Neural Probabilistic Forecasting of Symbolic Sequences With Long Short-Term Memory

Michael Hauser
Department of Mechanical Engineering,
The Pennsylvania State University,
University Park, PA 16802
e-mail: mzh190@psu.edu

Yiwei Fu
Department of Mechanical Engineering,
The Pennsylvania State University,
University Park, PA 16802
e-mail: yxf118@psu.edu

Shashi Phoha
Applied Research Laboratory,
The Pennsylvania State University,
University Park, PA 16802
e-mail: sxp26@arl.psu.edu

Asok Ray[1]
Professor
Fellow ASME
Department of Mechanical Engineering,
The Pennsylvania State University,
University Park, PA 16802
e-mail: axr2@psu.edu

This paper makes use of long short-term memory (LSTM) neural networks for forecasting probability distributions of time series in terms of discrete symbols that are quantized from real-valued data. The developed framework formulates the forecasting problem into a probabilistic paradigm as h_Θ*:* $X \times Y \rightarrow [0, 1]$ *such that* $\sum_{y \in Y} h_\Theta(x, y) = 1$*, where* X *is the finite-dimensional state space,* Y *is the symbol alphabet, and* Θ *is the set of model parameters. The proposed method is different from standard formulations (e.g., autoregressive moving average (ARMA)) of time series modeling. The main advantage of formulating the problem in the symbolic setting is that density predictions are obtained without any significantly restrictive assumptions (e.g., second-order statistics). The efficacy of the proposed method has been demonstrated by forecasting probability distributions on chaotic time series data collected from a laboratory-scale experimental apparatus. Three neural architectures are compared, each with 100 different combinations of symbol-alphabet size and forecast length, resulting in a comprehensive evaluation of their relative performances.*
[DOI: 10.1115/1.4039281]

1 Introduction

Time series forecasting is a well-studied problem across a diverse range of fields [1]. From the perspectives of dynamical systems, accurate forecasting of future states can be implemented within a feedback mechanism in the case where control actions can be used to drive the system to a desired state or away from an undesired state. To this end, linear stationary forecasting models [1] have been widely used. For example, autoregressive moving average (ARMA) models provide a standard benchmark for linear stationary time series forecasting; if the time series is nonstationary, then autoregressive integrated moving average models [1] are used to deal with these nonstationary processes.

Previous work has shown that even feedforward (FF) neural networks significantly outperform the linear ARMA model [2], which provides a valuable baseline for time series forecasting. In that work, a feedforward neural network was used to forecast symbol sequences derived from an experimental combustion apparatus. If the ARMA model made a forecast prediction in the same bin as the true trajectory, then it was deemed a correct prediction, otherwise it was incorrect. Under these conditions the simple feedforward neural network significantly outperformed ARMA. Additionally, it was shown in Ref. [2] that feed forward neural symbolic forecasting outperforms taking an expectation over symbol centroids to yield a mean trajectory, which is analogous to a nonlinear neural network ARMA model. The current paper is built upon the previous work [2] by systematically evaluating the relative performance of different neural architectures.

To overcome the limitations of linear models, forecasting time series with neural networks has been reviewed by Zhang et al. [3]. Neural networks can be made to be highly nonlinear, and with even a single hidden layer they have the ability to approximate any Borel measurable function [4]. Additionally, it is desirable to replace these single-valued hard predictions by forecasting probability distributions over symbolic states [5,6].

A forecasting paradigm requires evaluation of both a predicted value and the bounds on its accuracy [7]. Casting the problem in a symbolic-probabilistic setting provides a wide range of generalities; for example, symbols can be interpreted as emanating from a state space. Furthermore, they facilitate probabilistic predictions without prior assumptions on data distributions (e.g., second-order statistics). However, they lack the fidelity that comes with deterministic predictions, although this shortcoming can be partially alleviated by taking an expectation over the prediction space.

Statistical characterizations of symbol sequences have been extensively investigated within the framework of probabilistic finite state automata [8,9]. Within this framework, symbol emission probabilities are estimated, given a current state of the system. Estimation of this symbol emission probability matrix has been cast in a Bayesian framework as well [10].

This paper formulates the probabilistic forecasting problem in a classification sense by using the concept of deep neural networks such that, conditioned on its history, the desired output is a probability over symbols; the symbols partition the range space and each symbol corresponds to a class. This procedure is significantly different from a deterministic regression of the mean and its associated quantiles.

