Detecting Methane Outbreaks from Time Series Data with Deep Neural Networks

Krzysztof Pawłowski(✉) and Karol Kurach

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

Abstract. Hazard monitoring systems play a key role in ensuring people's safety. The problem of detecting dangerous levels of methane concentration in a coal mine was a subject of IJCRS'15 Data Challenge competition. The challenge was to predict, from multivariate time series data collected by sensors, if methane concentration reaches a dangerous level in the near future. In this paper we present our solution to this problem based on the ensemble of Deep Neural Networks. In particular, we focus on Recurrent Neural Networks with Long Short-Term Memory (LSTM) cells.

Keywords: Machine learning · Recurrent neural networks · Ensemble methods · Time series forecasting · Hazard monitoring systems

1 Introduction

Working in a coal mine historically has been a very hazardous occupation. Over time the conditions and safety improved substantially, in part thanks to advances in technology. Despite remarkable progress, it is still one of the most dangerous professions [6]. One of the dangers present is a possibility of explosion caused by high concentration of methane in the air. Therefore, it is of utmost importance to monitor methane concentration levels and to ensure they are within a safe range. If the concentration levels reach a critical threshold, the production line needs to be shut down [23], which is a costly interruption.

On the other hand, mining effectiveness positively depends on the pace of methane emissions. Therefore, in order to maximize the efficiency, a fine balance needs to be made between mining effectiveness and the safety. If one could predict methane concentration in the future and tell when it is likely to be dangerously high, one could reduce the speed of the operation and try to avoid reaching dangerous levels. Design of such an efficient prediction algorithm is a goal of IJCRS'15 Data Challenge: Mining Data from Coal Mines competition [16]. The problem is an example of supervised learning classification task, with data given in a form of non-stationary multivariate time series. Between the training and test data sets there is a significant concept drift.

K. Pawłowski and K. Kurach—Both authors contributed equally.

Y. Yao et al. (Eds.): RSFDGrC 2015, LNAI 9437, pp. 475–484, 2015.
DOI: 10.1007/978-3-319-25783-9_42

Different methods of tackling similar problems have been proposed in the literature. One of them is an application of Deep Neural Networks. While artificial neural networks have been known for a very long time, in recent years they have achieved spectacular results in areas such as computer vision [17,18] or speech recognition [10,14], in part due to advances in computing hardware, such as Graphics Processing Units. Recurrent Neural Network is an architecture well-suited for the processing of time series data [4,8]. We aim to test how well such methods perform in the competition. In this paper we present our solution which uses Recurrent Neural Network with Long Short-Term Memory cells [15,27], Deep Feedforward Neural Network and ensembling [5] techniques.

The rest of this paper is organized as follows. In Sect. 2 we present the problem in detail. Section 3 describes the Recurrent Neural Network model we used. In Sect. 4 we present ensembling technique, detail Deep Feedforward Neural Network model and analyse the results. Finally, in Sect. 5 we conclude the paper and propose future work.

2 Problem Statement

In this section we describe 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.

2.1 Data

The goal of *IJCRS'15 Data Challenge* competition is 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 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. 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

The problem presents the following challenges.

Imbalanced Data. Only about 2 % of the labels in the training set belong to the *warning* class, while the remaining belong to the *normal* class. A trivial solution that predicts *normal* for every label achieves 98 % accuracy, obviously without having any practical significance (and with a bad 0.5 mean-ROC score). Unless special precautions are taken, methods that heavily optimize just the prediction accuracy can have significant problems with this task.

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 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.

Noisiness. Seemingly small "meaningful" changes in sensor readings happen at the 1-second resolution. Most changes at this interval appear to be random fluctuations. That, combined with a large amount (16800) of readings per record, poses a severe danger of overfitting.

Large Data Size. The whole training set consists of over $868,560,000$ values. Just storing it in a computer memory requires 3.5 gigabytes of memory, when using 32-bit floating point representation. Thus storage and computational costs can be a significant constraint.

