sciendo

JAISCR, 2020, Vol. 10, No. 1, pp. 15 – 26
10.2478/jaiscr-2020-0002

ON TRAINING DEEP NEURAL NETWORKS USING A STREAMING APPROACH

Piotr Duda[1,*], Maciej Jaworski[1], Andrzej Cader[2], Lipo Wang[3]

[1]*Department of Computer Engineering, Częstochowa University of Technology, Częstochowa, Poland*

[2]*Clark University, Worcester, USA and Information Technology Institute, University of Social Sciences, Łódź, Poland*

[3]*School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore*

[*]*E-mail: piotr.duda@iisi.pcz.pl*

Submitted: 12th September 2019; Accepted: 18th November 2019

Abstract

In recent years, many deep learning methods, allowed for a significant improvement of systems based on artificial intelligence methods. Their effectiveness results from an ability to analyze large labeled datasets. The price for such high accuracy is the long training time, necessary to process such large amounts of data. On the other hand, along with the increase in the number of collected data, the field of data stream analysis was developed. It enables to process data immediately, with no need to store them. In this work, we decided to take advantage of the benefits of data streaming in order to accelerate the training of deep neural networks. The work includes an analysis of two approaches to network learning, presented on the background of traditional stochastic and batch-based methods.

Keywords: deep learning; data streams; convolutional neural networks

1 Introduction

Deep learning has become very popular both as a field of study for researchers [8, 24, 33] and as a practical tool in many applications [12, 36]. Different deep structures allow handling many real-world tasks, with effectiveness unimaginable just a few years ago. Among the others, convolutional neural networks allow processing big sets of images, recurrent neural networks are a great tool for Natural Language Processing, and deep belief networks are valuable in reconstructing the data.

The development of deep learning methods is strongly related to the increasing number of stored and labeled data. Despite the increase in computing speed, some deep neural networks still need a couple of days to obtain their best performance.

The training of a neural network can be split into two stages. The first one is a forward pass, in which data inputted into a network are processed by the subsequent layers until the final layer points out the predicted class. The backward pass allows changing the weights, associated with neuron pairs of neighboring layers. Depending on the amount of data used for weights updating in each learning step, two main types of training procedures can be distinguished: stochastic and batch-based. In the stochastic one, the error of prediction is backpropagated after each single data element is passed forward. In the batch-based approach, the weights are updated after processing the whole training set. Re-

cently, the most common is the intermediate strategy in which the neural network is learned using mini-batches of data. This approach inherits benefits from both stochastic and batch-based procedures. The training set is partitioned into several smaller subsets and the weights are updated after processing each subset.

The use of an appropriate learning algorithm for a multilayer neural network ensures that after each epoch the accuracy calculated on the training set will not decrease. On the other hand, all data from the learning set are processed in each epoch, also those that do not change weights in a significant way. Consequently, the network wastes resources on analyzing data that will not improve its effectiveness.

In light of the above premises, we decided to propose algorithms that process the training dataset in a significantly different manner than it is standardly done. Instead of repeatedly processing the entire training set, it is treated as a statistical population. From this population, learning samples are taken randomly in a continuous manner creating a specific stream of data. To prevent the frequent choices of data that do not affect a network's weight modification, additional value is assigned to each data element. This value determines the probability with which the data element is drawn to the stream. To process the data stream prepared in such a way, one of the stream data mining algorithms (SDM) can be applied. The well-designed SDM algorithm has to fulfill the following criteria:

- the dataset cannot be stored and considered as a whole. The data elements should be analyzed as fast as possible, and the memory should be released for newly incoming data,
- the processing time should be as limited as possible, as the number of the new coming data can be arbitrarily fast,
- the distribution of the data can change during the processing of the stream. Such an event is called a concept-drift and it can occur in every moment. The algorithm should be able to adjust to a time-varying environment.

The fulfillment of the above three criteria can also be beneficial for algorithms that allow fast learning of neural networks. While the first two (minimizing memory and processing time) do not require a broader explanation, it is worth paying attention to the third criterion, i.e. the adaptability of the algorithm to a time-varying environment.

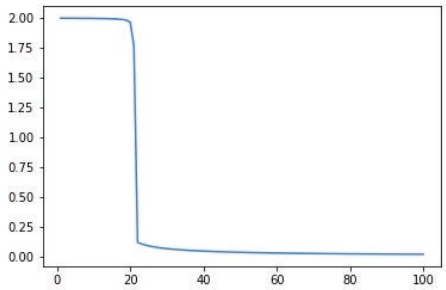


Figure 1. An example of the loss function values obtained for subsequent epochs

The values of the loss function presented in Figure 1 demonstrate an exemplary case of training a neural network, however, it vividly visualizes the stages in which most neural networks are trained. One can see that at the beginning stage of training the network the loss function has obtained similar, high, values. After about 20 epochs, the performance of the neural network has critically changed and the values of the loss function suddenly decreased. At this point, key data for neural network learning may have lost their relevance, while previously less important data could have become more meaningful. The moment of such a change can be indicated by an algorithm called the drift detector (DD). The DDs are a family of SDM algorithms whose major task is to indicate when the changes in an environment (concept-drift) have occurred. Combining DD with a learning algorithm can reduce the number of epochs required to obtain high accuracy.