The technical approach of the proposed method is to estimate symbol emission probabilities largely analogously to what is done in finite state automata with stochastic symbolic emissions from given states; however, it does so with deep neural networks. In contrast to a D-Markov model [11,12] that generates probabilities of emitting certain symbols given a finite history symbol sequence of length D, the proposed model generates symbol emission probabilities based on a finite history of time series of length D. The proposed model is also able to identify nonlinear relationships between past history and future symbols in the setting of dynamic data-driven application systems [13]. The primary contributions of this paper are in the formulation of a novel approach to probabilistic forecasting, namely, symbolic probabilistic forecasting, not in how to train neural networks.

The paper is organized in five sections including the present section. Section 2 introduces the mathematical preliminaries needed for algorithm development, with special attention payed to the long short-term memory (LSTM) neural network model. Section 3 describes the technical details of the proposed neural probabilistic forecasting algorithm. Section 4 presents the results of

[1]Corresponding author.

Contributed by the Dynamic Systems Division of ASME for publication in the Journal of Dynamic Systems, Measurement, and Control. Manuscript received April 17, 2017; final manuscript received January 8, 2018; published online March 30, 2018. Assoc. Editor: Dumitru I. Caruntu.

algorithm validation on experimental data, collected from a laboratory-scale combustion apparatus. Section 5 summarizes the contents and concludes the paper. Additionally, Sec. 5 suggests directions of future research to test the neural network models on vector time series data.

2 Mathematical Preliminaries

This section succinctly presents the key mathematical concepts that are necessary to develop the algorithms.

2.1 Probabilistic Formulation. Neural networks are a general class of algorithms that have nested compositions of affine transformations followed by a simple nonlinearity. They can be either strictly FF, or include some type of feedback mechanism. Neural networks with feedback are called recurrent and are usually preferred means of modeling temporal dynamics within a neural network framework.

Classification neural networks are mappings h_Θ: $X \times Y \to [0, 1]$ such that $\sum_{y\in Y} h_\Theta(x, y) = 1$, which is usually interpreted as $h_\Theta(x, y) := p(Y = y|X = x)$, where X is the finite dimensional state space, Y is the finite space of labels (i.e., symbol alphabet) and $\Theta = \{(W^l, b^l)\}_{l=0}^{L-1}$ is the parameterization of architecture class H such that $h_\Theta \in H$. The notation $h_\Theta(x) := h_\Theta(x, \cdot) = p(Y|X = x)$ will be used to represent the output distribution.

Classification neural networks are trained by minimizing a loss function $l(h_\Theta, q)$, where $q: X \times Y \mapsto [0, 1]$ such that $\sum_{y\in Y} q(x, y) = 1$ is the true classification model. It is noted that the true distribution q may not be an element of H, i.e., $q \notin H$, but because neural network models are very flexible, they may still well approximate q.

2.2 Long Short-Term Memory. Long short-term memory neural networks [14,15] are recurrent neural networks with a specific architecture in the hidden space which allows the network to act as if it has a memory unit it can read from and write to. Figure 1 presents a schematic diagram of the LSTM neural network structure for understanding how the different data elements are related-to and operate-on each other.

The LSTM has an input x_t, an output h_t and a cell state C_t that acts as the memory and is the central component of the LSTM. It is noted that the sigmoid function $\sigma(z) := 1/(1 + \exp(-z))$ has a range of 0 to 1, and the hyperbolic tangent function $\tanh(z) := (\exp(z) - \exp(-z))/(\exp(z) + \exp(-z))$ has range -1 to 1.

The candidate cell state $\tilde{C}_t$, its weightings i_t, and the forget gate [16] activations f_t are computed as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{1}$$

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3}$$

where W's and U's are appropriate weight matrices and the b's are bias vectors.

With * defined to be element-wise multiplication, the updated cell state C_t is then found as

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1} \tag{4}$$

and the output is obtained as a weighted version of the cell state

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_f) \tag{5}$$

$$h_t = o_t * \tanh(C_t) \tag{6}$$

Because Eqs. (1)–(3) and (5) depend only on x_t, h_t and h_{t-1}, their arguments are computed in parallel.

2.3 Time Series Symbolization. The procedure of time series symbolization is built by constructing a partition of the time series feature space, defining mutually exclusive and exhaustive regions over the feature space. Once a partition is defined over the feature space, the data are symbolized by mapping each data point to a corresponding symbol, which is uniquely identified with that partitioning region. With K symbols, the symbol set is defined to be $Y := \{y_1, y_2, \ldots, y_K\}$.

