Stochastic Weight Update in The Backpropagation Algorithm on Feed-Forward Neural Networks

Juraj Köščak, Rudolf Jakša, Member, IEEE and Peter Sinčák, Member, IEEE

Abstract—We will examine stochastic weight update in the backpropagation algorithm on feed-forward neural networks. It was introduced by Salvetti and Wilamowski in 1994 in order to improve probability of convergence and speed of convergence. However, this update method has also one another quality, its implementation is simple for arbitrary network topology. In stochastic weight update scenario, constant number of weights is randomly selected and updated. This is in contrast to classical ordered update, where always all weights are updated. We will describe exact implementation, and present example results on toy-task data with feed-forward neural network topology. Stochastic weight update is suitable to replace classical ordered update without any penalty on implementation complexity and with good chance without penalty on quality of convergence.

I. INTRODUCTION

The stochastic weight update was introduced by Salvetti and Wilamowski in 1994 [1] in order to improve probability of convergence and speed of convergence of the backpropagation algorithm. Besides the stochastic weight update, they examined another two stochastic methods: random pattern selection and randomized learning rate. On the XOR problem they demonstrated significant improvement in the learning speed and probability of convergence for every one from these methods, especially for randomized learning rate.

The backpropagation algorithm (Werbos, 1974; Rumelhart, McClelland, 1986) is one of most used learning algorithms today. Plain vanilla implementation and momentum implementation (Silva, Almeida, 1990) [2] are the predominant implementations on feed-forward topologies. Multilayer perceptron (MLP) is most used feed-forward topology, it is a layered architecture with fully interconnected layers.

The backpropagation through time (BPTT) [3] is most used backpropagation variant for recurrent topologies. The BPTT was originally described, and is well suited, for recurrent topologies with full connectivity. Although the BPTT implementation is not much more complex compared to feed-forward vanilla backpropagation, the feed-forward time-delay networks are preferably used for time-related problems. Time-delay networks are of the MLP design, so layered topology is most used topology with the backpropagation algorithm.

Sparse topologies became recently popular with the recurrent echo state networks (ESN) (Jaeger, 2001) [4]. However these are usually not based on backpropagation algorithm. ESN networks are based on so called reservoir of randomly interconnected neurons, so they have fully random sparse topology.

Another area where sparse topologies are employed are neuro-fuzzy architectures, typical is backpropagation based ANFIS (Jang, 1993) [5]. ANFIS-like neuro-fuzzy topologies are characterized by layered, but not fully interconnected structure. In contrast to randomly-connected ESN architectures, topologies of ANFIS-like structures are deterministic.

The last popular group of backpropagation related topologies are neocognitron (Fukushima, 1980) [6] and LeNet (LeCun et al., 1989) [7]. These so called convolutional neural networks do use deep layered topologies with five or more layers. The difference from the classical MLP network is however the weight reuse, where several links in the network share the same weight.

In modular neural networks several basic topologies are connected to bigger meta-topology. Sometimes it might be useful to process this meta-network as a single instance with complex topology. This happen for instance with so called actor-critic architectures for reinforcement learning, where two or more networks are back-propagated by backpropagation algorithm in single run. Beside a couple of actor-critic topologies most famous modular architectures are mixtures of experts (Jacobs, Jordan, Nowlan, & Hinton, 1991) [8] and NARA (Takagi, 1992) [9], which are general-use neural networks.

Pruning methods and cascade correlation (Fahlman, 1991) [10] style algorithms for incremental building of topology can lead to pseudo-random sparse topologies too.

II. MOTIVATION

Implementation of classical ordered weight update in backpropagation algorithm does rely on the order of weights computation in network. In classical MLP networks it is simple layer-by-layer computation from inputs to the output layer. Less common, but still convenient implementation is for fully interconnected network, like in the common BPTT case. The necessity to update weights from every neuron to every neuron, can actually bring us compact implementation - we don’t need to care about any exceptions.