The rest of the paper is organized as follows. In Section 2 the related works about deep learning methods and drift detectors are presented. The proposed modifications of the training phase of neural networks are described in Section 3. The results of numerical experiments performed on convolutional neural networks are presented in Section 4. Finally, Section 5 presents conclusions and planned future works.

2 Related works

In this chapter, we recall the most significant and the most recent papers about deep learning and drift detectors.

In recent years, the attention of machine learning researchers focused mainly on the deep neural networks [4, 19]. They are designed to solve hard AI problems, like image or speech recognition. In the literature one can find a variety of deep neural network structures and architectures. Each of them is designed to solve a specific task. For example, Convolutional Neural Networks and their derivatives are successfully used in image processing tasks. They usually consists of stacks of convolutional and sub-sampling layers, often followed by several dense layers. Some important example CNNs are LeNet [34], AlexNet [32] or VGG Net [40]. More recent architectures are, for example, DenseNet [25], GoogLeNet with Inception units [41], and Residual Networks [20] which contains connection between neurons from non-neighbouring layers. The mentioned CNN structures provide state-of-the-art performance on different benchmarks for object recognition tasks.

Recurrent Neural Networks are another type of neural network architecture. It allows processing sequences of data over time. The main problem in learning RNNs is the vanishing (or exploding) gradient [23]. The solution of the exploding gradient is to clip the value to some threshold if the gradients become too large. To solve the vanishing gradient problem, new neural units have been proposed. The first one is the Long Short-Term Memory (LSTM) [16], which contains gates responsible for modifying the value in the hidden neuron. Another type is the Gated Recurrent Unit (GRU) [9]. They are more simple than LSTMs and require less computational effort. Therefore, they are at this moment the most popular recurrent neurons used in deep learning applications.

The architectures presented above are mainly used for supervised learning tasks. However, there are also structures applicable for unsupervised learning, for example autoencoders [5], which learn how to reconstruct original data. There exists a variety of autoencoder types in the literature. The most popular are denoising autoencoders [42], in which the noise is introduced to original data during learning. Enforcing sparsity in the encoder layer leads to sparse autoencoders. In this type of autoencoder the size of the encoder layer can be higher than the input dimensionality. Another interesting type is the variational auto-encoder [30]. The deep neural structures for unsupervised learning tasks can be also formed based on the Restricted Boltzmann Machines [22] (RBM). The RBMs are two-layered networks which can learn good models of data distributions. They can be formed into stacks by connecting the visible layer of one RBM to the hidden layer of the previous one. An example of such a structure is the Deep Belief Networks (DBN) [21].

It should be also noted that several authors tried to merge the fields of deep learning and data stream mining. In [7] the authors combined the evolving deep neural network with the Least Squares Support Vector Machine. Deep neural networks were also successfully applied in semi-supervised learning task in the context of streamming data. In [26] the Deep Hybrid Boltzmann Machines and Denoising Autoencoders were proposed. It was demonstrated how such structures can used for online learning from data streams. In [39] the idea was to train the Deep Belief Network in unsupervised manner based on the unlabeled data from the stream. Then, few available labeled elements were used to occasionally fine-tune the model to the current data concept. In [28] and [27] the Authors proposed to apply the RBM as a concept drift detector. It was demonstrated that the properly learned RBM can be use to monitor possible changes in underlying data distribution. These method was further analyzed from the resource-awareness perspective in [29].

Another area that has recently attracted the attention of researchers is data stream analysis. Among the others, the methods to indicate the changes in an environment have been strongly investigated. The Drift Detection Method [15], the DDM algorithm, monitors the correctness of classification by the current model. Treating observations as a result of Bernoulli trials, the authors propose a statistical test to inform about warning or alarm state. The idea was improved in [3] as the EDDM algorithm. The Dynamic Streaming Random Forest (DSRF) was proposed in [2]. In this approach, after the initial phase of the subsequently generating a finite number of trees, the algorithm update statistics

defining thresholds for decision trees construction. Then the algorithm update forest with fixed percent of the trees. An algorithm measures entropy of incoming data as a drift detector. If the drift is detected, all the parameters of the algorithm are reset to initial values, and the algorithm replaces a specific number of trees in the forest, which number depends on a value of measured entropy. Its idea are extended in a paper [1]. In paper [18] the author propose the Adaptive Random Forests algorithm, which combine classical random forest procesure with Hoeffding's decision trees [14]. To react on changes in data stream a procedure based on ADWIN algorithm [6] and Page-Hinkley test [38] can be applied. In [10], the authors proposed WSTD algorithm, which applied Wilcoxon rank sum statistical test to improve false positive detection. In [11], the authors proposed to computing multiple model explanations over time and observing the magnitudes of their changes. The application of unsupervised methods, motivated by the statistical learning theory, are investigated in [37].

For more recent information about streaming algorithms and drift detectors the reader is referred to [13, 17] and [31].

3 Training techniques

In this Section, the mathematical formalism related to the presented methods will be introduced. Next, in the following subsections, the proposed methods to learn neural networks will be described.

