POLA: ONLINE TIME SERIES PREDICTION BY ADAPTIVE LEARNING RATES

Wenyu Zhang

Institute for Infocomm Research, Singapore

ABSTRACT

Online prediction for streaming time series data has practical use for many real-world applications where downstream decisions depend on accurate forecasts for the future. Deployment in dynamic environments requires models to adapt quickly to changing data distributions without overfitting. We propose POLA (**P**redicting **O**nline by **L**earning rate **A**daptation) to automatically regulate the learning rate of recurrent neural network models to adapt to changing time series patterns across time. POLA meta-learns the learning rate of the stochastic gradient descent (SGD) algorithm by assimilating the prequential or interleaved-test-then-train evaluation scheme for online prediction. We evaluate POLA on two real-world datasets across three commonly-used recurrent neural network models. POLA demonstrates overall comparable or better predictive performance over other online prediction methods.

Index Terms— Time series prediction, online learning, gradient learning, concept drift

1. INTRODUCTION

As data collection becomes more accessible, there is an increase in the amount of data available for analysis. In streaming settings, data is collected by continuously monitoring a system. A key challenge for model deployments in dynamic environments is concept drift, where the joint distribution of predictor and response variables change across time, such that static models can become obsolete [1]. Online learning techniques continuously update the model as new data streams in to adapt to the current state of the system [2, 3], in some cases with the constraint that samples are immediately discarded after being learned due to reasons such as limited storage, privacy and to reduce interference from outdated samples [4].

In this work, we focus on online time series prediction which is studied and applied in a wide range of domains such as in traffic monitoring, climate research and finance [5, 6, 7, 8]. Predictions can be used for downstream tasks such as anomaly detection and to aid decision making [9, 10]. Unlike in the traditional online learning framework, temporal correlations means that observations cannot assumed to be independently and identically distributed (i.i.d.). In this work, we also focus on deep recurrent neural network (RNN) models which are popularly used to capture complex temporal correlations [11]. Given sufficient data at training, these deep models are shown to have competitive prediction performance over conventional statistical time series models without excessive pre-processing and feature engineering [12, 13].

Deep models are typically learned by gradient descent algorithms. Most works design algorithms for stationary environments with i.i.d. samples [4], and others that do consider time series prediction require Hessian computations [5, 14] that are computationally intensive for neural networks with large number of parameters. We propose POLA (**P**redicting **O**nline by **L**earning rate **A**daptation) to

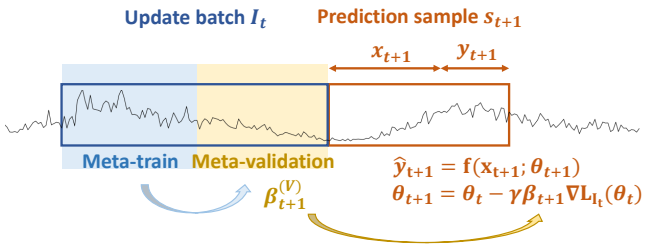

Fig. 1: Overview: POLA adapts the learning rate of SGD update through factor $\beta_{t+1} \in [0, 1]$ where the factor is meta-learned with the most recent data batch by assimilating online evaluation.

- Adapt quickly in dynamic environments without overfitting to current system state or noisy samples, by
- Automatically scheduling the online learning rate of the stochastic gradient descent (SGD) algorithm by assimilating online evaluation.

Figure 1 provides an overview of the proposed method. POLA estimates the learning rate with a novel meta-learning [15] procedure that assimilates the interleaved-test-then-train evaluation scheme for online prediction. Since SGD update only uses the most recently observed batch of data, POLA operates under the constraint that samples are discarded after being learned, which further makes it suitable for light-weight applications. We test our proposed approach on two publicly available real-world datasets across three commonly-used recurrent neural network architectures.

