

Stochastic Weight Matrix-based Regularization Methods for Deep Neural Networks^{*}

Patrik Reizinger¹ and Bálint Gyires-Tóth¹

Budapest University of Technology and Economics
Budapest, Hungary {rpatrik96@sch, toth.b@tmit}.bme.hu

Abstract. Generalization is one of the major challenges in most deep learning applications. Depending on the addressed problem, it can be reduced by several well-known methods. Despite the availability of a wide range of algorithms the problem cannot be considered as solved. The aim of this paper is to extend the available toolbox by introducing two generally applicable methods based on directly modifying the weight matrices – called Weight Matrix Modification (WMM). Weight Reinitialization uses a simplified Bayesian assumption with partially resetting a sparsely selected subset of the weights, while Weight Shuffling introduces non-white noise in an entropy- and weight distribution-invariant way. The latter can also be interpreted as an ensemble approach. The proposed methods are evaluated on benchmark datasets, such as MNIST, CIFAR-10 or the JSB Chorales database, and also on time-series modelling tasks. We report gains both regarding performance and entropy of the analyzed networks.

Keywords: Deep learning · Generalization · Regularization · Weight matrix.

1 Introduction

The importance of preventing the phenomenon of overfitting, i.e., the reduction of the generalization ability of a deep neural network, has gained more importance with the broadening of the spectrum of using deep learning for solving real-world problems. This is mainly due to the fact that artificial intelligence has become a tool widely used in the industry, where in several cases - due to cost

^{*} The research presented in this paper has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Informations), by the BME-Artificial Intelligence FIKP grant of Ministry of Human Resources (BME FIKP-MI/SC), by Doctoral Research Scholarship of Ministry of Human Resources (NKP-18-4-BME-394) in the scope of New National Excellence Program, by János Bolyai Research Scholarship of the Hungarian Academy of Sciences. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. Patrik Reizinger expresses his gratitude for the financial support of the Nokia Bell Labs Hungary.

restrictions, etc. - no feasible way exists to collect enough or perfectly representative data. The existing pool of methods is rather numerous, nevertheless new approaches can bring additional advantages. Before introducing our proposal, it is worth shortly summarizing the aspects of the most popular regularization methods. Traditional constraints on the weights (like L1 and L2 norms) add a prior to the weight matrix distributions, so the method takes its effect not directly on the weights, as the methods in section 3. Novel results [10] propose the usage of smaller batch sizes; more robust training and improved generalization performance is reported - while the variance of the minibatch, in general, tends to be higher. Dropout [14] and its generalization, DropConnect [15], are approaching the effect of ensemble networks by partially blocking different parts of the error propagation in the computational graph throughout training. Batch Normalization [6] affects the gradients, by taking a normalization step speed-ups are reported both regarding accuracy and rate of convergence.

Another approach of regularization, stemming from the scarceness of available data, is the augmentation of the dataset. Besides using transformations on the data samples, the role of noise is also to consider, because applying additive white noise in some cases also improves generalization, since it can smoothen the minibatch, while its long-term effect on the distribution of the samples is negligible. Hence the expected value of it equals zero. However, in the case of working with, e.g. time series or natural language, besides noise it is rather difficult to find the equivalent transformations of rotating, scaling, etc. which are extensively used in the case of computer vision for data augmentation. For time-series classification, there exist methods which can be thought to be of similar nature, such as Window Warping [4], which applies scaling for a randomly selected slice of a signal. A promising result regarding noise as means of regularization for reinforcement learning is given in [12], the use of additive noise in the parameter space contributed to significant improvements in several tasks. For recurrent networks, another type of regularization method has turned out to be successful: Zoneout [8] can be considered as a variant of Dropout for recurrent neural networks such as LSTM (Long Short-Term Memory) [5]. The main difference compared to Dropout is that Zoneout maintains at least the identity - i.e., instead of dropping some cells, the activation values from the last minibatch are used in the graph -, thus preserving information flow in the recurrent architecture.

This paper is organized as follows: section 2 briefly reviews the motivation for the proposed framework, section 3 includes the detailed overview of the proposed algorithms, the experimental setup is described in section 4, while the results are introduced in section 5 and discussed in section 6.