Let T be a training set and N be a number of elements in it. Originally this set consists of d-dimensional feature vectors X_i and labels c_i, for $i = 1,\ldots,N$

$$T = \{(X_i, c_i) | i = 1,\ldots,N, X_i \in \mathbf{A}, c_i \in \mathbf{L}\}, \quad (1)$$

where $\mathbf{A}$ is a d-dimensional feature space, and $\mathbf{L}$ is a finite set of labels. Now, let us add to these elements additional values, specifying the probability of drawing (*pod*) each element to the stream

$$T^S = \{((X_i, c_i), v_i) | (X_i, c_i) \in T, v_i \in (0,1)\}. \quad (2)$$

Now, based on the set T^S one can define a stream S_t consisting of t data elements as follows

$$S_t = (Y_1,\ldots,Y_t | Y_i = (X_{j_i}, c_{j_i}), 1 \le i \le t, 1 \le j_i \le N). \quad (3)$$

It is worth noticing that $Y_i \in S_t$, for $i = 1,2,...$, form a sequence of independent random variables.

Now, let us consider a neural network $f : \mathbf{A} \to \mathbf{L}$. Without loss of generality, we can assume that f is an l-layered neural network

$$f(X) = \phi_l \circ \phi_{l-1} \circ \cdots \circ \phi_1(X), \quad (4)$$

where X is the input vector. A single layer $\phi_j : \mathbf{Z_j} \to \mathbf{Z_{j+1}}$, where $\mathbf{Z_j}$ is an N_j dimensional space of $(j-1)$-th layer output values, $\mathbf{Z_0} = \mathbf{A}$, and $\mathbf{Z_l} = \mathbf{L}$, can be defined as follows

$$\begin{aligned} \phi_j(z) = \quad & [\rho_j^1(\textstyle\sum_{i=1}^{N_j} w_{i,1} z_i + b_1), \ldots, \\ & \rho_j^{N_{j+1}}(\textstyle\sum_{i=1}^{N_j} w_{i,N_{j+1}} z_i + b_{N_{j+1}})] \end{aligned} \quad (5)$$

where $z = [z_1,\ldots,z_{N_j}] \in \mathbf{Z_j}$, $w_{i,m}$ is a weight between the i-th neuron of the $(j-1)$-th layer and the m-th neuron of the j-th layer, ρ_j^m is an activation function for the m-th neuron on the j-th layer and b_m is the bias for the m-th vector.

During the backpropagation phase, the weights of each neuron on every layer are updated according to the following formula.

$$w_{i,m} := w_{i,m} - \eta \frac{\partial E}{\partial w_{i,m}}, \quad (6)$$

where $\eta > 0$ is the learning rate and E is a loss function. The procedure of computing the gradient of the loss function depends on the applied training strategy. In a stochastic approach, the gradient is computed based on a single observation. In a batch-based approach, the gradient is taken as an average over all the elements in the batch.

3.1 Boosting based training approach

In the proposed method we apply the idea of boosting and we use the strategy to train the neural network based on mini-batches. The mini-batch B_τ consists of n elements recently taken from the stream

$$B_\tau = (Y_{t-n},\ldots,Y_t), \quad (7)$$

where $\tau = 1,2,\ldots,$ is an index of the following mini-batches.

The training of neural networks proceeds in a standard way. Every data separately is processed

in a forward pass (after which the value of loss function $L(X_i)$ is computed) and in a backward pass (which is used to compute the gradients of weights). After computing errors and gradients for the last layer, the probabilities of drawing elements from the training set to the stream are updated. We consider three approaches.

The Only Wrongly Classified (OWC) approach

In this approach, only those data elements that have been misclassified have their *pod* values changed. During the forward phase, the predicted class is established. If it differs from the expected one, a new weight v_i' is set to fixed number

$$v_i' = \begin{cases} \lambda, & f(X_i) \neq c_i \\ v_i, & \text{otherwise.} \end{cases} \tag{8}$$

An exemplary value of λ can be $\lambda = 1/n$.

Loss Based (LB) approach

In this approach, every data element from a mini-batch changes its weight v_i' according to the value of loss function $L(X_i)$. It is expected that correctly classified elements obtain lower values of loss function than the misclassified ones. In consequence, the correctly classified elements will have lower weights and will be rarely chosen to the mini-batches in the next steps

$$v_i' = L(X_i)/M_i, \tag{9}$$

where M_i indicates the number of times the $-i$-th data element was drawn in the past. The denominator defined in such a way prevents from drawing the same element repeatedly, which can be problematic if the network cannot correctly classify some elements.

Normalized Loss Based (NLB) approach

Recently, one of the most commonly used activation functions is the *ReLU*. One of the consequences of its application is the fact some loss functions, like the mean squared error, can obtain arbitrarily high values. This may upset the proportion of *pod* values. For this reason, normalization of the loss values can be beneficial. We proposed to apply the hyperbolic tangent which transforms big values of loss function close to 1, and the small ones close to 0.

$$v_i' = tanh(L(X_i))/M_i, \tag{10}$$

where the meaning of M_i is the same as in (9).

It should be noted that in none of the above-mentioned approaches v_i' values represent the probability mass since function since it is not guaranteed that they sum up to 1. To ensure this property the *pod*s should be normalized after processing the whole mini-batch in the following way

$$v_i = \begin{cases} v_i'/Z, & \text{for } x_i \in B \\ v_i/Z, & \text{for } x_i \in T \backslash B \end{cases} \tag{11}$$

where Z is a normalization factor, given as

$$Z = \sum_{\{v_i' | X_i \in T\}} v_i'. \tag{12}$$

After processing one mini-batch of data, another one is gathered and the procedure is repeated until the stopping condition is fulfilled.

