2022 年臺灣國際科學展覽會
優勝作品專輯

作品編號	190040
參展科別	電腦科學與資訊工程
作品名稱	Enhancement of Online Stochastic Gradient Descent using Backward Queried Images
得獎獎項	四等獎

國　　家	South Korea
就讀學校	Shanghai American School
指導教師	
作者姓名	Gio Huh

關鍵詞　catastrophic interference、backward query、online learning

作者照片

Abstract Stochastic gradient descent (SGD) is one of the preferred online optimization algorithms. However, one of its major drawbacks is its predisposition to forgetting previous data when optimizing through a data stream, also known as catastrophic interference. In this project, we attempt to mitigate this drawback by proposing a new low-cost approach which incorporates backward queried images with SGD during online training. Under this new approach, we propose that for every new training sample through the data stream, the neural network is optimized using the corresponding backward queried image from the initial dataset. After compiling the accuracy of the proposed method and SGD under a data-stream of 50,000 training cases with 10,000 test cases and comparing our algorithm to SGD, we see substantial improvements in the performance of the neural network with two different MNIST datasets (Fashion and Kuzushiji), classifying the MNIST datasets at a high accuracy for the mean, minimum, lower quartile, median, and upper quartile, while maintaining lower standard deviation in performance, demonstrating that our proposed algorithm can be a potential alternative to online SGD.

1. Introduction

The size and quality of datasets are vital factors in the effective training of neural networks. Under ideal circumstances, datasets are elaborate and large enough to improve the generalization of the neural network. Realistically, however, obtaining such datasets may not be practical or feasible especially when the dataset may be given as a function of time or is too large to run at once.

Online machine learning is employed to overcome these impediments by sequentially utilizing the dataset. If a dataset is limited by time, the neural network is initially optimized with the original dataset and incrementally trained as more data become available. If a dataset is too large, a stream-based online learning algorithm is used [1,2]. In these cases, stochastic gradient descent (SGD) is normally chosen as the online machine learning algorithm because of its simplicity, increased update frequency, and low cost-per-literation [3,4]. This method, however, is susceptible to noisy gradient signals and variance, which diminish its convergence speed. Additionally, SGD gives priority to the most recent training cases, which reduces the value that previous training cases provide, and hence becomes prone to catastrophic forgetting [5]. A case in point is when the neural network is first optimized using a high-quality dataset and subsequently with a lower-quality data stream; the neural network will likely show suboptimal performance because the lower-quality training cases generate noise and cause the neural network to forget the old data. Due to these factors, SGD is not an ideal learning algorithm; there is a need for a more streamlined approach to online/stream-based machine learning.

Prior approaches have attempted to overcome the aforementioned limitations of SGD by improving SGD with rehearsal/pseudo-rehearsal methods [6,7,8,9] and regularization [10]. For example, one approach is locally weighted regression, a modeling technique centered around producing local models from subsets of the dataset and combining them into a single global model in order to reduce catastrophic forgetting. Another approach is rehearsal [6], which merges old data with newly received data for online learning. A variant of this is pseudo-rehearsal [7,8,9], which combines randomly generated data with the online machine learning process. The advantage of this approach is that the neural network can avoid saving old data. All of these approaches do show improvements in online machine learning; however, these approaches, by no means, are perfect, each having their limitations.

In this project, we suggest a low-cost learning algorithm that addresses the shortcomings of online SGD machine learning when the initial data are made accessible initially, and the remaining data are made available through a data stream. Specifically, we propose an adaptation of SGD that incorporates the backward queried samples of the initial data during the training of the remaining data. Through this algorithm, we aim to reduce catastrophic interference of online SGD while improving its ability to converge to minima. We benchmarked our learning algorithm using the Fashion-MNIST and Kuzushiji-MNIST datasets and then evaluated our results with those of the online SGD, using 10,000 unknown test cases.

