Continual Learning Framework With Modular Autoencoder And Expert Networks

Mingkun Tian Yuchao Gao
New York University
{mt4835, yg3285}@nyu.edu

Abstract

Continual learning has recently received significant attention, leading to the development of various approaches. In this report, we introduce a new continual learning algorithm designed for training a task-agnostic model utilizing autoencoder and expert networks. During training, our algorithm autonomously directs data to its corresponding task expert based on autoencoders' decisions, which hinge on the disparity in reconstruction loss between the given data and its records. Simultaneously, the autoencoder network undergoes training while data routing occurs. After training, the autoencoder network is being fully trained so that it provides accurate routing information of each task effectively. At inference, the model selects the task expert based on the autoencoder network's appropriate decision. Our experimental results show the algorithm's performance on two benchmark continual learning datasets, achieving decent results on different settings. We conclude with several suggestions for refining the autoencoders' decision-making process, particularly tailored to tasks with varying degrees of similarity. The code is available at https://github.com/jwl25b/AdvancedML_Final_Project

1 Introduction

Continual learning, also known as lifelong learning, is an approach in artificial intelligence that aims to enable models to learn from data over time, adapting and evolving as new information becomes available. A major challenge in continual learning is "catastrophic forgetting," where a model, when exposed to new tasks or data, forgets previously learned information. This phenomenon undermines the ability of AI systems to retain knowledge, making it difficult for them to generalize and perform effectively across diverse tasks.

To address this challenge, we proposed a framework built on autoencoder-expert networks which creates a continual learning model designed to manage the interplay between learning and retaining information effectively. In this framework, each expert is specialized to handle a specific type of task, while the autoencoder acts as a classifier, directing each batch of data to its corresponding task expert.

This classification is based on the disparity in reconstruction loss between the given data and its records. During training, the autoencoder distinguishes the type of task associated with a batch of data and directs it to the appropriate expert for training. Our approach allows each expert to specialize in a single type of task, facilitating efficient and effective learning across diverse datasets.

This report is organized as follows. Section 2 provides the background material and related work including the single-model approach and the multi-model approach. In Section 3, we detail the structure and algorithm of our proposed model. Section 4 provides the experiments conducted to validate the effectiveness of our approach. Conclusion and future work are discussed in section 5.

Preprint. Under review.

2 Background and Related Work

2.1 Single model with regularization based method for multi-task learning

In a single model implementation, one of the approaches to mitigate catastrophic forgetting is using regularization-based methods, which constrains weight updates to limit changes and prevents the loss of existing knowledge. Methods such as L2 regularization are commonly used to penalize large weight updates, ensuring model stability across diverse tasks.

A specific approach is the Elastic Weight Consolidation (EWC) framework Kirkpatrick et al. (2017). By regularizing important weights, EWC framework prevents significant changes during subsequent training sessions. However, while EWC is effective for tasks with overlapping solution spaces, it encounters difficulties with tasks that have vastly different or have non-overlapping solution spaces. This constraint can result in a decline in performance due to the difficulty of striking a balance between learning new tasks and preserving existing knowledge. encounter challenges in effectively managing the trade-off between acquiring new information and retaining previously acquired knowledge.

2.2 Generalist and specialist model networks for multi-task learning

Jacobs et al. (1991) were among the pioneers in employing a strategy of utilizing multiple models, with each model addressing specific tasks. In their work, they developed an adaptive mixture of experts, where each expert was represented by a neural network, to tackle multi-speaker vowel recognition. They also implemented a distinct gating network to decide which expert network to employ for each sample. However, a drawback of their approach was that every training sample had to be traversed through each expert for the gating function to be trained.

Aljundi et al. (2017) proposed a mixture of one generalist model and many specialist models to avoid this issue. Their approach involved training under-complete autoencoder networks as the generalist model, to grasp low-dimensional sub-spaces for individual tasks or domains. This enabled the network to acquire specialized models (experts) by leveraging knowledge transfer from the most relevant preceding task. Our model extends the concept of utilizing autoencoder networks within a hybrid learning setting, specifically designed for mini-batch training. This approach is flexible and can be adapted for task-agnostic scenarios with further modifications.