2. RELATED WORKS

Gradient descent algorithms are commonly used to train neural network models online by constantly tracking a specified performance measure such as prediction error, and are also applied to traditional statistical methods such as the autoregressive integrated moving average models [8, 16]. The learning rate can be adapted to achieve better convergence, avoid overfitting to noisy samples or to automatically adjust to shifts in data distribution [5, 14, 17, 18, 19]. RMSprop [20] is a popular algorithm that scales the learning rate by moving average of squared gradients based on the intuition that the magnitude of each weight update should be similar regardless of the actual gradient magnitude. In POLA, instead of explicitly applying an inductive bias into learning rate scaling, we propose learning this scaling directly from the data itself. For adaptation in non-stationary environments, previous works apply simplifying assumptions on the loss function and Hessian computations [5, 14], however second-order derivative calculation can be expensive and the resulting learning rates are subject to instability if the approximated Hessian is ill-conditioned. Other works adapt the learning rate to reduce the

adverse effect of outliers by monitoring distributional properties at neighboring data points [21, 22].

Popular alternatives to gradient descent algorithms include the online sequential extreme learning machine [23], which updates by recursive least squares. Extension to time series modeling includes variants with RNN structure [24], and moderating the extent of model update through a confidence coefficient based on the difference between prediction error and a set threshold [7]. Other works optimize for the model parameters by genetic programming [25], alternating parameter and hyperparameter optimization [26], or use concept drift detection to detect data distribution changes through specified rules and subsequently initiate model updates [6, 27, 28]. However, incorporating concept drift detection as a stand-alone step can introduce additional errors, and fixed rules may be too rigid or restrictive for real-world applications. POLA builds on the SGD algorithm so that the model is updated directly based on prediction loss without intermediate concept drift detection steps.

3. PROBLEM SETUP

For a process $\{Z_t\}$, we denote the observation at time t as $z_t \in \mathbb{R}^d$ with dimensionality d. At each time t, the prediction task is to use $x_t = [z_{t-m+1}, \ldots, z_t]$, a historical input sequence of length m, to predict $y_t = [z_{t+1}, \ldots, z_{t+n}]$, an output forecast sequence for the next n time steps. We denote the prediction sample at time t as $s_t = (x_t, y_t)$, that is, samples are obtained by a sliding window of stride 1 on the sequence $\{z_t\}$. Note that the samples observed up until time t are $\{s_i\}_{i=m}^{t-n}$. The samples $\{s_i\}_{i=t-n+1}^{t}$ are not observed yet since they consist the value z_{t+1}.

Let f be a prediction model parameterized by θ_t at time t. Then the model predicts

$$\hat{y}_t = f(x_t; \theta_t) \tag{1}$$

and it incurs a loss $\ell_t(\theta_t; x_t, y_t)$ such as the squared prediction error $\|y_t - f(x_t; \theta_t)\|_2^2$. At the next time step $t+1$, the value z_{t+1} and consequently the sample s_{t-n+1} is observed and can be used to update model parameters to θ_{t+1}. For completeness of notations, $\ell_t = 0$ for $1 \leqslant t < m$ when the number of observations is insufficient to form samples of the desired length.

Let b be the number of most recent samples that can be stored and learned at any time t, we also refer to b as the online batch size. We denote the samples available for model update at time t as $U_t = \{s_i\}_{i \in I_t}$ where $|I_t| \leqslant b$. In this paper, we only consider the common setting where the model is updated with data chunks of a fixed size [21, 23, 24], that is, when $|I_t| = b$ and explicitly $I_t = \{t-n-b+1, \ldots, t-n\}$. We assume $b > 1$ since batch update is generally less noisy than single sample update, and it also reduces the amount of computation required for frequent updates which may be inhibiting for light-weight applications.

4. PROPOSED METHOD: POLA

4.1. Formulation

The objective of online prediction is to learn model parameters or hypothesis to minimize the loss function ℓ_{t+1} at the future time step:

$$\theta_{t+1}^* = \operatorname*{argmin}_{\theta} \ell_{t+1}(\theta; x_{t+1}, y_{t+1}) \tag{2}$$

where y_{t+1} is unobserved when we estimate for θ_{t+1}^* at time t.