2. Related Works

The two foremost methods we would like to mention are rehearsal-based: rehearsal and pseudo-rehearsal. Rehearsal is a method which saves old data for the online training of the neural network [6], and pseudo-rehearsal is a method which pairs the data stream with replicated training cases generated using the current neural network [7,8,9]. The aim of both of these methods is to improve the neural network's ability to retain previous data and its faculty to learn different concepts through a data stream. Recently, there has been success using rehearsal [11] and pseudo-rehearsal based methods for vision tasks [12,13]. Specific variants of rehearsal-based method includes Locally Weighted Regression Pseudo-Rehearsal (LW-PR^2), a version of LWR, which attempts to mitigate catastrophic interference and improve SGD by retaining information through local models and combining them into a single global model; and Deep Generative Replay, which uses generative adversarial networks (GAN) to replicate past data with desired outputs. Unfortunately, all of these approaches are susceptible to the common limitations of rehearsal and pseudo-rehearsal due to the fact that rehearsal methods increasingly have to store training data while pseudo-rehearsal is computationally complex, making both methods relatively difficult implement into varying problems. Moreover, a unique drawback of LWR, which is employed by LW-PR^2, is that it is computationally taxing and requires thousands of local models to be accurate, which also poses as a problem when the dataset is relatively large as the task becomes time intensive.

Another way that SGD has been attempted to be improved is through regularization, which reduces the overfitting and catastrophic forgetting of the neural network through parameters [10].

All of the previous works mentioned above can be classified into two categories: improving SGD in a parametric way (regularization) or by retaining information from the previous dataset (rehearsal-based). While it is difficult to assess which method is superior in online training, each has their own advantages and drawbacks. Since training cases provide more information than parameters, in some sense, rehearsal-based methods may be more beneficial. Having said that, saving old data or generating representative data can be computationally costly. In this project, we try to capitalize on the advantages of utilizing the previous dataset during online training while reducing the computational cost of saving such data: we propose a simple but effective approach of using representative data by using backward queried image per class and feeding this image with new received data during online learning.

3. Proposed Method

Our method comes in the form of combining regular SGD with backward queried data. In this section, we explain SGD, then backward query, and finally our proposed method in relation to the previous two concepts.

3.1 Stochastic Gradient Descent

SGD is preferred in online learning because of its characteristics of tuning the neural network for every single sample as in (1).

$$W_{t+1} = W_t - \eta \cdot \nabla_W \mathbb{J}\big(W; x^{(i)}; y^{(i)}\big) \quad (1)$$

where W denotes weights, η denotes learning rate, and $x^{(i)}$ denotes each training sample of label $y^{(i)}$. In this formula, we observe two specific characteristics of SGD: its simplicity and updating frequency. The single training case per iteration makes SGD relatively simple compared to other gradient descents, such as mini-batch or batch gradient descent (GD), which averages out the gradients of different training cases within a batch or whole dataset. Similarly, SGD possesses a relatively fast updating frequency since it optimizes the neural network for every training case, instead of batches.

These characteristics make SGD distinct from other gradient descent. It is also the source behind SGD's drawbacks. Since SGD is updated for every training case, training cases could generate noise within the neural network, especially when the quality of training cases is poor. Moreover, as SGD frequently updates with new training cases, it is likely to forget the changes made by previous training cases (catastrophic interference).

3.2 Backward Query

In a normal feedforward neural network, values are first given into the input/initial layer. Then, it is multiplied with weights. The manipulated values are thereafter combined and inputted into the activation function of their respective nodes. The output of the activation function forms the input of the next layer. This pattern recurs until it reaches the output layer, where it is classified using a criteria.

To generate backward queried samples, this process of calculating values in a feedforward neural network is reversed. Instead of initially computing from the input layer, labels that represent a specific class are inputted into the output layer. It is then computed with the inverse of the activation function and multiplied with the weights to obtain new values corresponding with the previous layer. This process repeats until the values have backwardly propagated to the input layer. The final value becomes the backward queried sample of the aforementioned class.

The significance behind backward query is that the backward queried samples, in essence, becomes the representative data of the class and provides insight into the neural network because the values provide an imprint of how the weights and activate function save and compute the input value internally. This is advantageous for online training since we can capture the value of the entire dataset into a single case, which can be later used to retain previous information.

To derive an image of backward query, we train a neural network using 10,000 random samples out of the 60,000 training cases within the Fashion-MNIST dataset. Specifically, we train a multilayer perceptron with layers, [784, 256, 128, 100, 10], and learning rate, 0.01, using mini- batch gradient descent of batch size 32 for 100 epochs to find the backward queried sample. In [Fig. 1], the 10 images represent the backward queried sample of each class in the Fashion-MNIST dataset.

