Industrial Forecasting with Exponentially Smoothed Recurrent Neural Networks

Matthew F. Dixon*

February 2020

Abstract

Industrial forecasting has entered an era of unprecedented growth in the size and complexity of data which require new modeling methodologies. While many new general purpose machine learning approaches have emerged, they remain poorly understood and irreconcilable with more traditional statistical modeling approaches. We present a general class of exponential smoothed recurrent neural networks (RNNs) which are well suited to modeling non-stationary dynamical systems arising in industrial applications such as electricity load management and financial risk and trading. In particular, we analyze their capacity to characterize the non-linear partial autocorrelation structure of time series and directly capture dynamic effects such as seasonality and regime changes. Application of exponentially smoothed RNNs to electricity load forecasting, weather data and financial time series, such as minute level Bitcoin prices and CME futures tick data, highlight the efficacy of exponential smoothing for multi-step time series forecasting. The results also suggest that popular, but more complicated neural network architectures originally designed for speech processing, such as LSTMs and GRUs, are likely over-engineered for industrial forecasting and light-weight exponentially smoothed architectures capture the salient features while being superior and more robust than simple RNNs.

1 Background

Recurrent neural networks (RNNs) are the building blocks of modern sequential learning. RNNs use recurrent layers to capture non-linear temporal dependencies with a relatively small number of parameters [15]. They learn temporal dynamics by mapping an input sequence to a hidden state sequence and outputs, via a recurrent layer and a feedforward layer. However, despite the success of these and their successors, there appears to be a chasm between the statistical modeling literature (see e.g. [6, 20, 16]) and the machine learning literature (see e.g. [3, 29, 18, 30]).

While there have been many recent theoretical developments in recurrent neural networks from a dynamical systems perspective [22, 8, 27], there are still open fundamental questions as to how the type and properties of the time series data informs the choice of architectures. For example, [24] find empirical evidence that Echo State Networks are well suited for spatio-temporal modeling and Uber [32] won the 2019 M4 forecasting competition with a hybrid exponential smoothing-RNN method, which exponentiates the output of a neural network and combines it with a past observed time series level. There have been exhaustive empirical studies on the application of

*Matthew Dixon is an Assistant Professor in the Department of Applied Math, Illinois Institute of Technology. E-mail: matthew.dixon@iit.edu.
recurrent neural networks to prediction from financial time series data such as historical limit order book and price history [10, 4, 5, 31, 7, 26]. [31] find evidence that stacking networks leads to superior performance on intra-day stock data combined with technical indicators, whereas [2] combine wavelet transforms and stacked autoencoders with LSTMs on OHLC bars and technical indicators. [5] find evidence that dilated convolutional networks out perform LSTMs on various indices. [10] demonstrate that RNNs outperform feed-forward networks with lagged features on limit order book data.

There are still open fundamental questions as to how the type and properties of the time series data inform the choice of these architectures. One of the main contributions of this paper is to introduce a new class of RNNs, with supporting theoretical justification for architectural design choices.

A second challenge with applying neural networks to time series data is the over-reliance on extensive hyper-parameter tuning, a computationally complex global optimization problem with undesirable outcomes which are dependent on initial conditions for parameter choices. Moreover, little is known about how often to re-tune the parameters and the length of historical data used to train the model can affect the hyper-parameter tuning.

A third challenge is how to combine more traditional and informative diagnostics of time series data, such as stationarity and memory cut-off, into the model selection procedure. Such an approach is standard in the time series modeling literature [6] but absent in the machine learning literature. Finally, most of the aforementioned studies are engineering orientated and do not provide substantive insight to justify why the architectural choice is well suited to the dataset.

One of the main contributions of this paper is to cast RNNs into a time series modeling framework and rely on statistical diagnostics in combination with cross-validation to tune the architecture. The statistical tests characterize stationarity and memory cut-off length and provide insight into whether the data is suitable for longer-term forecasting and whether the model must be non-stationary.

It is well known that plain RNNs have difficulty in learning long-term dynamics, due in part to the vanishing and exploding gradients that can result from back propagating the gradients down through the many unfolded layers of the network [3, 29]. A particular type of RNN, called a Long Short-Term Memory (LSTM) [18, 30] was proposed to address this issue of vanishing or exploding gradients which essentially arises due to the shape of the activation function. A memory unit used in a LSTM allows the network to learn which previous states can be forgotten and alleviates the vanishing gradient. Partly for this reason, LSTMs have demonstrated much empirical success in the literature [13, 34].

The inclusion of forget, reset and update gates in the LSTM, and a slightly simpler variant — Gated Recurrent Units (GRUs) [9], provides a switching mechanism to forget memory while simultaneously updating a hidden state vector. These units do not, however, provide an easily accessible mathematical structure from which to study their time series modeling properties. As such, there is much opaqueness about the types of architectures which are best suited to prediction tasks based on data properties such as wide sense non-stationarity (see Appendix A). However we shall show that exponential smoothing not only alleviates the gradient problem but characterizes the time series modeling properties of these architectures.

The main outcome of applying this approach is that it partially identifies the choice of architecture based on both its time series properties and that of the data. The approach is general and easily extensible to GRUs and LSTMs with the inclusion of the reset gate and cell memory. In this paper, the input sequence is assumed to be of fixed length, although the methodology could be
extended to variable length as in sequence to sequence learning models [33]. A summary of the 
main contributions of this paper are

- We show how plain RNNs, with short-term memory, can be generalized to exhibit long-term 
memory with a smoothing scalar \( \alpha \)—smoothing also helps offset the infamous vanishing 
gradient problem in plain RNNs with only one extra parameter;

- We show how time series analysis guides the architectural parameters—the sequence length 
of the RNN can be determined from the estimated partial autocorrelogram, and tanh 
activation of the recurrent layer is needed for stability; and

- We demonstrate how a dynamic \( \alpha \)-RNN model for non-stationary data [11], a lightweight 
version of GRUs and LSTMs, has the ability to model complex non-stationary times series 
data with comparable performance.

The remainder of this paper is outlined as follows. Section 2 introduces the \( \alpha \)-RNN and Section 
3 applies time series analysis to guide the architectural properties. Section 4 introduces a 
dynamic version of the model and illustrates the dynamical behavior of \( \alpha \). Details of the training, 
implementation and experiments together with the results are presented in Section 6. Finally, 
Section 7 concludes with directions for future research.

2 \( \alpha \)-RNNs

Given auto-correlated observations of covariates or predictors, \( \mathbf{X}_t \), and responses \( \mathbf{Y}_t \) at times 
\( t = 1, \ldots, N \), in the time series data \( \mathcal{D} := \{ \mathbf{x}_t \}_{t=1}^{N} \), \( \mathbf{x}_t := (x_t, y_t) \), our goal is to construct an m-step 
\((m > 0)\) ahead times series predictor \( \hat{y}_{t+m} = F(\tilde{x}_t) \), of an observed target, \( y_{t+m} \), from a p length 
input sequence \( \bar{x}_t \)

\[
\mathbf{y}_{t+m} := F(\mathbf{x}_t) + u_t, \quad \text{where} \quad \bar{x}_t := \{x_{t-p+1}, \ldots, x_t\},
\]

\( x_{t-j} =: L^j[\mathbf{x}_t] \) is a \( j^{th} \) lagged observation of \( \mathbf{x}_t \in \mathbb{R}^d \), for \( j = 0, \ldots, p - 1 \) and \( u_t \) is the homoschedastic 
model error at time \( t \). We introduce the \( \alpha \)-RNN:

\[
\hat{y}_{t+m} = F_{W,b,\alpha}(\tilde{x}_t)
\]

where \( F_{W,b,\alpha}(\tilde{x}_t) \) is an \( \alpha \in [0, 1] \) smoothed RNN with weight matrices \( W := (W_h, U_h, W_y) \), and 
biases vectors \( b := (b_h, b_y) \).