2.3 Autoencoder

An autoencoder is essentially a specialized form of feed-forward network. Its training objective is to replicate its input as closely as possible in its output. Typically, autoencoders are constrained to approximate copies and are limited to reproducing input data that closely resembles what they were trained on. By necessitating the model to prioritize which elements of the input to replicate, it frequently uncovers valuable data features in the process (Goodfellow et al., 2016, pp. 499).

Autoencoders have been used in various computer vision algorithms. The main usage of autoencoders are dimension reduction and feature learning (Goodfellow et al., 2016, pp. 499). Meanwhile, they are increasingly used in anomaly detection tasks. In such scenarios, autoencoders are trained on regular data to establish a baseline of "normalcy," flagging any deviations from this norm as anomalies. This approach relies on the notion that anomalous instances typically exhibit significantly higher reconstruction loss compared to normal instances Li et al. (2023).

We follow the concept introduced in Li et al. (2023), implementing under-complete autoencoders. Initially, we considered utilizing the linear autoencoder as it learns a similar subspace as PCA. However, given that autoencoders with nonlinear functions exhibit better performance on dimensionality reduction compared to PCA Hinton and Salakhutdinov (2006), we modified our autoencoder with nonlinear activation functions.

3 Approach

In this section, we describe the main assumption and the structure of our algorithm. A sample training script is provided in our GitHub repo.

We consider the scenario that tasks' identities are unspecified and each task data are i.i.d. Furthermore, all tasks have identical batch sizes, and the batch sizes are sufficiently large to infer that the features described within a data batch can broadly represent the distribution of features across the entire feature space. Additionally, the data batches arrive sequentially with no restrictions on their order or the task identity.

Our primary assumption is that the autoencoder associated with a specific task performs better at reconstructing data from that task compared to data from other tasks. Another key assumption is that the distribution of the per-sample loss follows a Gaussian distribution. This assumption will be further discussed in section 4.3.

Our model mainly contains two parts: autoencoder and expert networks, and data batch routing algorithm

3.1 Autoencoder and Expert networks

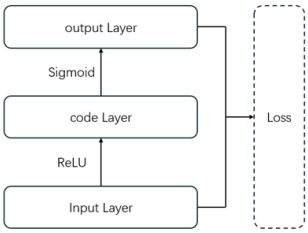
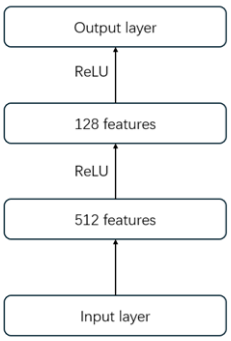


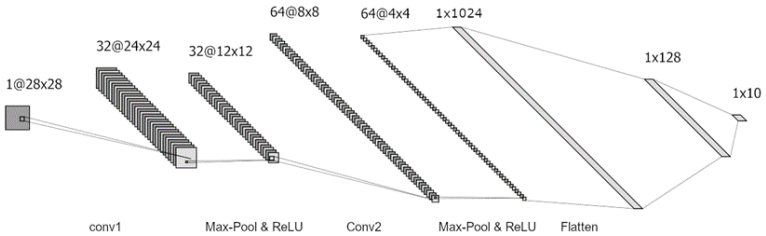
Figure 1: The Structure of Autoencoder

The autoencoder's structure is relatively straightforward (see Figure 1). it includes an input layer, a code layer, and an output layer. The dimensions of the input and output layers align with those of our algorithm's input. We conducted experiments with various sizes for the code layer, exploring dimensions ranging from 100 to 500 neurons, and determined an optimal value of 350 and 500 neurons for two datasets that we use. Additionally, we explored two types of loss functions: Binary Cross-Entropy (BCE) loss and Mean Square Error (MSE) loss, ultimately selecting BCE loss as our preferred loss function.

Furthermore, the autoencoder will retain specific information, including the mean and standard deviation of the training loss from the last processed data batch, for future use for the routing algorithm. The creation of the autoencoder will be based on decisions made by the data batch routing algorithm, and all autoencoders will maintain the aforementioned structure.



(a) MLP structure



(b) Small CNN structure

Figure 2: The Structures of Expert

For simplicity, we implement two types of experts including MLP classifier and a small CNN classifier. Figure 2a and 2b shows our implementation of MLP classifiers and small CNN classifiers. We focus on implementing experiments with the MLP classifier due to its simpler nature compared to

CNN networks, achieving faster iterations per second while maintaining reasonably decent accuracy. Moreover, the experts that initialized are identical so that we can set up a uniform baseline for evaluating the performance of the autoencoder network. Additionally, our experiments has the uniformed input size (28 by 28), but the output size might be various for different sets of tasks.