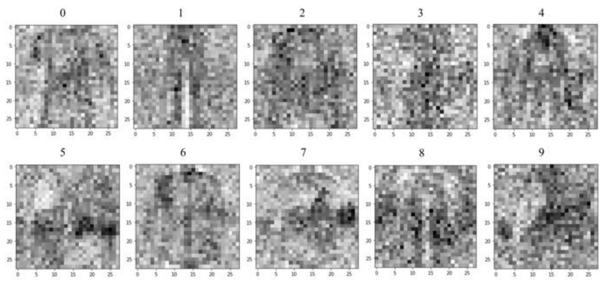
3.3 Our Method

In our approach, backward query was incorporated with SGD. We paired every training case with their corresponding backward queried data from the initial dataset. We then averaged the gradient of the two samples before updating the network. The proposed approach can be modeled using the formula in (2).

$$W_{t+1} = W_t - \eta \cdot \frac{1}{2}\left(\nabla_W \mathbb{J}\left(W; x^{(i)}; y^{(i)}\right) + \nabla_W \mathbb{J}\left(W; b^{(y)}; y^{(i)}\right)\right)$$
(2)

where W denotes weights, η denotes learning rate, $x^{(i)}$ denotes each training sample of label $y^{(i)}$, and $b^{(y)}$ denotes the backward queried data of label y.

The advantage of our method is that it is capable of mitigating catastrophic forgetting similar to rehearsal/pseudo-rehearsal methods while remaining more computationally effective than retaining old data or generating data through complex modeling.



[Fig. 1] Backward queried sample of Fashion-MNIST dataset.

4. Experiment

In evaluating the performance of SGD and our method, we establish our experimental settings and compare three gradient descents: GD, mini-batch, and SGD. Then, from these results, we determine mini-batch as the optimization algorithm for the initial dataset from the perspective of performance. Finally, we record the accuracy of SGD and the proposed method from 60,000 training samples, which are divided into the initial training set and streaming data. To measure the performance, 10,000 unknown samples are used.

4.1 Experimental Settings

I. Training & Test Data: We use two datasets, Fashion-MNIST and Kuzushiji-MNIST, both of which consists of 60,000 training cases and 10,000 test cases of 28 by 28 gray-scale pixel maps with 10 classes.

II. Neural Network Architecture: We implemented a multilayer perceptron (MLP) with Python and Numpy. The layers of the MLP are [784, 256, 128, 100, 10] where 784 and 10 are input and output layer respectively. We use sigmoid as the activation function, 0.01 as the learning rate, and a quadratic loss function as the loss function.

III. Initial Training Dataset and Data Stream: To emulate an online environment, the first 10,000 training cases is allocated to the initial dataset while the final 50,000 training cases is allocated to the subsequent data stream. To measure the performance, a separate 10,000 unknown test samples are used.

4.2 Implementation of Multilayer Perceptron and Proposed Method

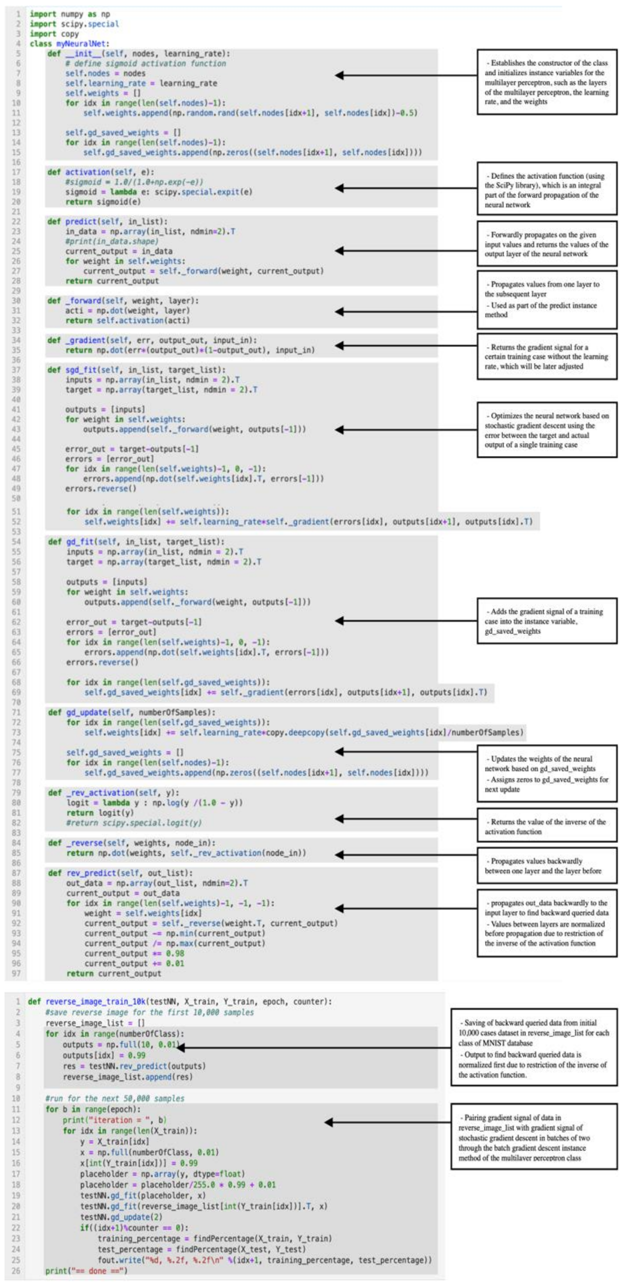
The implementations of both the multilayer perceptron class and the proposed method can be seen in [Fig. 2].