For each time step in the sequence \( s = t - p + 2, \ldots, t \), forward passes separately update a hidden 
internal state \( \hat{h}_s \in \mathbb{R}^H \), using the recurrence relations:

\[
\hat{h}_s = \sigma(W_h x_s + U_h \hat{h}_{s-1} + b_h), \\
\tilde{h}_s = \alpha \hat{h}_s + (1 - \alpha) \hat{h}_{s-1},
\]

where the input weight matrix \( W_h \in \mathbb{R}^{H \times d} \) and the recurrence weight matrix \( U_h \in \mathbb{R}^{H \times H} \). \( H \) 
is the number of hidden units and \( d \) is the dimensionality of the input space. \( \tilde{h}_s \in \mathbb{R}^H \) is an 
exponentially smoothed version of the hidden state \( \hat{h}_s \). When the output is continuous, the 
output from the final hidden state is given by:

\[
\hat{y}_{t+m} = W_y \hat{h}_t + b_y,
\]

with the starting condition in each sequence, \( \hat{h}_{t-p+1} = \sigma(W_h x_{t-p+1}) \).
3 Times Series Modeling

This section bridges the time series modeling literature \[6,20,21\] and the machine learning literature. We shall assume here for ease of exposition that the time series data is univariate \((d = 1)\), \(D := \{y_t\}_{t=1}^N\), and thus the predictor is endogenous.\(^1\)

Since autoregressive \((AR(p))\) models are well known in time series modeling, we find it instructive to show that plain RNNs are non-linear AR\((p)\) models. For ease of exposition, consider the simplest case of a RNN with one hidden unit, \(H = 1\). Without loss of generality, we set \(U_h = W_h = \phi, W_y = 0, b_y = \mu\). Then we can show by backward substitution that a plain-RNN, \(F_{W,b}(\tilde{x}_t)\), with sequence length \(p\), is a non-linear auto-regressive, \(NAR(p)\), model of order \(p\).

\[
\begin{align*}
\hat{h}_{t-p+1} &= \sigma(\phi y_{t-p+1}) \\
\hat{h}_{t-p+2} &= \sigma(\phi \hat{h}_{t-p+1} + \phi y_{t-p+2}) \\
\vdots &= \vdots \\
\hat{h}_t &= \sigma(\phi \hat{h}_{t-1} + \phi y_t) \\
\hat{y}_{t+m} &= \hat{h}_t + \mu \\
\end{align*}
\]

then

\[
\hat{y}_{t+m} = \mu + \sigma(1 + \sigma(\phi(L + \sigma(\phi(L^2 + \cdots + \sigma(\phi L^{p-1}) \cdots ))[y_t]). \tag{3}
\]

When the activation is the identity function \(\sigma := I_d\), then we recover the AR\((p)\) model

\[
\hat{y}_{t+m} = \mu + \sum_{i=0}^{p-1} \phi_i L^i [y_t], \quad \phi_i = \phi^i. \tag{4}
\]

with geometrically decaying autoregressive coefficients when \(|\phi| < 1\).

\(\alpha\)-RNNs The \(\alpha\)-RNN\((p)\) is almost identical to a plain RNN, but with an additional scalar smoothing parameter, \(\alpha\), which provides the recurrent network with “long-memory”\(^2\). To see this, let us consider a one-step ahead univariate \(\alpha\)-RNN\((p)\) in which the smoothing parameter is fixed. For each time step \(s = t - p + 2, \ldots, t\):

\[
\begin{align*}
\text{(output)} \quad \hat{y}_{t+1} &= W_y \hat{h}_t + b_y, \\
\text{(hidden state update)} \quad \hat{h}_s &= \sigma(U_h \hat{h}_{s-1} + W_h y_s + b_h), \\
\text{(smoothing)} \quad \tilde{h}_s &= \alpha \hat{h}_s + (1 - \alpha) \hat{h}_{s-1}. \\
\end{align*}
\]

This model augments the plain-RNN by replacing \(\hat{h}_{s-1}\) in the hidden layer with an exponentially smoothed hidden state \(\tilde{h}_{s-1}\). The effect of the smoothing is to provide infinite memory when \(\alpha \neq 1\). For the special case when \(\alpha = 1\), we recover the plain RNN with short memory of length \(p \ll N\).

---

\(^1\)The sequence of features is from the same time series as the predictor.

\(^2\)Long memory refers to autoregressive memory beyond the sequence length. This is also sometimes referred to as “stateful”. For avoidance of doubt, we are not suggesting that the \(\alpha\)-RNN has an additional cellular memory, as in LSTMs.
We can easily see this informally by simplifying the parameterization and considering the unactivated case. Setting $b_y = b_h = 0$, $U_h = W_h = \phi \in \mathbb{R}$ and $W_y = 1$:

\[
\begin{align*}
\hat{y}_{t+1} &= \hat{h}_t, \\
&= \phi(\hat{h}_{t-1} + y_t), \\
&= \phi(a\hat{h}_{t-1} + (1-\alpha)\hat{h}_{t-2} + y_t),
\end{align*}
\]

with the starting condition in each sequence, $\hat{h}_{t-p+1} = \phi y_{t-p+1}$. With out loss of generality, consider $p = 2$ lags in the model so that $\hat{h}_{t-1} = \phi y_{t-1}$. Then

\[
\hat{h}_t = \phi(a\phi y_{t-1} + (1-\alpha)\hat{h}_{t-2} + y_t)
\]

and the model can be written in the simpler form

\[
\hat{y}_{t+1} = \phi_1 y_t + \phi_2 y_{t-1} + \phi(1-\alpha)\hat{h}_{t-2},
\]

with auto-regressive weights $\phi_1 := \phi$ and $\phi_2 := a\phi^2$. We now see that there is a third term on the RHS of Eq. [9] which vanishes when $\alpha = 1$ but provides infinite memory to the model since $\hat{h}_{t-2}$ depends on $y_1$, the first observation in the whole time series, not just the first observation in the sequence. To see this, we unroll the recursion relation in the exponential smoother:

\[
\hat{h}_{t+1} = a \sum_{s=0}^{t-1} (1-\alpha)^s \hat{h}_{t-s} + (1-\alpha)^s y_1.
\]

where we used the property that $\hat{h}_1 = y_1$. It is often convenient to characterize exponential smoothing by the half-life — the number of lags needed for the coefficient $(1-\alpha)^s$ to equal a half, which is $s = -1/\log_2(1-\alpha)$. To gain further insight we use partial auto-correlations to characterize the memory of the model.

### 3.1 Partial Autocorrelation

We consider autoregressive time series models, with additive white noise of the form

\[
y_t = \tilde{y}_t + u_t, \quad u_t \sim N(0, \sigma^2_u),
\]

which carry a signature which allows its order, $p$, to be determined from “covariance stationary” time series data (see Appendix A for a definition of covariance stationarity). This signature encodes the memory in the model and is given by “partial autocorrelations”. Informally each partial autocorrelation measures the correlation of a random variable, $y_t$, with its $h$th lag, $y_{t-h}$, while controlling for intermediate lags. The partial autocorrelation must be non-zero to be able to predict $y_t$ from $y_{t-h}$. The sign of the partial autocorrelation is also useful for interpretability and describes the directional relationship between the random variables. The formal definition of the partial autocorrelation is now given.

**Definition 3.1** (Partial Autocorrelation). A partial autocorrelation at lag $h \geq 2$ is a conditional autocorrelation between a variable, $y_t$, and its $h$th lag, $y_{t-h}$ under the assumption that the values of the intermediate lags, $y_{t-1}, \ldots, y_{t-h+1}$ are controlled:

---

The assumption of Gaussian white noise is convenient but not necessary — any type of white noise is possible.
\[ \tilde{\tau}_h := \tilde{\gamma}_{t, t-h} := \frac{\tilde{\gamma}_h}{\sqrt{\tilde{\gamma}_{t,h} \tilde{\gamma}_{t-h,h}}}, \]

where

\[ \tilde{\gamma}_h := \tilde{\gamma}_{t, t-h} \]
\[ := \mathbb{E}[y_t - P(y_t \mid y_{t-1}, \ldots, y_{t-h+1}), y_{t-h} - P(y_{t-h} \mid y_{t-1}, \ldots, y_{t-h+1})] \]

is the lag-\(h\) partial autocovariance, \(P(W \mid Z)\) is an orthogonal projection of \(W\) onto the set \(Z\) and

\[ \tilde{\gamma}_{t,h} := \mathbb{E}[(y_t - P(y_t \mid y_{t-1}, \ldots, y_{t-h+1}))^2]. \quad (11) \]

The partial autocorrelation function (PACF) \(\tilde{\tau}_h : \mathbb{N} \rightarrow [-1, 1]\) is a map \(h \mapsto \tilde{\tau}_h\). The plot of \(\tilde{\tau}_h\) against \(h\) is referred to as the partial correlogram.

