CSE 599 Project: Network Plasticity

Alexey Yermakov
alexeyy@uw.edu

ABSTRACT

This research focuses on exploring a novel method for introducing plasticity to a neural network. This is done by taking an existing neural network trained on one objective, modifying the objective to be something totally different, and creating a new model that uses information learned previously to learn the new objective quicker. Then, after the new model training has completed, the old neurons are removed and end up with a model that is optimized for the new objective. The idea is that we should be able to use a previously trained model on some task to speed up training on a new task. In this work we explore simple scenarios to demonstrate how to introduce network plasticity to learn a new objective quicker than training from scratch. This has implications in deep RL such as sim-to-real problems and other non-stationary objectives.

KEYWORDS

Deep Learning; Convolutional Neural Networks; Multi-Layer Perceptrons; Capacity Loss; Plasticity

1 INTRODUCTION/BACKGROUND

Generalization is an important problem that needs to be solved in Reinforcement Learning (RL). With generalization, an RL algorithm trained in one situation should be able to adapt to a different environment: whether the reward function, state space, available actions, or transition probability distribution has changed. Different regimes of RL have been studied to obtain generalized RL algorithms, from trying to identify zero shot RL algorithms and their performance [6] to allowing for learning from a new environment in meta-RL [3]. However, recent work has observed that a more fundamental problem needs to be solved.

Training a neural network on a non-stationary objective leads to worse performance if the weights are not reset once the objective has changed [7]. This poses a problem in the deep RL framework where one might train a deep RL policy in a simulation and tries to apply it to a real-world scenario with a robot. An engineer or researcher may want the robot to continue learning from its mistakes, but the loss in a neural network's "capacity" to learn on the new objective will hinder the final performance of the robot. Thus, it is interesting to study the "plasticity" of a neural network: "the ability of a neural network to quickly change its predictions in response to new information" [8].

Existing research has attempted to reintroduce plasticity in a plethora of different ways. One method attempted this by introducing a newly initialized surrogate model $h_{\theta'}(x)$ with an existing trained model $h_\theta(x)$ [9]. These two models were combined to create a plasticity-enhanced model $\phi(x) = h_\theta(x) + h_{\theta'_1}(x) - h_{\theta'_2}(x)$, where $\theta'_1 = \theta'_2$ at initialization. Then, $h_\theta(x)$ and $h_{\theta'_2}(x)$ are frozen so that only $h_{\theta'_1}(x)$ is trained. This preserves the outputs at initialization for $\phi(x)$ to be equal to $h_\theta(x)$. The downside to this approach is that the final model has three times the parameters as the original model. Another approach attempted to use down-shifted model weights and then perturbing them [2], effectively bringing the parameters back to a state similar to their initialization. A third approach explored in two different papers used a teacher-student distillation model [5, 11]. Another approach was resetting the last layer of a model [10], which saw some popularity in the regime of pre-training and then fine-tuning LLMs. Further research explored resetting "useless" neurons [4, 12], changing activation functions [1], and modifying the loss function [7]. All of these approaches saw moderate success in retaining neural network plasticity, but none solved the problem outright. We seek to explore how to preserve network plasticity on a non-stationary objective so that a model can be trained quicker and more accurately than from scratch by using a previously trained model.

2 METHOD

In this project, we'll explore a different, novel technique that may be useful in reintroducing plasticity to a neural network after it has been trained on an objective. Our approach utilizes the straightforward intuition that a new model has plasticity and an old model has learned knowledge. So, after the training objective has changed, we slowly distill the old information into the new model through a slowly-varying parameter. For example, we first train a model on the original objective as in Figure 1. Then, we use the originally trained model by combining it with a newly initialized, but otherwise identical model as in Figure 2. Then, during the training procedure, we slowly vary α and β so that the information from the original model is distilled into the new model. Note that $\alpha : 0 \longrightarrow 1$ and $\beta = 1 - \alpha$. We also train a third model continuously throughout as another baseline.

Given that plasticity and capacity loss is easier to observe in smaller models [7, 8], our work on using a simple dataset and model is justified.

Our method is better than previous methods because:

- It utilizes weights from the initial model trained on the previous objective.
- It is simple to implement.
- The final model after training has the same number of parameters as training from scratch.

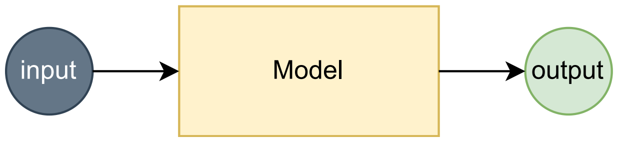

Figure 1: Original training method.

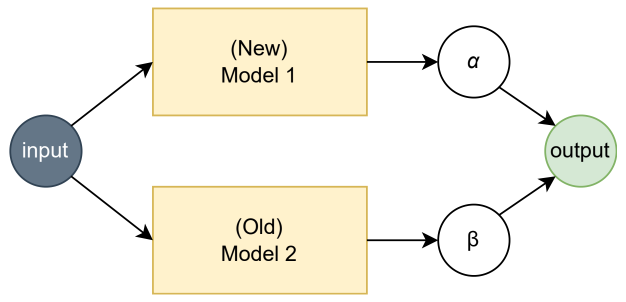

Figure 2: Modified/Expanded training method.