The pseudocode of the proposed procedure called the Boosting Based Training Algorithm (BBTA) is presented in Algorithm 1 for the OWC approach and in Algorithm 2 for the LB and the NLB approaches.

```
Input: S - data stream, M - batch size, λ
1 Collect a new batch B from the stream S;
2 for every data element in B do
3     Train the network on current element
        Calculate a predicted class;
4     if predicted class == expected class then
5         Update v_i according to (8);
6 for every data element in T do
7     Update pods according to (11)
8 Return to line 1;
```

Algorithm 1. The BBTA - OWC algorithm

```
Input: S - data stream, M - batch size
1 Collect a new batch B from the stream S;
2 for every data element in B do
3     Increase counter of drawn of the current
        element Train the network on current
        element Compute loss function for a
        current element;
4     Update v_i according to (9) or (10)
5 for every data element in T do
6     Update pods according to (11)
7 Return to line 1;
```

Algorithm 2. The BBTA - LB/NLB algorithm

3.2 Bagging based training with drift detector

The algorithms proposed in the previous Subsection has two strong drawbacks. First, as new values of *pod*s are recalculated in a single step only for part of data, some problems can occur. After processing every mini-batch, *pod*s of every unused data become depreciated. In consequence, a lot of time may pass until those data elements are sampled again. Secondly, if the drawn data have huge loss function values and they are still misclassified after being processed multiple times, then they can stop the learning process, especially in the case of the LB and NLB approaches. The data would have huge *pod* values and the subsequent mini-batch generated from the stream would be composed of the same data. In such a case, the network would be adjusted only to those data and it would quickly become overfitted.

Input: S - data stream, M - batch size, α, λ_C
1 CuSum = 0 ;
2 Collect a new batch B from the stream S;
3 **for** *every data element in B* **do**
4 Increase counter of drawn of the current element;
5 Train the network on current element;
6 Compute loss function for a current element;
7 Update v_i according to (9) or (10)
8 **for** *every data element in T* **do**
9 Update *pod*s according to (11)
10 Compute loss function on a validation set;
11 Update CuSum according to (13);
12 **if** $CuSum > \lambda_C$ **then**
13 Reinitialize *pod's* values;
14 *Return to line 1*;
15 **else**
16 *Return to line 2*;

Algorithm 3. The BBTA - LB/NLB algorithm

Based on the aforementioned motivation we decided to add a drift detector to indicate whether the neural network performance has changed significantly or not. The proposed procedure, called BBADD, is presented in Figure 3. If the algorithm detects changes in the performance of the neural network, it resumes all the *pod*s to equal values.

The CUSUM algorithm was applied as a drift detector. To monitor changes in loss function values the following value is computed

$$\begin{aligned} &Cus_0 = 0, \\ &Cus_i = max(0, Cus_{i-1} + L(B_{i-1}) - L(B_i) - \alpha), \end{aligned} \tag{13}$$

for $i = 1, 2, \ldots$, where $L(B_i)$ is a value of loss function in the i-th mini-bath and α is a fixed parameter. The drift is detected when Cus_i exceeds the value of threshold λ_C.

The proposed Boosting based training algorithm with a drift detector, called BBTADD, monitors changes in loss function values. To prevent overfitting, it requires an additional validation set, which is used to compute monitored *Cus* values. The details of the BBTADD algorithm are presented in Algorithm 3.

4 Experimental results

To demonstrate the performance of the proposed algorithms, the experiments were performed on the MNIST dataset [35], which contains 60000 gray-scale images of handwritten digits. Each image is of size 28×28. In experiments, we were generating a data stream from this dataset. Firstly, the data were mixed. Then, the mini-batches of size n were randomly collected.

To carry on the simulations a convolutional neural network was applied. The network consists of five layers. The first layer contains 32 channels. The kernel size was set to 3×3 and stride equal to one. The second layer has 64 channels, and the kernel size and strides are the same as in the first layer. Next, the max-pooling layer, with a filter set to 2×2 is inserted. After that, the dropout mechanism is applied with a probability factor equal to 0.25. The fourth layer of the network is a first fully connected layer, consisting of 128 neurons. The final layer, preceded by the dropout mechanism with a probability factor set to 0.5, contains 10 neurons. In the last layer, the softmax activation function was applied, whereas in the rest of the layers the *ReLU* activation functions were used. The errors were computed based on categorical cross-entropy loss function and the Adadelta optimizer [43] was used during the training phase. The details of the network are present in Figure 2