3.2 Routing algorithm at training phase

In the training phase, the provided data batch will be passed to all autoencoders, each of which will generate a per-batch loss. Based on our first assumption, the algorithm will select the optimal autoencoder-expert pairing by selecting the combination with the lowest loss. After the optimal autoencoder-expert pairing is selected, the provided data batch will be passed to this combination and generate a per-data loss.

Furthermore, our data routing algorithm relies on the Empirical Rule of the Gaussian distribution, which asserts that 99.7% of data points observed in a normal distribution fall within 3 standard deviations of the mean. According to our second assumption, for a well-trained autoencoder, iif over 99.7% of per-sample losses remain within 3 standard deviations of the mean recorded in the autoencoder, then those losses are attributed to the distribution generated by that autoencoder. Given that the well-trained autoencoder is exclusively trained with data from specific tasks, this implies that the data batch aligns with the distribution of those tasks. Conversely, if fewer than 99.7% of data points fall within 3 standard deviations of the distribution, the data batch is classified as a new task.

However, in the case of an under-trained autoencoder, more per-sample loss may fall outside the distribution. In such instances, a threshold is utilized for determination. If the count of outliers falls below this threshold, the data batch is considered to originate from the data distribution; otherwise, it is identified as a new task. We conducted several experiments based on the under-trained autoencoder and the result is discussed in the experiment section

An alternative approach for determining the origin of the data batch involves comparing the mean of the per-sample loss (this equals to per-batch loss), with the mean recorded in the best-performing autoencoder. However, based on our experiments, this method exhibits decent performance for tasks with less similarity, but it performs less effectively for tasks with greater similarity. We dive into the reasons behind this in section 4.4.

After determinied the origin of the data batch, the routing algorithm proceeds to either train the corresponding autoencoder and expert or initialize a new autoencoder and expert, integrating them into the network. In the end, the mean and standard deviation of the loss of the data batch are recorded in the autoencoder for routing future data batches.

3.3 Routing algorithm at inference phase

At inference, the algorithm uses similar routing rules to the training phase. The data batch undergoes processing by all autoencoders to determine the optimal autoencoder-expert pair and predictions are generated by the selected expert. In our experiment, we assume that no additional data batches from new tasks are passed to our model at the inference phase. However, in practical scenarios, this occurrence is possible. Therefore, in practice, we can utilize the detection rule similar to that implemented in the routing algorithm but with a lower threshold.

4 Experiments and Results

In our experiment, we use identical epoch numbers for both training and initializing experts, set at 10 epochs. However, with our third assumption, which states that the data batch broadly represents the distribution of features across the entire feature space, we use a higher epoch number (200) when initializing autoencoders. This ensures that the autoencoder captures the rough distribution of the task. Meanwhile, we use 10 epochs for training existing autoencoders so that they can learn more details of the distribution.

In each of our experiments below, we use a batch size of 300, representing 0.5% of the data for each task in the Permuted dataset and 2.5% of the data for each task in the Splitted and Shuffled dataset, and a outlier threshold is set to 0.2% of the batch size

4.1 Dataset and Dataset Modification

In our experiments, we primarily utilize two datasets: Permuted MNIST and Split MNIST. However, our model has poor performance on Split MNIST, particularly evident when using a basic autoencoder with a single code dimension, especially on one that is under-trained. For under-trained autoencoders, distinguishing between pairs of handwritten digits like (2,3), (1,7), (4,9), etc., might be a significant challenge due to the similarity among handwritten digits.

There are many potential solutions including using different structured autoencoders or incorporating convolutional layers into the autoencoder architecture. The solution we adopt involves randomly shuffling all input data using a fixed random number, thereby reducing the correlations between different handwritten digits.

Additionally, since the same random number is used to shuffle all data, we can reverse the shuffling process to enable the actual expert to learn the differences and similarities among various handwritten digits using the unshuffled data. We name this dataset as Splitted and Shuffled MNIST.

