# Deep Learning with Hybrid Noise Reduction Architecture: Time Series Application

Vanessa Haykal, Hubert Cardot, Nicolas Ragot

# Deep Learning with Hybrid Noise Reduction Architecture: Time Series Application

Vanessa Haykal , Hubert Cardot, Nicolas Ragot

*Université de Tours, LIFAT (EA 6300), France*

*{hubert.cardot / nicolas.ragot}@univ-tours.fr*

**Abstract**

Time series forecasting models have fundamental importance to various practical domains. It is desirable that these methods can learn non-linear dependencies and have a high noise resistance. In this paper, we propose a novel architecture using deep learning to address this challenge. We have adapted a new hybrid noise reduction architecture that use recursive error segments for learning and adjusting the predictions. The solution is based on a simultaneous fusion between the outputs of a Convolutional Neural Network (CNN) and a Long Short Term Memory (LSTM) network. This novel model is able to capture different types of properties which combination can substantially outperform their separate use. Applications involving electricity and financial datasets illustrate the usefulness of the proposed framework.

*Keywords:* Deep learning, convolutional neural network, long short-term memory, noise reduction, recursive error segments, time series forecasting.

## 1. Introduction

The main aim of time series modeling is to carefully collect and rigorously study the past observations to develop an appropriate model that describes the inherent structure of the series. Time series forecasting thus can be termed as the act of predicting the future by understanding the past. Due to the indispensable importance of time series forecasting in numerous practical fields

such as economics, finance, science and engineering, proper care should be taken to fit an adequate model to the underlying signals. It is obvious that a successful time series forecasting depends on an appropriate model fitting. Researchers have done many efforts over years to develop efficient models and to improve the forecasting accuracy. As a result, various important time series forecasting models have evolved in literature.

The well-known linear autoregressive moving average (ARMA) models [1] are examples of statistical formalism for time series analysis and forecasting. Many extension of ARMA models have failed to capture the behavior of complex series such as the generalized autoregressive conditional heteroskedasticity model [2, 3]. Therefore, researchers have introduced some machine learning techniques including Artificial Neural Network (ANN) [4], Support Vector Machine (SVM) [5], and Decision Trees [6] etc. The main challenge of these models has been to adjust their parameters like connection weights, architecture and learning algorithms. In this way, it has been usual to spend considerable computational resources in order to select the best model. These strategies commonly use some evolutionary computation techniques such as genetic algorithms [7] and particle swarm optimization [8, 9].

The turning point starts when introducing deep learning approaches that recently become very popular in various domains. The recurrent neural networks (RNN) models [10], for example, become very useful in recent natural language processing (NLP) research. Long Short Term Memory (LSTM) [11] and the Gated Recurrent Unit (GRU) [12] are considered as variants for RNN. They have significantly improved the state-of-the-art performance in machine translation, speech recognition and other NLP tasks as they can effectively capture the meanings of words based on the long-term and short-term dependencies among different documents [13, 14]. In the field of computer vision, the convolution neural network (CNN) models [15, 16] have shown outstanding performance by successfully extracting local and shift-invariant features at various granularity levels from input images. But deep neural networks have also received an increasing amount of attention in time series analysis [17, 18].

In this paper, we use deep learning models inside a hybrid noise reduction architecture to improve the forecasting results. The objective from this study is to maximize the model properties and minimize its error by using a novel hybrid structure. On the one hand, we use convolutional layers on the inputs to extract local features. On the other hand, we calculate the error segments for noise reduction based on a recursive residual method, and then we stack memory layers for long short-term dependencies. All the procedures are built on non-linear activation functions. The results of this work reveal that the proposed model is able to adjust the performance more efficiently than other approaches.

The rest of this paper is organized as follows: Section 2 introduces the basics of forecasting models used in this study namely the ARMA extension models, convolutional neural networks, and long short-term memory networks. Section 3 shows the key novelty of our study by explaining the properties and the structure of our proposed model. The overall experimental results are detailed in Section 4 where all models are compared based on a performance indicator. Finally, some concluding remarks are given in Section 5.

## 2. Forecasting Models

### *2.1. ARMA extensions*

Due to the linear limitations [19, 20] of the autoregressive moving average (ARMA) model, researchers have proposed different extensions for this model in order to improve its performance. In ARMA model, the future value of a variable is assumed to be a linear function of several past observations with their errors. It is modeled by the following equation:

$$\hat{x}_t = \sum_{i=1}^{p} \varphi_i x_{t-i} + \sum_{j=1}^{q} \theta_j (x_{t-j} - \hat{x}_{t-j})$$

where $\hat{x}_t$ is the predicted value at time $t$ and $x_{t-i}$ are the lagged inputs. The integers $p$ and $q$ are often referred as orders of the model. If $q = 0$, then the equation becomes an AR model of order $p$. When $p = 0$, the model reduces to

a MA model of order $q$. In some papers, the equation is formulated using the backshift operator $(B)$, where we consider $B^i x_t = x_{t-i}$. Many ARMA extensions models have been used in literature to capture the nonlinear characteristics of the data. Different studies [21, 22] work on the noise reduction architecture (NRA) illustrated in Figure 1. Many efforts in this direction have shown an improvement in the results by combining linear with non-linear models such as ANN and SVM etc.