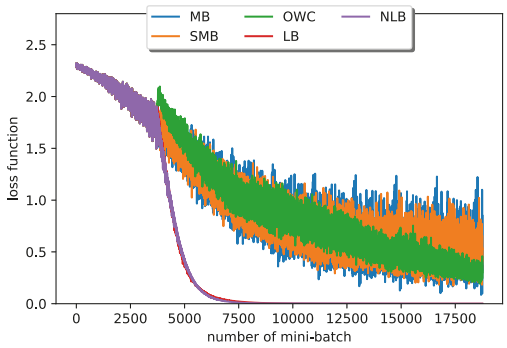
```
Model: "CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1179776
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

Figure 2. Summary of CNN.

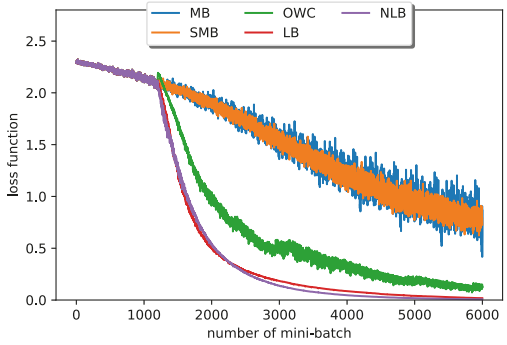
4.1 Analysis of BBT algorithm

To demonstrate the ability of the BBTA - OWC/LB/NLB algorithms, the obtained results are compared with the classical mini-batch learning approach (MB) and with a streaming approach (SMB) that does not include *pod* values. In the case of the MB method, the training set is divided into equally sized mini-batches. The subsequent mini-batches are applied to train the neural network. After the whole training set is processed, the procedure is repeated until the algorithm meets the stopping criteria. In the case of the SMB approach, the mini-batches are continuously generated from the training set. In each iteration, every data element has an equal chance to be found in a mini-batch.

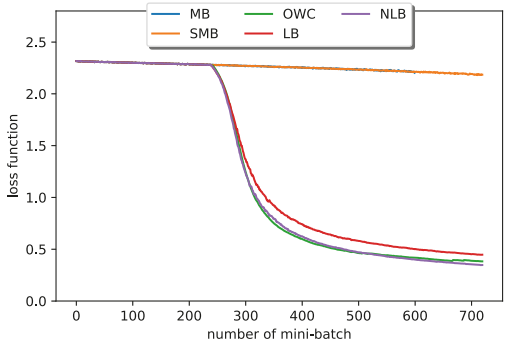
One of the crucial steps experiment preparation is to establish a mini-batch size. The bigger values result in faster training. On the other hand, the smaller values extend the time of computation but allow checking more network configurations. In the experiments, we investigated three different mini-batch sizes: $n = 32$, $n = 100$ and $n = 1000$. All simulations were run for ten epochs. Hence the total number of processed mini-batches is equal to 1875, 6000 and 600 for $n = 32$, $n = 100$ and $n = 1000$, respectively. To prevent chaotic behavior during the initial steps of training, the CNN was pre-trained in a classical manner at the beginning of each simulation.



(a) $n = 32$



(b) $n = 100$



(c) $n = 1000$

Figure 3. Loss function computed on mini-batches of various sizes: a) $n = 32$, b) $n = 100$, c) $n = 1000$.

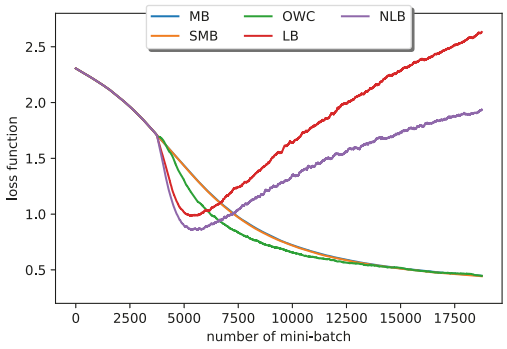
The values of the loss function obtained on subsequent mini-batches are presented in Figure 3. During the pre-training phase, each method presents the same values. After that, the influence of various approaches on the neural network performance differs significantly. Regardless of the applied mini-batch size, the MB and the SMB approaches demonstrate similar results, worse than

the other methods. It is important to note that the big differences between values obtained on the consecutive mini-batches are a consequence of their randomness. It seems that the LB and the NLB algorithms also achieve close values. However, it is worth noticing, that at the end of the experiment the loss function values for the NLB are always the lowest ones. The behavior of the OWC approach is the most sensitive to the mini-batch size. For small ones ($n = 32$) its performance is similar to the MB and the SMB methods. In the intermediate case ($n = 100$), its loss function values are somewhere between the values obtained for the other considered approaches. For very large mini-batches ($n = 1000$) the OWC competes for the best score with the NLB.

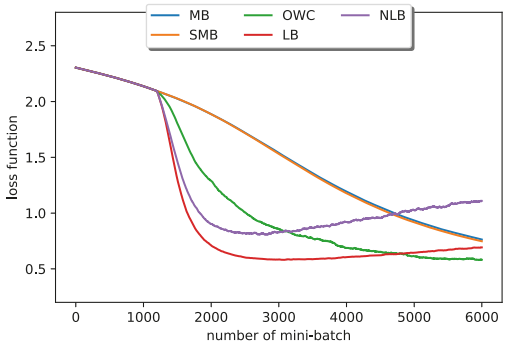
The initial picture of a well-functioning algorithm changes to the worse when we look at the values of the loss function calculated on the test set, see Figure 3.

One can see that in the case of the MB, the SMB, and the OWC methods, the convergence of their loss values seems to be similar to that presented in Figure 3. In the case of the LB and the NLB, after the initial big decrease in the loss function value, it starts to increase. The coincidence of low loss function values for training data and high values obtained for the test set indicate that in the cases the LB and NLB methods a small sample of data from the training set might obtain relatively high *pod* values. Consequently, the same data were drawn to the mini-batches. The model adjusted itself to those data and it lost the ability to generalize the knowledge. It is worth noticing that the problem occurred only in two cases, i.e. $n = 32$ and $n = 100$. The sufficiently large size of mini-batches solves this problem.

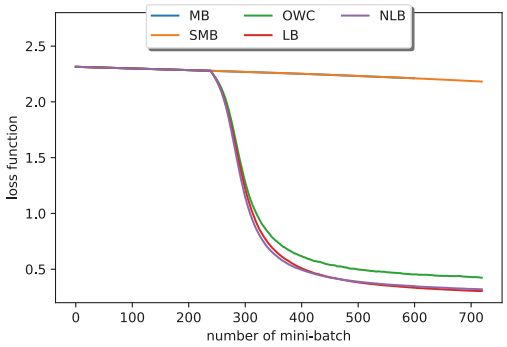
The accuracies obtained by the CNN, in the case of $n = 32$, are presented in Tables 1 and 2 for training and test data, respectively. The Tables include the mean accuracy over the whole training process, the maximal accuracy obtained during training and the final accuracy obtained after processing the last mini-batch. One can see that very good results of the LB and the NLB methods obtained for the training data and weak performance for the test set confirm the need for modification of these techniques.



(a) $n = 32$



(b) $n = 100$



(c) $n = 1000$

