# Detecting Hazardous Events from Sequential Data with Multilayer Architectures

Karol Kurach and Krzysztof Pawlowski*

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
{kkurach,kpawlowski236}@gmail.com

**Abstract.** Multivariate time series data play an important role in many domains, including real-time monitoring systems. In this paper, we focus on multilayer neural architectures that are capable of learning high level representations from raw data. This includes our previous solution based on Recurrent Neural Networks with Long Short-Term Memory (LSTM) cells. We build upon this work and present improved methods that aim to achieve higher prediction quality and better generalization to other similar tasks. We apply new deep neural architectures, minimize feature engineering and explore different ways of model selection. In particular, our focus on architectures includes networks with attention mechanism and convolutional networks. We tackle overfitting challenges in a presence of concept drift.

**Key words:** Time Series Forecasting, Recurrent Neural Networks, Ensemble Methods, Deep Learning

## 1 Introduction

Predicting future level of methane concentration in a coal mine is an important task, on which depends an efficiency of mining operation[12]. We focus on this supervised learning classification problem, which was also the topic of the *IJCRS'15 Data Challenge: Mining Data from Coal Mines* competition[6]. As a part of an earlier solution[8] we developed methods based on Deep Neural Networks, that achieved competitive results. However, a few opportunities for improvement were left. In this paper, we extend our previous work and develop improved methods. In particular, the main contributions of this paper are the following enhancements:

1. **Minimal feature engineering** that allows better generalization and easier application to other domains and tasks,
2. **More advanced neural architectures** to allow models with even greater learning capacity and higher prediction quality,
3. **Explore different model selection methods** in attempt to reduce overfitting, and in general investigate what techniques work best for problems with concept drift and highly correlated data at the same time.

* Both authors contributed equally.

The rest of this paper is organized follows. In Section 2 we describe the problem. In Section 3 we describe our previous, „baseline" solution which is based on Deep Neural Networks. In Section 4 we present the improved methods. Finally, Section 5 concludes the paper and proposes the future work.

## 2 Problem Statement

We start this section describing the data used in the competition. Then, we document the evaluation procedure, including the target measure to be optimized. Finally, we review the most important challenges relevant to the task.

### 2.1 Data

The goal of *IJCRS'15 Data Challenge* competition was to predict dangerous level of methane concentration in coal mines based on the readings of $28$ sensors. It is an example of supervised learning classification task. The data is split into training and test set, where the training set contains $51700$ records and the test set contains $5076$ records.

Each record is a collection of $28$ time series – corresponding to $28$ sensors that are installed in the coal mine. The sensors record data such as level of methane concentration, temperature, pressure, electric current of the equipment etc. Each of the time series contains $600$ readings, taken every second, for a total of $10$ minutes of the same time period for each sensor. That gives a total of $16800$ features per record. The time periods described in the training data set overlap and are given in a chronological order. For the test data, however, the time periods do **not** overlap and are given in random order.

For each record in the training set, three labels are given. The test set is missing the labels – it is the goal of the competition to predict those values. Each label instance can be either *normal* or *warning*. Those levels signify the amount of methane concentration, as recorded by the three known sensors, named *MM263*, *MM264* and *MM256*. The second-by-second readings of those sensors are described in time series mentioned in the previous paragraph. The predictions are to be made about the methane level in the future - that is during the period between three and six minutes after the end of the training (time series) period. If the level of methane concentration reaches or exceeds $1.0$, then the corresponding label should be *warning*. Otherwise, it should be *normal*.

### 2.2 Evaluation

The submissions consist of three predictions of label values, made for each of $5076$ records in the test set. Each prediction is a number – a higher value denotes a higher likelihood that the true label value is *warning*. The score is defined as a mean of *area under the ROC curve*, averaged over the three labels.

Participants may submit their predictions during the course of the competition. Until the finish of the competition, the participants are aware only of the score computed over *preliminary test set* – a subset of the whole test set that contains approximately $20\%$ of the records. This subset is picked at random by the organizers and is fixed for all competitors but it is not revealed to the participants which of the test records belong to

it. The participants may choose a single final solution, possibly taking into the account the scores obtained on the preliminary test set. However, the final score is computed over the *final test set* – remaining approximately $80\%$ of the test data. This score is revealed only after the end of the competition and is used to calculate the final standings – the team with the highest score is declared the winner.