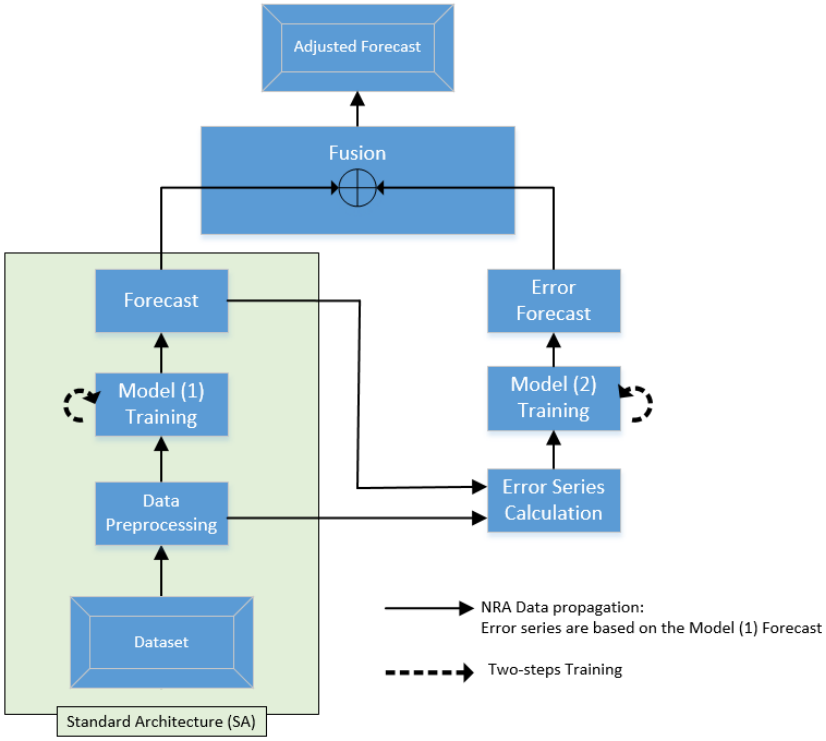


Figure 1: The Standard Architecture (SA) as part of the Noise Reduction Architecture (NRA)

The NRA works on a two-steps training procedure for correcting the forecast model. The first step reflects the elaboration of the trained forecaster, generally used as a Standard Architecture (SA), and the second step corresponds to the recursive error calculation procedure where we separately train the second model from the first model. The objective from this step is to detect any

new tendency of the phenomena not already captured by the original forecasting model. Consequently, the prediction of the recursive error series is used to correct the future values of the forecaster that deals with linear and nonlinear patterns.

### *2.2. Convolutional Neural Network*

In the last few years, deep neural networks have led to breakthrough results on a variety of pattern recognition problems, such as computer vision and voice recognition [23, 24, 25]. Convolutional neural network (CNN) extract features from input signal through a convolution operation with a filter (or kernel). The activation unit represents the result of the convolution of a kernel with an input signal. Each activated unit is connected only to a local region of the input. These units form a feature map as shown in Figure 2. These activated units filtered by

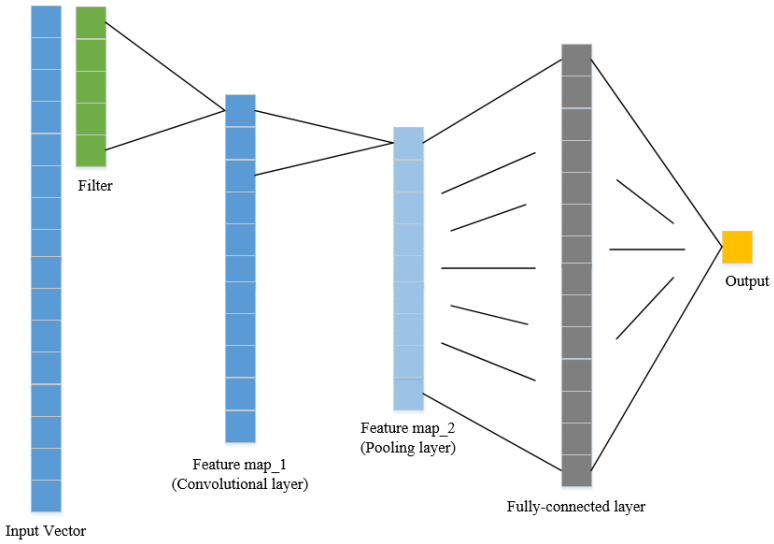


Figure 2: Overview of a convolutional neural network with 1D input

the same kernel share the same parameterization (weights and bias). The shared weights aspect of the convolutional layer reduces the total number of learnable parameters resulting a more efficient training process [26, 27]. To understand

the CNN, it might be sufficient to understand the formulation of convolution operator. A discrete convolution of two one-dimensional (1D) signals $f$ and $g$ is defined as:

$$(f * g)(i) = \sum_{j=-\infty}^{\infty} f(j)g(i-j)$$

where depending on this equation, nonexistent samples in the input may be considered as zero-padding values. A convolution at point $i$ is computed by shifting the signal $g$ over the input $f$ along $j$ and computing the weighted sum of the two signals. Moreover, we can also add some transfer functions such as the sigmoid and the rectifier linear unit (Relu) to capture the non-linear relationships. Another component of the CNN architecture is the pooling layer. It performs a down-sampling operation along the spatial dimensions. It is common to periodically insert it in-between successive convolutional layers using the *max* or *mean* operators. Its function is to progressively reduce the spatial size of the representation, hence to reduce the amount of parameters and computations in the network. Finally, the last layers are fully-connected and correspond to traditional ANN. The input to the first fully-connected layer is the set of all feature maps at the previous layer.

### *2.3. Long Short Term Memory Network*

Long Short Term Memory networks (LSTM) are a special kind of Recurrent Neural Network (RNN), they are among the most widely used models in Deep Learning for NLP today. LSTM are explicitly designed to avoid the long-term dependency problem through gating mechanism [28, 29, 30]. Remembering information for long periods is practically their default behavior.

The LSTM structure depicted in Figure 3 is implemented through the fol-

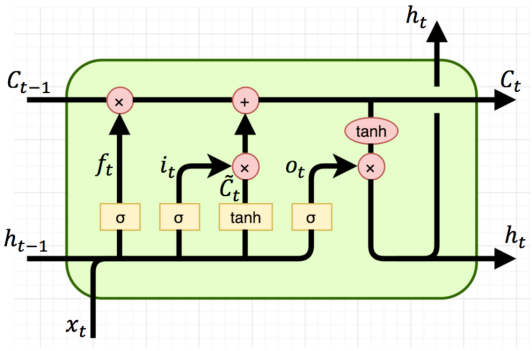


Figure 3: The components of the LSTM unit [29]

lowing equations:

$$
\begin{aligned}
f_t &= \sigma(w_{fh} h_{t-1} + w_{fx} x_t + b_f) \\
i_t &= \sigma(w_{ih} h_{t-1} + w_{ix} x_t + b_i) \\
\tilde{c}_t &= \tanh\left(w_{gh} h_{t-1} + w_{gx} x_t + b_g\right) \\
c_t &= f_t \times c_{t-1} + i_t \times \tilde{c}_t \\
o_t &= \sigma(w_{oh} h_{t-1} + w_{ox} x_t + b_o) \\
h_t &= o_t \times \tanh\left(c_t\right)
\end{aligned}
$$