2 Motivation

Considering the most widely-used approaches and the possibilities we decided to investigate more thoroughly the direct effects of regularization applied to the weight matrices. While doing so, we also considered that in contrast to

the real-world or artificial data used for the training process, parameters do not possess such a high degree of internal error-resilience, i.e., an improperly designed method can lead to significant degradation of the training, even to complete failure. This is a clear consequence of the weights being a compact structure of expressing information of the stochastic process intended to be modelled. Thus, either a 'universal weight scheme' or a correlated change seems to be feasible for weight modification methods, because we can assume that in those cases, the change of parameters will not cause significant degradation (e.g., changes, which cannot be corrected during the timeframe specified for early stopping, can cause the termination of the training process by triggering early stopping).

Given these considerations and constraints, our goal is to design robust, generally applicable regularization methods based on the direct modification of the weight matrices.

3 Proposed Methods

In this section, we describe the proposed two novel methods, collectively named Weight Matrix Modification. Both of them modify the weights explicitly, but the approach they use is different. Weight Reinitialization can be considered as a way of expressing distrust of backpropagation by discarding its parameter updates, while Weight Shuffling is a local approach, which constrains its intervention based on an assumption regarding the autocorrelation function of a specific weight matrix.

In this section W denotes a weight matrix, p a probability value, $\mathcal{L}$ the set of affected layers and c a constant (called *coverage*). Weight matrices are considered two dimensional as it is the general case (for CNNs, we only consider the 2D filters – not the filter bank as a whole). Nevertheless, it is done without loss of generalization, hence this property is not exploited in the proposed methods. Please note that both methods are concerning the weight matrices only and not the bias vectors, because our evaluations showed that including the bias cannot bring further advantages.

3.1 Weight Reinitialization

Although the 'universal weight scheme' mentioned in section 2 does not exist to solve all deep learning tasks, there are general concepts, which help performance improvements. Like the weight initialization scheme of [3], or the common choice of

$$W \sim U \left[-\frac{1}{\sqrt{n_i}}, \frac{1}{\sqrt{n_i}} \right], \quad (1)$$

where n_i is the number of columns of the weight matrix W , uses a uniform distribution U . From an information theoretical point of view, using uniformly distributed weights corresponds to the principle of maximal entropy, thus a high degree of uncertainty. However, the cited paper from Glorot and Bengio [3],

which can improve the convergence with normalizing the gradients, uses also a uniform parameter distribution.

These considerations provide us means for proposing a regularization technique with a Bayesian approach. We start with recursive weighted least squares, where the new prediction is based on past and current observations, each of them weighted by the inverse of their variance. Then, we modify it to be suitable for weight matrices: not the whole weight matrix is affected, just a sparse subset of it. Thus, the probability to carry out changes which result in a significant deterioration of the used metrics is decreased. Since we only modify a small amount of the weights, it seemed not to be feasible to compute the statistics of the whole layer (and for the sparse mask is newly generated in each minibatch, it is also not possible for all combinations without significant speed degradation). Thus, the choices of the algorithm are the given weight or sampling the initial uniform distribution.

Algorithm 1 describes the steps of Weight Reinitialization. It is carried out based on a probabilistic scheme, which is controlled by the p parameter. This approach reduces the probability to introduce a high level of degradation into the network, on the other hand, it also saves computational power, because the method will not be applied in every minibatch. First, a mask M is generated by sampling a uniform distribution U . Then a window position is selected randomly (its size is controlled by the c parameter) – which means that all items outside the window are cleared in M . After that the same p is used as a threshold for ensuring sparsity, thus clearing the majority of the elements within the window. W^* denotes the temporary matrix used for storing the samples from the distribution U_w , the distribution used for initializing the weights. Based on the mask values (which remained within the window after thresholding with p), the elements of the original weight matrix W_i get reinitialized.

Thus, if some parameters can be considered as outliers, which - we hypothesize - is presumably the consequence of overfitting. Consider the distribution of the weights of a given layer; if it is multimodal with some small peaks, those represent different momenta than the highest one. By reinitializing a random subset of weights, if weights from the small peaks of the weight distribution are chosen, overfitting can be reduced with doing a modification step towards maximal entropy, thus smoothing the histogram. On the other hand, reinitializing values near the mean would result in small changes only. If the mean of the weight matrix is not the same as the expected value of the trained weight matrix, then reinitializing a subset of weights expresses our distrust and is opting for maximal entropy, thus a uniform distribution. Independent of the value of the mean of the whole layer, the reinitialized subset itself has an expected value of zero (Equation 1), thus the method will not result in a significant change, meaning reduced sensitivity. An interesting side-effect of Weight Reinitialization is the fact that by reducing the Kullback-Leibler divergence between the actual distribution of the weights and the original one, the entropy is generally increased (at least momentarily), which will be investigated in section 5.