The PACF of the RNN(p) can be used to determine the lower bound on the sequence length in an \(\alpha\)-RNN(p). To see this, we first show that the partial autocorrelation of the \(\alpha\)-RNN(p) is time independent and has a sharp cut-off after \(p\) lags if \(\alpha = 1\).

**Theorem 1** (Partial autocorrelation of an \(\alpha\)-RNN(p)). The partial autocorrelation of the \(\alpha\)-RNN(p) is time independent and exhibits a cut-off after \(p\) lags: \(\tilde{\tau}_s = 0, s > p\) if \(\alpha = 1\). If \(\alpha \in (0, 1)\) the \(\alpha\)-RNN(p) has non-zero partial autocorrelations at lags beyond the sequence length.

See Appendix \[B\] for a proof. For \(\alpha \in (0, 1)\), the \(\alpha\)-RNN has non-zero partial autocorrelations \(\tilde{\tau}_s \neq 0, s > p\). It is easy to see this from the additional term containing \(\alpha\) in Equation (9). Further insight can be gained from Figure 1 which shows the fitted partial correlogram from data generated by an \(\alpha\)-RNN(3) with additive white noise. We observe that the memory is always longer than the sequence length of 3 when \(\alpha \in (0, 1)\). As \(\alpha\) approaches zero, the model has increasing memory.

The theorem and the properties of the \(\alpha\)-RNN(p) suggest to determine the sequence length \(p\) from the fitted PACF of each covariate. Moreover, the prediction horizon should not exceed that suggested by the maximum order of statistically significant partial autocorrelations.

### 3.2 Stability

Times series modeling also places emphasis on model ”stability” — this is the model attribute that past random disturbances decay in the model and the effect of lagged data becomes less relevant to the model output with increasing lag. We present the following theorem which shows that the stability of the \(\alpha\)-RNN model is determined solely by hidden state’s activation function, with the property that \(|\sigma(\cdot)| < 1|\).

**Theorem 2** (Stability of RNN(p) models). Suppose that there exists an invertible nonlinear function of the lag operator \(\Phi(L)\) of the form:

\[ y_t = \Phi^{-1}(L)[u_t] \]
\[ = (1 - \sigma(\phi \sigma(\phi \sigma(\phi \sigma(\ldots)))) + \ldots \]
\[ + \phi L^2) + \phi L^{-1}[u_t], \]

where, without loss of generality, we have again conveniently set \(W_x = U_h = \phi, W_y = 1\) and \(b_h = b_y = 1\). Then the RNN is stable if and only if \(|\sigma(x)| < 1\) for all finite \(x\).

\[ ^4\text{Although the model has no memory in the limit } \alpha = 0. \]
Figure 1: The fitted partial correlogram of univariate data generated by various additive noise \( \alpha \)-RNN\((3)\) models.

See Appendix C for a proof. In particular, the theorem justifies the common choice of \( \sigma(\cdot) := \tanh(\cdot) \) and while sigmoid is also another viable choice, it is too restrictive as it prohibits negative partial auto-correlations. We shall see in Section 6 that negative partial auto-correlations arise in time series data.

### 3.3 Vanishing gradient

It is well known that plain-RNNs exhibit a vanishing gradient problem [29, 3]. Following [29], we extend some of the analysis of BPTT to \( \alpha \)-RNNs. The \( \alpha \)-RNN\((p)\) BPTT can be written as:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{N} \frac{\partial L}{\partial W} = \sum_{t=1}^{N} \frac{\partial L}{\partial h_i} \sum_{k=t-p}^{t-1} \prod_{i=k+1}^{t-1} \frac{\partial h_i}{\partial h_{i-1}} U_h.
\]

for some generic loss function \( L_t \), where

\[
\frac{\partial h_i}{\partial h_{i-1}} = (1 - \alpha) + \alpha \frac{\partial h_i}{\partial h_{i-1}}
\]

\[
= (1 - \alpha) + \alpha \sigma'(I_i) U_h, \quad I_i := W_h x_i + U_h h_{i-1} + b_h.
\]

Substituting the expression for \( \frac{\partial h_i}{\partial h_{i-1}} \) into Equation 12 gives an expression proportional to

\[
\prod_{i=k+1}^{t-1} (1 - \alpha) + \alpha \sigma'(I_i) U_h, \quad I_i := W_h x_i + U_h h_{i-1} + b_h.
\]
When $\alpha = 1$ this expression is $\prod_{i=k+1}^{t-1} \sigma'(I_i)U_i$. When $\sigma(\cdot) := \tanh(\cdot)$, this product goes to zero with increasing function compositions due to the gradient of tanh vanishing with large $|x|$ and the product of small functions. However, when $\alpha \neq 1$, the additional term provides an additional contribution to the gradient which is non trivial for small $\alpha$ and independent of the input.

Thus the $\alpha$-RNN will not only alleviate the vanishing gradient problem, has guaranteed stability if we use choose a tanh activation function, but exhibits non-zero partial auto-correlations up to at least lag $p$ for $\alpha \in (0, 1]$.

The $\alpha$-RNN model can be trained by treating $\alpha$ as an additional parameter to be fitted with stochastic gradient descent. The choice to pre-determine that $\alpha$ is independent of time is obviously limited to stationary time series. While this is restrictive, it suggests that a simple statistical test of the time series can pre-determine the efficacy of the approach\footnote{For multivariate data, the test should be applied to each covariate.}. Moreover, if the data is covariance stationary, then the $\alpha$-RNN will preserve the stationarity of the partial auto-correlation structure, eliminating the need for more complex architectures such as GRUs and LSTMs (which are often motivated purely on the basis of the vanishing gradient problem). Such a procedure shall be demonstrated in Section\footnote{For multivariate data, the test should be applied to each covariate.}.

We can extend the model to non-stationary time series, which exhibit dynamic partial autocorrelation, by using a dynamic version of exponential smoothing.

### 4 Dynamic $\alpha_t$-RNNs

The extension of RNNs to dynamical time series models, suitable for non-stationary time series data, relies on dynamic exponential smoothing is a time dependent, convex, combination of the smoothed output, $\hat{h}_t$, and the hidden state $\tilde{h}_t$:

$$\tilde{h}_{t+1} = \alpha_t \hat{h}_t + (1 - \alpha_t)\tilde{h}_t,$$

(14)

where $\alpha_t \in [0, 1]$ denotes the dynamic smoothing factor which can be equivalently written in the one-step-ahead forecast of the form

$$\tilde{h}_{t+1} = \hat{h}_t + \alpha_t(\hat{h}_t - \tilde{h}_t).$$

(15)

Hence the smoothing can be viewed as a form of dynamic forecast error correction; When $\alpha_t = 0$, the forecast error is ignored and the smoothing merely repeats the current hidden state $\tilde{h}_t$, to the effect of the model losing its memory. When $\alpha_t = 1$, the forecast error overwrites the current hidden state $\tilde{h}_t$. The smoothing can also be viewed a weighted sum of the lagged observations, with lower or equal weights, $\alpha_{t-s} \prod_{r=1}^{s}(1 - \alpha_{t-r+1})$ at the $s \geq 1$ lagged hidden state, $\tilde{h}_{t-s}$:

$$\tilde{h}_{t+1} = \alpha_t \hat{h}_t + \sum_{s=1}^{t-1} \alpha_{t-s} \prod_{r=1}^{s}(1 - \alpha_{t-r+1})\tilde{h}_{t-s} + g(\alpha),$$

where $g(\alpha) := \prod_{r=0}^{t-1}(1 - \alpha_{t-r})\tilde{y}_1$. Note that for any $\alpha_{t-r+1} = 1$, the smoothed hidden state $\tilde{h}_{t+1}$ will have no dependency on all lagged hidden states $\{\tilde{h}_{t-s}\}_{s \geq r}$. The model simply forgets the hidden states at or beyond the $r^{th}$ lag.
4.1 Neural network exponential smoothing

While the class of $\alpha_t$-RNN models under consideration is free to define how $\alpha$ is updated (including changing the frequency of the update) based on the hidden state and input, a convenient choice is use a recurrent layer. Returning again to the more general setup with a hidden state vector $\hat{h}_t \in \mathbb{R}^H$, let us model the smoothing parameter $\hat{\alpha}_t \in [0, 1]^H$ to give a filtered time series

$$\tilde{h}_t = \hat{\alpha}_t \circ \hat{h}_t + (1 - \hat{\alpha}_t) \circ \tilde{h}_{t-1},$$

(16)

where $\circ$ denotes the Hadamard product between vectors. This smoothing is a vectorized form of the above classical setting, only here we note that when $(\alpha_t)_i = 1$, the $i^{th}$ component of the hidden variable is unmodified and the past filtered hidden variable is forgotten. On the other hand, when the $(\alpha_t)_i = 0$, the $i^{th}$ component of the hidden variable is obsolete, instead setting the current filtered hidden variable to its past value. The smoothing in Equation [16] can be viewed then as updating long-term memory, maintaining a smoothed hidden state variable as the memory through a convex combination of the current hidden variable and the previous smoothed hidden variable.

