CE888: Short Report
Continuous Learning Using Auto-Encoders

Reference Number: 17000264

Abstract— **The aim of the project is a discussing and a trial of implementing the novel idea of using auto-encoders for Continual Learning adapted in the area of machine learning. Even though machines are proved to be trained with high intelligence, still a part of that "intelligence" term is in doubt. The reason that is, is because machines suffer from the illness of forgetfulness, meaning that they tend to forget how to perform on tasks that they were trained before. During this project we will discuss about what Continual Learning is and how auto-encoders can contribute in the cumulative and autonomous development of artificial intelligence (AI) systems in retaining useful insights for the improvement of a prediction model. Previous attempts will be also discussed along with the attempt of the engaged auto-encoders presented in this project. The methodology that will be used is by permuting in a random but fixed way the pixels of the MNIST dataset and the CIFAR10 dataset. In addition, keras auto-encoders will be initialized and trained by getting fed with small batches of data. The best auto-encoder will be picked among all of the trained ones in respect to the lowest error. Finally, we will use the traditional way of evaluating the results using the "loss" score and the "accuracy" variables of TensorFLow open-source library.**

I. INTRODUCTION

Past is the time where the word "Intelligent" was used just to describe a person or an action that they did. Nowadays, the term is widely used in the area of Computer Science and more specifically in that of Machine Learning and Artificial Intelligence (AI).

The latter refers to the ability of machines being able to demonstrate intelligence similar to humans in solving tasks. During the last decade, deployment of machine learning techniques has been implemented in various settings with such efficiency that make tasks, which have been though as impossible to be tackled by a machine, look like a child's play (e.g chess,backgammon etc.)

A. Continual Learning and Artificial Intelligence

Definition 1.1: Continual Learning (CL) relies on the notion of learning continuously and adaptively about the external world and enabling the autonomous incremental development of ever more complex skills and knowledge.

In the context of Machine Learning, it means being able to smoothly update the prediction model to take into account different tasks and data distributions but still being able to re-use and retain useful knowledge and skills during time.

One might wonder however.*Why Continual Learning?* The key word here is adaptation. The ability to forge our system to deal with the continuous changing environment and demanding circumstances. Without doubt, future AI systems will mostly rely on Continuous Learning, opposed to algorithms that are trained offline. That is the way humans learn and AI systems will increasingly have the capacity to do the same.

B. Auto-Encoders

How can auto-encoders be introduced in the idea of Continual Learning? Before we answer this question we have to understand what auto-encoders are and what they do.

Definition 1.2: An auto-encoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. In other words,an auto-encoder is a neural network that tries to reconstruct its input.

In that way, given an input, auto-encoders are able to decompose the data, identify the strongest patterns that reveal the most significant features

through noisy and corrupted data so as to reconstruct itself.

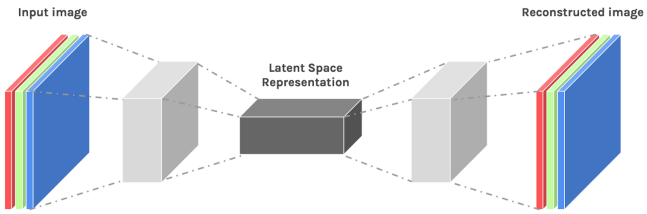

Fig. 1. Network level Representation

Hence, auto-encoders are essential in deep learning in a way. During this project we will employ the idea of creating auto-encoders to detect if there has been a change to our initial state using the MNIST and CIFAR10 datasets. But we will talk more these dataset later on.

II. BACKGROUND

As it is clear now, incremental learning is a key technique of Continual Learning in which we input data continuously and consistently to extend the knowledge of our model (i.e. to further train the model). It represents a vital aspect of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time or its size is out of system memory limits.

Unlike to humans though, computer tend to suffer from systematic forgetting. Let us provide with an example to make it more clear. Let us say that we enter in an office and trip over an obstacle that was there by accident. Next time that we will enter the same office(or probably enter on a similar one or we are about to experience a similar situation) we will probably take a careful look to our environment to check for the same obstacle. In contrast to that, learning procedure is perceived differently from machines. Even though humans will be careful for obstacles, in general, that may appear on their way, the respective machine generated model will be trained to avoid tripping on the specific obstacle. Trying to train it in avoiding possible obstacles with different shapes,looks and structures(or even being able to identify something as possible obstacle) may lead in forgetting about the initial obstacle that was trained to avoid. That is a main weakness in training a machine, the incapability in learning multiple tasks sequentially. And that is what Continual Learning is trying to approach as a problem and solve it.