3 EXPERIMENTS

For the experiments, we study the effect of plasticity on two models (a MLP and a CNN) on two different datasets (MNIST and CIFAR10). The model architectures and training hyperparameters for each of the four combinations are below:

MLP / MNIST (3,706,762 parameters):

- Linear Layer
  (input_dim: 784, output_dim: 1568)
- ReLU
- Linear Layer
  (input_dim: 1568, output_dim: 1568)
- ReLU
- Linear Layer
  (input_dim: 1568, output_dim: 1568)
- ReLU
- Linear Layer
  (input_dim: 1568, output_dim: 10)

MLP / CIFAR10 (132,206,602 parameters):

- Linear Layer
  (input_dim: 3072, output_dim: 6144)
- ReLU
- Linear Layer
  (input_dim: 6144, output_dim: 6144)
- ReLU
- Linear Layer
  (input_dim: 6144, output_dim: 6144)
- ReLU
- Linear Layer
  (input_dim: 6144, output_dim: 10)

CNN / MNIST (30,094 parameters):

- Convolutional Part
  - Conv2D
    (in_channels: 1, out_channels: 2, kernel_size: 3, padding: same)
  - ReLU
  - MaxPool2D
    (kernel_size: 2, stride: 2)
  - Conv2D
    (in_channels: 2, out_channels: 4, kernel_size: 3, padding: same)
  - ReLU
  - MaxPool2D
    (kernel_size: 2, stride: 2)
- Linear Part
  - Linear Layer
    (input_dim: 196, output_dim: 98)
  - ReLU
  - Linear Layer
    (input_dim: 98, output_dim: 98)
  - ReLU
  - Linear Layer
    (input_dim: 98, output_dim: 10)

CNN / CIFAR10 (447,814 parameters):

- Convolutional Part
  - Conv2D
    (in_channels: 3, out_channels: 6, kernel_size: 3, padding: same)
  - ReLU
  - MaxPool2D
    (kernel_size: 2, stride: 2)
  - Conv2D
    (in_channels: 6, out_channels: 12, kernel_size: 3, padding: same)
  - ReLU
  - MaxPool2D
    (kernel_size: 2, stride: 2)
- Linear Part
  - Linear Layer
    (input_dim: 768, output_dim: 384)
  - ReLU
  - Linear Layer
    (input_dim: 384, output_dim: 384)
  - ReLU
  - Linear Layer
    (input_dim: 384, output_dim: 10)

The optimizer we used was Adam with a learning rate of 0.001 for MNIST/MLP, 0.00005 for CIFAR10/MLP, 0.01 for MNIST/CNN, and 0.001 for CIFAR10/CNN. We used $\beta_1 = \beta_2 = 0.9$, a weight decay of 0.9, and $\epsilon = 1e - 9$ (inspired by the hyperparameters mentioned in [8]).

To observe the plasticity of the models on the datasets, we train the models for 10 epochs on a fixed objective before updating the training objective. The training objective can be updated in one of two ways. The first method is *class shuffling*. In this method, the ten labels for MNIST and CIFAR10 are shuffled randomly amongst themselves. Thus, two images with the label "1" will have the same, but different randomly assigned label (see Figure 4). The second method is is *full shuffling*. In this method, the ten labels for MNIST and CIFAR10 are shuffled completely randomly. This is done by creating a dummy MLP whose weights are never updated and using its initial predictions for class labels as the training labels. This follows the method in [7] (see Figure 5). The original dataset labels are provided below as well, in Figure 3. For 30 restarts using one of the two methods described we shuffle the dataset labels, introducing non-stationarity to the training objective.

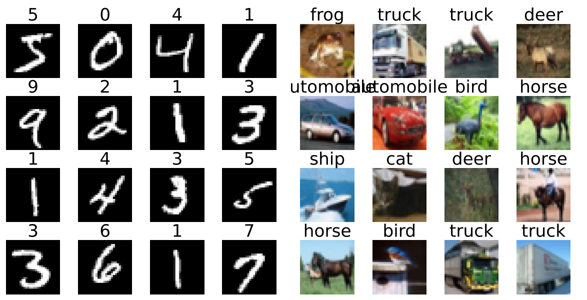
5 0 4 1 frog truck truck deer
9 2 1 3 utomobileautomobile bird horse
1 4 3 5 ship cat deer horse
3 6 1 7 horse bird truck truck

Figure 3: MNIST and CIFAR10 with no shuffling of labels.

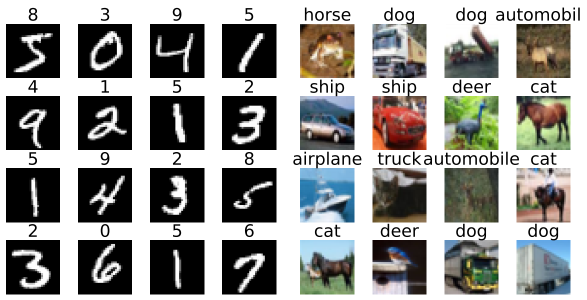
8 3 9 5 horse dog dog automobil
4 1 5 2 ship ship deer cat
5 9 2 8 airplane truckautomobile cat
2 0 5 6 cat deer dog dog

Figure 4: MNIST and CIFAR10 with class shuffling of labels.