Algorithm 1 Algorithm for Weight Reinitialization

Input: $\mathcal{L}, p, c$
for all $W_i \in \mathcal{L}$ **do**
 if $p < U[0; 1]$ **then**
 $M \leftarrow U[0; 1]$
 $M \leftarrow \text{window}(M, c)$
 $M \leftarrow \text{threshold}(M, p)$
 $W^* \leftarrow U_w$
 $W_i[M_{jk} == 1] = W^*$
 end if
end for

3.2 Weight Shuffling

As already mentioned in section 1, noise has a wide-range of applicability from the dataset to the weights. Instead of using Gaussian white noise, we consider to use non-white, correlated noise directly for a neighbourhood of weights. We refer to a neighbourhood of weights as a rectangular window with dimensions comparably smaller than the size of the weight matrix itself. Because the weights are intended to represent the characteristics of the underlying stochastic process, they should be correlated with each other (i.e. white noise itself is not appropriate for capturing the underlying process). Consequently, by selecting a continuous subset of the weights, the correlation should still hold, hence the process to be modelled stays the same. Shuffling them within the bounds of a local window can be interpreted as using correlated noise for that given subset. Interestingly, this transformation is invariant regarding of both the weight distribution and the entropy of the matrix – the latter of which will be investigated in section 5.

A more expressive way is to think about Weight Shuffling as a type of a ‘parameter ensemble’, i.e., weights in a neighbourhood are forced to fit well a range of input locations. As the ensemble approach, or the approximations of it – such as Dropout –, have been proven to be successful in practice due to the utilization (or simulation) of an averaging step, an improved degree of robustness is expected from Weight Shuffling as well.

The steps of Weight Shuffling are described in Algorithm 2. As in the case of Algorithm 1, we have a probabilistic first step which decides stochastically whether Weight Shuffling will be carried out or not. The mask M is sampled from a Bernoulli-distribution $\mathcal{B}$ directly, thus the density of the mask is independent of the parameter p . For the selection of the local window c is used as for Weight Reinitialization. Having generated the mask, the weights indexed by M are shuffled for each weight matrix of the layer subset $\mathcal{L}$ (filters of convolutional and gates of recurrent layers are handled independently from each other).

4 Experimental setup

Due to the fact that Weight Matrix Modification intervenes into the training loop on the lower levels in the software stack, we have searched for a framework

Algorithm 2 Algorithm for Weight Shuffling

Input: $\mathcal{L}, p, c$
for all $W_i \in \mathcal{L}$ **do**
 if $p < U[0; 1]$ **then**
 $M \leftarrow \mathcal{B}[0; 1]$
 $M \leftarrow \text{window}(M, c)$
 $\text{shuffle}(W_i[M])$
 end if
end for

which provides such a low-level interface. PyTorch [11] was chosen to be used for the implementation, because it makes possible to modify fine details of the training process. We utilized Ignite, a high-level library for PyTorch, which was built by the deep learning community to accelerate the implementation of deep learning models by providing a collection of frequently used routines.

The framework of the proposed methods is schematically depicted in Fig. 1: Ignite provides the wrapper which merges the PyTorch deep neural network model with Weight Matrix Modification with the help of its callback interface. Due to the fact that while designing Weight Matrix Modification neither architecture- nor task-specific assumptions were made, our approach provides great flexibility, which will be elaborated shortly. For hyperoptimization a Bayesian approach (tree of Parzen estimators) was used, based on a method shown in [1].

To prove the general applicability of the proposed framework, we set up a testing environment with different model architectures and datasets. Tests were carried out with fully connected, convolutional and LSTM [5] layers. In case of convolutional layers, regardless of the filterbank representation of the kernels, Weight Matrix Modification operations were, of course, restricted to operate only within one filter. For LSTM the effect of Weight Matrix Modification on different gates was investigated (the same holds for GRU [2] as well).