### 2.3 Challenges

We describe the two main challenges one needs to overcome when tackling this problem.

**Overlapping training periods.** Almost all adjacent training records overlap by $9$ out of the total $10$ minutes recorded in the time series. It clearly violates the assumption of i.i.d. that underpins the theoretical justification of many learning algorithms. In addition, due to the overlap, a classical cross-validation approach may result in splits very „similar" data across different folds and in turn yield over-optimistic estimates of the model performances.

**Concept drift.** Training and test data come from different time periods. The records in the training set are sorted by time, so it's easy to notice that there are very significant trends in the data that change along with the time. With test data samples taken at times belonging to a different interval than training samples, one can expect a severe concept drift - and indeed exploratory tests showed that classifier performance degrades on the test set, as compared to the same classifier's performance when it is evaluated on the interval of training data that was not used for its learning.

## 3 Baseline Model

In this section we present a brief summary of our solution to IJCRS'15 competition[8]. Our method consist of the two main parts: the Recurrent Neural Network with Long Short-Term Memory cells (which we refer to as "LSTM" later in the paper) and Deep Feedforward Neural Network ("DFNN"). Intuitively, the LSTM processes the whole input sequence, taking the order of sensor readings into account, while DFNN operates on the last 30 readings. Finally, we ensemble the predictions to improve the performance.

### 3.1 Long Short-Term Memory Model

Recurrent Neural Network (RNN) is a type of artificial neural network in which dependencies between nodes form a directed cycle. This kind of network is particularly suited for modeling sequential data, where the length of the input is not fixed or can be very long.

Long Short-Term Memory is an RNN architecture designed to be better at storing and accessing information than standard RNN [5]. LSTM block contains memory cells that can remember a value for an arbitrary length of time and use it when needed. It also has a special *forget gate* that can erase the content of the memory when it is no longer useful. All the described components are built from differentiable functions and trained during back-propagation step. The connections in LSTM cells vary slightly between implementations. The variant that we used is presented in Figure 1.

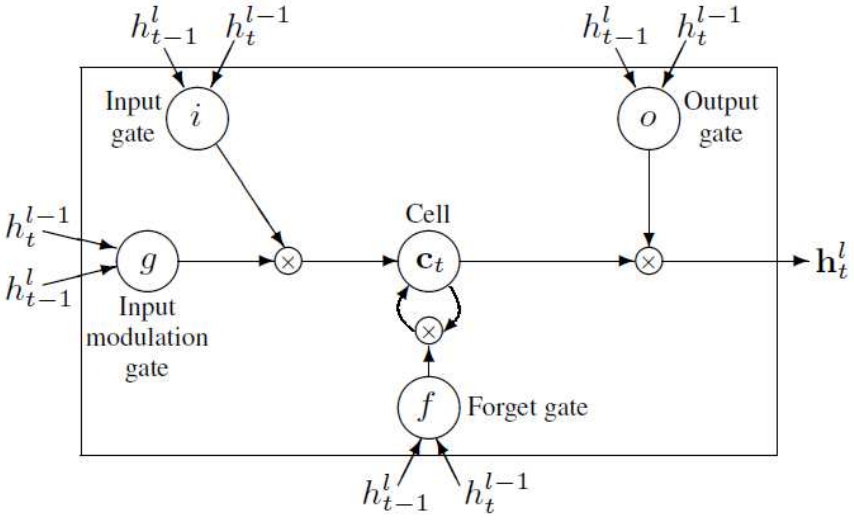


**Fig. 1.** A graphical representation of LSTM memory cells used in [16] and in our solution. The value $h_{t-1}^{l}$ represents the hidden state at the previous time step (same layer). The value $h_{t}^{l-1}$ represent the hidden state at the same time step (layer below).

**Architecture and training** The network that we used was unfolded to 60 time steps and trained using back-propagation through time[14]. The sensor values go through the hidden layer, which in this case consist solely of LSTM cells. At time step $t \in 1, .., 60$, the input for RNN are 28 average sensor values from seconds $[(t-1) * 10, t * 10)$.