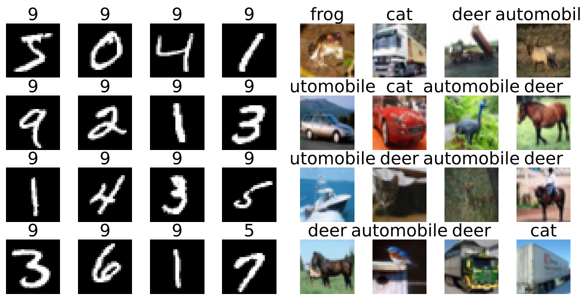
9 9 9 9 frog cat deer automobil
9 9 9 9 utomobile cat automobile deer
9 9 9 9 utomobile deer automobile deer
9 9 9 5 deer automobile deer cat

Figure 5: MNIST and CIFAR10 with full shuffling of labels.

For each dataset/model combination we train three models. The first is a model which gets re-initialized with each restart. The second is a model which is never re-initialized and trains continuously on the moving objective. The final model utilizes the modified/expanded model presented earlier. Every restart, the expanded model copies the training weights of the "New" portion of the expanded model from the previous restart into the "Old" portion, and re-initializes the "New" portion (see 2). For each restart, we take note of the training loss, testing loss, training accuracy, and testing accuracy of all three models.

Throughout training, α in our modified model starts out as having the value 0.2 (meaning the old model is dominant) and increases until it has value 1 at epoch 5 (meaning the new model is dominant), increasing by 0.2 every epoch. Note that $\beta = 1 - \alpha$. In intermediate states, both models have an effect on the output. Then, this output is passed to the Cross-Entropy loss function. It is important that the old model's weights are fixed throughout training and we only back-propagate through the new model. Then, we can also remove the old model after $\alpha = 1$ since it has no effect on the output.

4 RESULTS

To obtain quantitative results, we trained the three previously described models for each dataset/model architecture combination. Below are the plots for the training loss, training accuracy, testing loss, and testing accuracy for three models trained on each model/dataset combination, as described in section 3:

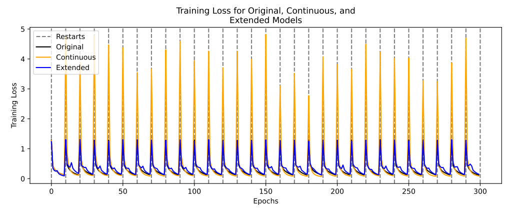

Figure 6: Training loss for the MLPs trained on MNIST with class shuffling.

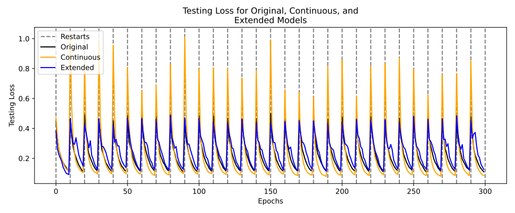

Figure 7: Testing loss for the MLPs trained on MNIST with class shuffling.

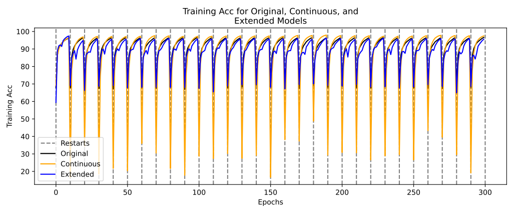

Figure 8: Training accuracy for the MLPs trained on MNIST with class shuffling.

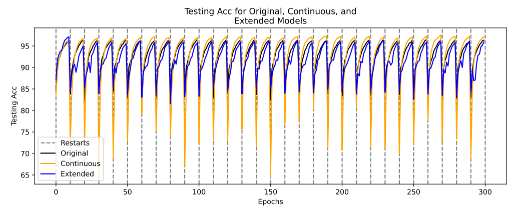

Figure 9: Testing accuracy for the MLPs trained on MNIST with class shuffling.

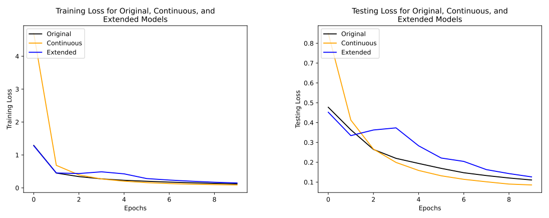

Figure 10: Last restart training and testing loss for the MLPs trained on MNIST with class shuffling.

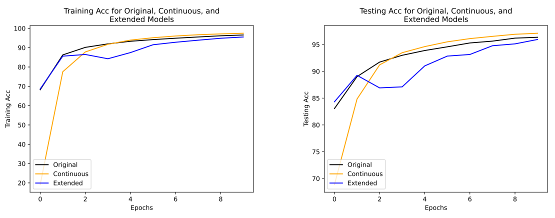

Figure 11: Last restart training and testing accuracy for the MLPs trained on MNIST with class shuffling.

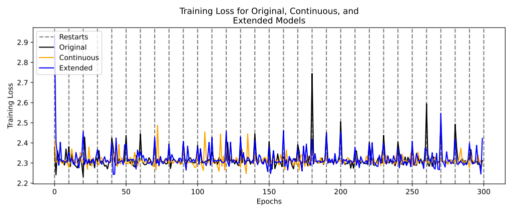

Figure 12: Training loss for the MLPs trained on MNIST with full shuffling.