The hidden variable is given by the semi-affine transformation:

$$\tilde{h}_t = \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h)$$

(17)

which in turns depends on the previous smoothed hidden variable. Substituting Equation [17] into Equation [16] gives a function of $\tilde{h}_{t-1}$ and $x_t$:

$$\tilde{h}_t = \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h)$$

(18)

$$= \hat{\alpha}_t \circ \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h) + (1 - \hat{\alpha}_t) \circ \tilde{h}_{t-1}.$$ (19)

We see that when $\alpha_t = 0$, the smoothed hidden variable $\tilde{h}_t$ is not updated by the input $x_t$. Conversely, when $\alpha_t = 1$, we observe that the hidden variable locally behaves like a non-linear autoregressive series. Thus the smoothing parameter can be viewed as the sensitivity of the smoothed hidden state to the input $x_t$.

The challenge becomes how to determine dynamically how much error correction is needed. As in GRUs and LSTMs, we can address this problem by learning $\hat{\alpha} = F(W_{\alpha}, U_{\alpha}, b_{\alpha})(\bar{x}_t)$ from the input variables with the recurrent layer parameterized by weights and biases $(W_{\alpha}, U_{\alpha}, b_{\alpha})$. The one-step ahead forecast of the smoothed hidden state, $\tilde{h}_t$, is the filtered output of another plain RNN with weights and biases $(W_h, U_h, b_h)$.

**Comparison with $\alpha$-RNNs** Figure [2a] shows the response of a univariate $\alpha$-RNN model when the input consists of two unit impulses and zeros otherwise. For simplicity, the sequence length is assumed to be 3 (i.e. the RNN has a memory of 3 lags), the biases are set to zero, all the weights are set to one and $\sigma(x) := tanh(x)$. The RNN loses memory of the unit impulse after three lags, whereas the RNNs with smooth hidden states maintain memory of the first unit impulse even when the second unit impulse arrives. The figure also shows an $\alpha_t$-RNN model, which although appears insignificant in this example, allows the model to fit non-stationary time series data. This is because the time dependent $\alpha_t$ results in dynamic partial autocorrelations. The response of $\hat{\alpha}_t$ to shocks can be seen in Figure [2b].

See Appendix 5 for a discussion of the differences between the $\alpha_t$-RNN and the GRU and LSTM.

---

6 Note that the weights have not been fitted here, we are merely observing the effect of smoothing on the hidden state for the simplest choice of parameter values.

7 Note that the value of $\alpha$ is initially set to one since no long-term memory is needed. Hence the response to the second shock at time $t = 12$ is more insightful.
5 Relationship to GRUs and LSTMs

5.1 GRUs

The $\alpha_t$-RNN model has no means to entirely reset its memory and become a feed-forward network (FFN). This is because the hidden variables update equation always depends on the previous smoothed hidden state, unless $U_h = 0$. By adding a reset layer, we recover a GRU:
smoothing: \[ \hat{h}_t = \hat{\alpha}_t \odot \hat{h}_t + (1 - \hat{\alpha}_t) \odot \hat{h}_{t-1} \]

smoother update: \[ \hat{\alpha}_t = \sigma^{(1)}(U_{\alpha} \hat{h}_{t-1} + W_{\alpha} x_t + b_{\alpha}) \]

hidden state update: \[ \hat{h}_t = \sigma(U_h \hat{\alpha}_t \odot \hat{h}_{t-1} + W_h x_t + b_h) \]

reset update: \[ \hat{r}_t = \sigma^{(1)}(U_r \hat{h}_{t-1} + W_r x_t + b_r). \]

When viewed as an extension of our \(\alpha_t\) RNN model, we observe that the effect of introducing a reset, or switch, \(\hat{r}_t\), is to forget the dependence of \(\hat{h}_t\) on the smoothed hidden state. Effectively, we turn the update for \(\hat{h}_t\) from a plain RNN to a FFN and entirely neglect the recurrence. The recurrence in the update of \(\hat{h}_t\) is thus dynamic. It may appear that the combination of a reset and adaptive smoothing is redundant. But remember that \(\hat{\alpha}_t\) effects the level of error correction in the update of the smoothed hidden state, \(\tilde{h}_t\), whereas \(\hat{r}_t\) adjusts the level of recurrence in the unsmoothed hidden state \(\hat{h}_t\). Put differently, \(\hat{\alpha}_t\) by itself cannot disable the memory in the smoothed hidden state (internal memory), whereas \(\hat{r}_t\) in combination with \(\hat{\alpha}_t\) can. More precisely, when \(\alpha_t = 1\) and \(\hat{r}_t = 0\), \(\hat{h}_t = \hat{\alpha}_t \odot \hat{h}_{t-1} + \hat{z}_t \odot \hat{c}_t\).

5.2 LSTMs

The \(\alpha_t\)-RNN model, like the GRU, provides a mechanism for propagating a smoothed hidden state — a long term memory which can be overridden and even turn the network into a plain RNN (with short memory) or even a memoryless FFN. More complex models using hidden units with varying connections within the memory unit have been proposed in the engineering literature with empirical success [18, 13, 34]. LSTMs are similar to GRUs but have a separate (cell) memory, \(c_t\), in addition to a hidden state \(h_t\). LSTMs also do not require that the memory updates are a convex combination. Hence they are more general than exponential smoothing. The mathematical description of LSTMs is rarely given in an intuitive form, but the model can be found in, for example, [18].

The cell memory is updated by the following expression involving a forget gate, \(\hat{\alpha}_t\), an input gate \(\hat{z}_t\) and a cell gate \(\hat{c}_t\)

\[ c_t = \hat{\alpha}_t \odot c_{t-1} + \hat{z}_t \odot \hat{c}_t. \]

In the terminology of LSTMs, the triple \((\hat{\alpha}_t, \hat{r}_t, \hat{z}_t)\) are respectively referred to as the forget gate, output gate, and input gate. Our change of terminology is deliberate and designed to provide more intuition and continuity with RNNs and the statistics literature. We note that in the special case when \(\hat{z}_t = 1 - \hat{\alpha}_t\) we obtain a similar exponential smoothing expression to that used in our \(\alpha_t\)-RNN. Beyond that, the role of the input gate appears superfluous and difficult to reason with using time series analysis.

When the forget gate, \(\hat{\alpha}_t = 0\), then the cell memory depends solely on the cell memory gate.
update \( \hat{c}_t \). By the term \( \hat{a}_t \circ c_{t-1} \), the cell memory has long-term memory which is only forgotten beyond lag \( s \) if \( \hat{a}_{t-s} = 0 \). Thus the cell memory has an adaptive autoregressive structure.

The extra “memory”, treated as a hidden state and separate from the cell memory, is nothing more than a Hadamard product:

\[
h_t = \hat{r}_t \odot \tanh(c_t),
\]

which is reset if \( \hat{r}_t = 0 \). If \( \hat{r}_t = 1 \), then the cell memory directly determines the hidden state.

Thus the reset gate can entirely override the effect of the cell memory’s autoregressive structure, without erasing it. In contrast, the \( \alpha \)-RNN and the GRU has one memory, which serves as the hidden state, and it is directly affected by the reset gate.

The reset, forget, input and cell memory gates are updated by plain RNNs all depending on the hidden state \( h_t \).

- Reset gate: \( \hat{r}_t = \sigma(U_r h_{t-1} + W_r x_t + b_r) \)
- Forget gate: \( \hat{a}_t = \sigma(U_a h_{t-1} + W_a x_t + b_a) \)
- Input gate: \( \hat{z}_t = \sigma(U_z h_{t-1} + W_z x_t + b_z) \)
- Cell memory gate: \( \hat{c}_t = \tanh(U_c h_{t-1} + W_c x_t + b_c) \).