For one-dimensional time series, two commonly used partitioning schemes are the uniform partitioning and the maximum entropy partitioning [17]. For general multidimensional time series, the right column clustering, Gaussian mixture models, or other unsupervised clustering techniques can be used [18]. This work uses uniform partitioning.

3 Algorithm Development

This section outlines an algorithm for neural probabilistic forecasting of symbol sequences. Given a time series interval up to the present $\{x(t - t')\}_{t'=0}^{l-1}$ of length l, the algorithm predicts the probability of the symbol y_k at a time instant $t+T$ as $p(Y(t+T) = y_k|X(t) = \{x(t - t')\}_{t'=0}^{l-1})$. The data can, thus, be organized into input–output pairs as follows:

$$\mathcal{D} := \{(\{x(n - t')\}_{t'=0}^{l-1}, y(n + T))\}_{n=l-1}^{N} \tag{7}$$

where l is the memory length of the time series obtained from the first minima of the mutual information with a time-shifted version of itself [19]), N is the number of data samples, and T is the future forecasting time-step. It is noted that $l \ll N$. The dataset is carefully organized so that the experiments conducted in Sec. 4.2 use exactly the same conditions for the different neural architectures at a given forecast-length and number-of-symbols pair. This data organization ensures fairness in generating exactly the same conditions for all neural architectures by comparing the plotted true trajectory for a fixed pair.

3.1 Mathematical Formulation. As described in Sec. 2.1, a neural network defines a mapping: $h_\Theta : X \times Y \mapsto [0, 1]$ such that $\sum_{y\in Y} h_\Theta(x, y) = 1$, where X and Y are the input and label spaces, and $\Theta = \{(W^l, b^l)\}_{l=0}^{L-1}$ is the parameterization. Because we are dealing with time series of the input and output spaces, the mapping used here is defined as $h_\Theta : (X \times T_1) \times (Y \times T_2) \mapsto [0, 1]$, where X is the input space, Y is the symbol alphabet set, and T_1 and T_2 are the forecast shifted time indices of X and Y, respectively. The definition of h_Θ is as follows:

$$h_\Theta(\{x(t - t')\}_{t'=0}^{l-1}) := p(Y(t + T)|X = \{x(t - t')\}_{t'=0}^{l-1}) \tag{8}$$

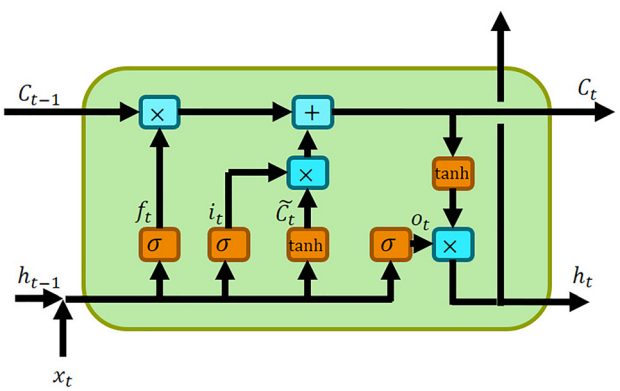

Fig. 1 Schematic of the LSTM neural network stucture. The nonlinear activations, σ and tanh, act elementwise on their respective input vectors. Similarly, multiplication and addition blocks operate on their input pairs elementwise.

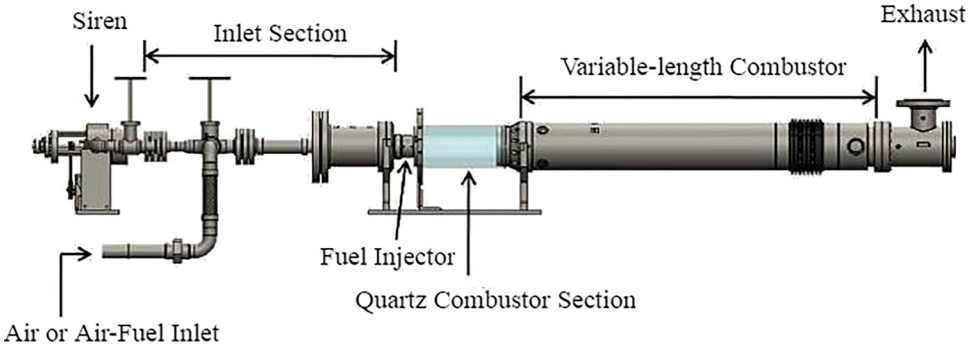

Fig. 2 Schematic diagram of the combustor apparatus

The forecasting problem is formulated within a probabilistic framework in terms of the objective function that is chosen as the cross entropy between the true probability distribution $q(x,y)$ such that, $\sum_{y\in Y} q(x,y) = 1$, and the model's symbol emission probability distribution $h_\Theta(x,y) = p(Y = y|X = x)$. The corresponding loss function is

$$l(h_\Theta(x), q(x)) := -\sum_{k=1}^{K} q(x, y_k)\log(h_\Theta(x, y_k)) \tag{9}$$

where for brevity, $x \equiv \{x(t - tt)\}_{t'=0}^{l-1}$ and y_k is realized at time $t+T$. Note that there are K symbols in the symbol set.