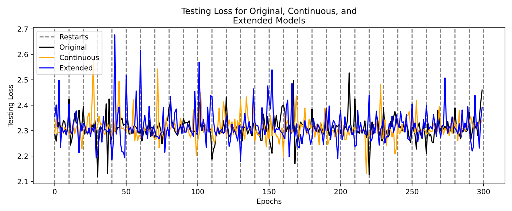

Figure 13: Testing loss for the MLPs trained on MNIST with full shuffling.

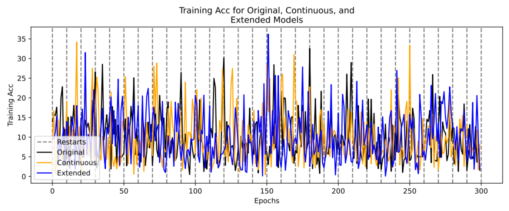

Figure 14: Training accuracy for the MLPs trained on MNIST with full shuffling.

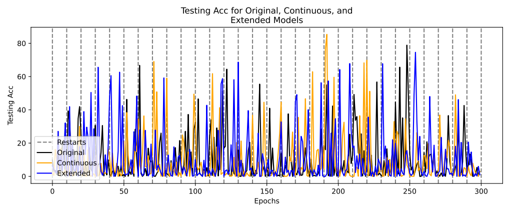

Figure 15: Testing accuracy for the MLPs trained on MNIST with full shuffling.

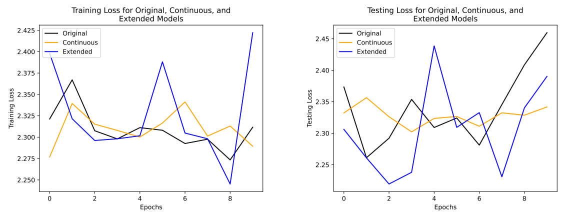

Figure 16: Last restart training and testing loss for the MLPs trained on MNIST with full shuffling.

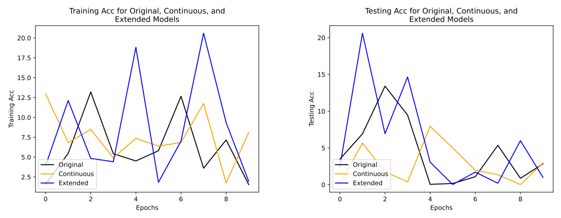

Figure 17: Last restart training and testing accuracy for the MLPs trained on MNIST with full shuffling.

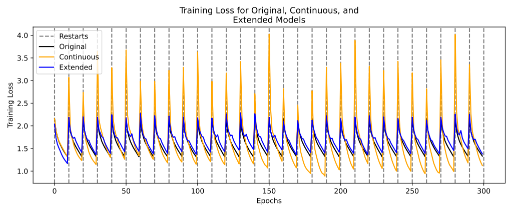

Figure 18: Training loss for the MLPs trained on CIFAR10 with class shuffling.

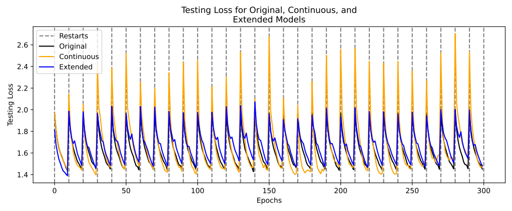

Figure 19: Testing loss for the MLPs trained on CIFAR10 with class shuffling.

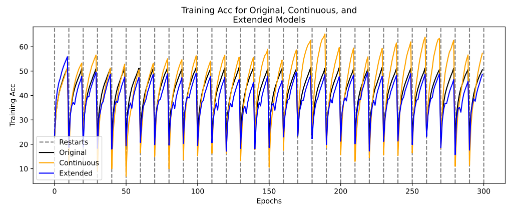

Figure 20: Training accuracy for the MLPs trained on CIFAR10 with class shuffling.

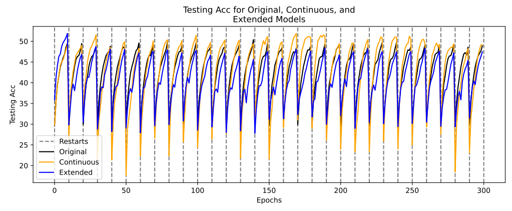

Figure 21: Testing accuracy for the MLPs trained on CIFAR10 with class shuffling.

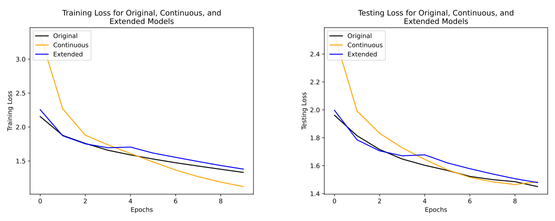

Figure 22: Last restart training and testing loss for the MLPs trained on CIFAR10 with class shuffling.