Like the \( \alpha \)-RNN, the LSTM can function as a short-memory, plain-RNN; just set \( \alpha_t = 0 \) in Equation [21]. However, the LSTM can also function as a coupling of FFNs; just set \( \hat{r}_t = 0 \) so that \( h_t = 0 \) and hence there is no recurrence structure in any of the gates. For avoidance of doubt, since the nomenclature doesn’t suggest it, all models in this paper can model long and short-term autoregressive memory. The \( \alpha \)-RNN couples these memories through a smoothed hidden state variable. The LSTM separates out the long memory, stored in the cellular memory, but uses a copy of it, which may additionally be reset. Strictly speaking, the cellular memory has long-short autoregressive memory structure, so it would be misleading in the context of time series analysis to strictly discern the two memories as long and short (as the nomenclature suggests). The latter can be thought of as a truncated version of the former.

6 Numerical Experiments

This section describes numerical experiments using financial time series data to evaluate the various RNN models. All models are implemented in v1.15.0 of TensorFlow [1]. Times series cross-validation is performed using separate training, validation and test sets. To preserve the time structure of the data and avoid look ahead bias, each set represents a contiguous sampling period with the test set containing the most recent observations. To prepare the training, validation and testing sets for m-step ahead prediction, we set the target variables (responses) to the \( t + m \) observation, \( y_{t+m} \), and use the lags from \( t - p + 1, \ldots, t \) for each input sequence. This is repeated by incrementing \( t \) until the end of each set. In our experiments, each element in the input sequence is either a scalar or vector and the target variables are scalar.

We use the SimpleRNN Keras method with the default settings to implement a fully connected RNN. Tanh activation functions are used for the hidden layer with the number of units found by time series cross-validation with five folds to be \( H \in \{5, 10, 20\} \) and \( L_1 \) regularization, \( \lambda_1 \in \{0, 10^{-3}, 10^{-2}\} \). The Glorot and Bengio uniform method [14] is used to initialize the non-recurrent weight matrices and an orthogonal method is used to initialize the recurrence weights as a random orthogonal matrix. Keras’s GRU method is implemented using version 1406.1078v, which applies
the reset gate to the hidden state before matrix multiplication. Similarly, the LSTM method in Keras is used. Tanh activation functions are used for the recurrence layer and sigmoid activation functions are used for all other gates. The AlphaRNN and AlphatRNN classes are implemented by the authors for use in Keras. Statefulness is always disabled.

Each architecture is trained for up to 2000 epochs with an Adam optimization algorithm with default parameter values and using a mini-batch size of 1000 drawn from the training set. Early stopping is used with a patience of 50 to 100 and minimum delta between $10^{-8}$ and $10^{-6}$. No randomization is used in the mini-batching sampling in order to preserve the ordering of the data. To evaluate the forecasting accuracy, we set the forecast horizon to up to ten steps ahead instead of the usual step ahead forecasts often presented in the machine learning literature — longer forecasting horizons are often more relevant due to operational constraints in industry applications and are more challenging when the data is non-stationary since the network’s fixed partial auto-correlation will not adequately capture the observed changing autoregressive structure. The numerical results are separated in two areas of interest: (i) properties of recurrent architectures on archetypal data properties such as seasonality and regime switching; and (ii) evaluation on real data to demonstrate their utility in industrial forecasting.

6.1 LLM Seasonality DGP

To characterize the ability of exponentially smoothed RNNs to directly capture seasonality from noisy data, without the need to separately deseasonalize the data, we generate hourly data from an additive local level model with daily seasonality and i.i.d. noise [17]:

\[
\text{observed series: } y_t = \mu_t + \gamma_t + u_t, \quad u_t \sim \mathcal{N}(0, \sigma^2_u),
\]
\[
\text{latent level: } \mu_t = \mu_{t-1} + \chi_t, \quad \chi_t \sim \mathcal{N}(0, \sigma^2_{\chi}),
\]
\[
\text{latent seasonal: } \gamma_t = \sum_{j=1}^{s-1} -\gamma_{t-j} + \omega_t, \quad \omega_t \sim \mathcal{N}(0, \sigma^2_{\omega}),
\]

for \( t \in \{s, \ldots, N\} \). Choosing \( s = 24 \), we simulate \( N = 10,000 \) observations under noise variances \( \sigma^2_u = 300, \sigma^2_{\chi} = 1, \sigma^2_{\omega} = 1 \). The first 8,000 observations are used for training and the remaining are used for testing.

The data is non-stationary — we accept the Null hypothesis of the augmented Dickey-Fuller test which states that the AR model contains a unit root. The test statistic is 0.317 and the p-value is 0.9230 (the critical values are 1%: -3.431, 5%: -2.862, and 10%: -2.567). We choose a sequence length of \( p = 30 \) based on the PACF (not shown as the DGP is known).

Figure 3 compares the PACFs of various architectures on the test set. Figure 3a shows the PACF on the generated data — the positive lags at 24, 48, 72 and 96, due to the seasonality, are clearly shown. Figure 3b and 3c show the PACF on the RNN and \( \alpha \)-RNN, both stationary architectures. Neither are able to capture the seasonality. On the other hand, the \( \alpha_t \)-RNN, the GRU and the LSTM, shown in Figure 3d-f, adequately capture the seasonality. This is an important observation as typically deseasonalization techniques such as differencing or convolutional filtering are needed. For completeness, the cross-validated parameters, using five folds, are shown in Table 1.
Figure 3: The PACFs are compared for various recurrent architectures over a local level seasonality dataset with a periodicity of 24 hours.

| Architecture | Parameters | \( \lambda \) | H |
|--------------|------------|------------|---|
| RNN          | 41         | 0          | 5 |
| \( \alpha \)-RNN | 132       | 0          | 10|
| \( \alpha_t \)-RNN | 86        | 0          | 5 |
| GRU          | 1,341      | 0          | 20|
| LSTM         | 1,781      | 0          | 20|

Table 1: The cross-validated parameters of the ten-step ahead forecasting models for the lbm+seasonality data generation process.

6.2 Multi-Regime DGP

We evaluate the various times series forecasting models on a simulated univariate non-stationary dataset. Specifically we assume the existence of a discrete latent variable on which a conditionally stationary random distribution exists. Such a model is dynamically representative of the types of noisy time series generated by many applications such as electricity demand motor neuron events from electroencephalography (EEG) data [23] and channel detection in cognitive radio [25]. Indeed regimes are ubiquitous in industrial applications such as recession periods in economic data, on-peak and off-peak periods in traffic and electricity demand.

\( N = 20000 \) observations of the univariate time series are generated from a two-regime AR(30) model. The two regime AR(30) model is a linear model of the form given in Appendix D.

To test that the time series is indeed non-stationary, we accept the Null hypothesis of the augmented Dickey-Fuller test at the 99% confidence level which states that the AR model contains a unit root and is thus non-stationary. The test statistic is \(-3.17\) and the p-value is 0.021 (the critical values are 1%: -3.434, 5%: -2.863, and 10%: -2.568). We choose a sequence length of \( p = 30 \) based on the PACF (not shown as the DGP is known).
Figure 12 compares the performance of the various forecasting networks. The ten-step ahead forecasts are compared in ascending order of the number of hidden layers — plain RNNs, $\alpha$-RNNs, $\alpha_t$-RNNs, GRUs and LSTMs. The figure shows that stationary models such as the plain RNN and the $\alpha$-RNN inadequately capture the regime change, with the peaks being severely underestimated. The latter effect is symptomatic of a model which has learned the auto-regressive structure in the lower regime, to the detriment of the other. All methods produce significant forecast error at the regime transition. We further observe minor differences in the performance of the GRU versus the $\alpha_t$-RNN model suggesting that the reset gate provides marginal effect for this dataset. We also observe no additional benefit in maintaining an additional cell memory. See Appendix 5 for further discussion on the cellular memory.

| Architecture | Parameters | $\lambda_1$ | H | MSE (test)          |
|--------------|------------|-------------|---|--------------------|
| RNN          | 461        | 0.001       | 20 | $4.220 \times 10^{-3}$ (0.0649) |
| $\alpha$-RNN | 462        | 0.001       | 20 | $4.040 \times 10^{-3}$ (0.0636) |
| $\alpha_t$-RNN | 941       | 0           | 20 | $1.299 \times 10^{-3}$ (0.0360) |
| GRU          | 371        | 0           | 10 | $1.204 \times 10^{-3}$ (0.0347) |
| LSTM         | 146        | 0.001       | 5  | $8.274 \times 10^{-4}$ (0.0288) |

Table 2: The ten-step ahead forecasting models for regime switching data are compared for various architectures using time series cross-validation.