Here, $i_t$, $f_t$, $o_t$ are called the input, forget and output gates, respectively. Note that they have the exact same equations, just with different weights ($w_{.h}$ is the recurrent connection at the previous hidden layer and current hidden layer, $w_{.x}$ is the weight matrix connecting the inputs to the current hidden layer). The input gate defines how much of the newly computed state for the current input you want to let through. The forget gate defines how much of the previous state you want to let through. Finally, the output gate defines how much of the internal state you want to expose to the external network. $\tilde{c}_t$ is a candidate hidden state that is computed based on the current input and the previous hidden state. $c_t$ is the internal memory of the unit. It is a combination of the previous memory, multiplied by the forget gate, and the newly computed hidden state, multiplied by the input gate. Thus, intuitively it is a combination of how we want to combine previous memory and the new input. $h_t$ is output hidden

state, computed by multiplying the memory with the output gate. Denote $\times$ as elementwise multiplication and $b_{\cdot}$ the bias term. The $\sigma$ and tanh represent the activation functions of the LSTM [31, 32, 33].

## 3. Proposed model

Despite the availability of numerous time series models, the accuracy of time series forecasting for many applications are not up to the needs. Consequently, many researches have argued that hybrid models could improve predictive performance. Indeed, their aim is to reduce the risk of using an inappropriate model by combining several models. Typically, the motivation for combining models comes from the assumption that a single one may not be sufficient to identify all the characteristics of the time series. A hybrid methodology [21, 22, 34, 35] that has different modeling capabilities is a good strategy for practical use. By combining interacting models, different aspects of the underlying patterns may be captured.

### *3.1. Model structure overview*

In this paper, we use different deep learning models fused inside a new hybrid noise reduction architecture (HNRA) with recursive forecast error. Recursive error prediction methods are frequently used in the field of ARMA extensions, as discussed in section 2.1. Based on this method, we propose a new model that generalize the principle of noise reduction and add additional properties based on deep learning. Figure 4 shows the structure of this novel hybrid noise reduction architecture to clarify the difference between our contribution and the state-of-the-art procedure as shown previously in Figure 1.

Firstly, the deep learning models of Figure 4 replace the classical linear models or non-linear algorithms used in Figure 1. CNN is applied to the sequential inputs using local receptive fields by taking into account the convolution operator and the LSTM is used for error correction mechanism to capture long-term dependencies. Secondly, in our model, the error correction mechanism is based

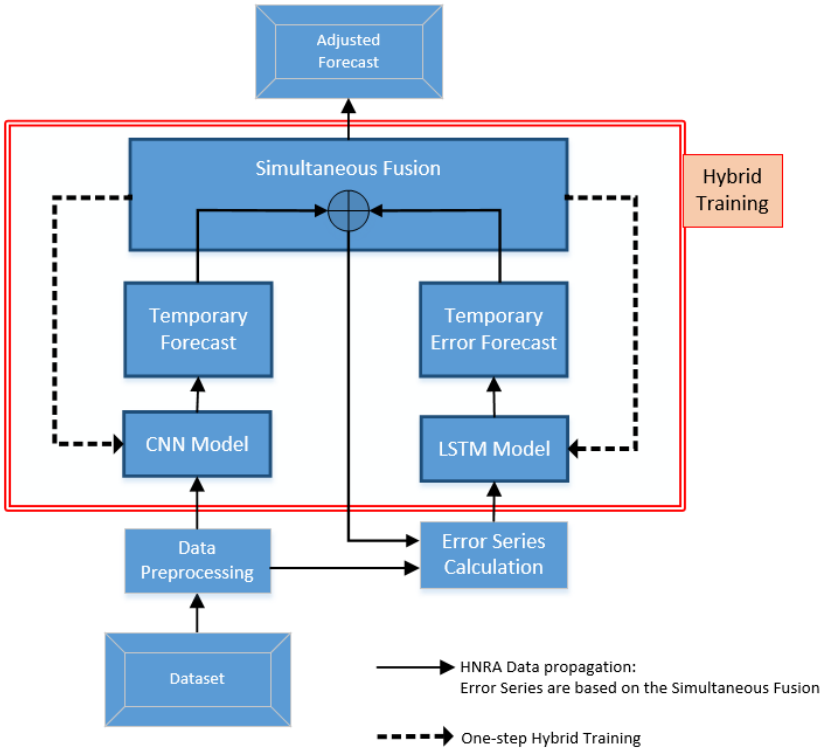


Figure 4: Novel Hybrid Noise Reduction Architecture (HNRA)

on the simultaneous fusion between the past inputs predictions and their lagged errors predictions, whereas in classical systems, as mentioned in Figure 1, the error series are based on the forecast of the first model only. Thirdly, in our model, both autoregressive inputs and recursive errors parameters are learnt jointly using one-step hybrid training procedure. During the training phase, we simultaneously optimize the cost function that integrates the whole parameters of both models, which differs from the classical recursive forecast error architecture (NRA) where a two-steps training is used.

In Table 1, we compare the different properties of our architecture against state-of-the-art models. The checklist refers mainly to ARMA extensions and to CNN-LSTM models. Indeed, in some recent literature review, we found that

| Properties | HNRA-CNN-LSTM | CNN-LSTM [34, 35] | ARMA Extensions [21, 22] |
|---|---|---|---|
| Sequential inputs | × | × | × |
| Error series | × | | × |
| Convolutional layers | × | × | |
| Memory layers | × | × | |
| Non-linear functions | × | × | × |
| Principle | HNRA | SA | NRA |

Table 1: Checklist comparison with our HNRA-CNN-LSTM

different articles use the principle of SA (Standard Architecture) with CNN and LSTM as a combination of successive layers where the output of CNN is considered as the input of LSTM. We mention it in this checklist to show exactly the difference between them. Contrarily, our model takes into consideration the error series properties which it is not the case of the SA-CNN-LSTM model. It is clear that the idea of the error segments is derived from the ARMA extensions using NRA architecture. By combining all the sub-models properties, we can get the NRA-CNN-LSTM and the HNRA-CNN-LSTM models as new contributions to explore. HNRA difference relies mainly on the one-step hybrid training and on the error calculation based on the final output as explained in Figure 1 and 4.