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 Long Short-Term Memory Model

This section describes the Recurrent Neural Network with Long Short-Term Memory (LSTM) [15] model, which is a crucial part of our final solution.

First, we give some background about the recurrent networks and formally define the dynamics of LSTM. Next, we describe how the data was preprocessed. Finally, we present the network architecture and describe in more detail the training procedure.

3.1 Overview

Recurrent Neural Network (RNN) is a type of artificial neural network in which dependencies between nodes form a directed cycle. This allows the network to preserve a state between subsequent time steps. 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. Parameters in RNNs are shared between different parts of the model, which allows better generalization.

LSTM is an RNN architecture designed to be better at storing and accessing information than standard RNN [11]. 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 backpropagation step.

The LSTM networks recently achieved state of the art performance in many tasks, including language modeling [21], handwriting [12] or speech [10] recognition, and machine translation [22]. There are several variants of LSTM that slightly differ in connectivity structure and activation functions. Definition 1 describes the architecture that we implemented based on the equations from [27].

Definition 1. *Let $h_t^l \in \mathbb{R}^n$ be a hidden state in layer l of the network at step t. We assume that h_t^0 is the input at time t. Similarly, let $c_t^l \in \mathbb{R}^n$ be a vector of long term memory cells in layer l at step t. We define $T_{n,m} : \mathbb{R}^n \to \mathbb{R}^m$ to be an affine transform ($x \to Wx + b$ for some W and b) and $\odot$ be a element-wise multiplication. Then, LSTM is a transformation that takes* 3 *inputs (h_{t-1}^l, h_t^{l-1}, c_{t-1}^l) and computes* 2 *outputs (h_t^l and c_t^l) as follows:*

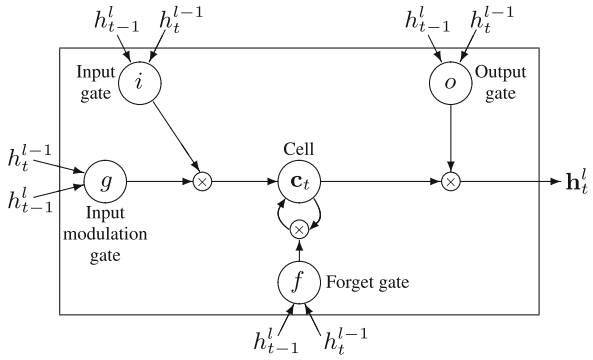

Fig. 1. A graphical representation of LSTM memory cells used in [27] and in our solution.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

In these equations, i, f, o, g are *input*, *forget*, *output* and *input modulation* gates respectively. The sigm and tanh functions are applied element-wise. This relationship is presented in Fig. 1.

3.2 Data Preprocessing

To address problems mentioned in Sect. 2.3, we apply several transformations to the original data.

First, for every sensor we normalize all of its readings to mean 0 and variance 1. This is a standard technique that improves convergence of gradient descent methods. To reduce overfitting and prevent algorithm from getting stuck in local minima, we shuffle training data after every epoch (one full pass over the data). We also address the problem of unbalanced classes by up-sampling with repetition from the *warning* class. It was done to ensure that at least 10 % of training examples are from the *warning* class.

Finally, we address the problem of large data. We modify the 600 input values from every sensor by grouping every 10 consecutive values and replacing them with their average.

3.3 Network Architecture and Training

As described in Sect. 3.2, we did not use the input sequence of 600 values directly but instead we grouped them. The network was unfolded to 60 time steps and trained using backpropagation through time [25]. This was mostly because of the practical purposes: for the original size the training took 10 times longer, we often exceeded the GPU memory and it was preventing us from testing bigger embedding or batch sizes. It could also negatively affect backpropagation, as the gradient was propagated for much longer. We tried other window sizes, but the value of 10 seemed to achieve a good trade-off between learning speed and quality.

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 network's hidden state h_n^1 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 algorithm and backpropagation through time. 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.