The cross entropy between two probability distributions q and h_Θ over the same sample space is intuitively understood as the average number of bits required to identify a realization drawn from the sample space if the coding scheme is optimized for an approximated distribution h_Θ, as opposed to the true distribution q [20]. Thus minimizing the cross entropy between these two distributions is a way of minimizing the distance, measured in bits, between them.

In a deterministic regression problem for forecasting h_Θ: $X \rightarrow Y$, the cost functional of cross entropy is taken to be the loss function [21]

$$l(h_\Theta(x), y) := y\log(h_\Theta(x)) + (1 - y)\log(1 - h_\Theta(x)) \tag{10}$$

where it is noted that $h_\Theta(x) \in \mathbb{R}$ and $y \in \mathbb{R}$ are not probability distributions, because they are actual values and are not symbolized in a finite state space; hence, they are not probabilistic and takes the form as a deterministic regression. Similarly, the $\ell_2(\mathbb{R})$-norm cost functional is also used for regression tasks, as opposed to determining probabilities of new classes and symbols. If used in a probabilistic sense the $\ell_2(\mathbb{R})$-norm cost functional assumes restrictive second-order statistics.

Training the neural network to make inferences involves taking the expected loss over Eq. (9)

$$L(\Theta; \mathcal{D}) := E_{(X,Y)}[l(h_\Theta(x, y), q(x, y))] \tag{11}$$

The best model is taken to be $\text{argmin}_{h_\Theta\in H} L(\Theta; \mathcal{D})$, which is trained by using Adadelta [22] in backpropagation over ten epochs with a batch size of 100. The rationale for using Adadelta is delineated in the following:

(1) Adadelta automatically updates the learning rate along each dimension. In this sense, it is more user-friendly than standard stochastic-gradient-descent methods that often require the user to manually tune the learning rate.
(2) Adadelta is an extension of Adagrad [23], but with the goal to ensure that the learning rate does not monotonically decrease to zero as the number of updates goes to infinity, as is the case with Adagrad.
(3) Root-mean-square (RMS) propagation has been tried in preliminary testing as the third update rule, which is an unpublished technique used in Geoff Hinton's Coursera lecture series on machine learning. Root-mean-square propagation yielded similar results as Adadelta.
(4) While stochastic gradient descent has been used in preliminary testing, Adadelta is consistently found to yield better minima in Eq. (11).

3.2 Testing Three Neural Network Architectures. Three different neural network architectures have been tested and compared. The first of these three architectures is a simple FF architecture with three hidden layers. The second is a LSTM architecture with a fully connected layer before and after the LSTM hidden layer. The third is a hybrid LSTM-FF architecture, which concatenates the outputs of the third hidden layer of the LSTM and the FF networks. All three architectures use a softmax classifier at the output and have a window size of eight time steps, i.e., $x \in \mathbb{R}^8$ with hidden layer dimensions equal to 50. These three architectures have also been tested with other datasets, all yielding very similar results. Therefore, in the selection of a network architecture, computational considerations are deemed to be very important.

Graphics processing unit implementations of the aforementioned three neural network architectures have been written in the Python library Theano [24–26]. The training would take, on the average, about 15 min per symbol size-forecast length pair. With three architectures and 100 pairs per architecture, the total training time has been about 3 days on the graphics processing unit.

4 Results and Discussion

This section presents the results and comparisons of the three proposed neural network algorithms, namely, LSTM, FF, and LSTM-FF running in parallel, on the chaotic experimental data collected from a combustor apparatus that is described in the following.