Since we estimate θ_{t+1}^* based on SGD, assuming $|I_t| = b$ such that model parameters are updated at step t with online batch size b, then the estimated parameters θ_{t+1} take the form

$$\begin{aligned} \theta_{t+1} &= \theta_t - \gamma \beta_{t+1} \nabla L_{I_t}(\theta_t) \\ \text{where } L_{I_t}(\theta_t) &= \sum_{i \in I_t} \ell_i(\theta_t; x_i, y_i) \end{aligned} \tag{3}$$

where we introduce a maximum learning rate γ and an additional factor $\beta_{t+1} \in [0, 1]$ to represent the overall learning rate as $\gamma_{t+1} = \gamma\beta_{t+1}$. Such decomposition is also used in [21]. When $|I_t| < b$ such that there are insufficient samples for a batch update, then $\theta_{t+1} = \theta_t$. The intuition for β_{t+1} is that the learning rate should be small if the update batch is not useful in helping the model adapt.

The adaptive update can be formulated as bilevel optimization for both θ_{t+1} and β_{t+1}:

$$\begin{aligned} \theta_{t+1}, \beta_{t+1} &= \operatorname*{argmin}_{\theta, \beta} \ell_{t+1}(\tilde{\theta}; x_{t+1}, y_{t+1}) \\ \text{subject to } \tilde{\theta} &= \operatorname*{argmin}_{\theta} \left\{ \gamma\beta L_{I_t}(\theta) - (1/2)\|\theta - \theta_t\|_2^2 \right\} \end{aligned} \tag{4}$$

and approximation of $L_{I_t}(\theta)$ with a first-order Taylor expansion about the point θ_t gives the adaptive SGD update in Equation 3 for θ_{t+1}, following proximal view of gradient descent [29]. We estimate β_{t+1} through a meta-learning procedure in Section 4.2.

4.2. Meta-learning For Learning Rate

The lower level objective in Equation 4 corresponds to the training set and the upper level objective corresponds to the test set. We create a meta-training set $U_t^{(T)}$ and meta-validation set $U_t^{(V)}$ by splitting U_t, the update batch of samples indexed by I_t, as

$$\begin{aligned} U_t^{(T)} = \{s_i\}_{i \in I_t^{(T)}} \text{ and } U_t^{(V)} = \{s_i\}_{i \in I_t^{(V)}} \\ \text{where } I_t^{(T)} = \{t-n-j\}_{j=\lceil b/2 \rceil}^{b-1} \text{ and } I_t^{(V)} = \{t-n-j\}_{j=0}^{\lceil b/2 \rceil - 1} \end{aligned} \tag{5}$$

such that the meta-training and meta-validation sets are a proxy to the training and testing procedure. The bilevel optimization in the meta-stage is then

$$\begin{aligned} \theta_{t+1}^{(V)}, \beta_{t+1}^{(V)} &= \operatorname*{argmin}_{\theta, \beta} L_{I_t^{(V)}}(\tilde{\theta}) \\ \text{subject to } \tilde{\theta} &= \operatorname*{argmin}_{\theta} \left\{ \gamma\beta L_{I_t^{(T)}}(\theta) - (1/2)\|\theta - \theta_t\|_2^2 \right\} \end{aligned} \tag{6}$$

and we approximate β_{t+1} with $\beta_{t+1}^{(V)}$. Empirically, we set β_{t+1} as the moving average of the most recently computed $\beta^{(V)}$'s.

4.3. Implementation

We present two versions of POLA corresponding to two approaches to estimate $\beta_{t+1}^{(V)}$, namely POLA-FS and POLA-GD, in Algorithm 1. POLA-FS searches for the optimal learning rate $\gamma_t = \gamma\beta_{t+1}^{(V)}$ in a finite set of candidates denoted $\mathcal{C}$. Replacing the solution to the lower level objective in Equation 6 with the SGD update, we get

$$\beta_{t+1}^{(V)} = \operatorname*{argmin}_{\beta \text{ s.t. } \gamma\beta \in \mathcal{C}} L_{I_t^{(V)}} \left(\theta_t - \gamma\beta \nabla L_{I_t^{(T)}}(\theta_t) \right). \tag{7}$$

The version POLA-GD optimizes for $\beta_{t+1}^{(V)}$ by gradient descent while freezing all other model parameters. Two additional hyperparameters define the gradient descent procedure, namely the number of

update steps k and the learning rate η. The update at each step is

$$\alpha^{(i+1)} = \alpha^{(i)} - \eta \nabla_\alpha L_{I_t^{(V)}} \left(\theta_t - \gamma \sigma(\alpha^{(i)}) \nabla L_{I_t^{(T)}}(\theta_t)\right)$$
$$\beta^{(i+1)} = \sigma(\alpha^{(i+1)}) \quad (8)$$

where $\sigma(\alpha) = \frac{1}{1+e^{-\alpha}}$ is the sigmoid function.

