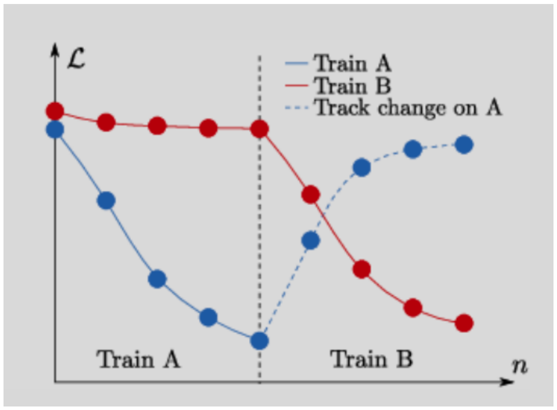
Report

1. When doing incremental learning, we usually have the following case: We train a model on one task, then train on another task using the weights from the previous task, and observe the “catastrophic forgetting” phenomenon shown in the below graph (from the lecture slides).

   Such phenomenon was also observed in when we trained our CNN model on classifying the first digits [0,1,2,3,4] (task 1) and then trained that model on classifying the other digits [5,6,7,8,9] (task 2) using the same weights (for corresponding digits) in task 1. After we trained the model on task 1, the test accuracy was very high, around 99.71% but it got to 0% after we trained the model on task 2 (as expected, the test accuracy on task 2 was very high after we trained the model on task 2, around 99.57%). The “seen classes” accuracy, meaning the average accuracy over the both tasks was 49.78%.

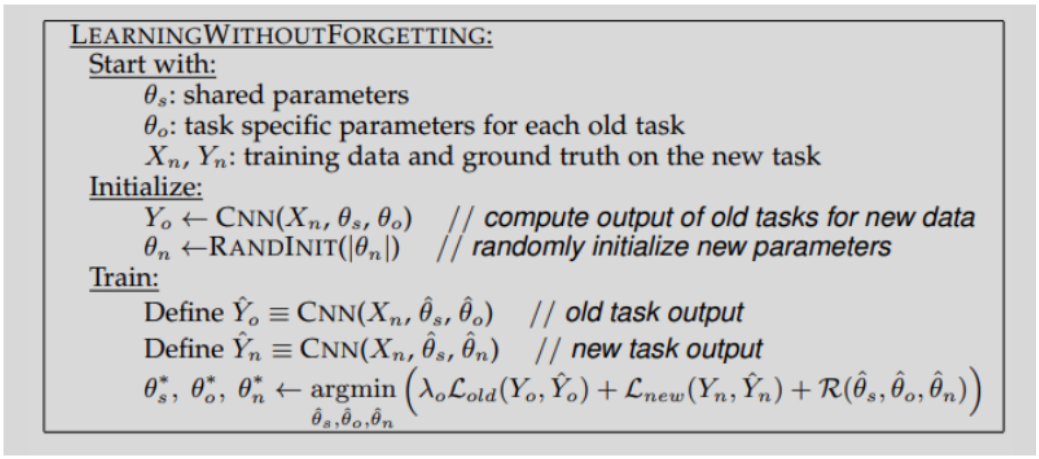
2. When we trained our model on task 1, and then used the pretrained backbone model for task 2 (with adding a new classifier layer), we got a better overall performance. The test accuracy on task 1 was high (as in problem 1), around 99.68%. The test accuracy on task 2 now got lower to 31.54%, while the overall accuracy was 65.61 %, much higher than what we had in problem 1. We can see that using the trained backbone from task 1 and freezing it for task 2 training (while training the newly added layer) has much lower test accuracy than training the whole pipeline for task 2 as we did in problem 1. However, the overall accuracy increased as we kept the backbone and the task 1 classification layer fixed, and had a separate layer for task 2.

3. Training our model for 3 epochs on the data containing the first 5 digits (task 1) and then training for 3 more epochs on the data containing all the digits (by adding a new classifier for task 2) resulted in much higher accuracy compared to what we had in problem 1. The test accuracy on task 1 and task 2 were 99.63% and 98.70%, respectively. The overall test accuracy was around 99.17%. Having such good performance was expected as we trained the model, which was trained only on the data containing the first 5 digits, on the data containing all digits for 3 epochs. In problem 1, in the second stage of our training, we only trained the model on the data containing the last 5 digit information.

4. The training setting is very similar to what we had in problem 3, but at the second stage of training, we have less amount of data containing the first 5 digits. As expected, the test accuracy on task 1 (after the second stage of training) was a bit lower compared to what we had in problem 3, around 87.41%. However, it is much better compared to problem 1, where we had 0% test accuracy for task 1, and it is slightly worse than the test accuracy for task 1 in problem 2. When it comes to task 2, the test accuracy was approximately 99.53%, which is a bit lower than what we had in problem 1, however it is much higher when compared to the test accuracy of task 2 in problem 2 and 3. The overall test accuracy was 99.47%.

   So, we can see that the training setting in problem 3 was the best in terms of overall test accuracy, however the training setting of problem 4 gave us pretty similar overall test accuracy while training on much less data.

5. We implemented the training to reflect the “Learning without Forgetting” scheme we learned in the class (without weight regularization in the loss function). Our implementation is generally similar to the algorithm provided below (from the lecture slides).



```
LEARNINGWITHOUTFORGETTING:
  Start with:
    $\theta_s$: shared parameters
    $\theta_o$: task specific parameters for each old task
    $X_n, Y_n$: training data and ground truth on the new task
  Initialize:
    $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$    // compute output of old tasks for new data
    $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$    // randomly initialize new parameters
  Train:
    Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$    // old task output
    Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$    // new task output
    $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n)\right)$
```

   We trained our model on task 1, and then trained the model on task 2 with an additional loss term. That loss term was computed by passing the task 2 data to the original model to get predicted labels, which were used as labels for that additional loss term (the model outputs gotten from training on task 2 were used as predictions in that loss).

   The test accuracy after the second stage training for task 1 and task 2 were around 14.55% and 97.61%, respectively. As we can see, we could achieve much better performance compared to problem 1 on task 1, however the test performance was much lower compared to what we had in problem 2,3 and 4. The test performance for task 2 was very similar to that of training settings in previous problems. As we increased the coefficient of the newly added loss term, the test performance for task 1 got higher while the test performance for task 2 got lower, as expected.