### *3.2. Model formulation*

The proposed model formulation is based on ARMA extensions, and it is generalized by using advanced non-linear modeling techniques, thanks to deep learning. The system could be presented as follows:

$$\begin{cases} \hat{x}_t = f_{HNRA}(f_{CNN}(x_{t-1}, ..., x_{t-p}), f_{LSTM}(e_{t-1}, ..., e_{t-q})) \\ e_t = x_t - \hat{x}_t \end{cases}$$

The details of the system formulations are given in the Appendix. The resulting output $\hat{x}_t$ of the proposed model is:

$$\hat{x}_t = \hat{x}_t^{(L'')} \oplus \hat{e}_t^{(L')} \qquad \longleftrightarrow \qquad \begin{cases} \hat{x}_t = \hat{x}_t^{(L'')} + \hat{e}_t^{(L')} \\ e_t = x_t - \hat{x}_t \end{cases}$$

where $\hat{x}_t^{(L'')}$ corresponds to the final layer output of the CNN and $\hat{e}_t^{(L')}$ is the final forecasted error derived from LSTM. Then, the calculated error $e_t$ is updated and reused for the next prediction.

### *3.3. Model diagram & learning process*

In order to facilitate the understanding of the model during the training phase, we draw its diagram in Figure 5 to describe the propagation steps of its mechanism. The main objective is to predict $\hat{x}_t$ at time $t$. We start with a window $[x_{t-1}, ..., x_{t-p}]$, it is an input signal containing the past observed values. In the diagram, it is expressed with an start point $x_{t-p}$ followed by a series of forward shift operators ($B^{-1}$) up to $x_{t-1}$. A filter with size $z$ slides over the input and computes the dot product between the window and the shared weights of CNN. A pooling layer down-samples the convolved array by averaging each $k$ neurons. Then, a fully connected layer is used to get the temporary output $\hat{x}_t^{(L'')}$ of CNN that is fused with error forecast $\hat{e}_t^{(L')}$. Initially, the error series are set to zero, then they are updated on each iteration. The final output $\hat{x}_t$ is then subtracted from the true observed value $x_t$ at the next step in order to get the forecast error $e_t$. We add backshift operators ($B$) on these errors to get their lagged window $[e_{t-1}, ..., e_{t-q}]$. Then, we stack a memory layer to the error segments for long short-term dependencies modeling (LSTM) which output is $\hat{e}_t^{(L')}$ used to compute $\hat{x}_t$.

In our example, we design the diagram with $p$ equal to six and $q$ equal to four, so the overall inputs of the model are equal to ten. We use five forward shift operator to get the lagged window of the observed values, and a filter slide gradually on the whole input signal with a length equal to three. Once we calculate the current error, we use four backshift operators to get the lagged window of the previous error values. We stack to it four LSTM units with a given initial condition for $c_0$ and $h_0$. The blue arrows represent the final regression value of each model, and the black arrow gives the final output prediction. We note that all the calculations are built on non-linear mathematical functions such as hyperbolic tangent.

Once the structure of the system is specified, it is ready for the training phase. The estimators are optimized by minimizing the cost function that combines all the parameters of the CNN and LSTM models. In our case, we use the mean squared error (MSE) as a cost function. The computation of this criterion is given as follows:

$$MSE = \frac{\sum_{n=1}^{N} (x_t - \hat{x}_t)^2}{N}$$

where the expected output is compared with the desired output. After that, the weights are adjusted accordingly using the stochastic gradient descent optimization. This process is repeated among the whole sample size $N$ with a specific number of epochs.