4.1 Description of the Combustor Apparatus. The apparatus is built upon a swirl-stabilized, lean-premixed, laboratory-scale combustor that has been used to perform the experimental investigation. Figure 2 shows a schematic diagram of the variable-length combustor, consisting of an inlet section, an injector, a combustion chamber, and an exhaust section. High-pressure air is delivered to the apparatus from a compressor after passing through filters to remove any liquid or solid particles that might be present in the inlet air. The air supply pressure is set to approximately 1.34 MPa using a dome pressure regulator. The air is preheated to a maximum temperature of 250 °C by an 88 kW electric heater. The fuel for this study is natural gas (approximately 95% methane) that is supplied to the combustor system at a pressure of approximately 1.48 MPa. The flow rates of the air and natural gas are measured by thermal mass flow meters. The desired

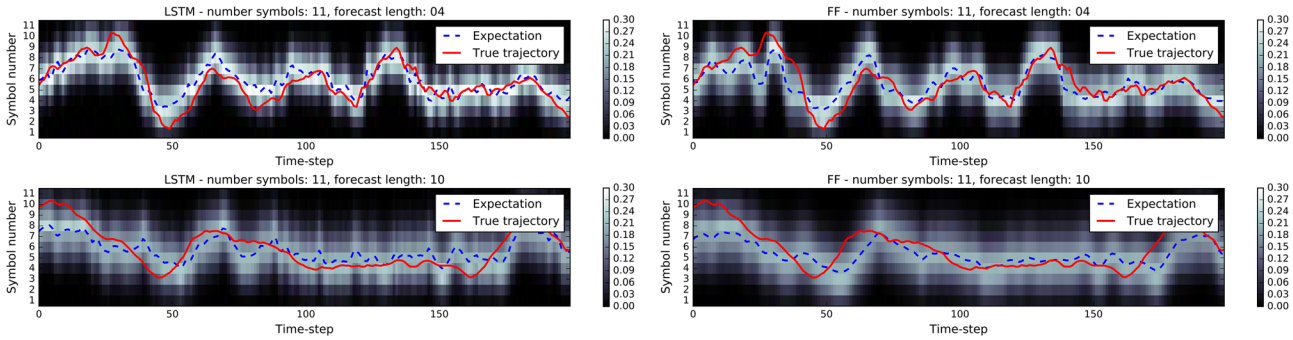

Fig. 3 Forecasts from the neural probabilistic framework; dark background corresponds to low probability and white corresponds to high probability. The LSTM and feed forward networks are compared, as well as forecast lengths of 4 and 10 time-steps. Solid lines are the true trajectory while dotted lines are the expectation over the forecasted probability distributions.

equivalence ratio and mean inlet velocity is set by adjusting these flow rates with needle valves.

4.2 Forecasting Physical Phenomena. Combustion and fluid processes, which are governed by physical process dynamics that can be represented by highly coupled, nonlinear partial differential equations, generate chaotic pressure signals; the underlying dynamics are very difficult to predict. The proposed algorithms, which serve the purpose of instability prediction, are dynamic data-driven [13,27], instead of solely relying on constitutive model equations and thermodynamic state relations. The goal here is to predict combustion instabilities for possible usage in a feedback control system to mitigate instabilities.

Tests were conducted at a nominal combustor pressure of 1 atm over a range of operating conditions. At each operating condition, time series of pressure data were collected for 8 s at sampling rate 8192 Hz, which is sufficiently long to capture the dynamic characteristics of the underlying process and which, in addition, can be considered to be approximately statistically stationary. Based on these test data, individual models have been compared by averaging over their element-wise relative error rates; reductions in the average error rate between algorithms have been examined.

Combustion instabilities are commonly defined by the RMS values of the pressure signals inside the combustion chamber of a trailing window, with the intuition that a larger RMS value implies existence of larger pressure fluctuations and thus larger instabilities. Figure 3 shows characteristic probabilistic forecasts, where black corresponds to low probability and white corresponds to high probability. It is seen that the LSTM makes more confident predictions than the FF architecture. As the forecast length is increased from 4 to 10 time steps, both architectures are less confident in their predictions and thus generate more diffuse prediction densities. This is expected because there are more uncertainties in the prediction with increased forecast length. The true trajectory in red in Fig. 3 is compared with the expectation over the forecasted probability distributions, where the centroid of the partitioned region is taken to be the expected value of the random variable.

Figure 4 shows how the different neural network architectures perform across all test data sets. These data sets are composed of all combinations of forecast lengths of 2, 4, 6,…, 20 and the symbol alphabet size (i.e., number of symbols) being 2, 3, 4,…, 11, making 100 unique combinations in total. Increasing the symbol alphabet size (i.e., number of symbols) enhances the fidelity of the model at the possible expense of model accuracy as seen in Fig. 4, because of either the inherit difficulty in forecasting at such a high resolution or the finite data length [10,12]. This general trend is seen across all three tested architectures. It is noted that the design parameters (i.e., fidelity, forecast length and accuracy) are application specific.