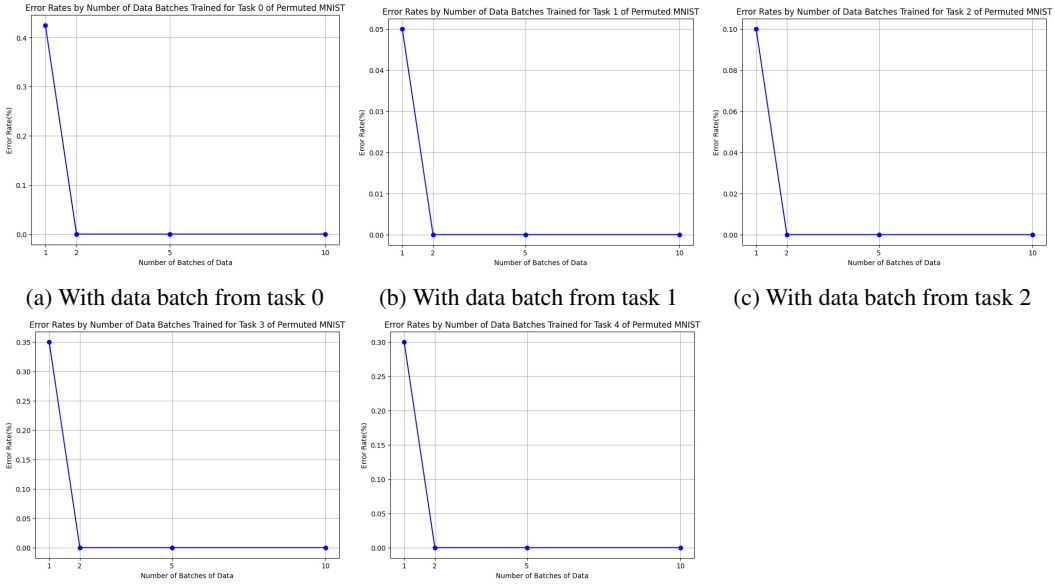
4.2 Data routing with under-trained autoencoder

To demonstrate that the under-trained autoencoders we train can classify data with high accuracy, allowing for effective routing to the appropriate expert, we conduct a series of experiments using under-trained autoencoders.

Initially, we set up five tasks, each corresponding to a potential individual autoencoder. We begin by training each autoencoder with a single random batch of data. Moreover, since each batch is designed to be associated with a specific task ID for error tracking, we can ensure that each of the autoencoder is only being trained with the data from a unique task.

After training, we evaluate the autoencoder by running it through the entire dataset and observing its performance in classifying every data batch. We count the number of batches that are mis-routed, and upon completion, calculate the error rate for each task's autoencoder.

Furthermore, we proceed to train each autoencoder with an additional batch from its intended task, increasing the total training data to two batches. We observed a reduction in the error rate compared to training with just one batch. We continue this pattern, training the autoencoders with five and ten batches of data, respectively, noting the further decline corresponding to the error rates.



(a) With data batch from task 0 (b) With data batch from task 1 (c) With data batch from task 2

(d) With data batch from task 3 (e) With data batch from task 4

Figure 3: Error rate of autoencoder trained with data batch in Permuted MNIST

We replicated this experiment 10 times using both datasets and Figure 3 refers to the result with Permuted MNIST. Figure 11 in Section 6) refers to the result with Splitted and Shuffle MNIST. Both results show that the average error rates for under-trained autoencoders remain low, even with only a single batch of training data. Furthermore, as more data is used for training, the error rates progressively decline to zero. This experiment thus confirms that the under-trained autoencoders can route data accurately, even with minimal training, and improve steadily with additional training, offering robust performance for data classification and routing.

4.3 Distribution of the per-sample loss: from under-trained to fully-trained autoencoder

In section 3, we assume that the distribution of the pre-sample loss is Gaussian distribution. In this section, we will prove that the distribution generally conforms to Gaussian distribution using the quantile-quantile plot. The experiments conducted below used the Permuted dataset.

We developed a script based on our approach to detect different tasks and initialize autoencoder networks. Initially, an under-trained autoencoder (trained with 1 data batch) is created and trained on the same task (Task 1).