Table 2 compares the average MSEs and their standard deviations each ten-step ahead forecasting model using time series cross validation with ten folds. To assess overfitting, the MSEs over the training periods are also provided for comparison with in and out-of-distribution model performance. The stationary models are observed to exhibit the worst MSE — the plain RNN and the $\alpha$-RNN are similar in performance, the latter alleviates the vanishing gradient problem. The $\alpha_t$-RNN and the GRU compare in performance suggesting the effect of the reset gate in the latter is marginal.
Figure 4: The ten-step ahead forecasts are compared for various recurrent architectures over a regime switching dataset whose regime alternates every 100 observations.

6.3 Short-term climate forecasting

The Jena climate modeling dataset was recorded at the Weather Station operated by the Max Planck Institute for Biogeochemistry in Jena, Germany. 14 different quantities (such as air temperature, atmospheric pressure, humidity, wind direction etc) were recorded every 10 minutes, over several years. The dataset contains 420,551 observations covering the period 2009-2016 [19]. We demonstrate how the different networks forecast the temperature using all lagged observed variables. Each covariate in the training and the test set is normalized using the moments of the
Figure 5: The partial autocorrelogram (PACF) for each of the covariates (features) used in the model. Some of the covariates exhibit a monotonically decaying PACF while others exhibit oscillatory decay, with positive and negative partial autocorrelations.

We choose a sequence length of \( p = 20 \) based on the PACF and perform a ten-step ahead forecast. Figure 6 compares the performance of the various forecasting networks and shows that stationary models such as the plain RNN and the \( \alpha \)-RNN adequately capture the temperature dynamics, even when the forecasting date moves further from the training period — this is expected because the partial autocorrelation is stationary and there is hence no secular drift in the error.

We reject the Null hypothesis of the augmented Dickey-Fuller test at the 99% confidence level for each covariate in favor of the alternative hypothesis that the data is stationary (contains at least one unit root). The largest test statistic is \(-3.81841\) and the p-value is \(0.002\) (the critical values are 1%: \(-3.431\), 5%: \(-2.862\), and 10%: \(-2.567\)). However, we observe some evidence of cyclical memory in some of the covariates as seen in Figure 5.

Viewing the results of time series cross validation, using the first 23,971 observations, in Table 3, we observe minor differences in the performance of the GRU versus the \( \alpha_t \)-RNN, suggesting that the reset gate provides no benefit for this dataset. In this case, we observe no additional benefit in the LSTM.

Table 3: The ten-step ahead climate forecasts are compared for various architectures using time series cross-validation.

| Architecture | Parameters | \( \lambda_1 \) | H | MSE (test) |
|--------------|------------|----------------|---|------------|
| RNN          | 261        | 0              | 10 | \( 3.099 \times 10^{-2} \) (0.176) |
| \( \alpha \)-RNN | 107        | 0.001          | 5  | \( 4.710 \times 10^{-2} \) (0.217) |
| \( \alpha_t \)-RNN | 216        | 0              | 5  | \( 3.611 \times 10^{-2} \) (0.190) |
| GRU          | 761        | 0.001          | 10 | \( 4.190 \times 10^{-2} \) (0.205) |
| LSTM         | 406        | 0              | 5  | \( 4.754 \times 10^{-2} \) (0.218) |
(a) The forecasts for each architecture and the observed out-of-sample time series.

(b) The errors for each architecture over the same test period.

Figure 6: The ten-step ahead forecasts of temperature using the Jena weather data with MSEs shown in parentheses.

6.4 Electricity consumption

$N = 30000$ observations of hourly power system loads are provided over the period from January 1st 2008 to June 20th 2011 for the DK2 bidding zone collected in the open power systems time series data \cite{28}. The consumption time series is chosen as it exhibits both short term cyclical patterns and longer-term trends. However, while these cyclical patterns correspond to peak and off-peak consumption periods, the transitions are gradual. Without attempting to deseasonalize the data nor de-trend it, we apply the various architectures as in the previous experiment.
We reject the Null hypothesis of the augmented Dickey-Fuller test at the 99% confidence level in favor of the alternative hypothesis that the data is stationary (contains no unit roots). The test statistic is $-10.991$ and the p-value is 0 (the critical values are 1%: -3.431, 5%: -2.862, and 10%: -2.567).

The PACF in Figure 7 is observed to exhibit seasonality. We choose a sequence length of $p = 30$ and perform a ten-step ahead forecast to highlight the limitations of not including high order lags.

Figure 12 compares the performance of the various networks and shows that plain RNN performs poorly, whereas the $\alpha_t$-RNN better captures the load dynamics. From Table 6, we further observe relatively minor differences in the performance of the GRU versus the $\alpha_t$-RNN, suggesting that the reset gate provides no benefit. We also observe no additional benefit in the LSTM.

| Architecture | Parameters | $\lambda_1$ | H | MSE (test) |
|--------------|------------|-------------|---|------------|
| RNN          | 461        | 0           | 20 | 0.1062 (0.326) |
| $\alpha$-RNN| 462        | 0           | 20 | $6.647 \times 10^{-2}$ (0.258) |
| $\alpha_t$-RNN| 941       | 0           | 20 | $3.197 \times 10^{-2}$ (0.179) |
| GRU          | 1,341      | 0           | 20 | $4.746 \times 10^{-2}$ (0.218) |
| LSTM         | 1,781      | 0           | 20 | $4.265 \times 10^{-2}$ (0.207) |

Table 4: The ten-step ahead forecasting models for electricity load are compared for various architectures using time series cross-validation.

6.5 Bitcoin forecasting

One minute snapshots of USD denominated Bitcoin mid-prices are captured from Coinbase over the period from January 1st to November 10th, 2018. We demonstrate how the different networks forecast Bitcoin prices using lagged observations of prices. The predictor in the training and the test set is normalized using the moments of the training data only so as to avoid look-ahead bias or introduce a bias in the test data. We accept the Null hypothesis of the augmented
(a) The forecasts for each architecture and the observed out-of-sample time series.

(b) The errors for each architecture over the same test period.

Figure 8: The ten-step ahead forecasts are compared for various architectures over the hourly electricity consumption dataset.
Table 5: The four-step ahead Bitcoin forecasts are compared for various architectures using time series cross-validation. The half-life of the $\alpha$-RNN is found to be 1.077 minutes ($\hat{\alpha} = 0.4744$).

Dickey-Fuller test as we can not reject it at even the 90% confidence level. The data is therefore stationary (contains at least one unit root). The largest test statistic is $-2.094$ and the p-value is 0.237 (the critical values are 1%: -3.431, 5%: -2.862, and 10%: -2.567). While the partial autocovariance structure is expected to be time dependent, we observe a short memory of only four lags by estimating the PACF over the entire history (see Figure 9).

We choose a sequence length of $p = 4$ based on the PACF and perform a four-step ahead forecast. We comment in passing that there is little, if any, merit in forecasting beyond this time horizon given the largest significant lag indicated by the PACF. Figure 10 compares the performance of the various forecasting networks and shows that stationary models such as the plain RNN and the $\alpha$-RNN least capture the price dynamics — this is expected because the partial autocorrelation is non-stationary.

Viewing the results of time series cross validation, using the first 30,000 observations, in Table 5 we observe minor differences in the performance of the LSTM, GRU versus the $\alpha_t$-RNN, suggesting that the reset gate and extra cellular memory in the LSTM provides negligible benefit for this dataset. In this case, we observe very marginal additional benefit in the LSTM, yet the complexity of the latter is approximately 50x that of the $\alpha_t$-RNN.
Figure 10: The four-step ahead forecasts of temperature using the minute snapshot Bitcoin prices (USD) with MSEs shown in parentheses. Note that the prices have been standardized.

6.6 High Frequency Trading Data

Our dataset consists of $N = 1,033,468$ observations of tick-by-tick Volume Weighted Average Prices (VWAPs) of CME listed ESU6 level II data over the month of August 2016 [12, 10].

We reject the Null hypothesis of the augmented Dickey-Fuller test at the 99% confidence level in favor of the alternative hypothesis that the data is stationary (contains no unit roots). The test statistic is $-5.243$ and the p-value is $7.16 \times 10^{-6}$ (the critical values are 1%: -3.431, 5%: -2.862, and 10%: -2.567).
The PACF in Figure 11 is observed to exhibit a cut-off at approximately 23 lags. We therefore choose a sequence length of $p = 23$ and perform a ten-step ahead forecast. Note that the time-stamps of the tick data are not uniform and hence a step refers to a tick.