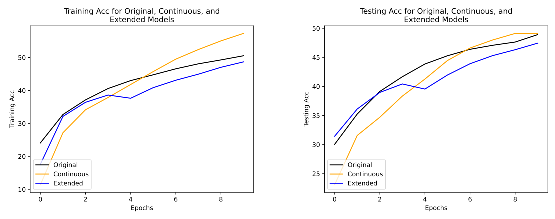

Figure 23: Last restart training and testing accuracy for the MLPs trained on CIFAR10 with class shuffling.

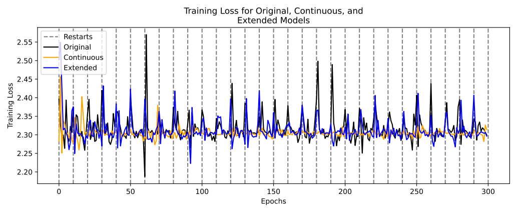

Figure 24: Training loss for the MLPs trained on CIFAR10 with full shuffling.

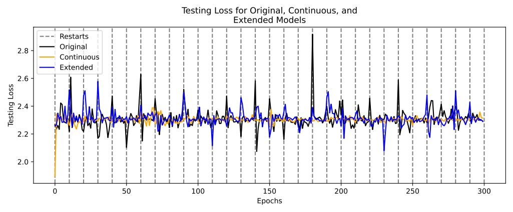

Figure 25: Testing loss for the MLPs trained on CIFAR10 with full shuffling.

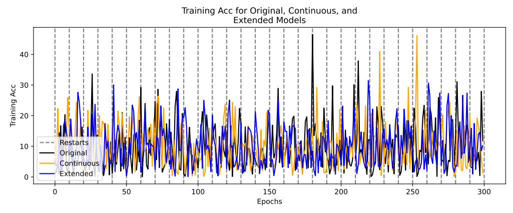

Figure 26: Training accuracy for the MLPs trained on CIFAR10 with full shuffling.

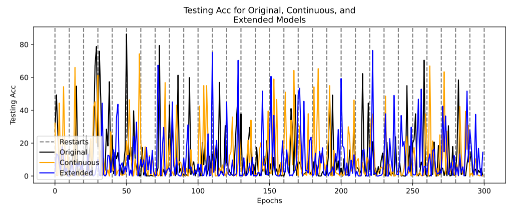

Figure 27: Testing accuracy for the MLPs trained on CIFAR10 with full shuffling.

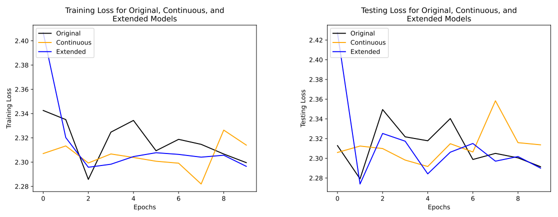

Figure 28: Last restart training and testing loss for the MLPs trained on CIFAR10 with full shuffling.

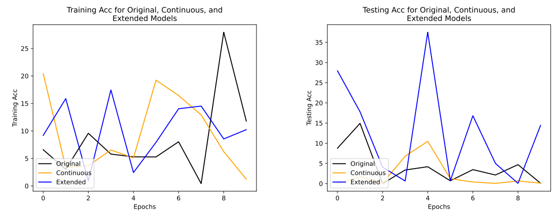

Figure 29: Last restart training and testing accuracy for the MLPs trained on CIFAR10 with full shuffling.

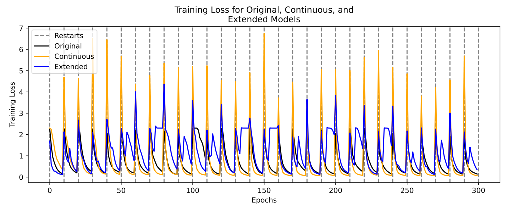

Figure 30: Training loss for the CNNs trained on MNIST with class shuffling.

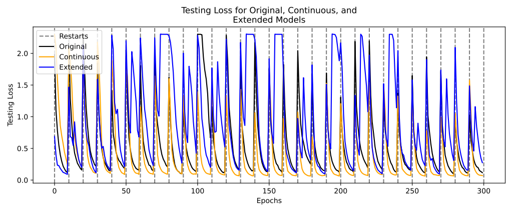

Figure 31: Testing loss for the CNNs trained on MNIST with class shuffling.

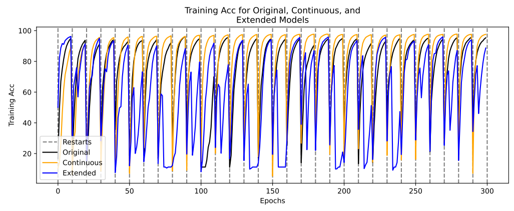

Figure 32: Training accuracy for the CNNs trained on MNIST with class shuffling.

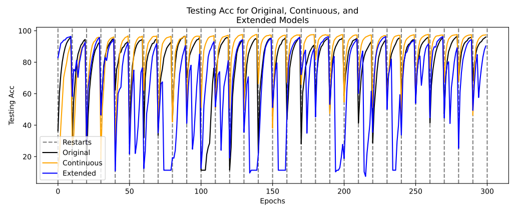

Figure 33: Testing accuracy for the CNNs trained on MNIST with class shuffling.