After processing the whole sequence, the last network's hidden state encodes all sensor averages in the same order in which they were seen. On top of this we build a standard supervised classifier (Multi-Layer Perceptron in this case) that predicts the binary outcome. The *warning* class is assigned a value of 1.0 and *normal* class is assigned a value of 0.0. The loss function used in the final model was Mean Squared Error. It performed better than Binary Cross Entropy loss which is typically used for binary classification

The training is done using standard Stochastic Gradient Descent. To improve the convergence of this algorithm, the data was normalized to mean $0$ and variance $1$. Also, the training data is randomly shuffled after every training epoch. We initialize all the parameters by sampling from uniform distribution. To avoid *exploding gradient* problem, the gradients are scaled globally during training, so that their norm is not grater than $1\%$ of parameters' norm. All models were trained using Torch[3] on a machine with a GPU card.

### 3.2 DFNN Model

*Deep feedforward neural network* (DFNN) is an *artificial neural network* with multiple layers of hidden neurons. One notable difference between DFNN and LSTM network (described in Section 3.1) is that DFNN architecture does not contain recursive connections – instead, every neuron of the previous layer is connected with every neuron of the next layer. We train the DFNN model with a *backpropagation* algorithm[13] that uses *stochastic gradient descent* (with *mini-batch* and *momentum*) as an optimization procedure to minimize the *root mean squared error* between numeric predictions and the target values. To avoid overfitting to the training set we use two regularization[4] methods: ad-hoc *early stopping*[9] and *dropout*[10].

**Feature engineering** For DFNN model we apply the following preprocessing steps:

1. scale the readings (separately for each sensor) to *mean* $0$ and *standard deviation* $1$
2. transform the values with $x \rightarrow log(1 + x)$ function,
3. compute *mean* and *standard deviation* for every sensor, taken over the last $30$ readings (30-second period),
4. keep the last $20$ readings for the sensor that corresponds to the target label,
5. discard all the original features.

Such preprocessing reduces the number of features from $16800$ $(28 * 600)$ to just $76$ $(28 * 2 + 20)$.

**Training and parameter tuning** For each target label we train a different DFNN model and tune its parameters independently. We perform model selection to optimize the performace on two sets. Initially, we use the validation set created from $20\%$ of the original data and train on the remaining $80\%$. For the final submission, we use the preliminary test set. See Subsection 3.3 for the discussion of model selection challenges.

### 3.3 Ensemble

The Baseline Model is an ensemble of two submodels - LSTM described in Subsection 3.1 and DFNN described in Subsection 3.2. More precisely, the ensemble procedure computes rank for each of the submodels (independently) and then, for each record it takes the arithmetical average of the corresponding ranks as a final prediction. Table 1 illustrates the scores that particular models achieve on the preliminary test set.

**Table 1.** Model scores

| AUC score \ label | MM263 | MM264 | MM256 |
|---|---|---|---|
| LSTM | **0.9599** | 0.9560 | 0.9605 |
| DFFN | - | **0.9773** | 0.9602 |
| ensemble | - | 0.9722 | **0.9683** |

As the Baseline Model we combine the best-performing methods for each target label. That is, for label *MM263* we use LSTM, for label *MM264* we use DFNN and for label *MM256* we use the ensemble of LSTM and DFNN.

subsectionResults and challenges

Baseline Model achieves the final score of $0.94$ – recall that this score is computed on the final test set and revealed only after the end of competition. While the score on the final test set is good, we noticed that Baseline Model achieves a much better score of $0.9685$ on the preliminary test set.

Such decrease in performance is probably caused by overfitting and rather easy to explain: we performed model selection based on the preliminary test scores. Therefore, it is not surprising that the very best model, as judged by its performance on the preliminary test set, does not achieve a similar performance on the final test set.

The cross-validation procedure is a standard Machine Learning methodology to deal with a danger of overfitting. We did not use it for optimizing the Baseline Model, because we observed a significant concept drift between the training and test set (as stated in Subsection 2.3). Our hypothesis was that the data more similar to the final test set would give more useful estimates of the model's final performance.

During the contest, we did not test this hypothesis and the obvious challenge is to verify, and possibly to refute it. We address this challenge in Subsection 4.3.

## 4 Improved Methods

We improve the algorithm described in Section 3, particularly to address the challenges described in Subsection 3.3. To that end, we:

1. **Minimize feature engineering** for better generalization to other time series tasks and to make the improved model easier to apply,
2. **Introduce new architectures** to increase model's capacity for learning,
3. **Improve model selection** to reduce the overfitting effect.

The rest of this section describes these improvements in detail.