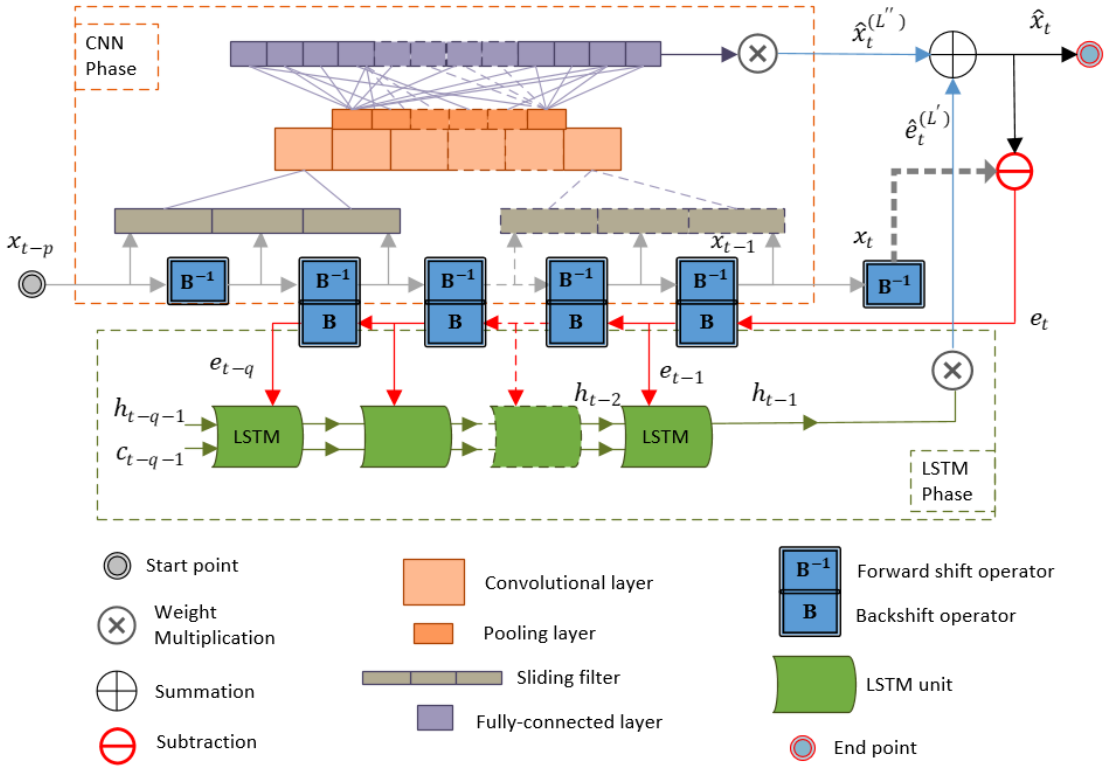


Figure 5: Novel approach diagram used during the learning phase

## 4. Experimentation

### *4.1. Data description & training*

Based on these references [35, 36], we take into account the same data samples to compare our results. For benchmarking, we used the Carbon Copy model. Its predictions are exactly the same value as the previous one. This simple model is not often used in literature, whereas it provides already interesting behavior since the original and predicted curves are close to each others.

In our study, we work on three datasets that are shown in Figure 6. The first dataset is the daily closing price of the Nikkei 225 index. The observations from 1984 to 2014 are used as an in-sample set and from 2014 to 2015 are considered as the out-sample set. The second dataset represents the electrical consumption that records the market capacity price at each hour in 2017 as shown in the second subplot of Figure 6. We can see that the series has large spread and sudden peak values occurrence that increase the forecasting difficulty of the algorithms. The sample contains three months as training data, and one month as testing data. Furthermore, the third dataset is the daily closing price of the S&P 500 index. The first observations from 1980 to 2014 are used as a training set. The testing set starts from 2014 to 2015. For all the datasets, we get the stationary phase by differentiating and normalizing on each signal. After that, we split the training sample into two parts to get a validation set. The goal is then to select the model that performed the best. We use the grid search method for hyper-parameters optimisation. It's a searching algorithm that intends to find the best combination among all the hyper-parameters. We train each of these models and evaluate them using the validation set. Indeed, deep learning models have different hyper-parameters that should be tuned in order to adjust the results. According to the following experimentation, the number of layers and units have an effective influence on the overall model performance.

For the Nikkei dataset, the obtained model topology has three convolutional layers where each one is followed by a mean pooling layer. We use 82, 64 and

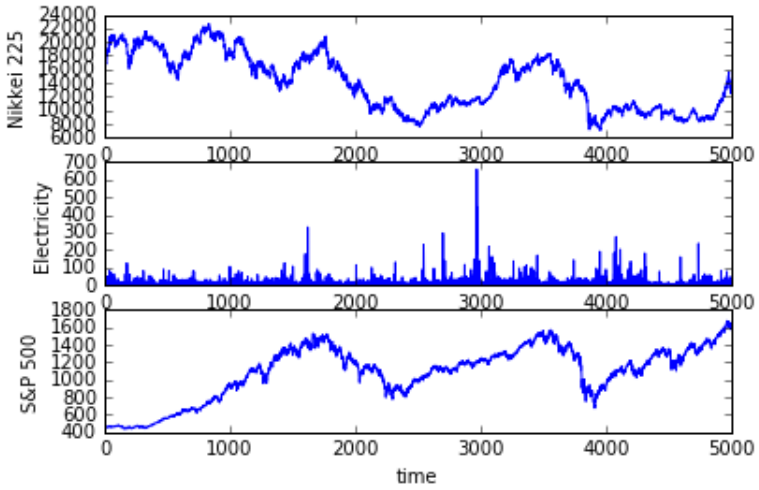


Figure 6: Overview of the three datasets: Nikkei 225, Electricity & S&P 500 (from up to down)

20 filters respectively. A fully connected layer is added to the architecture with 30 neurons. Using the temporary forecast of CNN as discussed previously in section 3, we can make the calculation of errors and take into account twelve lagged values to create the error segments. We stack one LSTM layer with 43 units. The model use a window with twenty-two observations.

For the Electricity dataset, the model is based on the past twenty-five sequential observations. In our topology, we use two convolutional layers where each one is followed by a mean pooling layer. We use 64 and 32 filters respectively. A fully connected layer is added to the architecture with 32 neurons. Using the temporary forecast of CNN, we can make the calculation of errors and take into account fourteen-lagged values to create the error segments. We add two LSTM layers with 25 and 8 cells respectively.

For the S&P 500 dataset, the model is based on the past thirty-one sequential observations. In our topology, we use three convolutional layers where each one is followed by a mean pooling layer. We use 128, 32 and 32 filters respectively. A fully connected layer is added to the architecture with 50 neurons. Using

the temporary forecast of CNN, we can make the calculation of errors and take into account seventeen-lagged values to create the error segments. We add two LSTM layers with 30 and 10 cells respectively.

### *4.2. Results*

In these experiments, we used the root mean squared error (RMSE) as a performance indicator already mentioned in Section 3.3. A small value of RMSE means that the predicted time series values are closed to the actual values. Based on the test sample and the optimal models described in Section 4.1, we produced the summary of the RMSE values in Table 2. In our proposed model, we step through the test sample to make future predictions by multiplying the CNN coefficients with the observed test signals and the LSTM coefficients with the error test window. For each new input series, we calculate the actual error and update the error series lag values so that we can estimate the error at the next time step. Consequently, for the new error segment, we mainly replace the previous estimated errors by their true observed values as explained in [37].

As shown in Table 2, the RMSE of Nikkei 225 decreases from 185.37 as obtained in [36] to 177.3. Even when comparing our model to recent models like CNN-LSTM, we found a decline in the error from 178.47 to 177.3 and like NRA-CNN-LSTM, it is reduced from 178.29 to 177.3. For the electricity dataset, we find a slight decrease in the RMSE comparing to [35] from 16.89 to 16.53. The RMSE of the NRA-CNN-LSTM corresponds to 16.8, and then it diminishes to 16.53 when replaced by the HNRA. Since the series has a high peaks, applying a denoising technique on inputs may help the deep learning algorithms to improve their performance. For S&P 500 in [36], the RMSE drops from 14.8 to 12.4. Even when comparing our model to already existing models like CNN-LSTM, we found a decrease in the error from 13.19 to 12.4 and like NRA-CNN-LSTM, the RMSE goes down from 12.86 to 12.4. These results illustrate the interest of the hybrid noise reduction architecture on this experimentation.