Figure 4. Loss function computed on the test data for various sizes of mini-batches: a) $n = 32$, b) $n = 100$, c) $n = 1000$.

Table 1. The mean, maximal and final accuracies obtained by various algorithms on the training mini-batches.

Algorithm	Mean	Maximal	Final
MB	0.7599	**1.0**	0.9375
SMB	0.7608	**1.0**	0.8125
OWC	0.7559	**1.0**	**1.0**
LB	0.89154	**1.0**	**1.0**
NLB	**0.89205**	**1.0**	**1.0**

Table 2. The mean, maximal and final accuracies obtained by various algorithms on the test sets.

Algorithm	Mean	Maximal	Final
MB	0.76779	0.88040	0.87949
SMB	0.76855	**0.88179**	**0.88169**
OWC	**0.77530**	0.86720	0.86690
LB	0.59989	0.72610	0.59100
NLB	0.66487	0.71899	0.69470

4.2 Analysis of BBTADD algorithm

The simulations presented in the previous Section demonstrate the need for modification of the BBAT LB and NLB algorithms. Now, the performance of the BBTADD algorithm, described in Section 3.2, will be demonstrated and discussed. To highlight the advantages of the proposed method its performance is compared with the NB approach. In the conducted experiments, the parameters of CUSUM drift detector are set to $\alpha = 0.00001$ and $\lambda_C = 0.002$. Their low values are dictated by small differences between subsequent loss function values on the validation set. The size of mini-batches is set to $n = 32$, as it was the worst scenario for the LB and the NLB approaches in Section 4.1.

The values of loss function computed on training mini-batches and testing data are presented in Figs 5 and 6, respectively. One can see that each of the considered algorithms behave unstable on the training data, which is caused by the small size of the batch. At the beginning of training, the variance of loss function values of the LB and the NLB methods is smaller than in the case of the MB approach. However, due to the reinitialization of the *pod* values their convergence rate was lowered.

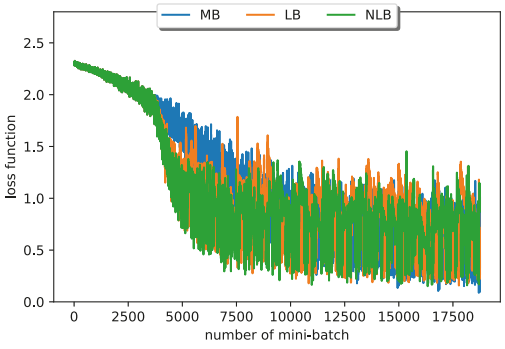


Figure 5. the loss function computed on the training mini-batches for $n = 32$

Results presented in Figure 6 are as expected. The value of the loss function for the LB and the NLB methods decreases on the test dataset faster than in the case of the MB approach.

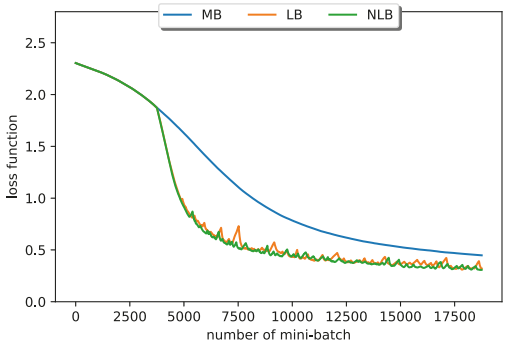


Figure 6. The loss function computed on test data for $n = 32$

After processing 18750 mini-batches the drift has been indicated 1376 and 1334 times for LB and NLB, respectively. Better adjustment of the drift detector parameters can improve the stability of these methods on the training mini-batches.

The accuracies on training and testing data, presented in Tables 3 and 4, respectively, confirm the effectiveness of the proposed methods. Comparing the results on the test set, presented in Tables 2 and 4, the usability of the drift detector application is confirmed.

Table 3. The mean, maximal and final accuracies obtained by the BBATDD MB/LB/NLB methods on the training mini-batches.

Algorithm	Mean	Maximal	Final
MB	**0.74494**	**1.0**	0.90625
LB	0.68139	**1.0**	**0.9375**
NLB	0.696	**1.0**	0.53125

Table 4. The mean, maximal and final accuracies obtained by the BBATDD MB/LB/NLB methods on the test sets.

Algorithm	Mean	Maximal	Final
MB	0.75194	0.8834	0.88289
LB	**0.78996**	**0.9245**	**0.9196**
NLB	0.78812	0.9193	0.9182

5 Conclusion

In this paper, we explored the possibility of improving methods for learning deep neural networks by applying techniques commonly used in the data streams analysis. Replacing epoch-based learning by randomly creating subsequent mini-batches allowed the neural network training to be more efficient. The idea taken from the boosting algorithm resulted in a faster decrease of the loss function values. However, it has turned out that the proposed techniques may result in drawing the same elements into the mini-batches. The application of the drift detector helped to eliminate this problem effectively.

The conducted experiment demonstrated that the proposed approach opens up a number of research threads. Finding other strategies for determining *pod* values can significantly improve learning. The application of a dedicated drift detector can allow the system to better react to changes in the trained model. Moreover, the application of the proposed techniques to other convolutional neural network architectures, like VGG Net or DenseNet, or other structures, like restricted Boltzman Machines or recurrent neural network, would be an interesting task.

Acknowledgments

This work was supported by the Polish National Science Centre under grant no. 2017/27/B/ST6/02852.

References

[1] Abdulsalam, H., Martin, P., and Skillicorn, D. S.; Streaming random forests. In 11th International Database Engineering and Applications Symposium (IDEAS 2007), pp. 225–232.