### 4.1 Minimize Feature Engineering

Recall from Section 3 that some components of our Baseline Model required a significant feature engineering, particularly as described in Subsection 3.2. Such approach, while effective in practice, makes model less generalizable as the feature engineering

steps depend on the problem at hand. If one could reduce this process to minimum, it would be much easier to apply the methods to other multivariate time series problems. That is what we aim for. To that end, we limit feature engineering only to the following two operations:

- **Data normalization**, in regards to *mean* and *standard deviation*. This is a standard Machine Learning procedure, and as such it should be applicable to almost any problem. Without data normalization, and thus with data at different scales, it could be difficult to control the optimization procedure and the regularization.
- **Downsampling the data**. That is, replacing groups of adjacent values in the time series with their average. We do not optimize the downsampling granularity for the best possible score, it is only set to fit in the memory and decrease computing time.

### 4.2 New Architectures

In Section 3 we described a solution that was an ensemble of Recurrent Neural Network and Deep Feedforward Neural Network. We investigate how we could improve on this architecture and train a powerful model without the need of ensembling techniques. To this end, we propose several modifications to our baseline network. This includes changes to the LSTM training procedure, adding attention mechanism and applying convolutional layers.

**LSTM improvements** In the RNN network described in the Section 3.1, the target binary value is predicted from $h_n$ (the last hidden state). This architecture unfortunately has some drawbacks: there is only one signal at the end of the sequence, which can be distorted during long back-propagation. As a result, the model will not learn much from the data at the begin of the sequence. To address this problem, we modify the architecture to predict the target label at every time step (from every $h_i$).

Another improvement is related to the fact that the network had only one hidden layer. An easy modification is to train the network with $l \geq 2$ vertical layers. Such a network has a capacity to express several different transitions happening at one time step.

**Attention mechanism** One of the recently introduced improvements for LSTMs is the attention mechanism[2]. The general idea behind this modification is based on the observation that LSTMs process long sequences with a limited memory. After processing the whole input data, the LSTM needs to encode all of it in the internal memory cells. For longer sequences, it is impossible to do without some information loss. The attention mechanism solves this problem by allowing the network to automatically search for parts of the input that are relevant. There are two main types of attention: *hard attention*[1], which is based on Reinforcement Learning techniques and *soft attention*[15] that is differentiable and trained by back-propagation. In our work, we focus on the latter and add a similar mechanism to the network.

**Convolutional networks** Deep Neural Networks achieve state of the art performance in image recognition[7]. The best performing models are built using convolutional and subsampling layers[11]. The weight sharing in convolution allows the network to significantly reduce the number of parameters to train, and as a result detect features irrespective of their position in the image. Inspired by 2D convolutions used for images, we train a 1D convolutions for the time series data. This can be compared to training moving window "patterns" in the sequential data, and searching for them at the test time. One of the main advantages of this approach is simplicity of training, compared to more complex recurrent networks.

### 4.3 Better Model Selection

Recall from Subsection 3.3 that Baseline Model experiences quite heavy drop in score when it is applied to the final test set, from the results it achieved on the preliminary test set. It is caused by fact, that we performed model selection with preliminary test set – a classical example of overfitting. We believe that, given heavy distribution drift, such choice had some justification. Yet it could be beneficial to compare different schemes of model selection, which we describe below.

**Standard k-fold cross-validation** is probably one of the most common Machine Learning method. However, in our problem the data overlap by $9$ out of $10$ minutes, which makes adjacent record very similar. If each record is assigned to a random fold, as standard k-fold cross validation requires, the procedure could favor models that are overfit to the training data.

**Deterministic cross-validation** One improvement to cross-validation is by partitioning the records in such a way, that the number of overlapping instances assigned to different folds is minimized. To that end a procedure similar to k-fold cross-validation can be used. Instead of assigning records randomly to the folds, one sorts all the records chronologically and then divides them into equal, continuous, folds. For example, in 5-fold procedure, the first fold contains the fifth of the earliest records, the second fold contains the second fifth of the earliest records and so on.

**Repeated deterministic cross-validation** Another improvement to deterministic k-fold cross-validation could be to perform it on a fixed number of continuous intervals, repeated and together covering the whole set. That makes records belonging to the same fold more diverse and still does not assign too many overlapping records to different folds.

The partitioning of methods mentioned above are illustrated in Table 2.