Figure 12 compares the performance of the various networks and shows that plain RNN performs poorly, whereas the $\alpha_t$-RNN better captures the VWAP dynamics. From Table 6, we further observe relatively minor differences in the performance of the GRU versus the $\alpha_t$-RNN, again suggesting that the reset gate and extra cellular memory in the LSTM provides no benefit. In this case, we find that the GRU is approximately 10x the complexity of the $\alpha_t$-RNN with very marginal benefit.
Figure 12: The ten-step ahead forecasts of VWAPs are compared for various architectures using the tick-by-tick dataset.

(a) The forecasts for each architecture and the observed out-of-sample time series.

(b) The errors for each architecture over the same test period.

Table 6: The ten-step ahead forecasting models for VWAPs are compared for various architectures using time series cross-validation. The half-life of the \(\alpha\)-RNN is found to be 2.398 periods (\(\hat{\alpha} = 0.251\)).
7 Conclusion

Industrial forecasting has entered an era of unprecedented growth in the size and complexity of data which require new modeling methodologies. This paper presented a general class of exponential smoothed recurrent neural networks (RNNs) which are well suited to modeling non-stationary dynamical systems arising in industrial applications such as electricity load management and financial risk and trading. In particular, we demonstrated how they characterize the non-linear partial autocorrelation structure of time series and directly capture dynamic effects such as seasonality and regime changes. Application of exponentially smoothed RNNs to electricity load forecasting, weather data and financial time series, such as minute level Bitcoin prices and CME futures tick data, highlighted the efficacy of exponential smoothing for multi-step time series forecasting. In all of these examples, we show that exponentially smoothed RNNs are well suited to forecasting, being much simpler than more complex architectures such as GRUs and LSTMs, yet retaining the most important aspects needed for forecasting non-stationary series. These methods scale to large numbers of covariates and complex data. The experimental design and architectural parameters, such as the predictive horizon and model parameters, can be determined by simple statistical tests and diagnostics, without the need for extensive parameter optimization. Moreover, unlike traditional time series methods such as ARIMA models, these methods are shown to be unconditionally stable without the need to pre-process the data.
References

[1] Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. (2016). TensorFlow: A System for Large-scale Machine Learning. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI’16, pp. 265–283.

[2] Bao, W., J. Yue, and Y. Rao (2017, 07). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLOS ONE 12(7), 1–24.

[3] Bayer, J. (2015). Learning sequence representations. Technische Universität München.

[4] Borovkova, S. and I. Tsiamas (2019). An ensemble of lstm neural networks for high-frequency stock market classification. Journal of Forecasting 38(6), 600–619.

[5] Borovykh, A., S. Bohte, and C. W. Oosterlee (2017). Conditional time series forecasting with convolutional neural networks.

[6] Box, G. and G. M. Jenkins (1976). Time Series Analysis: Forecasting and Control. Holden-Day.

[7] Chen, S. and L. Ge (2019). Exploring the attention mechanism in lstm-based hong kong stock price movement prediction. Quantitative Finance 19(9), 1507–1515.

[8] Chen, Z., J. Zhang, M. Arjovsky, and L. Bottou (2019). Symplectic recurrent neural networks.

[9] Chung, J., Ç. Gülçehre, K. Cho, and Y. Bengio (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR abs/1412.3555.

[10] Dixon, M. (2017). Sequence classification of the limit order book using recurrent neural networks. Journal of Computational Science.

[11] Dixon, M. and J. London (2021). Time series modeling with α-RNNs. Technical report.

[12] Dixon, M. F., N. G. Polson, and V. O. Sokolov (2019). Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading. Applied Stochastic Models in Business and Industry 35(3), 788–807.

[13] Gers, F. A., D. Eck, and J. Schmidhuber (2001). Applying LSTM to Time Series Predictable through Time-Window Approaches, pp. 669–676. Berlin, Heidelberg: Springer Berlin Heidelberg.

[14] Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics.

[15] Graves, A. (2013). Generating sequences with recurrent neural networks. CoRR abs/1308.0850.

[16] Hamilton, J. (1994). Time series analysis. Princeton, NJ: Princeton Univ. Press.

[17] Harvey, A. (1990). Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press.

[18] Hochreiter, S. and J. Schmidhuber (1997, November). Long short-term memory. Neural Comput. 9(8), 1735–1780.
[19] Jena (2016). Jena climate data. Collected by the Jena weather station by the Max Planck Institute for Biogeochemistry. url: https://www.bgc-jena.mpg.de/wetter.

[20] Kirchgässner, G. and J. Wolters (2007). Introduction to Modern Time Series Analysis. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

[21] Li, D. and K. Zhu (2020). Inference for asymmetric exponentially weighted moving average models. Journal of Time Series Analysis 41(1), 154–162.

[22] Lim, S. H., L. T. Giorgini, W. Moon, and J. S. Wettlaufer (2019). Predicting rare events in multiscale dynamical systems using machine learning.

[23] Lisi, G., D. Rivela, A. Takai, and J. Morimoto (2018). Markov switching model for quick detection of event related desynchronization in eeg. Frontiers in Neuroscience 12, 24.

[24] McDermott, P. L. and C. K. Wikle (2018). Deep echo state networks with uncertainty quantification for spatio-temporal forecasting.

[25] Mikaeil, A. M., B. Guo, X. Bai, and Z. Wang (2014, Nov). Hidden markov and markov switching model for primary user channel state prediction in cognitive radio. In The 2014 2nd International Conference on Systems and Informatics (ICSAI 2014), pp. 854–859.

[26] Måkinen, Y., J. Kannaiinen, M. Gabouj, and A. Iosifidis (2019). Forecasting jump arrivals in stock prices: new attention-based network architecture using limit order book data. Quantitative Finance 19(12), 2033–2050.

[27] Niu, M. Y., L. Horesh, and I. Chuang (2019). Recurrent neural networks in the eye of differential equations.

[28] OPS (2019). Open power systems data. Total load excl. transmission system losses in DK2 in MW (DK_2_load_actual_net_consumption_tso). url: https://data.open-power-system-data.org/time_series.

[29] Pascanu, R., T. Mikolov, and Y. Bengio (2012). On the difficulty of training recurrent neural networks.

[30] Schmidhuber, J. and S. Hochreiter (1997). Long Short-Term Memory. Neural Comput 9(8), 1735–1780.

[31] Sirignano, J. and R. Cont (2019). Universal features of price formation in financial markets: perspectives from deep learning. Quantitative Finance 19(9), 1449–1459.

[32] Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting 36(1), 75 – 85. M4 Competition.

[33] Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, pp. 3104–3112. Curran Associates, Inc.

[34] Zheng, J., C. Xu, Z. Zhang, and X. Li (2017, March). Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network. In 2017 51st Annual Conference on Information Sciences and Systems (CISS), pp. 1–6.
A Time Series Modeling Definitions

Definition A.1 (Time series). A time series is one observation of a stochastic process, over a specific interval: \(\{y_t\}_{t=1}^{N}\).

Definition A.2 (Autocovariance). The \(j\)th autocovariance of a time series is \(\gamma_j := \mathbb{E}[(y_t - \mu_t)(y_{t-j} - \mu_{t-j})]\) where \(\mu_t := \mathbb{E}[y_t]\).

Definition A.3 (Covariance (weak) stationarity). A time series is weak (or wide-sense) covariance stationary if it has time constant mean and autocovariances of all orders:

\[
\mu_t = \mu, \quad \forall t \\
\gamma_{jt} = \gamma_j, \quad \forall t.
\]

As we’ve seen, this implies that \(\gamma_j = \gamma_{-j}\): the autocovariances depend only on the interval between observations, but not the time of the observations.

Definition A.4 (Autocorrelation). The \(j\)th autocorrelation, \(\tau_j\) is just the \(j\)th autocovariance divided by the variance:

\[
\tau_j = \frac{\gamma_j}{\gamma_0}.
\] (23)

Definition A.5 (White noise). White noise, \(\phi_t\), is i.i.d. error which satisfies all three conditions:

1. \(\mathbb{E}[\phi_t] = 0, \forall t;\)
2. \(\mathbb{V}[\phi_t] = \sigma^2, \forall t;\) and
3. \(\phi_t\) and \(\phi_s\) are independent, \(t \neq s, \forall t, s.\)

Gaussian white noise just adds a normality assumption to the error. White noise error is often referred to as a “disturbance”, “shock” or “innovation” in the time series literature.