Algorithm 1 POLA

Input: Historical sequence length n, forecast horizon m, online batch size b, maximum learning rate γ, moving average window q

Additional input for POLA-FS: Finite set of candidate learning rates $\mathcal{C}$

Additional input for POLA-GD: Number of update steps k, learning rate η

Output: Predictions $\hat{y}_t$ for $t = 1, \ldots, T$

procedure POLA
  Initialize $S = 0$, $B = []$
  for $t = 1 : T$ **do**
    Predict $\hat{y}_t = f(x_t, \theta_t)$
    if $S == b$ **then** ▷ batch update
      Collect indices $I_t = \{t - n - b + 1, \ldots, t - n\}$ of update batch
      Create meta-training set $U_t^{(T)}$ and meta-validation set U_t^T as in Equation 5
      Optimize for $\beta_{t+1}^{(V)}$ as in Equation 7 for POLA-FS and Equation 8 for POLA-GD
      B.append$\left(\beta_{t+1}^{(V)}\right)$
      Set $\beta_{t+1} = \frac{1}{q} \sum_{i=1}^{q} B[-i]$
      Update $\theta_{t+1} = \theta_t - \gamma \beta_{t+1} \nabla L_{I_t}(\theta_t)$
      if $\beta_{t+1} > 0$ **then**
        $S = 0$ ▷ reset counter
    else
      $\theta_{t+1} = \theta_t$
      $S = S + 1$ ▷ update counter

5. EXPERIMENTS

We test our proposed POLA on 2 publicly-available time series datasets with 3 commonly-used recurrent network structures: Recurrent Neural Network (RNN), Long Short-Term Memory network (LSTM), and Gated Recurrent Unit network (GRU). We use recurrent networks with 10 neurons. For all datasets, we allow a pre-training period with 700 samples for hyperparameter tuning and use prequential or interleaved-test-then-train evaluation scheme on the rest of the samples, denoted as the online phase, as is common in streaming data applications. We use an online batch size $b = 10$ and evaluate for multi-step prediction, which is more difficult than one-step prediction since the model needs to learn temporal patterns instead of simply aggregating the most recently observed values [30]. We evaluate prediction performance with normalized root-mean-square error (RMSE) where RMSE is scaled by the standard deviation of the data to allow comparison across datasets. All results presented are averaged across runs over 10 different seeds.

We compare with a variety of time series and neural network methods. We include a popular exponential smoothing method Holt-Winters which is shown to be a strong baseline for time series forecasting [12], and an extreme-learning machine method for time series OR-ELM [24]. For neural network methods, the Pre-trained RNN is trained with offline configurations of 500 epochs with learning rate 0.1 and batch size 32 on the pre-training samples and not updated in the online phase. Follow The Leader (FTL) RNN is retrained every b steps with all historical data. We limit the number of batches to match that in the pre-training phase to constrain online training time. MAML is a meta-learning method that aims to learn parameters offline which can generalize better to unseen data distributions, and we apply the variation of MAML for sequential data in [31] to RNN. We also include 3 gradient-descent online methods applied on recurrent networks. The first uses SGD with constant learning rate, the second uses RMSprop [20] which is a popular algorithm for training neural networks with element-wise adaptive learning rates, and the third is a weighted gradient (WG) algorithm that adapts the learning rate of SGD based on whether the current sample is an outlier or change point [21]. Since WG is designed for one-step prediction [21], we implement WG to adapt the learning rate using only the first-step prediction. The maximum learning rate γ for these gradient-descent methods including POLA is determined by hyperparameter tuning on the set $\mathcal{C} = \{1, 0.1, 0.01, 0.001, 0.0001, 0\}$ by splitting the pre-training samples into two-thirds for training and one-thirds for validation. This is the same set $\mathcal{C}$ used in POLA-FS. Hyperparameters for POLA-GD is set to $k = 3$ and $\eta = 0.1$, and the POLA moving average window q is tuned on $\{1, 3, 5, 7, 9\}$ on the pre-training samples.