Referring to Fig. 4, Table 1 lists the average relative reduction in error over all pairwise combinations, where the relative error reduction is calculated as mean $(1 - A/B)$, and A/B is element-wise division of the matrices A and B from Fig. 4. In conclusion the LSTM performs the best, followed closely by LSTM-FF and the last is FF.

Figure 5 shows the errors, weighted by the distance between the predicted symbol and the true symbol. This is perhaps a better performance measure of the architectures because the error rate should be penalized more heavily by the deviation of the predicted symbol from the true symbol. The weighted error can be interpreted as the average distance between the predicted and true

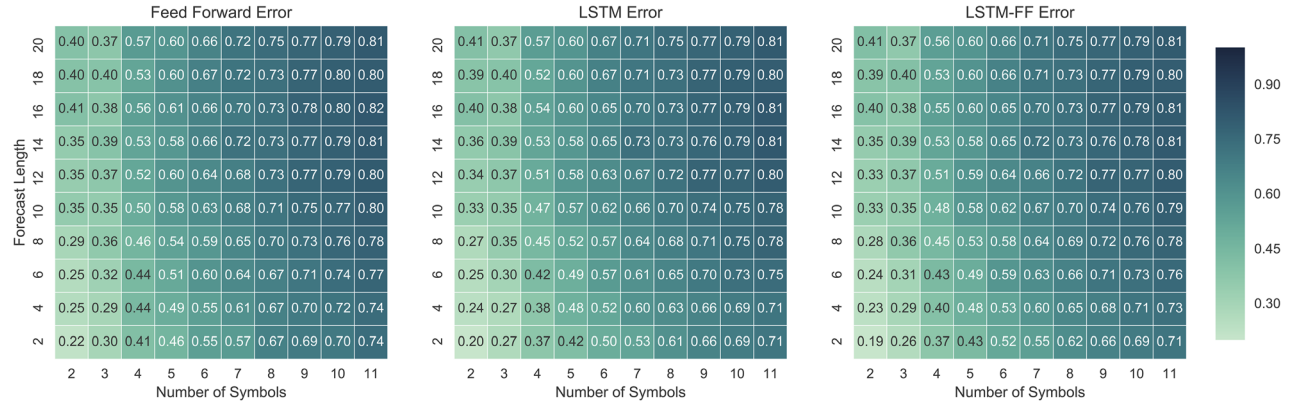

Fig. 4 Test error rates for different combinations of forecast length and number of symbols. The lighter shade corresponds to a lower error rate while a darker shade corresponds to a higher error rate. When averaged over all pairwise combinations, the relative reductions in error can be seen in Table 1.

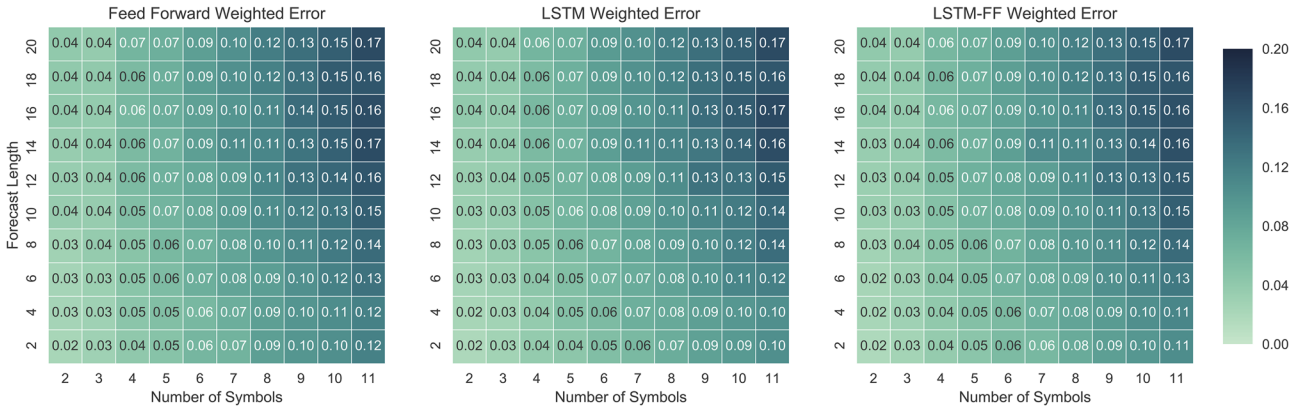

Fig. 5 Weighted error rates for different combinations of forecast length and number of symbols, where the weighting factor is determined by the distance between the predicted symbol and the true symbol (determined by partition centroid). The weighted error can be interpreted as the average distance between the predicted and true symbol, scaled between 0 and 1. The lighter shade corresponds to a lower error rate while a darker shade corresponds to a higher error rate. When averaged over all pairwise combinations, the relative reductions in weighted error can be seen in Table 2.