For evaluation several benchmark datasets were investigated, such as MNIST [9] (traditional and row-by-row sequential approach), CIFAR-10 [7] or JSB Chorales [13]. Beside classification, regression problems were also investigated, for that purpose an artificially generated (synthetic) dataset was used. The synthetic time series were fed into the neural network both with and without colored noise, and consisted of the linear combination of sinusoidal, exponential and logarithmic functions.

5 Results and discussion

During the analysis of the results several factors were considered. The three most important factors were

- Performance (accuracy/MSE (Mean Squared Error))
- Entropy

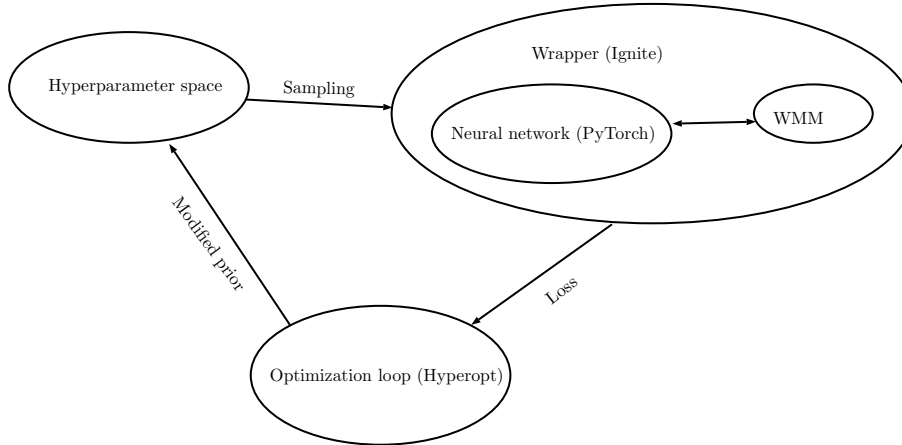


Fig. 1: Structure of the framework for Weight Matrix Modification

– Robustness.

Our analysis intended to identify the tasks where the proposed approach can turn out to be advantageous, thus we could even make useful conclusions with datasets such as MNIST, where state-of-the-art results are near perfect. For each dataset and each of the proposed methods we have conducted more than 2000 experiments to decrease the variance the stochastic processes involved in the proposed methods introduced to the training.

To clearly identify the effects of Weight Matrix Modification and to prevent overfitting originating from the simple nature of some datasets, the networks used for evaluation were not very deep (maximum depth was 5 layers). During evaluation, a baseline model (for each task, respectively) and network variants using L2-regularization, Batch Normalization [6] or Dropout [14] were used for comparison.

The figures included in this section are comparing different aspects of the networks. We compared the average of the best results utilizing Weight Matrix Modification to the best baseline results. The effect of the proposed methods regarding performance metrics is depicted in Fig. 2 (higher is better, we converted accuracy and MSE to be able to display them on the same scale), where datasets were omitted with a relative change smaller than 0.1%.

Regarding performance, the change is not significant for MNIST (traditional and sequential approach) and JSB Chorales, in the case of the synthetic dataset performance suffers even from degradation. On the other hand, for CIFAR-10 Weight Reinitialization brought a performance boost of 7%, while Weight Shuffling one of 3%. The biggest performance increase (over 10%) was experienced in the case of the synthetic sequential dataset with colored noise.

For the tasks utilizing recurrent networks (sequential MNIST, JSB Chorales, synthetic data) the best results were achieved by different choices of the LSTM