4 Final Ensemble Model

We improve the quality of our prediction by making an ensemble model. In this section we give an overview of the ensemble technique, describe the extra base learner that was used in conjunction with the method described in Sect. 3 and document the procedure we used to obtain the final prediction.

4.1 Ensembling Methods

Ensemble methods combine results of multiple "base" learning algorithms, to form a single prediction that often achieves a better performance than any of individual base algorithm alone [5]. To give an example, one of the simplest forms of ensembling is to average the predictions of base algorithms. Ensembling works the best when base algorithms are accurate and, importantly, diverse. Diversity can mean that errors produced by the base algorithms are slightly correlated. Intuitively, in the mentioned example of ensembling by averaging, the prediction quality increases because uncorrelated errors "average out". The more sophisticated ensembling algorithms include bagging [2], boosting [7] or stacking [26].

4.2 Deep Feedforward Neural Network as a Base Learner

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 Sect. 3) 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 use DFNN, together with LSTM, as base learners in an ensemble model. We train the DFNN with a *backpropagation* algorithm [24] 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 [9] methods: ad-hoc *early stopping* [19] and *dropout* [20].

Feature Engineering. To balance the impact of different features, further reduce overfitting and decrease the computational cost we preprocess the training and test data in the following way:

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

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

We also preprocess the training target values in the way described below. As mentioned in Subsect. 2.1, there are three labels for each training record with the *normal* class for future methane concentration levels under 1.0 and the *warning* class for methane concentration levels at or above 1.0. Recall from Subsect. 2.3, that an overwhelming majority of adjacent training periods overlap by exactly 9 minutes. That two facts together allow us to reconstruct the exact amount of methane concentration for a given record – to that end, we simply peek sensors values a few records in advance. We take reconstructed values as targets instead of the original label values, while also discarding all the training records for which such reconstruction is not possible (it turns out that less than 1000 records out of 51700 training records are discarded).

Training and Parameter Tuning. For each target label we train a different DFNN model and tune its parameters independently, to optimize the performance on:

– (initially) the validation set – created by us as 20 % of original training data. We train the model on the remainder of training data,

Table 1. Tuned parameter values for DFNN

Target label	MM263	MM264	MM256
activation	-	sigmoid	ReLU
layer sizes	-	76-15-5-2-1	76-25-7-3-1
dropout	-	none	.5
learning rate	-	0.1	0.1
mini-batch size	-	30	30
number of epochs	-	550	233

Table 2. 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**

– (finally) the preliminary test set. See Subsect. 4.4 for more discussion of model selection challenges.

We describe the final values of the most important parameters in Table 1. Parameters for the first target label (*MM263*) are omitted because for that label, the DFNN model fails to generate quality predictions for all the parameter combinations we tried.

4.3 Ensembling in Our Solution

Our final solution is an ensemble of two base models - LSTM described in Sect. 3 and DFNN described in Subsect. 4.2.

We decide to forgo the complex ensembling schemes that would require retraining of the models and perhaps additional parameter tuning. Instead, we consider a few simple averaging methods and finally we choose the method that gives the best AUC score – that is averaging the ranks of the base models' predictions. Table 2 illustrates the scores of particular models that are achieved on the preliminary test set. As the final submission we take the model that is a combination of the best-performing methods for each target label value. 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.

4.4 Results and Discussion

Our final ensemble model achieved a score of 0.94 and the 6th place in the competition. It is interesting to mention that on the preliminary test set, our

method obtained much higher score of 0.9685 which, if not decreased, would correspond to the 1st place.

Such a significant drop in score can be explained by overfitting. It was not a surprise, as we deliberately chose to perform model selection on the preliminary test set – that is, we submitted the model that achieved the best score on that set. Usually one would perform a cross-validation on the training set to perform model selection. The reason we decided not to, was because of the significant differences between the training and the test distributions (concept drift), as mentioned in Subsect. 2.3. We wanted to avoid a situation when the model is overly tuned to the training test and as a result does not generalize well on the test set. As we had only one shot for a final submission, it is not clear if the traditional (cross-validation) approach would have worked better – it can be an interesting topic for further research.