Table 1 Relative reduction in error

LSTM/FF	LSTM/LSTM-FF	LSTM-FF/FF
2.57%	0.587%	1.99%

symbol, scaled between 0 and 1. The general trend reaffirms that the error rates increase as the forecast length increases, as well as when the number of symbols increases. When averaged over all pairwise combinations, the relative reductions in weighted error can be seen in Table 2, where again the relative reduction in weighted error is calculated from Fig. 5 as mean $(1 - A/B)$, and A/B is element-wise division of matrices A and B.

It is seen that the LSTM algorithm outperforms the FF algorithm, because the recurrent feedback mechanism in LSTM is specifically designed to accommodate time series data. Similarly, the LSTM outperforms the LSTM-FF, likely because the feed forward aspect of LSTM-FF is corrupting the dynamic features learned by LSTM.

5 Summary, Conclusions, and Future Work

This brief paper has developed a neural probabilistic framework for forecasting time series from neural network architectures in terms of discrete symbols as opposed to real values without any major limiting assumptions such as second-order statistics. Although discrete symbolization is subjected to potential loss of resolution compared to continuous data, it enhances performance robustness especially if the signal-to-noise ratio is not high [28]. In particular, the proposed method yields performance robustness by taking the expectation over the forecasted probability distribution. This formulation of probabilistic forecasting is suitable for continuously varying uncertain dynamical systems, in which control actions operate at a slower time scale than that of system dynamics; such systems are prevalent in diverse mechanical engineering applications.

Three neural probabilistic architectures, namely, FF, LSTM, and combined LSTM and FF neural networks running in parallel, have been tested on experimental (time series) data of chaotic pressure oscillations collected from a laboratory-scale combustor apparatus. The performance of these three architectures has been comprehensively compared, where each architecture has been trained for 100 combinations of forecast length and number of symbols, so that application-specific requirements can be optimally chosen. It is found that, on the average, the LSTM performs the best, followed by LSTM-FF, and finally the FF.

Table 2 Relative reduction in weighted error

LSTM/FF	LSTM/LSTM-FF	LSTM-FF/FF
3.94%	0.997%	2.98%

The future research should focus on comparison of the aforementioned three neural architectures with AR(I)MA models on user-defined, controlled chaotic datasets. For example, such datasets can be generated from solutions of the Duffing equation [29] that can be studied as a two-dimensional time-series trajectory in the phase-space with multidimensional symbolization techniques (e.g., k-means), as described in Sec. 2.3. While there are many other issues that need to be resolved by further theoretical and experimental research, the authors suggest comparison of the present work with various other applications (e.g., Cheng et al. [30]) as a topic of future research.

Acknowledgment

The first author has been supported by PSU/ARL Walker Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsoring agencies. The authors would like to thank Professor Domenic Santavicca and Mr. Jihang Li for kindly providing the experimental data used in this work.

Funding Data

- U.S. Air Force Office of Scientific Research (AFOSR) (Grant No. FA9550-15-1-0400).

References

[1] Montgomery, D. C., Jennings, C. L., and Kulahci, M., 2015, *Introduction to Time Series Analysis and Forecasting*, Wiley, Hoboken, NJ.
[2] Hauser, M., Fu, Y., Li, Y., and Ray, A., 2017, "Probabilistic Forecasting of Symbol Sequences With Deep Neural Networks," American Control Conference (ACC), Seattle, WA, May 24–26, pp. 3147–3152.
[3] Zhang, G., Patuwo, B. E., and Hu, M. Y., 1998, "Forecasting With Artificial Neural Networks: The State of the Art," Int. J. Forecasting, **14**(1), pp. 35–62.
[4] Hornik, K., Stinchcombe, M., and White, H., 1989, "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, **2**(5), pp. 359–366.
[5] Gneiting, T., 2008, "Editorial: Probabilistic Forecasting," J. R. Stat. Soc. Ser. A, **171**(2), pp. 319–321.
[6] Gneiting, T., and Katzfuss, M., 2014, "Probabilistic Forecasting," Annu. Rev. Stat. Appl., **1**(1), pp. 125–151.
[7] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M., 2015, *Time Series Analysis: Forecasting and Control*, Wiley, Hoboken, NJ.