4.3 Comparison among GD, SGD, and Mini-batch

After implementing the multilayer perceptron class, we compared batch gradient descent, SGD, and mini-batch gradient descent using the total (60,000 training cases and 10,000 test cases) Fashion-MNIST and Kuzushiji-MNIST dataset for 500 epochs and recorded the model's accuracy for every 10 epochs.

In evaluating [Fig. 3] and [Fig. 4], there are important observations we can consider in choosing our gradient descent for the initial dataset. First off, it is evident that batch gradient descent shows suboptimal performance for both Kuzushiji-MNIST and Fashion-MNIST for every recorded epoch in comparison to SGD and mini-batch. Next, comparing SGD and mini-batch for the Kuzushiji-MNIST, we see that the two gradient descents show similar performance; however, over time, we see that mini-batch surpasses SGD in performance. For Fashion-MNIST, we can see that mini-batch is superior to SGD since mini-batch consistently improves over each epoch while SGD seems to fluctuate due to noise.

Through this comparison, we can see that although SGD is advantageous for online learning due to its low-cost-per-iteration, it is easily affected by noise and shows poor convergence, which results from catastrophic interference. In contrast, the mini-batch shows less fluctuation due to the fact that the batch samples dampen noise from the data and alleviate catastrophic forgetting, making mini-batch the suitable optimization algorithm.



```python
import numpy as np
import scipy.special
import copy
class myNeuralNet:
    def __init__(self, nodes, learning_rate):
        # define sigmoid activation function
        self.nodes = nodes
        self.learning_rate = learning_rate
        self.weights = []
        for idx in range(len(self.nodes)-1):
            self.weights.append(np.random.rand(self.nodes[idx+1], self.nodes[idx])-0.5)

        self.gd_saved_weights = []
        for idx in range(len(self.nodes)-1):
            self.gd_saved_weights.append(np.zeros((self.nodes[idx+1], self.nodes[idx])))

    def activation(self, e):
        #sigmoid = 1.0/(1.0+np.exp(-e))
        sigmoid = lambda e: scipy.special.expit(e)
        return sigmoid(e)

    def predict(self, in_list):
        in_data = np.array(in_list, ndmin=2).T
        #print(in_data.shape)
        current_output = in_data
        for weight in self.weights:
            current_output = self._forward(weight, current_output)
        return current_output

    def _forward(self, weight, layer):
        acti = np.dot(weight, layer)
        return self.activation(acti)

    def _gradient(self, err, output_out, input_in):
        return np.dot(err*(output_out)*(1-output_out), input_in)

    def sgd_fit(self, in_list, target_list):
        inputs = np.array(in_list, ndmin = 2).T
        target = np.array(target_list, ndmin = 2).T

        outputs = [inputs]
        for weight in self.weights:
            outputs.append(self._forward(weight, outputs[-1]))

        error_out = target-outputs[-1]
        errors = [error_out]
        for idx in range(len(self.weights)-1, 0, -1):
            errors.append(np.dot(self.weights[idx].T, errors[-1]))
        errors.reverse()

        for idx in range(len(self.weights)):
            self.weights[idx] += self.learning_rate*self._gradient(errors[idx], outputs[idx+1], outputs[idx].T)

    def gd_fit(self, in_list, target_list):
        inputs = np.array(in_list, ndmin = 2).T
        target = np.array(target_list, ndmin = 2).T

        outputs = [inputs]
        for weight in self.weights:
            outputs.append(self._forward(weight, outputs[-1]))

        error_out = target-outputs[-1]
        errors = [error_out]
        for idx in range(len(self.weights)-1, 0, -1):
            errors.append(np.dot(self.weights[idx].T, errors[-1]))
        errors.reverse()