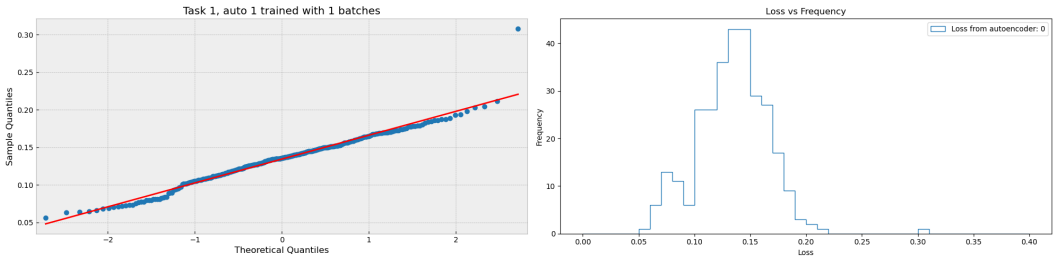


Figure 4: QQ-plot with loss from autoencoder trained with 1 batch in Permuted MNIST

Figure 4 shows the QQ-plot of the loss respects to the Gaussian distribution. It is evident that the loss generated by the correct autoencoder conforms to a Gaussian distribution.

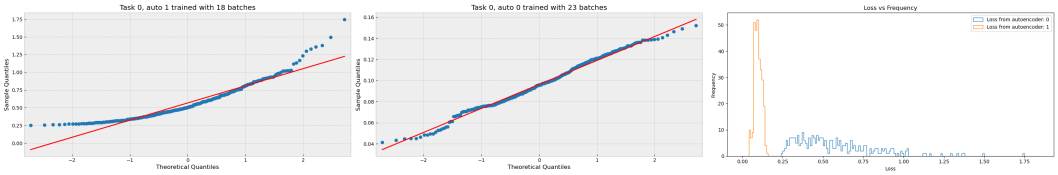


Figure 5: QQ-plot with losses from autoencoders trained with small number of batches in Permuted MNIST

Then, as the algorithm executes, the data routing algorithm detects a new task (Task 0) and proceeds to initialize a new autoencoder (auto 0). Figure 5 shows the QQ-plot for two losses with both autoencoders trained with task 0 and task 1. It is clear to see that the loss from the autoencoder trained with the same task conforms to a Gaussian distribution. And the loss from the autoencoder trained with different task is more likely to be a right-skewed Gaussian distribution.

Furthermore, this phenomena can be found in the autoencoder in various training steps (see figure 12 in Section 6).

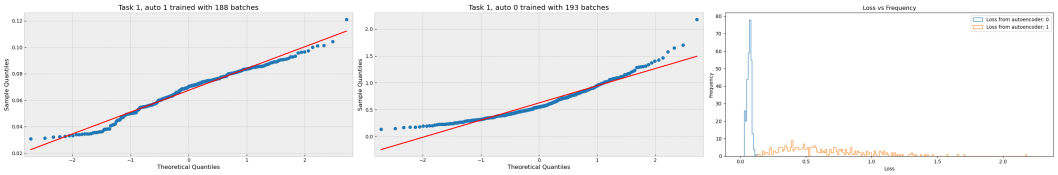


Figure 6: QQ-plot with data batch from Task 1

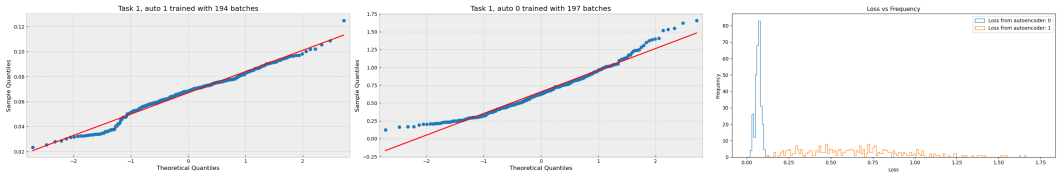


Figure 7: QQ-plot with data batch from Task 1

However, when training is almost finished, the loss distribution from the autoencoder trained on identical tasks tends to be a lightly right-and-left-tailed Gaussian distribution, with a larger mean and smaller standard deviation. The loss distribution from the autoencoder trained on different tasks is more likely to be a heavily right-tailed and lightly left-tailed Gaussian distribution, with a smaller mean and larger standard deviation (see Figures 6 and 7).

The same experiment can be replicated using the Splitted and Shuffled MNIST, with a different number of tasks (see Figure 13 in Section 6). Despite the potential variations in distribution details, we can still draw the similar conclusion regarding the general conformity of the loss distribution to a Gaussian distribution.