In Figure 7, we visualize the RMSE percentage change of all the models among all the datasets. It is calculated by differencing and dividing the new

| | Models | RMSE | Percentage Change (%) |
|---|---|---|---|
| Nk225 | Carbon Copy | 188.17 | 1.51 |
| | GA-WA [36] | 185.37 | 0 |
| | LSTM | 184.60 | -0.41 |
| | CNN | 179.79 | -3.01 |
| | CNN-LSTM | 178.47 | -3.72 |
| | NRA-CNN-LSTM | 178.29 | -3.81 |
| | HNRA-CNN-LSTM | 177.30 | -4.35 |
| Electricity | Carbon Copy | 20.57 | 21.78 |
| | RF [35] | 19.46 | 15.21 |
| | LSTM [35] | 17.35 | 2.72 |
| | CNN [35] | 17.30 | 2.42 |
| | CNN-LSTM [35] | 16.89 | 0 |
| | NRA-CNN-LSTM | 16.80 | -0.53 |
| | HNRA-CNN-LSTM | 16.53 | -2.13 |
| S&P500 | Carbon Copy | 15.17 | 2.5 |
| | GA-WA [36] | 14.80 | 0 |
| | LSTM | 14.67 | -0.87 |
| | CNN | 14.01 | -5.33 |
| | CNN-LSTM | 13.19 | -10.87 |
| | NRA-CNN-LSTM | 12.86 | -13.10 |
| | HNRA-CNN-LSTM | 12.40 | -16.21 |

Table 2: RMSE values for different models on the test samples

RMSE result by the already existing one. The percentage is equal to zero for the best existing model in the literature review. All the models that are based on deep learning give a high decrease compared to other classical artificial intelligence models. Many adjustments have been made using the NRA and HNRA principles. Following the histograms, the HNRA outperforms the NRA in the three cases. The percentage change has an improvement of 4.35% for Nikkei 225. Further, it has a slight enhancement of 2.13% for the electricity dataset and a significant amelioration of 16.21% for the S&P 500.

As a complement of information, we notice that applying the auto-correlation function on the final errors of the carbon copy (worst RMSE) and the HNRA-

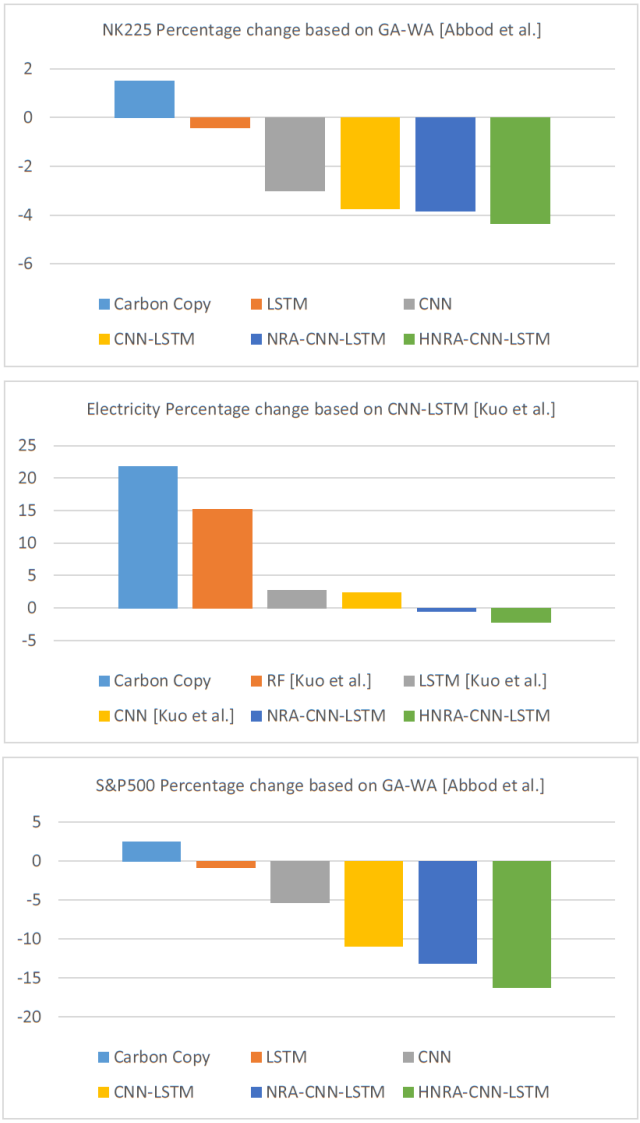


Figure 7: RMSE percentage change comparison for each dataset

CNN-LSTM gives us values that tend to zero for both models. Consequently, we can assume the existence of white noise due to the closeness of the predicted and original series on both forecasters. It is important to calculate it to verify this assumption, but it is not considered as a sufficient indicator in our experiment