5.1. Dataset Description

Sunspot: The monthly sunspot number dataset[1] contains the arithmetic mean of the daily sunspot number over all days of each calendar month from January, 1749 to July, 2020. Each cycle in the sunspot number series is approximately 11 years. We set the length of historical data $n = 48$ which corresponds to 4 years as used by [32] for annual data, and forecast length $m = 5$ which is 5 months.

Household Power Consumption: The household power consumption dataset[2] is a multivariate dataset that records the electricity consumption of a household from December 16, 2006 to November 26, 2010. Due to missing values, we take daily averages and forward padded any remaining missing values. The measurements for global active power, global intensity and voltage are used for prediction. Other variables are not included due to sparse or low readings. We set the length of historical data $n = 28$ which is 4 weeks, and forecast length $m = 3$ which is 3 days.

These datasets are known to exhibit concept drift [32, 33].

5.2. Online Prediction Performance

Table 1 presents the prediction performance of RNN and other time series models. Holt-Winters and OR-ELM generally perform worse than RNN methods, likely because RNN has better modeling capacity for complex temporal patterns. MAML has the highest RMSE, possibly because the small amount of pre-training data does not have sufficiently diverse examples of data distribution changes to generalize to those in the online phase. Both Pre-trained and FTL RNN incur higher errors than the gradient-descent online RNNs, which indicate the presence of concept drift between the pre-training and online phase, and also within the online phase. Both versions of POLA have the best performances for both datasets, showing that the automatic learning rate scheduler is effective in helping the model adapt.

[1] http://www.sidc.be/silso/datafiles

[2] https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption

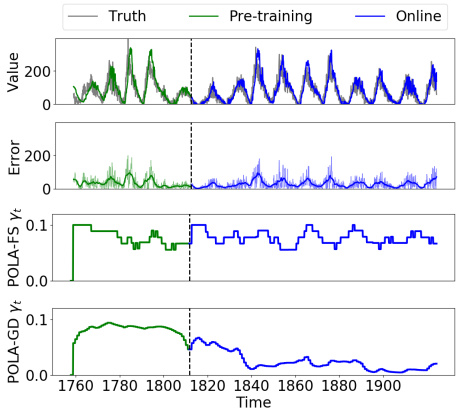

Fig. 2: Predictions (5-step ahead), errors (smoothed across 2 years) and POLA learning rates for Sunspot dataset.

The additional meta-learning procedure is expected to increase computation time. Approximately in seconds on the Sunspot dataset with 3259 observations, Online-SGD and Online-RMSprop takes 3.5s, WG and POLA-FS takes 6.5s, POLA-GD takes 40s with k=1 and 250s with k=3 update steps. Figure 2 plots the POLA-FS predictions, errors and learning rates. POLA-GD predictions and errors are visually similar. We observe different behaviors in the learning rate γ_t. From Equation 8 for POLA-GD, the rate of change of γ_t depends directly on η, hence we see slower-moving changes with a low η. The sigmoid shape also means that $\sigma(\alpha)$ will tend to be near 0 or 1, which is reflected through POLA-GD γ_t being near 0 or 0.1.

We evaluate online recurrent network methods on other architectures in Table 2. POLA achieve comparable or better performance than competing methods. The improved performance of Online-RMSprop in LSTM compared to the other architectures could be due to the larger number of model parameters, such that element-wise learning rates can provide localized adaptation. However while POLA is consistently better or on par with baseline Online-SGD, RMSprop is inconsistent and in some cases performs much worse.