4.4 Contrasting Decision-Making Strategies: Outlier-Based vs. Mean-Based

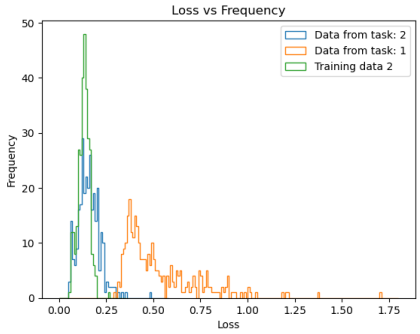


Figure 8: Loss distribution of training data batch and new data batches in Permuted MNIST

One of the methods that is mentioned in the section 3.2 to distinguish new tasks is using the mean of the per-sample loss (that is equal to per-batch loss). This method could have a decent performance for those tasks with less similarity, for example, the Permuted MNIST. Figure 8 provides an illustration of the loss data from the under-trained (1 batch) autoencoder applied to the Permuted MNIST dataset. The figure shows that the distributions of batches from the corresponding and other tasks are fairly distinguishable, with their means substantially separated.

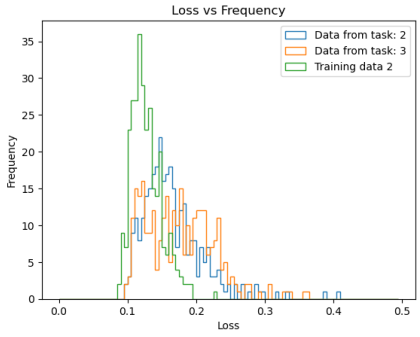
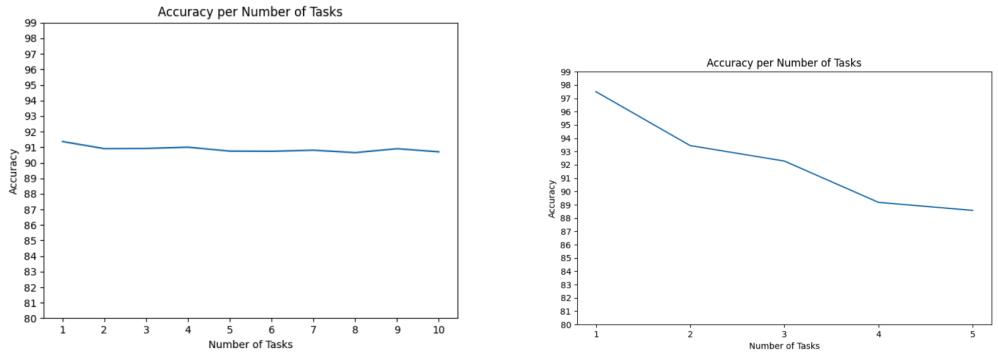


Figure 9: Loss distribution of training data batch and new data batches in Splitted and Shuffled MNIST

However, under identical configurations, there can be a considerable performance decline when dealing with tasks exhibiting greater similarity. Figure 9 illustrates this scenario with loss data from Splitted and Shuffled MNIST datasets.

Figure 9 is an example that the means of the two loss distributions are closely aligned, raising a challenge in analyzing and determining the mean threshold. However, the distribution of the loss with a different task is broader, resulting in a higher incidence of outliers compared to losses within the same task. This observation highlights the robustness of the outlier-based method when encountered with tasks of higher similarity.

4.5 General Accuracy Test with Permuted MNIST and Splitted and Shuffled MNIST



(a) Permuted MNIST (b) Splitted and Shuffled MNIST

Figure 10: Average Accuracy w.r.t two datasets

We evaluate the overall average accuracy of our algorithm across two datasets: the Permuted MNIST with 5 replications and the Split and Shuffled MNIST with 10 replications. For the Permuted MNIST dataset, we incrementally increase the number of tasks from 1 to 10, while for the Split and Shuffled MNIST dataset, we use the range between 1 and 5 tasks.