        for idx in range(len(self.gd_saved_weights)):
            self.gd_saved_weights[idx] += self._gradient(errors[idx], outputs[idx+1], outputs[idx].T)

    def gd_update(self, numberOfSamples):
        for idx in range(len(self.gd_saved_weights)):
            self.weights[idx] += self.learning_rate*copy.deepcopy(self.gd_saved_weights[idx]/numberOfSamples)

        self.gd_saved_weights = []
        for idx in range(len(self.nodes)-1):
            self.gd_saved_weights.append(np.zeros((self.nodes[idx+1], self.nodes[idx])))

    def _rev_activation(self, y):
        logit = lambda y : np.log(y /(1.0 - y))
        return logit(y)
        #return scipy.special.logit(y)

    def _reverse(self, weights, node_in):
        return np.dot(weights, self._rev_activation(node_in))

    def rev_predict(self, out_list):
        out_data = np.array(out_list, ndmin=2).T
        current_output = out_data
        for idx in range(len(self.weights)-1, -1, -1):
            weight = self.weights[idx]
            current_output = self._reverse(weight.T, current_output)
            current_output -= np.min(current_output)
            current_output /= np.max(current_output)
            current_output *= 0.98
            current_output += 0.01
        return current_output
```

- Establishes the constructor of the class and initializes instance variables for the multilayer perceptron, such as the layers of the multilayer perceptron, the learning rate, and the weights

- Defines the activation function (using the SciPy library), which is an integral part of the forward propagation of the neural network

- Forwardly propagates on the given input values and returns the values of the output layer of the neural network

- Propagates values from one layer to the subsequent layer
- Used as part of the predict instance method

- Returns the gradient signal for a certain training case without the learning rate, which will be later adjusted

- Optimizes the neural network based on stochastic gradient descent using the error between the target and actual output of a single training case

- Adds the gradient signal of a training case into the instance variable, gd_saved_weights

- Updates the weights of the neural network based on gd_saved_weights
- Assigns zeros to gd_saved_weights for next update

- Returns the value of the inverse of the activation function

- Propagates values backwardly between one layer and the layer before

- propagates out_data backwardly to the input layer to find backward queried data
- Values between layers are normalized before propagation due to restriction of the inverse of the activation function

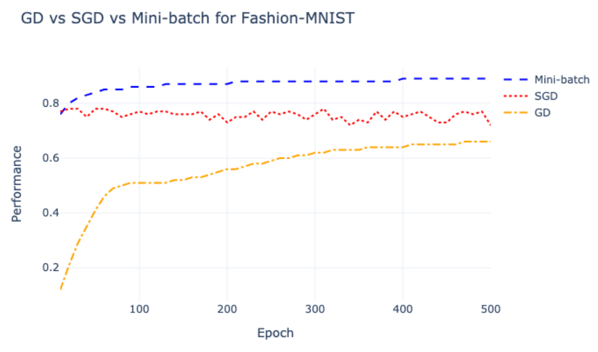
```python
def reverse_image_train_10k(testNN, X_train, Y_train, epoch, counter):
    #save reverse image for the first 10,000 samples
    reverse_image_list = []
    for idx in range(numberOfClass):
        outputs = np.full(10, 0.01)
        outputs[idx] = 0.99
        res = testNN.rev_predict(outputs)
        reverse_image_list.append(res)

    #run for the next 50,000 samples
    for b in range(epoch):
        print("iteration = ", b)
        for idx in range(len(X_train)):
            y = X_train[idx]
            x = np.full(numberOfClass, 0.01)
            x[int(Y_train[idx])] = 0.99
            placeholder = np.array(y, dtype=float)
            placeholder = placeholder/255.0 * 0.99 + 0.01
            testNN.gd_fit(placeholder, x)
            testNN.gd_fit(reverse_image_list[int(Y_train[idx])].T, x)
            testNN.gd_update(2)
            if((idx+1)%counter == 0):
                training_percentage = findPercentage(X_train, Y_train)
                test_percentage = findPercentage(X_test, Y_test)
                fout.write("%d, %.2f, %.2f\n" %(idx+1, training_percentage, test_percentage))
    print("== done ==")
```