METHOD	NORMALIZED RMSE	
	Sunspot	Power
Holt-Winters	0.991	NA
OR-ELM	0.822	NA
Pre-trained	0.572	0.816
FTL*	0.572	0.820
MAML	1.295	1.023
Online-SGD	0.552	0.775
Online-RMSprop	0.536	0.809
WG	0.552	NA
POLA-FS	0.532 ± 0.002	**0.769** ± 0.003
POLA-GD	**0.500** ± 0.002	0.773 ± 0.005

Table 1: Prediction performance of RNN and time series models. Results of univariate methods on multivariate dataset are marked NA. * FTL retrains with entire historical data.

5.3. Sensitivity Analysis

We conduct sensitivity analysis on the hyperparameters of POLA-GD and the online batch size b. From Table 3, we see that the prediction performance of POLA-GD RNN can depend on the choice of hyperparameters k and η. Overall, a larger number of update steps k tend to give better performance as the optimization can search for a better solution of $\beta_{t+1}^{(V)}$.

MODEL	METHOD	NORMALIZED RMSE	
		Sunspot	Power
LSTM	Online-SGD	0.532	0.821
	Online-RMSprop	0.517	**0.794**
	WG	0.532	NA
	POLA-FS	0.534 ± 0.009	0.802 ± 0.005
	POLA-GD	**0.512** ± 0.006	0.806 ± 0.071
GRU	Online-SGD	0.526	**0.768**
	Online-RMSprop	0.521	0.786
	WG	0.526	NA
	POLA-FS	0.508 ± 0.002	0.769 ± 0.003
	POLA-GD	**0.489** ± 0.002	**0.768** ± 0.003

Table 2: Prediction performance of recurrent network models. Results of univariate methods on multivariate dataset are marked NA.

# STEPS (k)	LEARNING RATE (η)	NORMALIZED RMSE	
		Sunspot	Power
1	0.1	0.515	0.773
2	0.1	0.504	0.772
3	0.1	0.500	0.773
1	0.01	0.525	0.777
2	0.01	0.520	0.776
3	0.01	0.516	0.775

Table 3: Prediction performance of POLA-GD RNN learned with different hyperparameters.

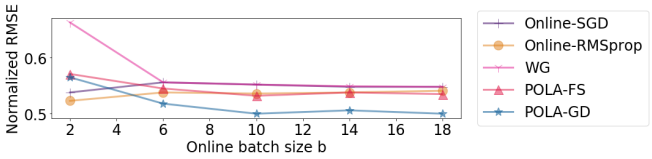

Fig. 3: Normalized RMSE across different online batch sizes b with RNN model on Sunspot dataset.

Figure 3 shows the performance of online RNN methods using different online batch size b for $b \in \{2, 6, 10, 14, 18\}$. POLA has comparable or better performance than competing methods at $b \geqslant 10$. Since each batch of samples is split into two at the meta-stage, a larger b helps to estimate the learning rate more accurately.

6. CONCLUSION

In this paper, we propose POLA to automatically adapt the SGD learning rate to adjust recurrent neural network models for predicting real-time in dynamic environments. The learning rate is estimated using a novel meta-learning procedure that assimilates the interleaved-test-then-train evaluation. POLA demonstrates overall comparable or better predictive performance over other competing methods across multiple datasets and network architectures.

7. REFERENCES

[1] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54 – 71, 2019.

[2] Brendan McMahan, "Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization," *Proceedings of Machine Learning Research*, vol. 15, pp. 525–533, 2011.

[3] Lin Xiao, "Dual averaging method for regularized stochastic learning and online optimization," in *NIPS*. 2009.

[4] H. Brendan McMahan, "A survey of algorithms and analysis for adaptive online learning," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 3117–3166, Jan. 2017.

[5] Kohei Miyaguchi and Hiroshi Kajino, "Cogra: Concept-drift-aware stochastic gradient descent for time-series forecasting," in *AAAI*, 2019.

[6] T. Liu, S. Chen, S. Liang, S. Gan, and C. J. Harris, "Fast adaptive gradient RBF networks for online learning of nonstationary time series," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2015–2030, 2020.

[7] Junjie Lu, Jinquan Huang, and Feng Lu, "Time series prediction based on adaptive weight online sequential extreme learning machine," *Applied Sciences*, vol. 7, pp. 217, 2017.