B Proof of Partial Autocovariance Property of \(\alpha\)-RNNs

Proof. Let’s first consider a RNN(1) process, i.e. \(\alpha = p = 1\). The lag-1 partial autocovariance is

\[
\tilde{\gamma}_1 = \mathbb{E}[y_t - \mu, y_{t-1} - \mu] = \mathbb{E}[\tilde{y}_t + \mu_t, y_{t-1} - \mu],
\] (24)

and using the RNN(1) model with, for simplicity, a single recurrence weight, \(\phi\):

\[
\tilde{y}_t = \sigma(\phi y_{t-1})
\] (25)

gives

\[
\tilde{\gamma}_1 = \mathbb{E}[\sigma(\phi y_{t-1}) + u_t - \mu, y_{t-1} - \mu] = \mathbb{E}[y_{t-1}\sigma(\phi y_{t-1})],
\] (26)

where we have assumed \(\mu = 0\) in the second part of the expression.

Continuing with the lag-2 autocovariance gives:

\[
\tilde{\gamma}_2 = \mathbb{E}[y_t - P(y_t | y_{t-1}), y_{t-2} - P(y_{t-2} | y_{t-1})],
\] (27)

and \(P(y_t | y_{t-1})\) is approximated by the RNN(1):
\[
\hat{y}_t = P(y_t | y_{t-1}) = \sigma(\phi y_{t-1}).
\] (28)

and \(P(y_{t-2} | y_{t-1})\) is approximated by the backward RNN(1):

\[
\hat{y}_{t-2} = P(y_{t-2} | y_{t-1}) = \sigma(\phi(\hat{y}_{t-1} + u_{t-1})),
\] (29)

so that we see, crucially, that \(\hat{y}_{t-2}\) depends on \(u_{t-1}\) but not on \(u_t\). Substituting the backward RNN(1) and \(u_t = y_t - \hat{y}_t\) into Equation 27 gives

\[
\hat{y}_2 = \mathbb{E}[u_t, y_{t-2} - \sigma(\phi(\hat{y}_{t-1} + u_{t-1}))],
\] (30)

and \(y_{t-2} - P(y_{t-2} | y_{t-1})\) hence depends on \(\{u_{t-1}, u_{t-2}, \ldots\}\). Thus we have that \(\hat{y}_2 = 0\).

Now suppose \(\alpha \in (0, 1)\). Repeating the above we have the backward \(\alpha\)-RNN:

\[
\hat{y}_{t-2} = P(y_{t-2} | y_{t-1}) = \sigma(\phi(\alpha(\hat{y}_{t-1} + u_{t-1}) + (1 - \alpha)\tilde{h}_t))
\] (31)

and we see that, by virtue of the dependency of \(\tilde{h}_t\) on \(y_t\) and hence \(\hat{y}_t + u_t\), that the lag-2 autocovariance is no longer zero.

Now consider the lag-2 partial autocovariance of the RNN(2) process, again with \(\alpha = 1\). Using the backward RNN(2) model:

\[
\hat{y}_{t-2} = P(y_{t-2} | y_{t-1}) = \sigma(\phi\sigma(\phi(\hat{y}_t + u_t) + y_{t-1})),
\] (32)

which depends on \(u_t\) and hence the lag-2 partial autocovariance:

\[
\hat{y}_2 = \mathbb{E}[u_t, y_{t-2} - \sigma(\phi\sigma(\phi(\hat{y}_t + u_t) + \hat{y}_{t-1} + u_{t-1}))],
\] (33)

is not zero. It follows by induction that lag-s partial autocorrelations

\[
\hat{y}_s = \mathbb{E}[u_t, y_{t-s} - P(y_{t-s} | y_{t-s+1}, \ldots, y_{t-1})] = 0, s > p,
\] (34)

since \(P(y_{t-s} | y_{t-s+1}, \ldots, y_{t-1})\) is approximated by the backward RNN(p):

\[
\hat{y}_{t-s} = P(y_{t-s} | y_{t-s+1}, \ldots, y_{t-1})
= \sigma(\phi\sigma(\phi\sigma(\ldots, \phi\sigma(\hat{y}_{t-s+p} + u_{t-s+p}) + \ldots + \hat{y}_{t-s+p-1} + u_{t-s+p-1}) + \ldots) + \hat{y}_{t-1} + u_{t-1})(36)
\]

Thus the PACF for an \(\alpha\)-RNN(p) has a cut-off at \(p\) lags when \(\alpha = 1\). With long memory, i.e. \(\alpha \in (0, 1)\), \(\tilde{\tau}_s \neq 0, s > p\) and hence the minimum memory of \(\alpha\)-RNN(p) model with \(\alpha \in (0, 1]\) is \(p\).

Such a property can be used to identify the order of the RNN model from the estimated PACF and hence determines the sequence length in the \(\alpha\)-RNN which is guaranteed to have at least the same order for \(\alpha \in (0, 1]\).

C Proof of RNN stability theorem

Proof. The proof proceeds by induction. We first consider the RNN(1) model:

\[
y_t = \Phi^{-1}(L)[u_t] = (1 - \sigma(\phi L))^{-1}[u_t],
\] (37)
where for ease of exposition we have set $W_y = 1, U_h = W_x = \phi \in \mathbb{R},$ and $b_h = b_y = 0$ without loss of generality. Expressing this as an infinite dimensional non-linear moving average model

$$ y_t = \frac{1}{1 - \sigma(\phi L)} [u_t] = \sum_{j=0}^{\infty} (\sigma(\phi L))[u_t]^j, \quad (38) $$

and the infinite sum will be stable when the $(\sigma(\cdot))^j$ terms do not grow with $j,$ i.e. $|\sigma| < 1$ for all values of $\phi$ and $y_{t-1}.$ In particular, the choice $\text{tanh}$ satisfies the requirement on $\sigma.$ For higher order models, we follow an induction argument and show first that for a RNN(2) model we obtain

$$ y_t = \frac{1}{1 - \sigma(\phi^2 L) + \phi L} [u_t] $$

$$ = \sum_{j=0}^{\infty} \sigma^j(\phi^2 L) + \phi L)[u_t], $$

which again is stable if $|\sigma| < 1$ and it follows for any model order that the stability condition holds.

It follows that lagged unit impulses of the data strictly decay with the order of the lag when $|\sigma| < 1.$ Again by induction, at lag 1, the output from the hidden layer is

$$ h_t = \sigma(\phi 1 + \phi 0) = \sigma(\phi 1). \quad (39) $$

The absolute value of each component of the hidden variable under a unit vector impulse at lag 1 is strictly less than 1:

$$ |h_t| = |\sigma(\phi 1)| < 1, \quad (40) $$

if $|\sigma(\phi)| < 1$ and each element of $\phi 1$ is finite. Additionally if $\sigma$ is strictly monotone increasing then $|h_t|$ under a lag two unit innovation is strictly less than $|h_t|$ under a lag one unit innovation

$$ |\sigma(\phi 1)| > |\sigma(\phi^2 1)|. \quad (41) $$

The choice of tanh or sigmoid activation is therefore suitable for RNNs with finite weights and input.

### D Regime Switching DGP

The simulated regime switching data is given by:

$$ y_{t+1} = \mu_t + \sum_{i=1}^{30} \phi_{t,i} y_{t-i+1} + \epsilon_t, \epsilon_t \sim N(0, \sigma_n^2) \quad (42) $$

where $\sigma_n = 1 \times 10^{-4}, y_0 = 0.38$ and for $j = 0, \ldots, 150$

$$ \phi_{t,i} = \begin{cases} \phi_0, & t = jL, \ldots, (j + 1)L, \mod(j, 2) = 0, \\ \phi_1, & t = jL, \ldots, (j + 1)L, \mod(j, 2) \neq 0, \end{cases} \quad (43) $$

with lag parameters $\phi_0 = 0.02, \phi_1 = 0.01$ and regime length $L = 100$ and

$$ \mu_t = c + \mu(-1)^t, \ t = jL, \ldots, (j + 1)L, \ j = 0, \ldots, 150. \quad (44) $$

where $\mu = 0.1$ and $c = 0.14.$