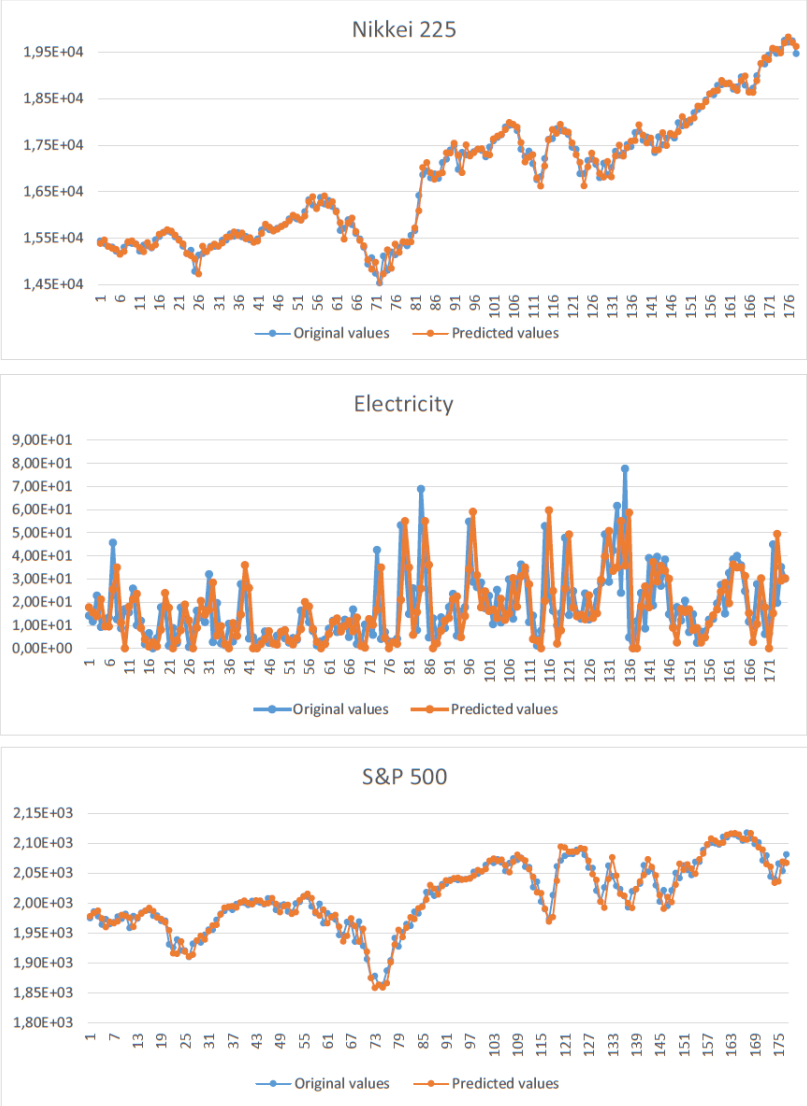


Figure 8: Original versus predicted time series for each dataset

to evaluate the overall forecast system, since we get null values to the worst model as well as to the best model. In this case, the RMSE is more efficient and commonly used to evaluate the results. In Figure 8, we plot the original and the predicted series for each test sample of the three datasets.

## 5. Conclusion

The objective of this study is to maximize the model properties and minimize its error by using a novel hybrid structure known as the HNRA-CNN-LSTM. The approach combines CNN and LSTM using a hybrid noise reduction architecture. It shows how to model and use recursive errors to correct predictions and improve the overall time series forecast performance. The hybrid model exploits the strength of CNN model as well as LSTM model to determine different patterns. CNN uses the convolution operator to extract local important features from the input data, whereas the long-term dependencies of LSTM can correct these predictions. Both sequential inputs and error series are learnt simultaneously to minimize the global loss function of the model.

Based on the RMSE values, we can conclude that deep learning approaches are more efficient than existing reference models such as random forest or support vector machines. Further, the noise reduction architecture is better than the standard approaches since it corrects and minimizes the error. Hence, the NRA-CNN-LSTM outperforms the single CNN and LSTM forecasting methods. It integrates non-linear models that are trained separately. We also conclude that our HNRA model outperforms the state-of-the-art approaches by achieving the smallest RMSE values for the three datasets. It can be adequate to work on it for complex non-linear problems. Its hybrid training mechanism is more efficient than the two-steps procedure training used in NRA. For future research, applying some denoising techniques [38, 39, 40] on the input data may be convenient to adjust the results.

## References


[1] G. E. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung, Time series analysis: forecasting and control, John Wiley & Sons, 2015.

[2] R. F. Engle, Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation, Econometrica: Journal of the Econometric Society (1982) 987–1007.

[3] T. Bollerslev, Generalized autoregressive conditional heteroskedasticity, Journal of econometrics 31 (3) (1986) 307–327.

[4] G. Zhang, B. E. Patuwo, M. Y. Hu, Forecasting with artificial neural networks:: The state of the art, International journal of forecasting 14 (1) (1998) 35–62.

[5] J. A. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural processing letters 9 (3) (1999) 293–300.

[6] M. S. Lauretto, B. B. Silva, P. M. Andrade, Evaluation of a supervised learning approach for stock market operations, arXiv preprint arXiv:1301.4944, 2013.

[7] Z. Michalewicz, Evolution strategies and other methods, in: Genetic Algorithms+ Data Structures= Evolution Programs, Springer, 1996, pp. 159–177.

[8] P. S. d. M. Neto, G. G. Petry, R. L. Aranildo, T. A. Ferreira, Combining artificial neural network and particle swarm system for time series forecasting, in: Neural Networks, 2009. IJCNN 2009. International Joint Conference on, IEEE, 2009, pp. 2230–2237.

[9] V. G. Gudise, G. K. Venayagamoorthy, Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks, in: Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706), IEEE, 2003, pp. 110–117.

[10] J. L. Elman, Finding structure in time, Cognitive science 14 (2) (1990) 179–211.

[11] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[12] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint arXiv:1412.3555, 2014.

[13] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473, 2014.

[14] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal processing magazine 29 (6) (2012) 82–97.

[15] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[16] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks 3361 (10) (1995) 1995.

[17] S. Dasgupta, T. Osogami, Nonlinear dynamic boltzmann machines for time-series prediction., in: AAAI, 2017, pp. 1833–1839.

[18] R. Yu, Y. Li, C. Shahabi, U. Demiryurek, Y. Liu, Deep learning: A generic approach for extreme condition traffic forecasting, in: Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, 2017, pp. 777–785.

[19] A. Jain, A. M. Kumar, Hybrid neural network models for hydrologic time series forecasting, Applied Soft Computing 7 (2) (2007) 585–592.

[20] P. S. de Mattos Neto, G. D. Cavalcanti, F. Madeiro, Nonlinear combination method of forecasters applied to pm time series, Pattern Recognition Letters 95 (2017) 65–72.

[21] G. P. Zhang, Time series forecasting using a hybrid arima and neural network model, Neurocomputing 50 (2003) 159–175.

[22] P. R. A. Firmino, P. S. de Mattos Neto, T. A. Ferreira, Error modeling approach to improve time series forecasters, Neurocomputing 153 (2015) 242–254.