[2] Abdulsalam, H., Skillicorn, D. B., and Martin, P.; Classifying evolving data streams using dynamic streaming random forests. In International Conference on Database and Expert Systems Applications (2008), Springer, pp. 643–651.

[3] Baena-Garcia, M., del Campo-Avila, J., Fidalgo, R., Bifet, A., Gavalda, R., and Morales-Bueno, R.; Early drift detection method. In Fourth International Workshop on Knowledge Discovery from Data Streams (2006).

[4] Bengio, Y.; Learning deep architectures for AI. Foundations and Trends in Machine Learning 2, 1 (2009), 1–127.

[5] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H.; Greedy layer-wise training of deep networks. In Proceedings of the 19th International Conference on Neural Information Processing Systems (Cambridge, MA, USA, 2006), NIPS'06, MIT Press, pp. 153–160.

[6] Bifet, A., and Gavaldà, R. Adaptive learning from evolving data streams. In International Symposium on Intelligent Data Analysis (2009), Springer, pp. 249–260.

[7] Bodyanskiy, Y., Vynokurova, O., Pliss, I., Setlak, G., and Mulesa, P.; Fast learning algorithm for deep evolving gmdh-svm neural network in data stream mining tasks. In 2016 IEEE First International Conference on Data Stream Mining Processing (DSMP) (Aug 2016), pp. 257–262.

[8] Bologna, G., and Hayashi, Y.; Characterization of symbolic rules embedded in deep dimlp networks: a challenge to transparency of deep learning. Journal of Artificial Intelligence and Soft Computing Research 7, 4 (2017), 265–286.

[9] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR abs/1412.3555 (2014).

[10] deBarros, R. S. M., Hidalgo, J. I. G., and de Lima Cabral, D. R.; Wilcoxon rank sum test drift detector. Neurocomputing 275 (2018), 1954–1963.

[11] Demsar, J., and Bosnic, Z.; Detecting concept drift in data streams using model explanation. Expert Systems with Applications 92 (2018), 546–559.

[12] Deng, L., Hinton, G., and Kingsbury, B.; New types of deep neural network learning for speech recognition and related applications: An overview. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (2013), IEEE, pp. 8599–8603.

[13] Ditzler, G., Roveri, M., Alippi, C., and Polikar, R.; Learning in nonstationary environments: A survey. IEEE Computational Intelligence Magazine 10, 4 (2015), 12–25.

[14] Domingos, P., and Hulten, G.; Mining high-speed data streams. In Proc. 6th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining (2000), pp. 71–80.

[15] Gama, J., Medas, P., Castillo, G., and Rodrigues, P.; Learning with drift detection. In Brazilian Symposium on Artificial Intelligence (2004), Springer, pp. 286–295.

[16] Gers, F. A., and Schmidhuber, J.; Recurrent nets that time and count. In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium (July 2000), vol. 3, pp. 189–194 vol.3.

[17] Gomes, H. M., Barddal, J. P., Enembreck, F., and Bifet, A.; A survey on ensemble learning for data stream classification. ACM Computing Surveys (CSUR) 50, 2 (2017), 23.

[18] Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., and Abdessalem, T.; Adaptive random forests for evolving data stream classification. Machine Learning 106, 9-10 (2017), 1469–1495.

[19] Goodfellow, I., Bengio, Y., and Courville, A.; Deep Learning. MIT Press, 2016.

[20] He, K., Zhang, X., Ren, S., and Sun, J.; Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016), pp. 770–778.

[21] Hinton, G. E., Osindero, S., and Teh, Y.-W.; A fast learning algorithm for deep belief nets. Journal of Neural Computation 18, 7 (July 2006), 1527–1554.

[22] Hinton, G. E., Sejnowski, T. J., and Ackley, D. H.; Boltzmann machines: Constraint satisfaction networks that learn. Tech. Rep. CMU-CS-84-119, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1984.

[23] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J.; Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[24] Hou, Y., and Holder, L. B.; On graph mining with deep learning: Introducing model r for link weight prediction. Journal of Artificial Intelligence and Soft Computing Research 9, 1 (2019), 21–40.

[25] Huang, G., Liu, Z., v. d. Maaten, L., and Weinberger, K. Q.; Densely connected convolutional networks. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017), pp. 2261–2269.

[26] II, A. G. O., Giles, C. L., and Reitter, D.; Online semi-supervised learning with deep hybrid boltzmann machines and denoising autoencoders. CoRR abs/1511.06964 (2015).

[27] Jaworski, M., Duda, P., and Rutkowski, L.; On applying the Restricted Boltzmann Machine to active concept drift detection. In Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (Honolulu, USA, 2017), pp. 3512–3519.