A lot of research has been done on the so-called catastrophic forgetting of the machines. A first approach that will be discussed is ”An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks” written by Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, Yoshua Bengio [2]. The paper elaborates on the fact that when a machine is trained on a task and then trained on a second one, the resulted model forget how to perform the first task. An investigation is conducted on the extent to which the catastrophic forgetting problem occurs for modern neural networks, comparing both established and recent gradient-based training algorithms and activation functions. To begin with, this project discusses the regularization of the neural network and that is was done using the dropout training algorithm that generalizes performance [4]. What dropout algorithm actually does is to train in an extreme efficient way, multiple neural networks and averaging their predictions. Continuing to the training process, activation functions were used to configure the learning parameters. Given the problematic selection of hyper-parameters of deep learning methods due to humans’ bias in terms of familiarity with that specific method or conflict of interest, random hyper-parameter [1] was used to resolve the matter.

What this paper actually has shown is that dropout algorithm was able to perform best in performing on ”old task” and ”new task”, where ”old task” and ”new task” refers to the actual tasks that the machine was trained in different steps. Even though its ability in preventing forgetfulness of these tasks was not explained in whole, putting aside some controversies in the usage of activation functions [8], it is undeniable that the resulted outcome using dropout was proved as beneficial in keeping record of previous ”experiences”.

Another worth mentioned approach in curing machines tendency in forgetting was done by Kirkpatrick in the paper of ”Overcoming catastrophic

forgetting in neural networks" [5].

To have an ease introduction about it,we know that when a neural network system undergoes training for a specific task(let us call it task A), it it able to optimize a specific set of parameters to maintain its ability in solving similar tasks is the upcoming future. But when it is the case that a different task(let us call that task B) is inputted as training data, what happen is that the neural network optimizes its previous learning parameters is favor of Task B, hence forgetting its ability in performing task A effectively and efficiently.

A practical approach introduced in Kirkpatrick's paper [5] was by providing conjugated data at once (i.e. providing Task A and Task B simultaneously) in order to achieve an average optimization of the learning parameters that would lead in fair performance in both tasks. The way that was tested was by training a single agent using deep-learning techniques to play multiple Atari games ([6], [7]). Although that seems to solve a part of the problem, one could easily notice that it could only be possible just with a small amount of data.

However, their work presents an algorithm, named elastic weight consolidation (EWC), that appears to provide a distinctive solution in catastrophically learning. The way it works is similar to the way brain's synaptic consolidation does. Using a weighted value, it slows the training procedure down if it appears to be irrelevant in comparison to the previous taught task.

III. Methodology

Before starting with the methodology that will be used in this project, we will present the dataset that will be used.

A. MNIST DATASET

The MNIST(Modified National Institute of Standards and Technology) dataset is a huge database consisted of handwritten digits used in machine learning for training purposes in the area of image processing.

Fig. 2. Sample images from MNIST test dataset.

It has a training set of 60.000 examples, and a test set of 10.000 examples. It is a subset of a larger set available from NIST.

B. CIFAR10 DATASET

An additional dataset that will be used is the CIFAR10(Canadian Institute for Advanced Research) dataset. CIFAR10 dataset is a collection of images that are used in training machine learning and computer vision algorithms. It contains 60.000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6.000 images of each class.

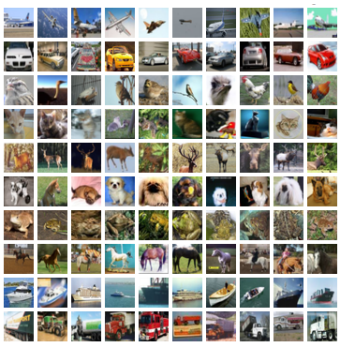

Fig. 3. Sample images from CIFAR10 dataset.

Due to its large amount of data just a subset will be used in our analysis.

C. KERAS AUTO-ENCODERS

Definition: 3.1: Auto-Encoders are artificial neural networks with the aim of learning how to represent a set of given data. The way it works is, given an original output, the encoder(data compression algorithm) demolishes the input by reducing its dimensions, creating a compressed representation

of it [3]. Finally, through a decompression function (decoder) it tries to represent itself as accurately as possible. The following figure represents a plain representation of an auto-encoder.

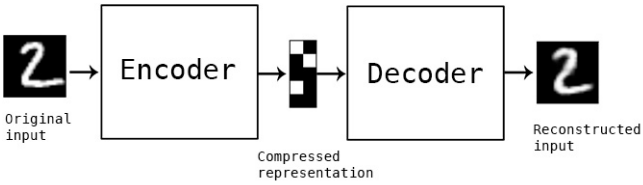

Fig. 4. Auto-Encoders.