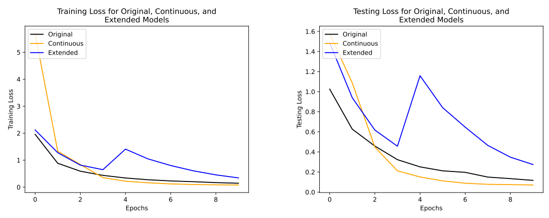

Figure 34: Last restart training and testing loss for the CNNs trained on MNIST with class shuffling.

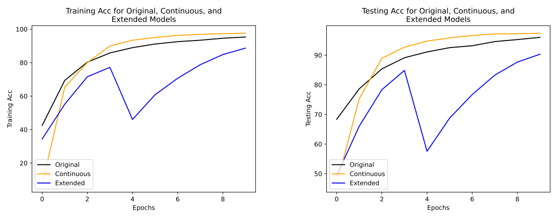

Figure 35: Last restart training and testing accuracy for the CNNs trained on MNIST with class shuffling.

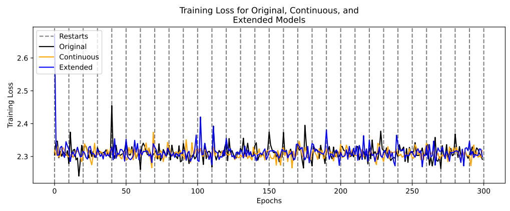

Figure 36: Training loss for the CNNs trained on MNIST with full shuffling.

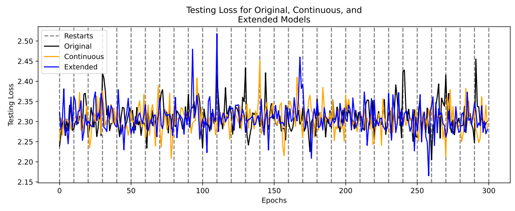

Figure 37: Testing loss for the CNNs trained on MNIST with full shuffling.

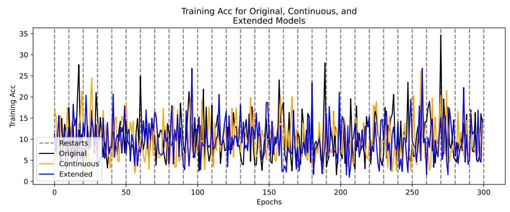

Figure 38: Training accuracy for the CNNs trained on MNIST with full shuffling.

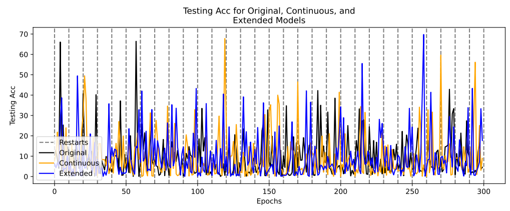

Figure 39: Testing accuracy for the CNNs trained on MNIST with full shuffling.

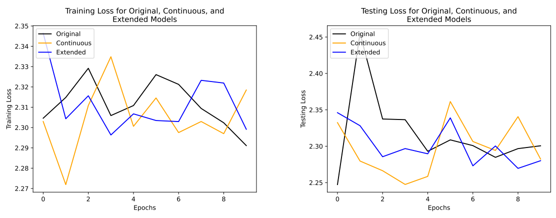

Figure 40: Last restart training and testing loss for the CNNs trained on MNIST with full shuffling.

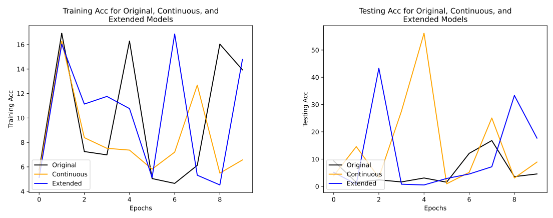

Figure 41: Last restart training and testing accuracy for the CNNs trained on MNIST with full shuffling.

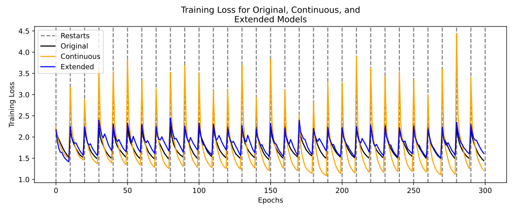

Figure 42: Training loss for the CNNs trained on CIFAR10 with class shuffling.

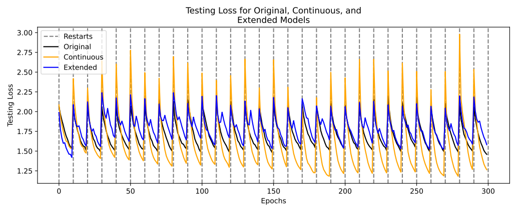

Figure 43: Testing loss for the CNNs trained on CIFAR10 with class shuffling.

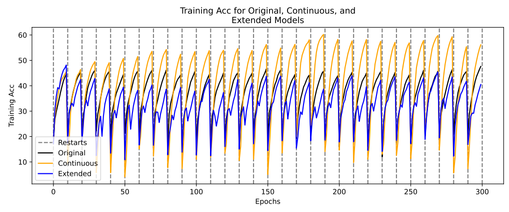

Figure 44: Training accuracy for the CNNs trained on CIFAR10 with class shuffling.