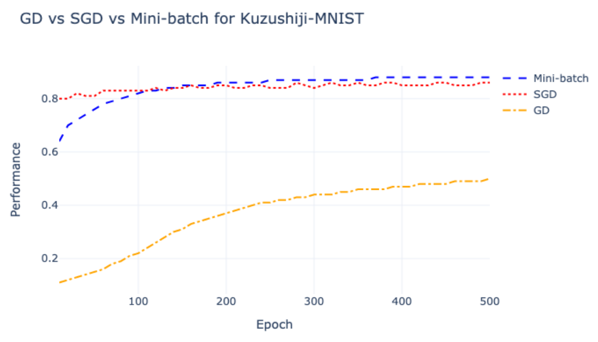
- Saving of backward queried data from initial 10,000 cases dataset in reverse_image_list for each class of MNIST database
- Output to find backward queried data is normalized first due to restriction of the inverse of the activation function.

- Pairing gradient signal of data in reverse_image_list with gradient signal of stochastic gradient descent in batches of two through the batch gradient descent instance method of the multilayer perceptron class

[Fig. 2] Implementation of Multilayer Perceptron and Proposed Method



[Fig. 3] Performance of GD, SGD, and Mini-batch for Fashion-MNIST over 500 epochs



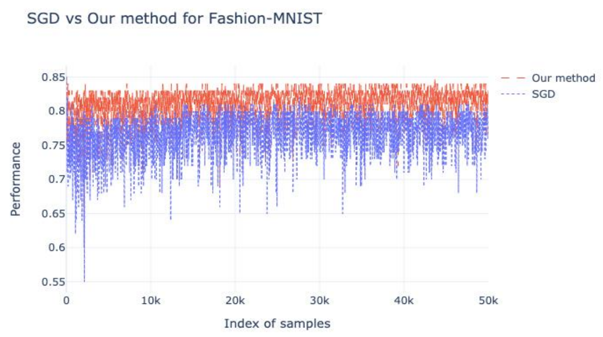
[Fig. 4] Performance of GD, SGD, and Mini-batch for Kuzushiji-MNIST over 500 epochs

4.4 Experiment with SGD and the Proposed Method

As known, mini-batch is the most suitable choice when the entire dataset is given, so we used it as the optimization algorithm for the initial 10,000 samples, which will be optimized for 100 epochs after which mini-batch approaches saturation. After the initial training, we used the remaining 50,000 samples as the stream data and updated the model using SGD or the proposed method.

During online training of the data stream, we measure the neural network's performance every 10 new samples. [Fig. 5] and [Table 1] below show the results of SGD and our method for Fashion-MNIST while [Fig. 6] and [Table 2] below present the results for Kuzushiji-MNIST. Table 1 and 2 display the statistics of SGD and the proposed method.

From the graphs, box plots, and tables that have been extracted, it is perceivable that our proposed method performs to a greater standard than SGD.

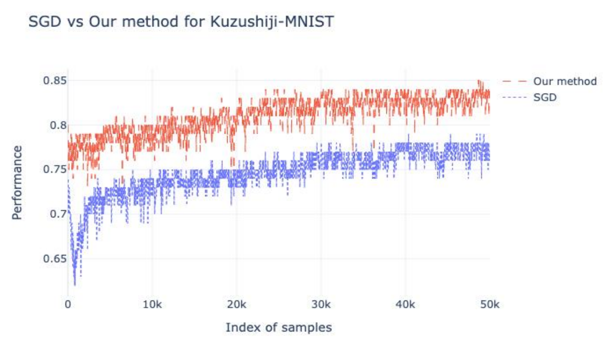


[Fig. 5] Performance of the proposed method and SGD over a data stream derived from Fashion-MNIST