**Table 2.** Illustration of cross-validation partitionings for 20 records (5 folds). Records are sorted chronologically. Different letters denote assignment to different folds

| c-v method/record number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| standard (randomized) 5-fold | C | E | C | C | C | E | E | A | A | E | D | B | D | B | B | B | D | A | D | A |
| deterministic 5-fold | A | A | A | A | B | B | B | B | C | C | C | C | D | D | D | D | E | E | E | E |
| 2-repeated deterministic 5-fold | A | A | B | B | C | C | D | D | E | E | A | A | B | B | C | C | D | D | E | E |

**Rolling validation** In all methods described so far, future data can be used to train model that is then validated on past instances. Such inversion of chronology is not always desirable, especially if significant trends exist. In practical real-time application one could never predict the present based on the knowledge learned „from the future". To implement this constraint, and possibly increase accuracy of estimations, one can partition the records in the same way as for deterministic cross-validation. However, only folds earlier (chronologically) than validation fold are allowed to take part in training the model. As the estimate of the measure, one simply takes average of the results from all $(k-1)$ the runs. Table 3 demonstrates the partitioning.

**Table 3.** Illustration of rolling validation partitioning (5 folds). Records are sorted chronologically. *„T"* denotes that record is used for training; *„V"* denotes that record is used for validation; *„-"* denotes that record is not used at all

| run/record number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| run 1 | T | T | T | T | V | V | V | V | - | - | - | - | - | - | - | - | - | - | - | - |
| run 2 | T | T | T | T | T | T | T | T | V | V | V | V | - | - | - | - | - | - | - | - |
| run 3 | T | T | T | T | T | T | T | T | T | T | T | T | V | V | V | V | - | - | - | - |
| run 4 | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | V | V | V | V |

## 5 Conclusion

The *IJCRS'2015 Data Challenge* competition was an opportunity to develop and test methods of solving multivariate time series problem that features a concept drift. Our first, baseline solution based on Deep Neural Networks achieved a competitive score of $0.94$. We build upon it to further enhance the performance and generalization. The contribution of this paper are improved methods of approaching such tasks. In particular, we apply new deep neural architectures, minimize feature engineering and explore different ways of model selection. The immediate future work is to obtain precise results of the above methods and analyze them. The next step would be to evaluate how the described methods generalize to different datasets.

## References


1. Ba, J., Mnih, V., Kavukcuoglu, K.: Multiple object recognition with visual attention. CoRR abs/1412.7755 (2014), http://arxiv.org/abs/1412.7755
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. CoRR abs/1409.0473 (2014), http://arxiv.org/abs/1409.0473
3. Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: A matlab-like environment for machine learning. In: BigLearn, NIPS Workshop. No. EPFL-CONF-192376 (2011)
4. Girosi, F., Jones, M.B., Poggio, T.: Regularization theory and neural networks architectures. Neural computation 7(2), 219–269 (1995)
5. Graves, A.: Generating sequences with recurrent neural networks. CoRR abs/1308.0850 (2013), http://arxiv.org/abs/1308.0850
6. Janusz, A., Ślęzak, D., Sikora, M., Wróbel, L., Stawicki, S., Grzegorowski, M., Wojtas, P.: Mining data from coal mines: IJCRS'15 Data Challenge. In: Proceedings of IJCRS'15. LNCS, Springer (2015), in print November 2015
7. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
8. Pawłowski, K., Kurach, K.: Detecting methane outbreaks from time series data with deep neural netowrks. In: Proceedings of IJCRS'15. LNCS, Springer (2015), in print November 2015
9. Prechelt, L.: Early stopping-but when? In: Neural Networks: Tricks of the trade, pp. 55–69. Springer (1998)
10. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)
11. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. CoRR abs/1409.4842 (2014), http://arxiv.org/abs/1409.4842
12. Szlązak, N., Obracaj, D., Borowski, M., Swolkień, J., Korzec, M.: Monitoring and controlling methane hazard in excavations in hard coal mines. AGH Journal of Mining and Geoengineering 37 (2013)
13. Werbos, P.: Beyond regression: New tools for prediction and analysis in the behavioral sciences (1974)
14. Werbos, P.J.: Generalization of backpropagation with application to a recurrent gas market model. Neural Networks 1(4), 339–356 (1988)
15. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A.C., Salakhutdinov, R., Zemel, R.S., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. CoRR abs/1502.03044 (2015), http://arxiv.org/abs/1502.03044
16. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. CoRR abs/1409.2329 (2014), http://arxiv.org/abs/1409.2329