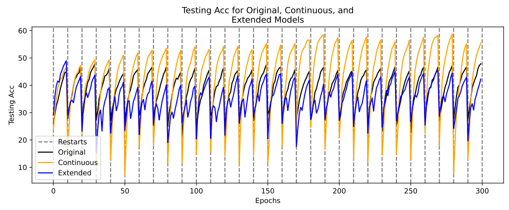

Figure 45: Testing accuracy for the CNNs trained on CIFAR10 with class shuffling.

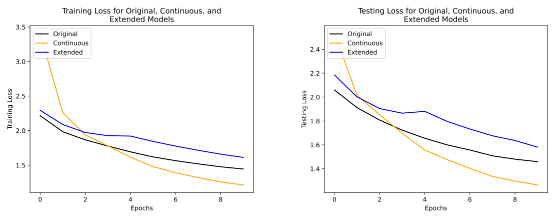

Figure 46: Last restart training and testing loss for the CNNs trained on CIFAR10 with class shuffling.

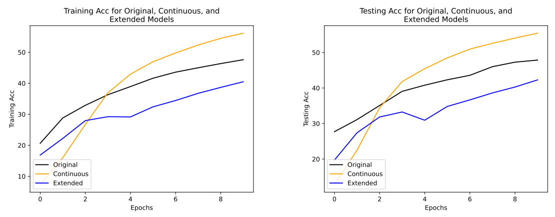

Figure 47: Last restart training and testing accuracy for the CNNs trained on CIFAR10 with class shuffling.

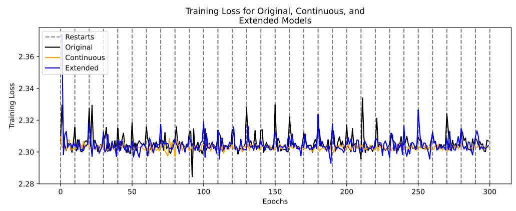

Figure 48: Training loss for the CNNs trained on CIFAR10 with full shuffling.

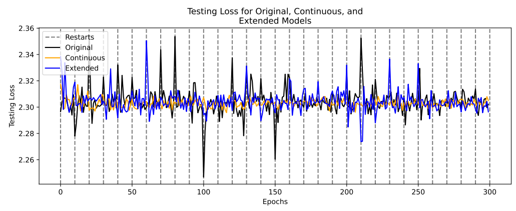

Figure 49: Testing loss for the CNNs trained on CIFAR10 with full shuffling.

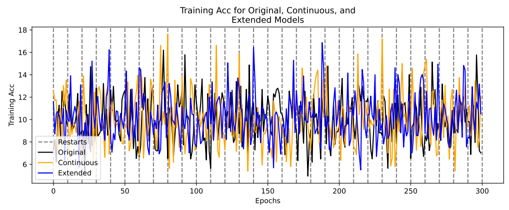

Figure 50: Training accuracy for the CNNs trained on CIFAR10 with full shuffling.

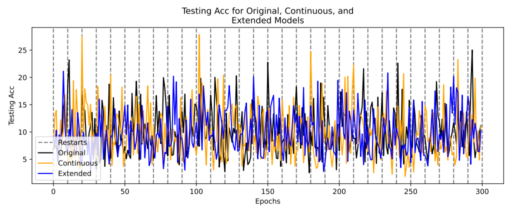

Figure 51: Testing accuracy for the CNNs trained on CIFAR10 with full shuffling.

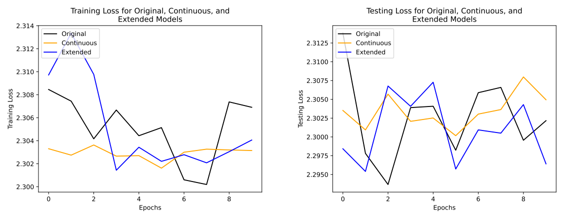

Figure 52: Last restart training and testing loss for the CNNs trained on CIFAR10 with full shuffling.

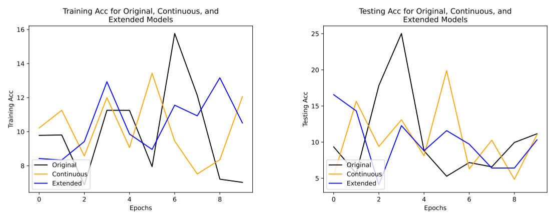

Figure 53: Last restart training and testing accuracy for the CNNs trained on CIFAR10 with full shuffling.

5 DISCUSSION

The above figures show some interesting results and behaviors. In general, it appears that full label shuffling causes the models to not be able to learn at all. This is observed from the training and testing loss and accuracy being totally noisy independent of the model being used, and showing no marked improvements per epoch over any restart. I'm not sure what's causing this behavior, as it seems to be implemented the same way as described in [8] and was a way they tested network plasticity. Fortunately, the figures for class shuffling are much more informative.

For the MLP trained on MNIST with class shuffling, we see that after each restart, the continuously trained model performs worse initially (compared to the other two models). This is likely due to it having just been trained on a different objective and not realizing it is now being trained on a new objective. However, over time, it surpasses the other two models in terms of final loss and accuracy for both the training and testing splits! This is fascinating, since it's a totally different result than what was seen in [8] and [7]. We would expect the continuously trained model to not fare better as more restarts are performed due to network capacity loss. Furthermore, we see that the extended model seems to be performing poorly compared to the original, newly initialized model. This shows us that the extended model is not using previously learned behavior efficiently. This is likely due to the effect of the previous model having information from the previous objective being counter-productive to the learning of the extended model's newly initialized hyperparameters.