[23] K. Fukushima, S. Miyake, Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, in: Competition and cooperation in neural nets, Springer, 1982, pp. 267–285.

[24] D. H. Hubel, T. N. Wiesel, Receptive fields and functional architecture of monkey striate cortex, The Journal of physiology 195 (1) (1968) 215–243.

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

[26] H. Lee, R. Grosse, R. Ranganath, A. Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in: Proceedings of the 26th annual international conference on machine learning, ACM, 2009, pp. 609–616.

[27] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

[28] F. A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with lstm, Neural computation 12 (10) (2000) 2451–2471.

[29] O. Christopher, Understanding lstm networks, Understanding LSTM Networks-Colah's Blog, 2015.

[30] S. Hochreiter, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, A Field Guide to Dynamical Recurrent Neural Networks (2001) 237–244.

[31] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE transactions on neural networks 5 (2) (1994) 157–166.

[32] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: Proceedings of the 49th annual

meeting of the association for computational linguistics: Human language technologies-volume 1, Association for Computational Linguistics, 2011, pp. 142–150.

[33] W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization, arXiv preprint arXiv:1409.2329, 2014.

[34] G. Lai, W.-C. Chang, Y. Yang, H. Liu, Modeling long-and short-term temporal patterns with deep neural networks, in: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, ACM, 2018, pp. 95–104.

[35] P.-H. Kuo, C.-J. Huang, An electricity price forecasting model by hybrid structured deep neural networks, Sustainability 10 (4) (2018) 1280.

[36] B. Al-hnaity, M. Abbod, Predicting financial time series data using hybrid model, in: Intelligent Systems and Applications, Springer, 2016, pp. 19–41.

[37] P. R. A. Firmino, P. S. de Mattos Neto, T. A. Ferreira, Correcting and combining time series forecasters, Neural networks 50 (2014) 1–11.

[38] I. Khandelwal, R. Adhikari, G. Verma, Time series forecasting using hybrid arima and ann models based on dwt decomposition, Procedia Computer Science 48 (2015) 173–179.

[39] P. Sugiartawan, R. Pulungan, A. K. Sari, Prediction by a hybrid of wavelet transform and long-short-term-memory neural network, International Journal of Advanced Computer Science and Applications 8 (2) (2017) 326–332.

[40] A. Levinskis, Convolutional neural network feature reduction using wavelet transform, Elektronika ir Elektrotechnika 19 (3) (2013) 61–64.

## Appendix

As discussed in Section 3.2, more details of the proposed model formulations are given in this part. Recall our HNRA-CNN-LSTM model, we get the

following system:

$$\begin{cases} \hat{x}_t = f_{HNRA}(f_{CNN}(x_{t-1}, ..., x_{t-p}), f_{LSTM}(e_{t-1}, ..., e_{t-q})) \\ e_t = x_t - \hat{x}_t \end{cases}$$

By developping these functions, we get the following mathematical formulations.

*Output of the convolutional layers:*

$$a_j^{(l)}(\tau) = \phi(\sum_{f=1}^{F^{(l-1)}} [\sum_{s=1}^{S^{(l-1)}} k_{if}^{(l-1)}(s) * a_f^{(l-1)}(\tau - s)] + b_j^{(l-1)})$$

We consider $P(a_{F^{(l-1)}}^{(l^{"}-1)}(\tau))$ as an output of the pooling layer. It is common to periodically insert it between convolutional layers.

*Output of the fully connected layers:*

$$m_i^{(l'')} = \phi'(\sum_{z=1}^{m^{(l''-1)}} w_{i,z}^{(l''-1)}(a_{F^{(l-1)}}^{(l''-1)}(\tau)) + b_i^{(l''-1)})$$

*Output of the CNN regressor:*

$$\hat{x}_t^{(L'')} = w_{xm} m_i^{(L'')} + b_{xm}$$

*LSTM gates:*

$$\begin{pmatrix} i' \\ f' \\ o' \\ g' \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} e_t^{(l'-1)} \\ h_{t-1}^{(l')} \end{pmatrix}$$

*LSTM internal cell state:*

$$c_t^{(l')} = f' \times c_{t-1}^{(l')} + i' \times g'$$

*LSTM hidden state:*

$$h_t^{(l')} = o' \times \tanh(c_t^{(l')})$$

*Output of the LSTM regressor:*

$$\hat{e}_t^{(L')} = w_{eh} h_t^{(L')} + b_{eh}$$

*HNRA principle:*

$$\hat{x}_t = \hat{x}_t^{(L'')} \oplus \hat{e}_t^{(L')} \qquad \longleftrightarrow \qquad \begin{cases} \hat{x}_t = \hat{x}_t^{(L'')} + \hat{e}_t^{(L')} \\ e_t = x_t - \hat{x}_t \end{cases}$$

## Nomenclature

$\phi, \phi'$ Activation functions $\in \{sigmoid, tanh, relu\}$

$a_f^{(0)}$ Initial input $p$-vector

$a_f^{(l>1)}$ Hidden feature map $f$ in layer *(l)*

$b_j^{(l-1)}, b_i^{(l''-1)}$ Bias

$e_t^{(0)}$ Initial error $q$-vector

$e_t^{(l'>1)}$ Hidden error state in layer *(l')* in timestep $t$

$F^{(l-1)}$ Number of feature maps in layer *(l-1)*

$K_{if}^{(l-1)}$ Kernel convolved over feature map $f$ in layer *(l-1)* to create the feature map $i$ in layer *(l)*

$L'', L'$ The final layer of the CNN and LSTM respectively.

$m^{(l''-1)}$ Number of units in layer *(l"-1)*

$m_i^{(l'')}$ The output of $i^{th}$ unit with layer *(l")* of the fully connected network

$P(.)$ Pooling operator $\in \{max, mean\}$

$S^{(l-1)}$ Length of kernels in layer *(l-1)*

$T_{n,m}$ Affine Transformation *(Wx+b)* from $R^n$ to $R^m$ space [33]

$w_{i,z}^{(l''-1)}$ The weighted connection from $z^{th}$ unit layer in *(l"-1)* to the $i^{th}$ unit layer *(l")*