[8] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir, "Online learning for time series prediction," 2013, vol. 30 of *Proceedings of Machine Learning Research*, pp. 172–184.

[9] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134 – 147, 2017, Online Real-Time Learning Strategies for Data Streams.

[10] Chung Chen and Lon-Mu Liu, "Forecasting time series with outliers," *Journal of Forecasting*, vol. 12, no. 1, pp. 13–35, 1993.

[11] Sascha Krstanovic and Heiko Paulheim, "Ensembles of recurrent neural networks for robust time series forecasting," in *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 11 2017, pp. 34–46.

[12] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *International Journal of Forecasting*, 2020.

[13] Wenyu Zhang, Devesh K. Jha, Emil Laftchiev, and Daniel Nikovski, "Multi-label prediction in time series data using deep neural networks," *IJPHM*, vol. 10, no. 26, 2019.

[14] Tom Schaul, Sixin Zhang, and Yann LeCun, "No more pesky learning rates," 2013, vol. 28 of *Proceedings of Machine Learning Research*, pp. 343–351, PMLR.

[15] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey, "Meta-learning in neural networks: A survey," *ArXiv*, 2020.

[16] Chenghao Liu, S. Hoi, P. Zhao, and Jianling Sun, "Online ARIMA algorithms for time series prediction," in *AAAI*, 2016.

[17] Peter Bartlett, Elad Hazan, and Alexander Rakhlin, "Adaptive online gradient descent," in *NIPS*, 2007.

[18] A. P. George and W. Powell, "Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming," *Machine Learning*, vol. 65, pp. 167–198, 2006.

[19] S. Jenni and P. Favaro, "Deep bilevel learning," in *ECCV*, 2018.

[20] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky, *Neural Network for Machine Learning*, Accessed September 24, 2020, `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[21] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya, "Robust online time series prediction with recurrent neural networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 816–825.

[22] Haimin Yang, Zhisong Pan, and Qing Tao, "Robust and adaptive online time series prediction with long short-term memory," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1–9, 12 2017.

[23] Guang-Bin Huang, Nanying Liang, Hai-Jun Rong, Paramasivan Saratchandran, and Narasimhan Sundararajan, "On-line sequential extreme learning machine.," in *Proceedings of the IASTED International Conference on Computational Intelligence*, 01 2005, vol. 2005, pp. 232–237.

[24] J. Park and J. Kim, "Online recurrent extreme learning machine and its application to time-series prediction," in *IJCNN*, 2017, pp. 1983–1990.

[25] N. Wagner, Z. Michalewicz, M. Khouja, and R. R. McGregor, "Time series forecasting for dynamic environments: The dyfor genetic program model," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 4, pp. 433–452, 2007.

[26] Hongyuan Zhan, Gabriel Gomes, Xiaoye Li, Kamesh Madduri, and Kesheng Wu, "Efficient online hyperparameter optimization for kernel ridge regression with applications to traffic time series prediction," *ArXiv*, 11 2018.

[27] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars, "Task-free continual learning," in *CVPR*, 2019.

[28] Dushyant Rao, Francesco Visin, Andrei Rusu, Yee Teh, Razvan Pascanu, and Raia Hadsell, "Continual unsupervised representation learning," in *Neurips*, 2019.

[29] Nicholas Polson, James Scott, and Brandon Willard, "Proximal algorithms in statistics and machine learning," *Statistical Science*, vol. 30, 02 2015.

[30] Souhaib Ben Taieb, Gianluca Bontempi, Amir Atiya, and Antti Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert Systems with Applications*, vol. 39, 08 2011.

[31] Anusha Nagabandi, I. Clavera, Simin Liu, Ronald S. Fearing, P. Abbeel, S. Levine, and Chelsea Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv*, 2019.

[32] A. Miranian and M. Abdollahzade, "Developing a local least-squares support vector machines-based neuro-fuzzy model for nonlinear and chaotic time series prediction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 2, pp. 207–218, 2013.

[33] W. Zhang, D. Gilbert, and D. Matteson, "ABACUS: Unsupervised multivariate change detection via bayesian source separation," in *SDM*, 2019.