For the MLP trained on CIFAR10 with class shuffling, we see more or less the same behavior as the MLP trained on MNIST with class shuffling, except that the continuous model is performing even better in terms of final losses and accuracies. This could be showing that an opposite effect is happening compared to what was initially stated to be plasticity loss: as the objective is changing, the continuous model is able to learn *better* than the other models! The effect is perhaps exaggerated in CIFAR10 because it contains more data than MNIST, and thus more information can be learned.

For the CNN trained on MNIST with class shuffling, we see more or less the same behavior as the MLPs, except that the extended model is performing even worse. This could be a result of the CNN models containing significantly less parameters that can be trained, so the negative effect of the extended model's frozen parameters are much more difficult to overcome for the trainable parameters.

For the CNN trained on CIFAR with class shuffling, we see more or less the same behavior as the MLPs. We once again see that the continuous model is achieving better results at the end of every restart. In fact, from Figure 45 we see that the 10 epochs per restart isn't enough for the CNN to learn the dataset labels, and so the continuous model is able to extrapolate information from all previous restarts to achieve a higher testing accuracy every subsequent restart! This appears to be a clear indication of network plasticity *not* being lost.

6 CONCLUSION

This work explores the effects of a changing objective on simple models. In particular, a CNN and MLP are trained on MNIST and CIFAR10 with two different non-stationary objectives. One case is where the class labels are shuffled per-class and the other is where the class labels are shuffled per-image. It appears that shuffling the labels per-image results in the models being unable to learn any association from image to label. This makes intuitive sense, since there is not way even a human would be able to discern from two images showing a “1” which belongs to the class “6” and which to the class “2”. Thus, the above discussion focused on network plasticity in the context of class shuffling.

The results show that continuously trained models on a non-stationary objective appear to be benefiting from the restarts. In particular, for the continuous CNN trained on CIFAR10, it is shown that the testing accuracy improves per restart! This indicates that network plasticity is not, in fact, being lost with a non-stationary objective. Furthermore, the proposed extended model appears to be harmed by the inclusion of parameters from the previous objective. Thus, we can't conclude that the model from Figure 2 adds plasticity to the model, allowing it to learn quicker than a newly-initialized model. This is seen from the newly-initialized model consistently performing better than the extended model across all model/dataset combinations for class shuffling.

Future work can explore the effects of training more epochs per restart, playing around with the relationship between α and β per epoch for the extended model, and trying to train larger, more complex models on MNIST and CIFAR10. Training more epochs per restart could allow the models to exhibit plasticity loss. It is possible that training 10 epochs is not enough for the network to lose its plasticity before the objective changes. Additionally, the linear relationship implemented in the experiments between epoch and α could be harming the expanded model's performance on all tasks. Lastly, it is possible that the number of trainable parameters in the networks used in the experiments are not large enough to be able to train on fully shuffled labels, explaining the poor performance seen for all model/dataset combinations. The idea here is that maybe the model would over-fit to the training data, which would then show network plasticity loss better.

The entire codebase is available on GitHub [13].

7 REFERENCES

REFERENCES

[1] Zaheer Abbas et al. “Loss of plasticity in continual deep reinforcement learning”. In: *Conference on Lifelong Learning Agents*. PMLR. 2023, pp. 620–636.

[2] Jordan Ash and Ryan P Adams. “On warm-starting neural network training”. In: *Advances in neural information processing systems* 33 (2020), pp. 3884–3894.

[3] Jacob Beck et al. “A survey of meta-reinforcement learning”. In: *arXiv preprint arXiv:2301.08028* (2023).

[4] Shibhansh Dohare, Qingfeng Lan, and A Rupam Mahmood. “Overcoming policy collapse in deep reinforcement learning”. In: *Sixteenth European Workshop on Reinforcement Learning*. 2023.

[5] Maximilian Igl et al. “Transient non-stationarity and generalisation in deep reinforcement learning”. In: *arXiv preprint arXiv:2006.05826* (2020).

[6] Robert Kirk et al. “A survey of generalisation in deep reinforcement learning”. In: *arXiv preprint arXiv:2111.09794* 1 (2021), p. 16.

[7] Clare Lyle, Mark Rowland, and Will Dabney. “Understanding and preventing capacity loss in reinforcement learning”. In: *arXiv preprint arXiv:2204.09560* (2022).

[8] Clare Lyle et al. “Understanding plasticity in neural networks”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 23190–23211.

[9] Evgenii Nikishin et al. “Deep reinforcement learning with plasticity injection”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[10] Evgenii Nikishin et al. “The primacy bias in deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2022, pp. 16828–16847.

[11] Simon Schmitt et al. *Kickstarting Deep Reinforcement Learning*. 2018. arXiv: 1803.03835 [cs.LG].

[12] Ghada Sokar et al. “The dormant neuron phenomenon in deep reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 32145–32168.

[13] Alexey Yermakov. https://github.com/yyexela/CSE599_Project. 2024.