Figure 10 shows the result with both dataset. This indicate that our algorithm performs consistently and stably across all tasks on the Permuted MNIST dataset. This indicates that the autoencoder effectively avoids mis-routing and the potential bottleneck of the model is the expert network. However, on the Split and Shuffled MNIST dataset, the algorithm's performance deteriorates as the number of tasks decreases, this indicates that the autoencoder network might have mis-routing the data batch with increasing number of tasks. As discussed in Section 4.1, this phenomenon can be attributed to the similarity between different tasks. Despite our pre-processing method decreased the similarity, it is hard to eliminate it completely.

5 Conclusion and Future Works

In this report, we propose a continual learning algorithm designed to address the challenges of task-agnostic learning and catastrophic forgetting. By incorporating autoencoders and expert networks, the framework offers a modular and flexible approach to continual learning across various datasets. Our model demonstrates its capacity to handle diverse tasks effectively, delivering decent results across different settings in experimental evaluations on benchmark datasets.

The autoencoder-based expert system presents a promising avenue for high-accuracy data classification and routing. While current experiments demonstrate its potential, one of the phenomena we observed, particularly evident in the Splitted and Shuffled MNIST dataset, is the mis-routing at the beginning of training and the creation of duplicated autoencoder-expert networks midway through to the end of training.

One of the factors in our implementation is the fixed threshold. Given that the data batch may inadequately represent the distribution of features across the entire dataset, the fixed threshold could be too low initially and too high later in the training process. Using a dynamic threshold, we can

start with a higher outlier threshold at the beginning and gradually decrease it during training. This approach ensures that our algorithm avoids mis-routing the dataset at the outset and doesn't ignore any data batches from new tasks.

Furthermore, our algorithm exhibits linear (or super-linear given the duplicates are created) time and space complexity. This underscores the importance of optimizing the design of the expert component. By exploring and integrating different single-model approaches such as EWC, we can integrate a single expert responsible for all tasks with higher similarity. This strategy minimizes resource utilization in terms of time and space, enhancing the system's ability to effectively manage various tasks.

By tackling these challenges, we aim to create a robust, efficient system that continues to deliver reliable performance across diverse datasets and tasks.

References

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. 2017. Expert Gate: Lifelong Learning with a Network of Experts. arXiv:1611.06194 [cs.CV]

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

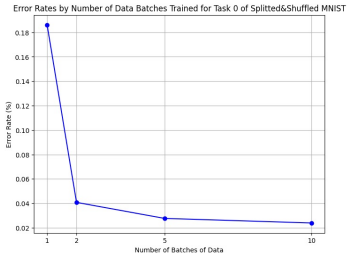
G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507. `https://doi.org/10.1126/science.1127647` arXiv:https://www.science.org/doi/pdf/10.1126/science.1127647

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Computation* 3, 1 (03 1991), 79–87. `https://doi.org/10.1162/neco.1991.3.1.79` arXiv:https://direct.mit.edu/neco/article-pdf/3/1/79/812104/neco.1991.3.1.79.pdf

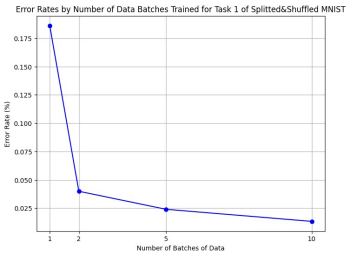
James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526. `https://doi.org/10.1073/pnas.1611835114` arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114

Jin Li, Kleanthis Malialis, and Marios M. Polycarpou. 2023. Autoencoder-based Anomaly Detection in Streaming Data with Incremental Learning and Concept Drift Adaptation. In *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE. `https://doi.org/10.1109/ijcnn54540.2023.10191328`

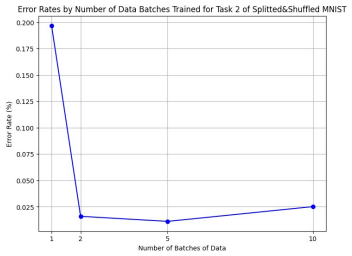
6 Appendix



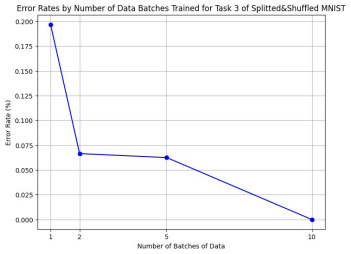
(a) With data batch from task 0



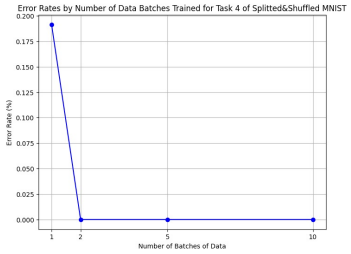
(b) With data batch from task 1



(c) With data batch from task 2



(d) With data batch from task 3



(e) With data batch from task 4

Figure 11: Error rate of autoencoder trained with data batch in Splitted and Shuffled MNIST