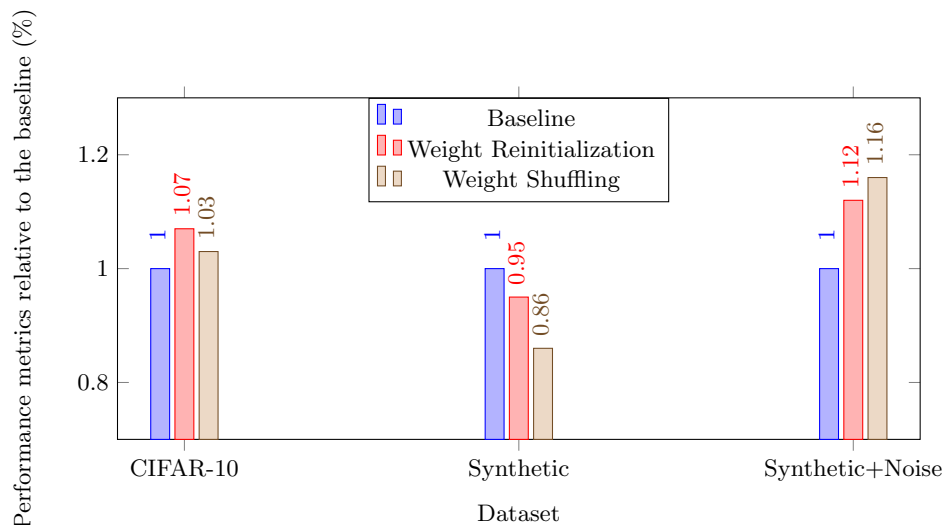


Fig. 2: The effect of Weight Matrix Modification on performance metrics (higher values are better)

gates. In case of Weight Reinitialization the modification of the weights of the forget gate was proven to be the best for the sequential MNIST task, while for the other three problems the cell gate was the best choice. For Weight Shuffling the output gate was preferred (synthetic + JSB Chorales) – the sequential MNIST task resulted in modifying the weights of the last, fully connected layer.

Partially due to the fact that the proposed methods have effect onto the entropy of the networks – Weight Reinitialization is expected to increase, while Weight Shuffling to retain it – so we decided to compare also entropy values for the different tasks. These results are shown in Fig. 3 (lower is better), where we also compare to the respective baselines.

For MNIST the change is not too significant, but in general Weight Matrix Modification was able to compress the networks by 2 – 9%, except Weight Shuffling for the not sequential task. For CIFAR-10, the compression is more significant, both methods rank in the same range.

In the case of the only multilabel dataset, JSB Chorales, the results were rather astonishing: while Weight Reinitialization has not brought a significant change, Weight Shuffling compressed the network by almost a factor of two. This superiority of Weight Shuffling compared to Weight Reinitialization regarding data compression is also characteristic of both variants of the synthetic dataset. Nevertheless, in the latter case Weight Reinitialization also reduced entropy for the synthetic data, but there is a difference of 27% between the noiseless and noisy cases.

Generally speaking, Weight Shuffling with its entropy-invariant approach was able to reduce entropy more than Weight Reinitialization. Despite using a prob-

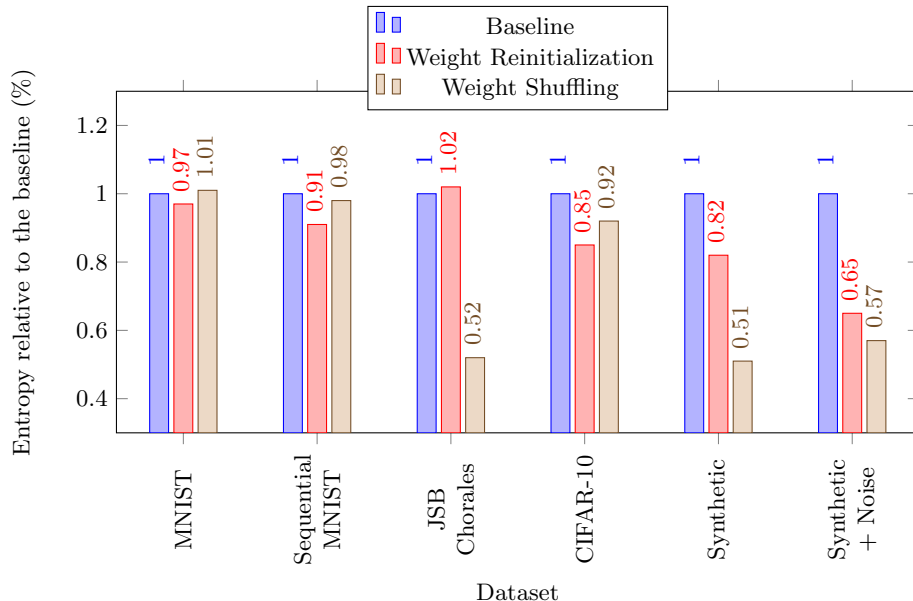


Fig. 3: The effect of Weight Matrix Modification onto the entropy of neural networks (lower values are better)

ability distribution according to the principle of maximal entropy, Weight Reinitialization is also able to compress the networks. The analysis of the temporal change of entropy in the case of Weight Reinitialization is shown in Fig. 4, where the legend shows the $\mathcal{L}$ parameters of Algorithm 1, also including the gates of the LSTM layer and the ranking according to MSE (the smaller the number, the lower the MSE). Although the entropy momentarily increases due to Weight Reinitialization, at the end it can help compressing the network, as shown in Fig. 4.