With sparse topologies, missing links often bring up some additional code into implementation to handle them. With backpropagation the problem might become even more complicated as we have to care about order of links. For instance, layered recurrent topology for BPTT algorithm might become difficult to handle, as we have to unfold relatively complex structure into several time steps and be able to evaluate it in both directions.
With stochastic weight update the order of links is not important. In any topology we just create a pool of links, select the number of them, and update them in random order. This can simplify the link update part in backpropagation implementations for complex topologies. With some future development, the stochastic weight update might become a part of efficient backpropagation implementation for recurrent networks.

Simplification of implementation might bring some penalty on the quality of results. However, Salvetti and Wilamowski shown on the XOR problem that opposite is probably true, and introduction of stochastic processes into backpropagation can actually improve quality of convergence. We will try to examine the approach again with the circle-in-square problem and discuss possible variations.

### III. STOCHASTIC WEIGHT UPDATE

Incorporation of some stochastic feature into learning process of backpropagation algorithm can help to avoid getting stuck network in a local minima on error surface [1]. One of ways how to do it, is to use stochastic weight update. Maybe not all weights are required to be updated, not all of them have the same influence on the network error. Some of them are more important, have higher values, some of them are close to zero. Their influence is lower, almost none. If we don’t update all weights, we can speed up the learning process, because we save some computation time and decrease the CPU consumption.

**ALGORITHM I**

**IMPLEMENTATION OF BACKPROPAGATION ALGORITHM FOR MLP**

1. For $i = 0; i < all\text{Required}\text{Iterations};$
2. For $p = 0; p < all\text{Patterns}\text{Count};$
3. propagateSignalThroughAllLayers($);$
4. countErrorSignalOfLayers($);$
5. adaptWeightsBasedOnErrorSignal($);

Where $i$ is number of iteration (number of learning cycle), and $p$ is number of presented pattern.

Stochastic weight update does not require heavy modification of the classic error backpropagation algorithm (Algorithm I). Slight difference is in the process of updating weights, after an error signal was computed. A real random number is generated from the range $(0, 1)$. If its value is less than 0.5, then this weight will be updated, otherwise will be left not modified (Algorithm III, for comparison classical ordered weight update is Algorithm II). This is 50% chance that the weight is going to be selected and updated. The value from the range is chosen pseudo randomly with approximately uniform distribution (like the random number generation in Java language [11]).

If we want to increase weight selection chance over 100% we can add another loop with parameter, which control how many times this random selection should be done. If set to 1 it is equal to the loop missing. With raising this parameter, probability that all weights will be selected and updated is increasing.

### IV. EXPERIENCE

To test the stochastic weight update we chosen the circle-in-square problem – a classification toy-task benchmark from the ART networks community (see Fig. 1). We wanted to check that the stochastic weight update will not reduce quality of backpropagation learning, and also to explore the influence of ratio of weights updated per pattern on the results.

Salvetti and Wilamowski in their original paper didn’t prescribed exact implementation, they just state: “Instead of being always updated in each step, weights are updated one by one in random manner. In this way, each time a pattern is presented not all weights are updated as usual but just the randomly selected ones.” Also noting: “In order to keep track of the number of calculations involved in the process, one iteration is considered when the same number of weight updates occur as if it was a standard backpropagation iteration.” The actual implementation and also the methodology of evaluation of results, thus, might differ. Although, we don’t see reasons for obtained results to be dramatically different.

**ALGORITHM II**

**CLASSICAL ORDERED WEIGHT UPDATE FOR MLP**

```
adaptWeightsBasedOnErrorSignal()
1. For $l = 0; l < num\text{Of}\text{Layers};$
2. For $w = 0; w < all\text{Weights}\text{Of}\text{Layer};$
3. adaptWeight($);
```

Where $l$ is number of layer, and $w$ is number of weight to be updated.