5 Conclusion

In this paper we presented our Deep Neural Network-based solution to *IJCRS'15 Data Challenge: Mining Data from Coal Mines* contest. It achieved a competitive score of 0.94 and the 6th place. The approach we developed should generalize well to other multivariate time series prediction problems. The obtained results confirm that methods based on Deep Neural Networks are not only effective for processing time series data, but also do not require extensive feature engineering to perform well. As expected, ensembling improves the quality of the prediction.

It would be interesting to see, if more advanced artificial neural network architectures, such as LSTMs with attention mechanism [1] or Neural Turing Machines [13], could achieve even better results. Another topic worth exploring are methods of handling the concept drift in context of parameter tuning and model selection, which was one of the main challenges in this task.

References

1. 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
2. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
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. Connor, J.T., Martin, R.D., Atlas, L.E.: Recurrent neural networks and robust time series prediction. IEEE Trans. Neural Netw. **5**(2), 240–254 (1994)
5. Dietterich, T.G.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, p. 1. Springer, Heidelberg (2000)
6. Donoghue, A.: Occupational health hazards in mining: an overview. Occup. Med. **54**(5), 283–289 (2004)
7. Freund, Y., Schapire, R., Abe, N.: A short introduction to boosting. J.-Jpn. Soc. Artif. Intell. **14**(771–780), 1612 (1999)
8. Giles, C.L., Lawrence, S., Tsoi, A.C.: Noisy time series prediction using recurrent neural networks and grammatical inference. Mach. Learn. **44**(1–2), 161–183 (2001)

9. Girosi, F., Jones, M.B., Poggio, T.: Regularization theory and neural networks architectures. Neural comput. **7**(2), 219–269 (1995)
10. Graves, A., Mohamed, A.R., Hinton, G.: Speech recognition with deep recurrent neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2013, pp. 6645–6649. IEEE (2013)
11. Graves, A.: Generating sequences with recurrent neural networks, CoRR abs/1308.0850 (2013). http://arxiv.org/abs/1308.0850
12. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **31**(5), 855–868 (2009)
13. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines, CoRR abs/1410.5401 (2014). http://arxiv.org/abs/1410.5401
14. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural comput. **9**(8), 1735–1780 (1997)
16. Janusz, A., Ślęzak, D., Sikora, M., Wróbel, L., Stawicki, S., Grzegorowski, M., Wojtas, P.: Mining data from coal mines: IJCRS 2015 data challenge. In: Proceedings of IJCRS 2015. LNCS, Springer (2015), in print November 2015
17. 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)
18. Le, Q.V.: Building high-level features using large scale unsupervised learning. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2013, pp. 8595–8598. IEEE (2013)
19. Prechelt, L.: Early stopping - but when? In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, p. 55. Springer, Heidelberg (1998)
20. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
21. Sundermeyer, M., Schlüter, R., Ney, H.: Lstm neural networks for language modeling. In: INTERSPEECH (2012)
22. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
23. Szlązak, N., Obracaj, D., Borowski, M., Swolkień, J., Korzec, M.: Monitoring and controlling methane hazard in excavations in hard coal mines. AGH J. Min. Geoengineering **37**, 105–116 (2013)
24. Werbos, P.: Beyond regression: new tools for prediction and analysis in the behavioral sciences, Ph.D. thesis, Harvard University, Cambridge (1974)
25. Werbos, P.J.: Generalization of backpropagation with application to a recurrent gas market model. Neural Netw. **1**(4), 339–356 (1988)
26. Wolpert, D.H.: Stacked generalization. Neural Netw. **5**(2), 241–259 (1992)
27. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization, CoRR abs/1409.2329 (2014). http://arxiv.org/abs/1409.2329