The robustness of the proposed methods is also a crucial factor. Fig. 5 shows the distribution of MSE w.r.t $p*c$ (c is denoted as the coverage parameter) during Bayesian hyperoptimization. The $\mathcal{L}$ parameter is color-coded in the figures, *none* denotes models without Weight Matrix Modification (i.e. the baseline). In the case of Weight Reinitialization (Fig. 5a) the plot has a horizontally elongated form, which means smaller sensitivity to parameter change (the less elongation in the y-direction, the better). Although using Weight Reinitialization imposes a higher variance due to its stochastic nature, it is clear to see that better results can be achieved also with it. A rather unexpected observation was, that in the case of Weight Shuffling (Fig. 5b) the bounding ellipse is rotated with 90° compared to Fig. 5a, thus showing higher stochastic sensitivity - nevertheless, the Bayesian optimization loop resulted in several good parameter sets.

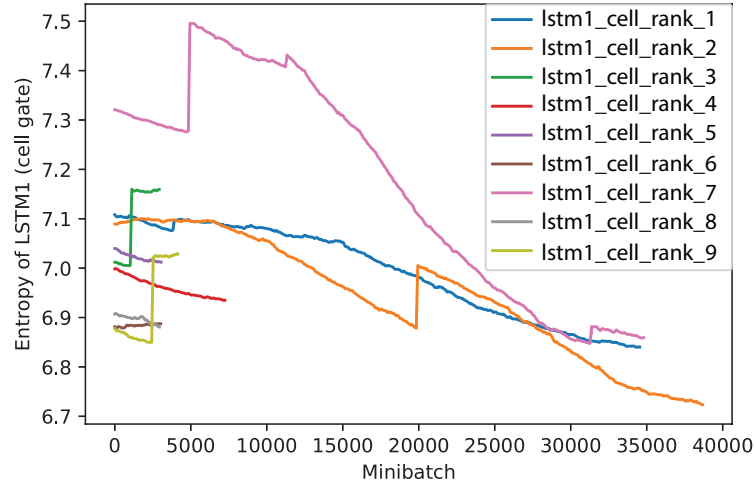


Fig. 4: Temporal entropy change during Weight Reinitialization (JSB Chorales, lower values are better)

6 Conclusions

In this paper two, weight matrix-based regularization methods were proposed and investigated for deep neural network applications. The advantages of both Weight Reinitialization and Weight Shuffling are among others their architecture- and task-independence. Our evaluations resulted in the following conclusions: both methods are capable of reducing overfitting, first of all in the case of noisy data, but also for CIFAR-10 better results were reported. On the other hand, Weight Reinitialization and Weight Shuffling also have shown their potential concerning model compression.

References

1. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems (NIPS)* pp. 2546–2554 (2011). <https://doi.org/2012arXiv1206.2944S>
2. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* (2014). <https://doi.org/10.1074/jbc.M608066200>
3. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. *Pmlr* **9**, 249–256 (2010). <https://doi.org/10.1.1.207.2059>
4. Guennec, A.L., Malinowski, S., Tavenard, R., Guennec, A.L., Malinowski, S., Tavenard, R., Augmentation, D., Guennec, A.L., Malinowski, S., Tavenard, R.: Data Augmentation for Time Series Classification using Convolutional Neural Networks. In: *ECML/PKDD workshop on advanced analytics and learning on temporal data* (2016)