[28] Jaworski, M., Duda, P., and Rutkowski, L.; Concept drift detection in streams of labelled data using the Restricted Boltzmann Machine. In 2018 International Joint Conference on Neural Networks (IJCNN) (2018), pp. 1–7.

[29] Jaworski, M., Rutkowski, L., Duda, P., and Cader, A.; Resource-aware data stream mining using the Restricted Boltzmann Machine. In Artificial Intelligence and Soft Computing (Cham, 2019), L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. M. Zurada, Eds., Springer International Publishing, pp. 15–24.

[30] Kingma, D. P., and Welling, M.; Stochastic gradient vb and the variational auto-encoder. In Second International Conference on Learning Representations, ICLR (2014), vol. 19.

[31] Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., and Wozniak, M.; Ensemble learning for data stream analysis: A survey. Information Fusion 37 (2017), 132–156.

[32] Krizhevsky, A., Sutskever, I., and Hinton, G. E.; Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[33] LeCun, Y., Bengio, Y., and Hinton, G.; Deep learning. Nature 521, 7553 (2015), 436.

[34] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P.; Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 11 (Nov 1998), 2278–2324.

[35] LeCun, Y., and Cortes, C.; Mnist handwritten digit database (2010); http://yann.lecun.com/exdb/mnist/

[36] Mamoshina, P., Vieira, A., Putin, E., and Zhavoronkov, A.; Applications of deep learning in biomedicine; Molecular pharmaceutics 13, 5 (2016), 1445–1454

[37] Mello, R. F., Vaz, Y., H.Grossi, C., and Bifet, A.; On learning guarantees to unsupervised concept drift detection on data streams; Expert Systems with Applications 117 (2019), 90–102

[38] Page, E. S.,Continuous inspection schemes; Biometrika 41, 1/2 (1954), 100–115

[39] Read, J., Perez-Cruz, F., and Bifet, A., Deep learning in partially-labeled data streams; In Proceedings of the 30th Annual ACM Symposium on Applied Computing (New York, NY, USA, 2015), SAC '15, ACM, pp. 954–959

[40] Simonyan, Karen; Zisserman, A., Very deep convolutional networks for large-scale image recognition; eprint arXiv:1409.1556 (2014)

[41] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., Going deeper with convolutions, In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2015), pp. 1–9

[42] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A., Extracting and composing robust features with denoising autoencoders; In Proceedings of the 25th International Conference on Machine Learning (New York, NY, USA, 2008), ICML '08, ACM, pp. 1096–1103

[43] Zeiler, M. D., Adadelta: an adaptive learning rate method; arXiv preprint arXiv:1212.5701 (2012)

Piotr Duda received the M.Sc. degree in mathematics from the Department of Mathematics, Physics, and Chemistry, University of Silesia, Katowice, Poland, in 2009. He obtained the Ph.D. degree and Sc.D. in computer science with the Institute of Computational Intelligence, Częstochowa University of Technology, Częstochowa, Poland in 2015 and 2019, respectively. His current research interests include deep learning and data stream mining.

Maciej Jaworski received the M.Sc. (Hons.) degree in theoretical physics from Jagiellonian University, Kraków, Poland, in 2009, and the M.Sc. degree in applied computer science from the AGH University of Science and Technology, Cracow, in 2011. He obtained the Ph.D. degree and Sc.D. degree in computer science with the Institute of Computational Intelligence, Częstochowa University of Technology, Częstochowa in 2015 and 2019, respectively. His current research interests include computational intelligence, data stream mining and neural networks.

Andrzej Cader is a professor at University of Social Science in Łódź, Poland. He received the Ph.D. degree in biocybernetics and biomedical engineering from the Medical University in Łódź, Poland. His research interests include intelligent systems science based on several architectures such as neural networks and complex systems. He is also currently engaged in time series analysis and chaos theory.

Lipo Wang received the Bachelor degree from National University of Defense Technology (China) and Ph.D. from Louisiana State University (USA). His research interest is artificial intelligence/machine learning with applications to communications, image/video processing, biomedical engineering, and data mining. He has authored 320 papers, of which 110 are in journals. He has authored 2 monographs and edited 20 books. His work has been cited 7,800 times in Google Scholar. He was/will be keynote speaker for 40 international conferences. He was President of the Asia-Pacific Neural Network Assembly (APNNA) and received the APNNA Excellent Service Award.