D. ANALYSIS

Our collected data will be acquired through a fixed but random permutation of the pixels of both the MNIST and the CIFAR10 datasets, deriving three permuted tasks for each of the train and test set of each one, summing to a total of 12 different tasks. Both of the these two steps in collecting the permuted tasks play a significant role to the training of the neural network that follows. The randomness of the shuffling prevent us from introducing bias in our model where as the fixed permutation of the pixels is a way to avoid inputing unnecessary noise in our data. Just for visualization and demonstrating purposes, the data was first reshuffled randomly. The limitation of omitting the fixed way of permutation will possibly result in a relatively high error rate score.

In addition, we will instantiate n of keras/neural network auto-encoders and associate an instance of a classifier with each one, using scikit-learn. What this actually means is that a n amount of neural networks will be created and each one of these will be matched with a classifier. In that way, our neural networks will be ready in receiving data and be trained.

IV. EXPERIMENTS

Moving on to the experimental state, the approach will be quite straightforward. Within each task, the training of the neural network and the classifier will be defined with the usual and -if I may- old-fashion way. Given that the data will be already shuffled, we will process it by sending small batches into the neural networks and the associated classifiers. Training our neural networks and classifiers in that way, we are going to receive n different error scores. Secondly, out of these scores we will track the lowest one and pick the respective batch that this auto-encoder and classifier were trained. Finally, by doing so, we will already have picked our ”favorite” auto-encoder and classifier.

Having created all that setup, there is the further need of evaluation. The way it will be done is by taking advantage and engaging all the tasks with our ”favorite bucket” of neural network and auto-encoder in order for the setup to be evaluated.

The final step of the experiment will be to repeat the same procedure with one main difference. Instead of instantiating n encoders in the starting point, we will increase the amount to $n + 1$. The rationality behind of this step is to configure a setup that will prevent working with a fixed amount of auto-encoders from the starting point and create an auto-encoder ”online” if the observed error is too high.

Similar studies as that one of ”An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks” [2] showed that using the dropout training algorithm, the best obtained network, judging from the validation set score, had almost 57% more parameters than the best network trained without dropout. Regarding their approach using the MNIST dataset as well following a procedure of pixel permutation it was shown that dropout algorithm improved the optimal’s model performance.

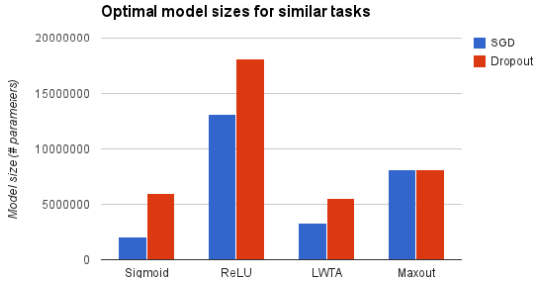

Fig. 5. Optimal model size with and without dropout on the input reformatting tasks.

Figure 5 shows the performance of the dropout

Another study, that of [5], which was actually based on the "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks" project, actually tried to represent the same results. Using dropout regularization with early stopping configuration, they tried to maximize its performance. The way it was implemented was by observing the test error variation on the validation test and in the case of 5 subsequent increases of the error rate, the procedure was terminated and the training of the next dataset was following after. The network's weights were, then, reseted to the lowest average validation error of all the previous datasets. A brief summary of their results is represented in the figure below

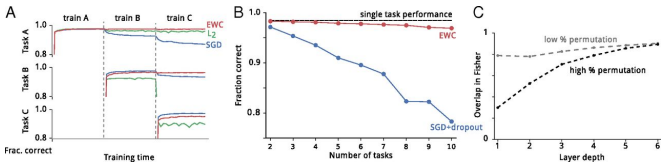

Fig. 6. Results on the permuted MNIST task

V. Discussion

Regarding the evaluation of the results, the loss error will be taken into account. The way it will be interpreted will be quite straightforward. Using the TensorFlow open-source library, the loss score will be evaluated. The lower the value of loss, the better the model.

The loss value is calculated using both the test and the validation set. Moreover, it measures how well the model is performing in regard to these sets. In other words, it is the sum of the errors that was made for every example individually, either in training or the validation set. Therefore, the prior objective will be to minimize that score as more as possible. The way it can be done is by trying different optimization approaches in the settings of the weight vector values.

Another variable that will contribute in gaining insights of the conducted experiments will be the "accuracy" variable. When models parameters are tuned and the training procedure is over, "accuracy" will calculate the percentage of the misclassification. The test samples will have already been inputted in the model and in comparison to the desirable output, the number of mistakes will be recored (value of one(1) stands for a mistake where as value of zero(0) means the exact opposite).

VI. Conclusion

To conclude, we will be able to gain a slight insight of how Continual Learning can be achieved in engaging neural networks and auto-encoders.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[2] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[3] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[4] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[5] 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. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[6] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[7] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[8] Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182, 2013.