[8] Dupont, P., Denis, F., and Esposito, Y., 2005, "Links Between Probabilistic Automata and Hidden Markov Models, Probability Distributions, Learning Models and Induction Algorithms," Pattern Recognit., **38**(9), pp. 1349–1371.
[9] Rozenberg, G., and Salomaa, A., 1997, *Handbook of Formal Languages: Beyonds Words*, Vol. 3, Springer Science & Business Media, Berlin.
[10] Wen, Y., Mukherjee, K., and Ray, A., 2013, "Adaptive Pattern Classification for Symbolic Dynamic Systems," Signal Process., **93**(1), pp. 252–260.
[11] Ray, A., 2004, "Symbolic Dynamic Analysis of Complex Systems for Anomaly Detection," Signal Process., **84**(7), pp. 1115–1130.
[12] Mukherjee, K., and Ray, A., 2014, "State Splitting and Merging in Probabilistic Finite State Automata for Signal Representation and Analysis," Signal Process., **104**, pp. 105–119.
[13] Darema, F., 2005, "Dynamic Data Driven Applications Systems: New Capabilities for Application Simulations and Measurements," Fifth International Conference on Computational Science (ICCS), Atlanta, GA, May 22–25, pp. 610–615.
[14] Hochreiter, S., and Schmidhuber, J., 1997, "Long Short-Term Memory," Neural Comput., **9**(8), pp. 1735–1780.
[15] Graves, A., 2012, "Supervised Sequence Labelling," *Supervised Sequence Labelling With Recurrent Neural Networks*, Springer, New York, pp. 5–13.
[16] Gers, F. A., Schmidhuber, J., and Cummins, F., 2000, "Learning to Forget: Continual Prediction With LSTM," Neural Comput., **12**(10), pp. 2451–2471.
[17] Li, Y., Chattopadhyay, P., and Ray, A., 2015, "Dynamic Data-Driven Identification of Battery State-of-Charge Via Symbolic Analysis of Input–Output Pairs," Appl. Energy, **155**, pp. 778–790.
[18] Hauser, M., Li, Y., Li, J., and Ray, A., 2016, "Real-Time Combustion State Identification Via Image Processing: A Dynamic Data-Driven Approach," American Control Conference (ACC), Boston, MA, July 6–8, pp. 3316–3321.
[19] Abarbanel, H., 2012, *Analysis of Observed Chaotic Data*, Springer Science & Business Media, New York.
[20] Cover, T. M., and Thomas, J. A., 2012, *Elements of Information Theory*, Wiley, Hoboken, NJ.
[21] Nasr, G. E., Badr, E., and Joun, C., 2002, "Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand," Fifteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS), Pensacola, FL, May 14–16, pp. 381–384.
[22] Zeiler, M. D., 2012, "Adadelta: An Adaptive Learning Rate Method," preprint arXiv:1212.5701.
[23] Duchi, J., Hazan, E., and Singer, Y., 2011, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," J. Mach. Learn. Res., **12**, pp. 2121–2159.
[24] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y., 2012, "Theano: New Features and Speed Improvements," preprint arXiv:1211.5590.
[25] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y., 2010, "Theano: A CPU and GPU Math Compiler in Python," Ninth Python in Science Conference, Austin, TX, June 28–July 3, pp. 1–7.
[26] Theano Development Team, 2016, "Theano: A Python Framework For Fast Computation Of Mathematical Expressions," e-print arXiv:1605.02688.
[27] Sarkar, S., Chakravarthy, S., Ramanan, V., and Ray, A., 2016, "Dynamic Data-Driven Prediction of Instability in a Swirl-Stabilized Combustor," Int. J. Spray Combust., **8**(4), pp. 235–253.
[28] Graben, P. B., 2001, "Estimating and Improving the Signal-to-Noise Ratio of Time Series by Symbolic Dynamics," Phys. Rev. E, **64**(5), p. 051104.
[29] Thompson, J., and Stewart, H., 1986, *Nonlinear Dynamics and Chaos*, Wiley, Chichester, UK.
[30] Cheng, L., Liu, W., Hou, Z.-G., and Yu, J., 2015, "Neural Network Based Nonlinear Model Predictive Control for Piezoelectric Actuators," IEEE Trans. Ind. Electron., **62**(12), pp. 7717–7727.