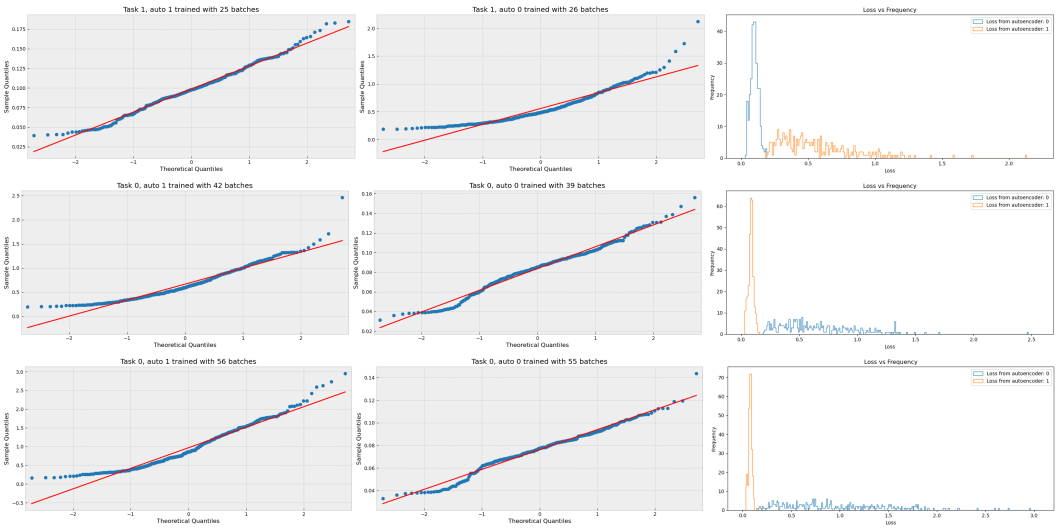


Figure 12: QQ-plot with losses from autoencoders trained with a medium amount of batches in Permuted MNIST

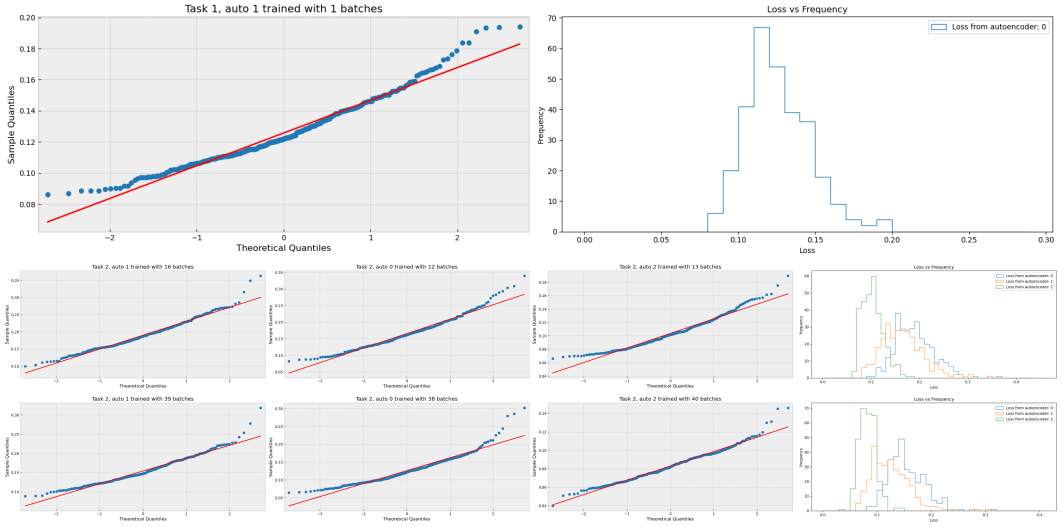


Figure 13: QQ-plot with losses from autoencoders trained with different amount of batches in Splitted and Shuffled MNIST