[Table 1] Comparison of SGD and Proposed Method with Fashion-MNIST

Statistic	Iterative Method	
	SGD	Proposed Method
Mean	0.77	0.81
Standard	0.026	0.019

Statistic	Iterative Method	
	SGD	Proposed Method
Deviation		
Minimum	0.55	0.66
Lower Quartile	0.76	0.81
Median	0.77	0.82
Upper Quartile	0.79	0.83
Maximum	0.85	0.85



[Fig. 6] Performance of the proposed method and SGD over a data stream derived from Kuzushiji-MNIST

[Table 2] Comparison of SGD and Proposed Method with Kuzushiji-MNIST

Statistic	Iterative Method	
	SGD	*Proposed Method*
Mean	0.75	0.81
Standard Deviation	0.024	0.020
Minimum	0.62	0.73
Lower Quartile	0.73	0.80
Median	0.75	0.82
Upper Quartile	0.76	0.83
Maximum	0.79	0.85

First, assessing the graphs, our method was able to retain or improve the initial performance of the neural network at a greater rate than SGD. In the figure of Fashion-MNIST, when the performance of SGD dropped significantly after the first 5,000 samples, our method was able to stay relatively close to the original performance of the neural network. We can also see a similar conclusion in the figure of Kuzushiji-MNIST as SGD saw a significant decline in performance in the first few thousand samples while our method was generally able to improve on the previous neural network's accuracy.

The tables also display a similar verdict. In [Table 1] and [Table 2], we also precisely see that the proposed method performs more favorably than SGD in most aspects: our method had a higher accuracy than SGD for the mean, minimum, lower quartile, median, and upper quartile, while maintaining lower standard deviation, or fluctuation.

In general, our method was more effective at improving the generalization of the neural network, reducing the loss in performance from the data-stream, and diminishing the variance in the performance of the neural network from catastrophic interference.

5. Conclusion

In this project, we proposed an online learning algorithm using backward queried data, and, from our results, we were able to validate the significance of our method for the application of online training. This is because we saw a reduction in the standard deviation of SGD and an improvement in its performance for both Fashion-MNIST and Kuzushiji-MNIST. Even more, the results demonstrate that our method is a viable option for real world applications of online machine learning due to the fact that we were able to enhance the ability of the neural network with relatively low cost and easy implementation.

Still, it should be cautioned that our approach was exclusively benchmarked with the task of image classification using two datasets with fixed variables, such as the batch-size or the point during online training at which the backward-queried samples are derived. It is, hence, vital to continue investigating the effects of backward query on online training with different tasks, datasets, as well as different variables of backward query. Nevertheless, our project provides some optimistic insights into the potential of backward query in online machine learning.

References

[1] N. Cesa-Bianchi, P.M. Long, and M. Warmuth, Worst-case quadratic loss bounds for prediction using linear functions and gradient descent. *IEEE Transactions on Neural Networks*, 7(3):604-619, 1996.

[2] J. Kivinen and M.K. Warmuth, Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1-63, 1997.

[3] L. Bottou and Y. LeCun. Large scale online learning. In S. Thrun, L. Saul, and B. Scho ̈lkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[4] T. Zhang. Solving large scale linear prediction

problems using stochastic gradient descent algorithms. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, Banff, Alberta, Canada, 2004.

[5] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

[6] R. Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[7] A. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pages 65–68. IEEE, 1993.

[8] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Journal of Neural Comput- ing, Artificial Intelligence and Cognitive Research*, 7:123–146, 1995.

[9] A. Robins. Sequential learning in neural networks: A review and a discussion of pseudore- hearsal based methods. *Intelligent Data Analysis*, 8(3):301–322, 2004.

[10] R. Tibshirani, Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, B.*, 58(1):267–288, 1996.

[11] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5533–5542. IEEE, 2017.

[12] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[13] C. Atkinson, B. McCane, L. Szymanski, and A. V. Robins. Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting. *CoRR*, abs/1812.02464, 2018.

【評語】190040

This project uses backward queried images with SGD to enhance the online stochastic gradient descent method. Two experiments were conducted using Fashion-MINIST and Kuzushiji-MINIST datasets, and the results were presented and discussed. Overall, this work is technically sound and solid. It would be great if the proposed work could be applied to some real problems to further strengthen the impacts of this research.