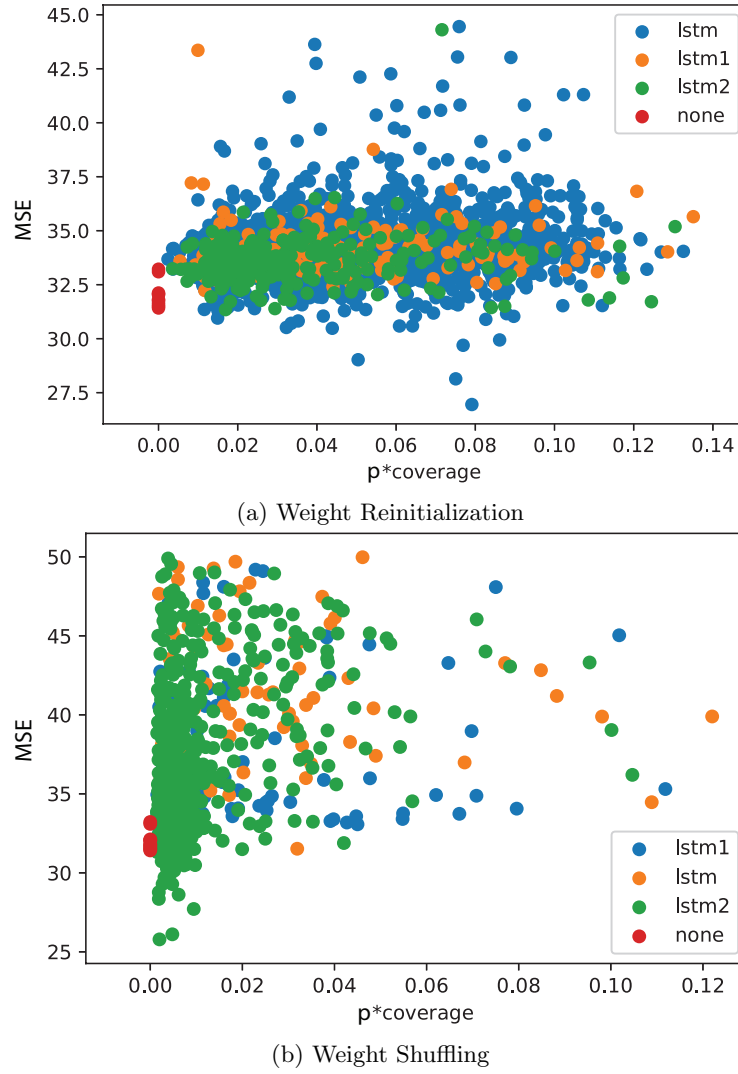


Fig. 5: Illustration of the robustness examination for the synthetic dataset (with noise)

5. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. *Neural Computation* **9**(8), 1735–1780 (1997). <https://doi.org/10.1162/neco.1997.9.8.1735>
6. Ioffe, S., Szegedy, C.: Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* (2015)
7. Krizhevsky, A., Hinton, G.: Learning Multiple Layers of Features from Tiny Images. *Tech. rep., Citeseer* (2009). <https://doi.org/10.1.1.222.9220>
8. Krueger, D., Maharaaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N.R., Goyal, A., Bengio, Y., Courville, A., Pal, C.: Zoneout: Regulariz-

- ing RNNs by Randomly Preserving Hidden Activations. arXiv pp. 1–11 (2016). <https://doi.org/10.1227/01.NEU.0000210260.55124.A4>, <http://arxiv.org/abs/1606.01305>
9. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11), 2278–2323 (1998). <https://doi.org/10.1109/5.726791>
 10. Masters, D., Luschi, C.: Revisiting Small Batch Training for Deep Neural Networks. arXiv pp. 1–18 (2018). <https://doi.org/10.1016/j.biortech.2007.04.007>, <http://arxiv.org/abs/1804.07612>
 11. Paszke, A., Chanan, G., Lin, Z., Gross, S., Yang, E., Antiga, L., Devito, Z.: Automatic differentiation in PyTorch. *Advances in Neural Information Processing Systems* 30 (Nips), 1–4 (2017)
 12. Plappert, M., Houthoofd, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M.: Parameter Space Noise for Exploration. arXiv pp. 1–18 (2017), <http://arxiv.org/abs/1706.01905>
 13. Raś, Z., Wieczorkowska, A.: *Advances in Music Information Retrieval, Series: Studies in Computational Intelligence*, Vol. 274. Springer-Verlag, Berlin Heidelberg (2010)
 14. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **15**, 1929–1958 (2014). <https://doi.org/10.1214/12-AOS1000>
 15. Wan, L., Zeiler, M., Zhang, S., LeCun, Y., Fergus, R.: Regularization of neural networks using dropconnect. *Icml* (1), 109–111 (2013). <https://doi.org/10.1109/TPAMI.2017.2703082>