**ALGORITHM III**

**OUR STOCHASTIC WEIGHT UPDATE FOR MLP**

```
stochasticAdaptWeightsBasedOnErrorSignal()
1. For $l = 0; l < num\text{Of}\text{Layers};$
2. For $w = 0; w < all\text{Weights}\text{Of}\text{Layer};$
3. if randomNum > 0.5
4. adaptWeight($);
```

Where $l$ is number of layer, and $w$ is number of weight to be updated. The randomNum is generated by pseudo-random generator with uniform distribution, and the 0.5 represents 50% chance that the current weight will be adapted. The 0.5 constant can be changed into any number from $(0, 1)$.

| Fig. 1. | Visualization of the circle-in-square problem. The task is to classify vectors of $x$ and $y$ coordinates into two classes - these which belong into circle a these which are outside. |
class, square class. Available data are divided into training set and testing set. In the training set we had 1001 samples, 492 of which represented the square class and 509 of samples represented the circle class. Testing data set consisted of 10002 samples, the square class was represented by 5004 samples, and circle class was represented by 4998 samples. Visualization of data set is on the Fig. 1.

We will compare the stochastic weight update with classical ordered update on multilayer perceptron with backpropagation algorithm. We will use the same number of neurons, the same sigmoid function with the steepness $\lambda$ and the same learning rate $\gamma$. The topology consists of one hidden layer, two input neurons, which do represent $x$ and $y$ coordinates of the points, and one output neuron, which classifies the required class. Networks are trained with online learning method - after each sample (point) the error signal is computed and weights are updated.

In our case for all networks, one iteration during learning process means, that after each training sample, error signal is counted and then weights are updated. In the stochastic implementation not all weights are updated in single iteration (only the selected ones), some weights keep their values from previous state. All experiments were running ten times with the same settings. As the result of experiment is chosen the best result and this is used to create the output figure.

In the first experiment (Fig. 2, Table IV) stochastic weight update is compared with regular update on well trained net-}

![Ordered Update](image1.png)  ![Stochastic Update](image2.png)

**Fig. 2.** Visualization of result of well trained networks with classical ordered weight update and stochastic weight update with backpropagation algorithm with the same settings. Results are comparable.

Second experiment is presented to show the effect of the ratio of weights updated per pattern in the stochastic weight update. The parameters of learning are different from first experiment. The setup is positioned into situation where classical weight update was not able to achieve comparable results with stochastic weight update using the same number of hidden neurons. So presented classical weight update network has more hidden neurons. Other networks parameters are the same for both networks. The number of iterations is the same 2600 iterations to compare with previous experiment.

Chosen levels o ratio are 25%, 50%, 75% and 100%.
Note that ratio 100% is actually the same behavior as if using classical ordered weight update. Classification results are displayed in the Table V. The best results were actually achieved with the 50% ratio (see also Fig. 3 for visualization of obtained results). The course of training is illustrated on the Figure 4. Overall course of training seems similar for all ratios except the 100% when the algorithm got stacked. Note that actual amount of computations per iteration is different for every ratio.

Our experience shows, that the classical ordered weight update is more sensitive to the changes of the learning rate \( \gamma \). If the learning rate is greater, the ordered update requires more neurons in hidden layer (see second experiment and Table V).

Fig. 4. The course of mean squared error of stochastic weight update backpropagation with different selection ratios. Overall course of training seems similar for all ratios except the 100% when the algorithm got stacked. This graph corresponds to the results in Table V and on Figure 3.

V. DISCUSSION

Our results confirm the results of Salvetti and Wilamowski on XOR problem - stochastic weight update performs well and in some cases can be superior to classical ordered update in backpropagation algorithm.

In our experience this stochastic update is also less sensitive to the settings of learning rate, compare to the classical ordered update. Thus it can be used in cases where it is not easy to find out the working configuration for the network.

We also want to underline the implementation possibilities which the stochastic weight update might open. Complexity of implementation of this weight update is less dependent on the network topology than the classical ordered weight update is, while overall complexity of implementation is not much higher than the classical one. This can be harnessed in backpropagation implementations with non-standard, sparse, or random network topologies.

ACKNOWLEDGMENT

This work was supported by the project implementation of Center of Information and Communication Technologies for Knowledge Systems (project number: 26220120020) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES