Forecasting Methods: An Overview

Thomas J. Delaney

Contents

1 Introduction

An overview of forecasting methods, from the simplest to the most up to date.

1.1 What is Forecasting?

A *time series* is list of measurements or recordings ordered by the time at which those measurements were taken, or those recordings were made. *Forecasting* is the act of predicting the future value of a quantity (usually a time series) with some measure of confidence. Every forecasting method consists of constructing a model of the series, and basing a mean prediction and prediction interval on that model.

1.1.1 Example: The Naïve Method

For example, one of the simplest forecasting methods is the *naïve* method. This consists of forecasting that the next value of the time series will be sampled from a normal distribution (aka. Gaussian distribution) with mean equal to the latest value of the time series, and standard deviation equal to the standard deviation of the one step changes in the time series. Using this model, the forecasted value for the time series is same as the latest value, and the prediction intervals are calculated using the z-distribution.

More formally, if $y_1, \dots, y_T \in \mathbb{R}$ is a time series and $d_1, \dots, d_{T-1}$ are the one step differences of that series, then according to the naïve model

$$\hat{y}_{T+1|1,\dots,T} \sim \mathcal{N}(y_T, \sigma^2) \tag{1}$$

where σ^2 is the variance of $d_1, \dots, d_{T-1}$. As for prediction intervals, since the z-statistic for 95% is 1.96, the 95% prediction interval for $\hat{y}_{T+1|1,\dots,T}$ is $\pm 1.96\sigma$.

2 Residuals, Normality& Prediction Intervals

In a forecasting context, a residual is the difference between a given time series value and the one step forecast for that value, $e_t = y_t - \hat{y}_{t|1,\dots,t-1}$. Given some forecasting model,these residuals are usually used to determine the amount of noise in the model and calculate the prediction intervals. It should be noted that since the z-distribution is always used to calculate the prediction intervals, the residuals of the model must be distributed normally. Otherwise the intervals will be inaccurate. Effectively, the noise in any model is always modelled as white noise,

$$\epsilon_t \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

where σ^2 is the variance of the residuals.

In the example given in section 1.1.1, since the model forecasts the next value in the series to be the same as the last, the residuals are the one step differences of the series. Another way of formulating the model given by equation 1 is

$$\hat{y}_{T+1|1,\dots,T} = y_T + \epsilon_t \tag{3}$$

3 Random Walks, Transformations, & Stationarity

3.1 Random Walk Forecasting

Random walk is another simple method for forecasting in which the one step change in the time series is modelled. The simplest version models the one step change as a normal distribution with zero mean. This is the same as the naïve model in section 1.1.1. The model can be expanded to include a *drift* term so that

$$\hat{y}_{T+1|1,\dots,T} - y_T \sim \mathcal{N}(c, \sigma^2) \tag{4}$$

where $c \in \mathbb{R}$ and σ^2 is the variance of the residuals. If $c > 0$ then the series will show an upward linear trend, and if $c < 0$ the series will show a downward linear trend. Another way of formulating this model is

$$\hat{y}_{T+1|1,\dots,T} = y_T + c + \epsilon_t \tag{5}$$

where ϵ_t is defined by equation 2, as usual.

3.2 Seasonally Adjusted Random Walk

A random walk model can be used to forecast time series with a seasonal pattern. If there are m seasons, that is, if the seasonality has a period of m time-steps then the forecasted value will be based on the value of the previous season, rather than the previous time step

$$\hat{y}_{T+1|1,\dots,T} - y_{T+1-m} \sim \mathcal{N}(c, \sigma^2) \quad \text{or} \quad \hat{y}_{T+1|1,\dots,T} = y_{T+1-m} + c + \epsilon_t \tag{6}$$

These random walk models are sometimes called *difference models* because the forecast is based upon the difference between previous values of the series, with some lag applied.

3.3 Transformations for Stationarity

A stationary time series is one whose properties do not depend on the time at which the series is observed, or more precisely if $\{y_t\}$ is a stationary time series, then for all s, the distribution of $(y_t, \dots, y_{t+s})$ does not depend on t. A white noise time series (equation 2) is an example of a stationary series. The aim of most simple forecasting models is to capture all the variance of a time series, except for some additional or multiplicative white noise. This means that the model will often consist of applying transformations to the time series until the series is stationary.

For example, the naïve model captures the amplitude or *level* of the time series by taking the one step difference, leaving only white noise thereafter. A seasonal random walk model with a non-zero drift parameter will capture the amplitude of the time series at each season, and the overall trend of the time series, leaving only white noise thereafter. Some non-linear trends can be captured using a Box-Cox transformation.

$$y_{t,(\lambda)} = \begin{cases} \frac{y_t^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(y_t) & \text{if } \lambda = 0 \end{cases} \tag{7}$$

If multiplicative noise is observed in the time series, it's common to apply a log transformation in order to the make the noise additive, or a Box-Cox transformation with $\lambda = 0$. In the case where the time series may take values less than or equal to zero, the Yeo-Johnson transforamtion is used to capture non-linearity,

$$y_{t,(\lambda)} = \begin{cases} \frac{(y_t+1)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, y_t \geq 0 \\ \ln(y_t + 1) & \text{if } \lambda = 0, y_t \geq 0 \\ -\frac{[(-y_t+1)^{2-\lambda} - 1]}{2-\lambda} & \text{if } \lambda \neq 2, y_t < 0 \\ -\ln(-y_t + 1) & \text{if } \lambda = 2, y_t < 0 \end{cases} \tag{8}$$

where $0 \leq \lambda \leq 2$.

4 ARIMA

4.1 Auto-regression models

Auto-regression models are exactly what they sound like. They use a linear combination of some of the previous elements of the time series to forecast the upcoming values. The number of values used is determined by the *degree*, p. A degree-p auto-regressive model can be written as

$$\hat{y}_{T|1,\dots,T-1} = c + \phi_1 y_{T-1} + \phi_2 y_{T-2} + \cdots + \phi_p y_{T-p} + \epsilon_t \tag{9}$$

where ϵ_t is the usual white noise. A model of this kind of often denoted as $AR(p)$.

Notice that the c term can capture a linear trend in the series. An $AR(1)$ model with $\phi_1 = 1$ is equivalent to a random walk with a trend.

In order for the modelled series to be stationary some restrictions must be applied to the parameter values. Specifically, for an $AR(p)$ model the roots of the polynomial

$$z^p - \sum_{i=1}^{p} \phi_i z^{p-i} \tag{10}$$

must lie within the complex unit circle, $|z_i| < 1$. Practically, this amounts to having $|\phi_1| < 1$ for an $AR(1)$ model, and $|\phi_2| < 1$, and $\phi_1 + \phi_2 < 1$ for $AR(2)$.

The most suitable order for an $AR(p)$ model can be determined by looking at the ACF and $PACF$ of the time series. The lag beyond which the $PACF$ appears to be zero is the most suitable order.

4.2 Moving-average models

Moving average model is a poor name for this model, but it is the conventional name. A better name would be *residual-regression model* because the forecast is a linear combination of past residuals. A moving average model or order q can be written as

$$\hat{y}_{T|1,\dots,T-1} = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{11}$$

where ϵ_t is the usual white noise. This type of model is usually denoted $MA(q)$.

If a moving average model can be expressed as an infinite degree AR model, the moving average model is called *invertible*.

4.3 ARIMA models

Auto-regression, difference, and moving average models can be combined. A combination of an $AR(p)$, d-step difference, and $MA(q)$ model is denoted $ARIMA(p, d, q)$. $ARIMA$ stands for *Auto-regressive integrated moving average*. To construct an $ARIMA(p, d, q)$ model, first $\{y_t\}$ is transformed to $\{y'_t\}$ by taking the d-step difference, then

$$\hat{y}'_{T|1,\dots,T-1} = c + \phi_1 y'_{T-1} + \cdots + \phi_p y'_{T-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{12}$$

The degree of the AR and the order of the MA that should be used can be determined from a combination of the ACF and the $PACF$. The number of differences to use is generally two or fewer. This can be determined by examining the long term trend of the data.

4.3.1 Seasonal ARIMA models

In order to capture seasonal behaviour in the time series, a *seasonal ARIMA* model can be used. This is an additional $ARIMA$ model where all the lags are the length of one season. The length of a season will have to be found by inspection. The additional seasonal $ARIMA$ terms are just multiplied by the standard $ARIMA$ terms.

5 Error, Trend, Smoothing models (ETS models)

5.1 Simple Exponential Smoothing

If a time series has no clear trend, seasonality, or cycle but appears to vary around some fixed amplitude, simple exponential smoothing might be a suitable forecasting method. In simple exponential smoothing, the forecast is an weighted sum of the series's previous values where the weights decrease exponentially going backwards in time.

$$\hat{y}_{T+1|T,\dots,1} = \alpha y_T + \alpha(1-\alpha) y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots \tag{13}$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter. If α is close to 0, a large weight will be given to values in the distant past. If α is close to 1, more recent values will get a larger weight, which usually makes sense. In the extreme case where $\alpha = 1$ the forecast will be same as the mean forecast of the naïve method.

Basically just decompose the time series into level, trend, and seasonal components. The seasonal component can be multiplicative or additive. Simple exponential smooting is usually written in one of two ways. The weighted average form is

$$\hat{y}_{T+1|T,\dots,1} = \alpha y_T + (1-\alpha)\, \hat{y}_{T|T-1,\dots,1} \tag{14}$$

Note that the second term on the right hand side is a forecasted value, and therefore accounts for all the values of $\{y_t\}$ where $t < T$.

The component form is

$$\hat{y}_{T+1|T,\dots,1} = \ell_T \tag{15}$$

$$\ell_T = \alpha y_T + (1-\alpha)\ell_{T-1} \tag{16}$$

Equation 15 is called the *forecast equation*, and equation 16 is called the *smoothing equation* or the *level equation*. The practicality of the component form will be clearer when trend and seasonal components are introduced.

When implementing exponential smoothing, the α and ℓ_0 are parameters that must be fitted. The usual way of fitting the parameters is by minimising the sum of the squared errors between the timer series and the fitted values,

$$SSE = \sum_{t=1}^{T} \left(y_t - \hat{y}_{t|t-1}\right)^2 \tag{17}$$

This is a non-linear optimisation problem for which there is no analytical solution, so an optimisation algorithm must be used. Usually this will be built into whatever programming language is used for implementation.

5.2 Holt's Linear Trend Method

Simple exponential smooting can be extended to time series with a trend by including another smoothing equation.

$$\hat{y}_{T+h|T,\dots,1} = \ell_T + hb_T \tag{18}$$
$$\ell_T = \alpha y_T + (1-\alpha)\left(\ell_{T-1} + b_{T-1}\right) \tag{19}$$
$$b_T = \beta^*\left(\ell_t - \ell_{t-1}\right) + (1-\beta^*)b_{T-1} \tag{20}$$

where $0 \leq \beta^* \leq 1$. Equations 21 and 22 are the forecast and level equations as before. Equation 23 is called the *trend equation*. The trend for a given timestep is a weighted sum of the change in the level beteen this time-step and the last time-step, and the previous trend. The level equation is influenced by the trend of the previous step. The forecast equation has a trending component.

Holt's linear method will give a forecast that is linear in the forecast horizon. A slightly better approach is to used the damped version.

$$\hat{y}_{T+h|T,\dots,1} = \ell_T + \left(\phi + \phi^2 + \cdots + \phi^h\right) b_T \tag{21}$$
$$\ell_T = \alpha y_T + (1-\alpha)\left(\ell_{T-1} + \phi b_{T-1}\right) \tag{22}$$
$$b_T = \beta^*\left(\ell_t - \ell_{t-1}\right) + (1-\beta^*)\phi b_{T-1} \tag{23}$$

where $0 < \phi < 1$ is the damping coefficient. This method gives a forecast which is linear initially but approaches

$$\ell_T + \frac{\phi}{1-\phi} b_T \quad \text{as } h \to \infty \tag{24}$$

5.3 Holt-Winter's Seasonal Method

Introducing another smoothing equation can account for seasonal variations. The seasonal component can be additive or multiplicative. The additive version is,

$$\hat{y}_{T+h|T,\dots,1} = \ell_T + hb_T + s_{T+h-m(k+1)} \tag{25}$$
$$\ell_T = \alpha(y_T - s_{T-m}) + (1-\alpha)\left(\ell_{T-1} + b_{T-1}\right) \tag{26}$$
$$b_T = \beta^*\left(\ell_t - \ell_{t-1}\right) + (1-\beta^*)b_{T-1} \tag{27}$$
$$s_T = \gamma\left(y_T - \ell_{T-1} - b_{T-1}\right) + (1-\gamma)\, s_{t-m} \tag{28}$$

where k is the integer part of $(h-1)/m$, i.e. the number of seasons in the forecast horizon, and $0 \leq \gamma \leq 1-\alpha$. Equation 32 is called the seasonal equation. This equation shows that the seasonal component is the weighted average of the seasonal index from this year $(y_t - l_{t-1} - b_{t-1})$ and last year s_{t-m}. The quantity $y_t - s_t$ is the *deseasonalised series*.

The multiplicative version is very similar

$$\hat{y}_{T+h|T,\dots,1} = (\ell_T + hb_T)s_{T+h-m(k+1)} \tag{29}$$

$$\ell_T = \alpha \frac{y_T}{s_{T-m}} + (1-\alpha)\left(\ell_{T-1} + b_{T-1}\right) \tag{30}$$

$$b_T = \beta^* \left(\ell_t - \ell_{t-1}\right) + (1-\beta^*) b_{T-1} \tag{31}$$

$$s_T = \gamma \frac{y_T}{\ell_{T-1} + b_{T-1}} + (1-\gamma)\, s_{t-m} \tag{32}$$

with the same resitrictions on the parameters. In this case, the deseasonalised series is y_t/s_t.

It's also possible to dampen the Holt-Winter's seasonal method in the same way as Holt's Linear method by introducing a damping parameter ϕ.

5.4 State Space Models

These expoential smoothing models are lacking an estimate for their error. When an error estimation is added to make these models complete, they are usually called *state space* models.

Equations 15 and 16 can be re-written as

$$y_t = \ell_{t-1} + \epsilon_t \tag{33}$$

$$\ell_t = \ell_{t-1} + \alpha\epsilon_t \tag{34}$$

where $\epsilon_t = y_t - \ell_{t-1}$, the tth residual. Equation 33 is called the measurement equation and equation 34 is called the state equation.

Errors can also be multiplicative. If

$$\epsilon_t = \frac{y_t - \hat{y}_{t|t-1,\dots,1}}{\hat{y}_{t|t-1,\dots,1}} \tag{35}$$

then the measurement and state equations become

$$y_t = \ell_{t-1}(1+\epsilon_t) \tag{36}$$

$$\ell_t = \ell_{t-1}(1+\alpha\epsilon_t) \tag{37}$$

In both cases $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

As an example Holt's linear method with additive errors is defined by

$$y_t = \ell_{t-1} + b_{t-1} + \epsilon_t \tag{38}$$

$$\ell_t = \ell_{t-1} + b_{t-1} + \alpha\epsilon_t \tag{39}$$

$$b_t = b_{t-1} + \beta\epsilon_t \tag{40}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and $\beta = \alpha\beta^*$.

5.5 ETS($\cdot, \cdot, \cdot$)

State space models are often denoted by the method error, trend, and seasonality used. The type of error can be additive, A, or multiplicative M. The type of trend can be none N, additive A, and additive but dampened A_d. The type of seasonality can be none, additive, or multiplicative. There are 18 different types of state space model. They are all summarised in figure 1.

ADDITIVE ERROR MODELS

Trend	Seasonal		
	N	A	M
N	$y_t = \ell_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$	$y_t = \ell_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/\ell_{t-1}$
A	$y_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + b_{t-1})$
A_d	$y_t = \ell_{t-1} + \phi b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + \phi b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = \phi b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + \phi b_{t-1})$

MULTIPLICATIVE ERROR MODELS

Trend	Seasonal		
	N	A	M
N	$y_t = \ell_{t-1}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$	$y_t = (\ell_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \alpha(\ell_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + s_{t-m})\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A	$y_t = (\ell_{t-1} + b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A_d	$y_t = (\ell_{t-1} + \phi b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$

Figure 1: All possible ETS models. From [1].

6 Modelling the Volatility: ARCH, GARCH

Until this section, all of the models mentioned in this document have noise with a constant variance. The models in this section are used for modelling the variance in the noise. ARCH stands for auto-regressive conditional heteroskedasticity. The 'skedasticity' part of the heteroskedasticity comes from the Greek *skedastikos*, meaning 'able to disperse'. A *homoskedastic* variable is dispersed by the same amount all the time, i.e. with constant variance, whereas a *heteroskedastic* variable has a variance that changes over time.

An ARCH model models the variance of the noise as an auto-regressive variable. If ϵ_t is the noise of the model, say

$$\epsilon_t = \sigma_t z_t \tag{41}$$

where $z_t \sim \mathcal{N}(0, 1)$ is white noise, and

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 \tag{42}$$

is an $AR(q)$ model for the variance of the noise. When an $AR(q)$ model is used to model the volatility of a time series, the volatility at any given time step will be correlated with the previous steps. If the volatility is modelled using white noise, the volatility will be uncorrelated.

If an $ARMA(q, p)$ is used to model the volatility, this is called a generalised auto-regressive conditional heteroskedasticity model, or GARCH model. In this case

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 \tag{43}$$

where $\omega = \alpha_0 + \beta_0$. Introducing the moving average model part allows the model to quickly respond to sudden changes in the volatility. The auto-regressive part then propagates these changes forward.

$ARCH(q)$ and $GARCH(q, p)$ models are popular in financial forecasting since the volatility is rarely white noise, and forecasting volatility accurately can be profitable. The $GARCH(1, 1)$ model is most popular in this context.

7 Ensemble Methods

Apply several simpler models and get many forecasts. Then aggregate these forecasts and their prediction intervals in one way or another. Seems like this always improves the overall performance.

8 Meta-learning

Meta-learning in forecasting refers to the process of learning which of a collection of forecasting methods is most suitable to use in a given situation, or learning how to combine a collection of methods into an ensemble approach. When building a meta-learning framework, there are three distinct steps that are usually necessary.

8.1 Building the Meta-learning Model

8.1.1 Augmenting the Dataset

Adding to the dataset with simulated time series. In order to take the machine learning approaches necessary in meta-learning, a large and balanced time series dataset is required. Therefore it can be necessary to augment whole the dataset, or augment specific

subsets of the dataset in order have an equal number of examples of each type of time series. When simulating time series, care should be taken to create series which have similar characteristics to the observed time series. Time series can be simulated using pre-defined functions in `R` such as `auto.arima` or `ets`.

Additional time series can also be created by bootstrapping, that is sampling chunks from existing time series. When deciding what size the sampled chunks should be, the period of the data's cycles and seasonality must be taken into account. If the bootstrapped data will be used to train an RNN that uses LSTMs or GRUs, any long term dependencies must be taken into account.

8.1.2 Feature Extraction

Feature extraction. In order to train the meta-learning model, features must be extracted from each of the time series and included in the training set along with the time series itself. Examples of features used by meta-learning models are: first autocorrelation coefficient, first autocorrelation coefficient of the first differenced series, first autocorrelation of the twice-differenced series, spectral entropy, the α and β parameter of an $ets(A, A, N)$ model, *lumpiness*, seasonal period, trend strength from an STL decomposition, series length, etc., etc. It is normal to extract dozens of different features from each time series. The time series combined with the features extracted from that series make up one training example.

8.1.3 Training

The first step to training the model is to apply each of the forecasting methods to each time series. Then to measure the accuracy of each forecast. Then to choose a loss function for the meta-learning model, the minimisation of which will minimise the forecasting error of the meta-learning model. Then to use some optimisation algorithm to find the meta-learning parameters that minimise the loss function. In the case where the model combines the collection of forecasting methods, another simpler model will be required to calculate prediction intervals, and that model must be trained in a similar way.

8.2 Using the Meta-learning Model

Building the meta-learning model is usually done offline before the model is used to forecast new time series. This means that as long as the time taken to train the model is reasonable (less than two days for example) this time can be disregarded, and forecasts can be obtained very quickly.

In order to obtain a forecast from the meta-learning model, the features mentioned in section 8.1.2 are extracted. Then the features and the time series itself are fed in to the model, and the model will return a forecast and prediction intervals. It's **that** easy!

8.3 Examples

8.3.1 Feature-based Forecast-Model Selection (FFORMS)[2]

Talagala et al (2018) trained a random forest to classify the best forecasting method for a given time series out of a collection of ARIMA and ETS models. They used the datasets of the M1 and M3 competitions, and augmented these datasets using ARIMA and ETS models to simulate new time series. They extracted thirty-three different features from each time series. They trained a different classifier for yearly, quarterly, and monthly data from the M1 and M3 competitions. The FFORMS model consistently ranks in the top most accurate forecasting methods for forecasting both the M1 and the M3 series and most often ranks as the most accurate method.

A schematic diagram of the FFORMS framework is shown in figure 2. This diagram is general enough to give a good understanding of a generalised meta-learning framework.

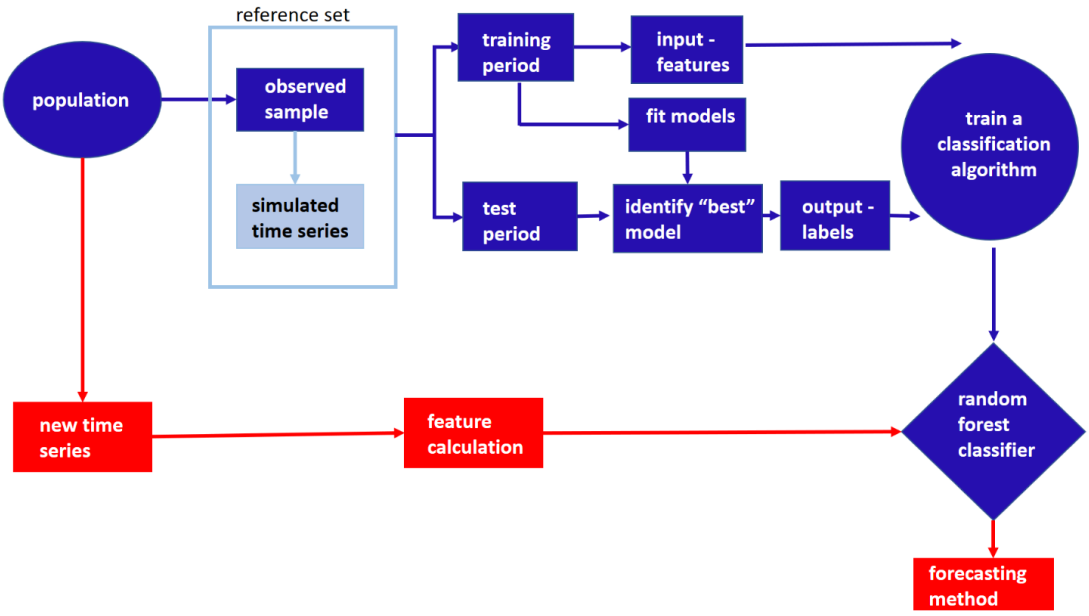


Figure 2: Schematic diagram of the FFORMS framework. The offline phase is shown in red, the online/training phase is shown in blue.

8.3.2 Meta-weighted Ensemble[3]

Montero-Manso et al (2018) trained a gradient boosted tree model to learn the weights for a weighted ensemble of simple forecasting methods, and entered their model in the M4 forecasting competition. They used the M4 competition dataset. No augmentation was necessary. While training the model they performed a search over the hyper-parameter space of the gradient boosted tree model, using cross-validation to find the best hyper-parameters. They extracted forty-two features from each of the time series in training.

To use the model for any new time series, the features are extracted and the forecasts of the simple methods are calculated. The model then takes the time series and the features and outputs a vector of real numbers the same length as the number of forecasting methods. These numbers are transformed into probabilities using a softmax function.

The *softmax* function is a squashing function which can take an arbitrary vector of real numbers $\mathbf{x} \in \mathbb{R}^n$ and transform it into a vector of values between 0 and 1 that sum to 1. Formally,

$$\sigma : \mathbb{R}^n \to \left\{ \sigma(\mathbf{x}) \in \mathbb{R}^n : \sigma(\mathbf{x})_j > 0 \forall j, \sum_{j=1}^{n} \sigma(\mathbf{x})_j = 1 \right\} \tag{44}$$

$$\sigma(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_{j=1}^{n} e^{\mathbf{x}_j}} \tag{45}$$

The softmax function is used in multi-class classification where a linear regression model is passed through a softmax function to give categorical probabilities. In this case there is one probability for each of the forecasting methods. These probabilities are used to combine the forecasts linearly in order to produce a single forecast.

In order to calculate prediction intervals, the model calculates the prediction intervals radius of a subset of the forecasting methods. A subset is taken for computational cost reasons. These radii are combined linearly using a weight matrix which is learned in training to give an upper and lower value for the interval for each time step in the forecasting horizon.

9 Time Series Features

9.1 Spectral Entropy

9.1.1 Shannon's Entropy

Shannon's Entropy aka. Information Entropy is a measure of the unpredictability of a random variable. If X is a random variable with possbile values $x_1, \ldots, x_N$ and associated probabilities $P(X = x_1), \ldots, P(X = x_N)$, the entropy of the random variable X is

$$H(X) = -\sum_{n=1}^{N} P(X = x_n) \log P(X = x_n) \tag{46}$$

$$= \sum_{n=1}^{N} P(X = x_n) \log \frac{1}{P(X = x_n)} \tag{47}$$

The entropy of a continuous random variable is defined similary, but with integrals instead of summations.

If the variable is uniformly distributed across its values, then its entropy is maximised. If X only ever takes one of its possible values, then $H(X) = 0$. Usually the $\log$ in equation 46 is taken to the base 2, and the entropy is therefore measured in *bits*.

9.1.2 Frequency Representation of Time Series

Any time series can be expressed as a function of freqencies instead of as a function of time. The Fourier transformation transforms a time series into its frequency representation

$$x(f) = \int_{-\infty}^{\infty} e^{-2\pi i f t} x(t) dt \tag{48}$$

The *power spectrum* of the time series is $S(f) = |x(f)|^2$. The normalised power spectrum is

$$P(f) = \frac{S(f)}{\sum_{f'} S(f')} \tag{49}$$

The normalised power spectrum of a time series takes the form of a probability distribution and therefore defines a random variable over the frequencies f'. The information entropy of this random variable is the *spectral entropy* of the time series.

9.2 Stability & Lumpiness

If a time series of length L is divided up into non-overlapping windows of τ time-steps, and a variance is measured from the data within each window, then the *stability* is the mean of the variances, and the *lumpiness* is the variance of the variances. There are `R` functions for both of these quantities.

10 Recurrent Neural Networks

Recurrent neural networks (RNNs) are linear combinations of non-linear *activation* functions applied to inputs combined in a way that is specifically designed for performing calculations on sequences, including time series.

In figure 3 a schematic diagram of a simple recurrent neural network is shown. The xs are the inputs, for example a given time series. The as are the activation values. The ys are the outputs. In this example network, the activations and outputs are defined by

$$a^{\langle t \rangle} = f(W_{aa}a^{\langle t-1 \rangle} + W_{ax}x^{\langle t \rangle} + b_a) \tag{50}$$

$$= f(W_a[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_a) \tag{51}$$

$$y^{\langle t \rangle} = g(W_{ya}a^{\langle t \rangle} + b_y) \tag{52}$$

where f and g are some non-linear *activation functions*, such as $\tanh$, or the sigmoid function. There are a lot of possible variations to the network architecture. The output for step t could be used as part of the input to step $t + 1$ for example, or there could be connections that skip over one or more time steps, or outputs may not be produced until after all the inputs have been processed. The activation functions need not be just one function, there could be several functions connected to make up a smaller network which gives the activation function.

For choosing activation functions there are a lot of potential functions from which to choose. There are some characteristics that would be beneficial for an activation function.

Non-linearity It has been proven that a two layer neural network with non-linear activation functions can approximate any function. Also, to capture non-linear dynamics requires non-linear functions.

Continuously Differentiable This makes gradient based optimisation easier.

Monotonicity If the activation function is monotonic, the error surface for a one-layer neural network is guaranteed to be convex. Meaning that there is only one optimum point, no local optima where an optimisation algorithm can get stuck.

Smooth (Infinitely Differentiable) functions with a Monotonic derivative These have been shown to generalised better in some cases (apparently).

Approximates the identity function near the origin If the activity function looks like the identity function $f(x) = x$ near the origin, then the network will learn efficiently when the paramters are initialised to small values. So this gives a good starting point for optimisation. If $f(0) = 0$, $f'(0) = 1$, and f' is continuous around 0, then it's fair to say that f approximates the identity around the origin.

The range or codomain of the function will have an effect on the network also. If the range of the function is restricted to some interval then gradient descent optimisation methods tend to be stable as the value of the gradient is very small for most values. On the other hand, the training process will suffer from the vanishing gradient problem causing learning to slow down as training goes on. $\tanh$ and the sigmoid function are examples of activation functions with ranges restricted to an interval. If the range of the function is unrestricted, training tends to be more efficient as the gradient of the activation function doesn't disappear, but a smaller learning rate may be necessary to find the optimum. The rectified linear unit and any other activation fuctions like that have unrestricted ranges. [1]

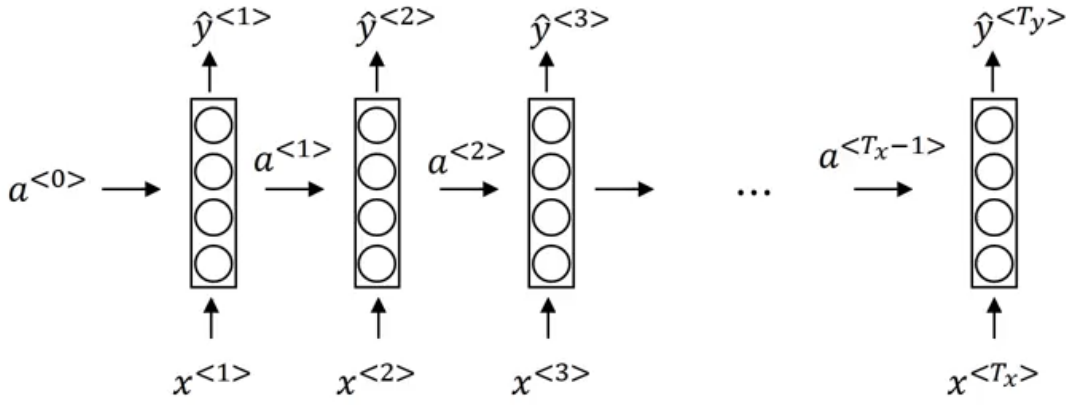


Figure 3: Schematic diagram of a simple recurrent neural network. The xs are inputs, the as are the activations, and the ys are outputs.

The backpropagation algorithm is used to train RNNs, just like any neural network. If the gradient of the loss function is propagated back through all of the time steps in the input sequence, the gradient may become exponentially small, or exponentially large. This is called the vanishing gradient problem, or exploding gradient problem, respectively. A vanishing gradient will make it difficult for information or errors (and error correction) to propagate along the RNN and will inhibit learning over time. An exploding gradient will make learning impossible as the gradient becomes too large to compute.

[1] A fairly comprehensive list of candidate activation functions can be found here: `https://en.wikipedia.org/wiki/Activation_function`

The vanishing gradient problem is more common in RNNs, and the exploding gradient problem is easier to deal with. Therefore, many of the innovations in RNNs are attempts to solve the vanishing gradient problem. The most well known and widely used of these is the *long-short term memory unit*, or LSTM (see section 10.1.1). There have been some innovations on the LSTM idea. The Gated Recurrent Unit (GRU) and the Just Another Net (JANET) will be covered here.

The LSTM, GRU, and JANET are examples of 'cells' or 'neurons' that can be stacked and connected in order to build the RNN. There have also been innovations in the way the RNNs are connected, i.e. the *architecture* of the network.

10.1 Cell Innovations: LSTM, GRU, JANET

10.1.1 Long-Short Term Memory Unit (LSTM)

Each of the neurons in the RNN can take the form of an LTSM. So the LSTM will take an input $x^{\langle t \rangle}$ and an activation $a^{\langle t-1 \rangle}$ from the previous time step, and will output an activation. The activation could then be used to create an output for the RNN. The LSTM is defined by

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c) \tag{53}$$

$$\Gamma_u = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u) \tag{54}$$

$$\Gamma_f = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f) \tag{55}$$

$$\Gamma_o = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o) \tag{56}$$

$$c^{\langle t \rangle} = \Gamma_u \odot \tilde{c}^{\langle t \rangle} + \Gamma_f \odot c^{\langle t-1 \rangle} \tag{57}$$

$$a^{\langle t \rangle} = \Gamma_o \odot \tanh(c^{\langle t \rangle}) \tag{58}$$

where $\odot$ represents element-wise multiplication and σ represents the sigmoid function. The $c^{\langle t \rangle}$ quantity can be interpreted as an internal value for the LSTM. The LSTM is sometimes referred to as a 'cell', so the letter 'c' is used. The $\tilde{c}^{\langle t \rangle}$ quantity can be interpreted as a candidate for the updated internal value which is calculated using the current input $x^{\langle t \rangle}$ and the activation of the previous time-step $a^{\langle t-1 \rangle}$ in equation 53. Equations 54, 55, and 56 are the *update*, *forget*, and *output* gates respectively. The first two of these gates together with equation 57 control the update to the internal value of the cell. The updated internal value $c^{\langle t \rangle}$ will be a combination of the candidate update and the previous internal value $c^{\langle t-1 \rangle}$. The activation of the unit $a^{\langle t \rangle}$ is a non-linear function of the updated internal value scaled by the output gate. A schematic diagram of an LSTM can be seen in figure 4.

The whole network of the LSTM and in particular equation 57 allows for errors, and therefore gradients, to propagate back through the network without diminishing in size. Equation 57 defines a state within the cell that may not be affected by incoming inputs and activations depending on the update and forget gates. Since the update and forget gates are defined using the sigmoid function, these gates must be between 0 and 1. Furthermore, the value of these gates is likely to be pushed towards 0 or 1 by the back propagation process. So, these gates can be intuitively interpreted as a vector of binary values of the same length as the input. Therefore the parameters W_u and W_f should allow the RNN to

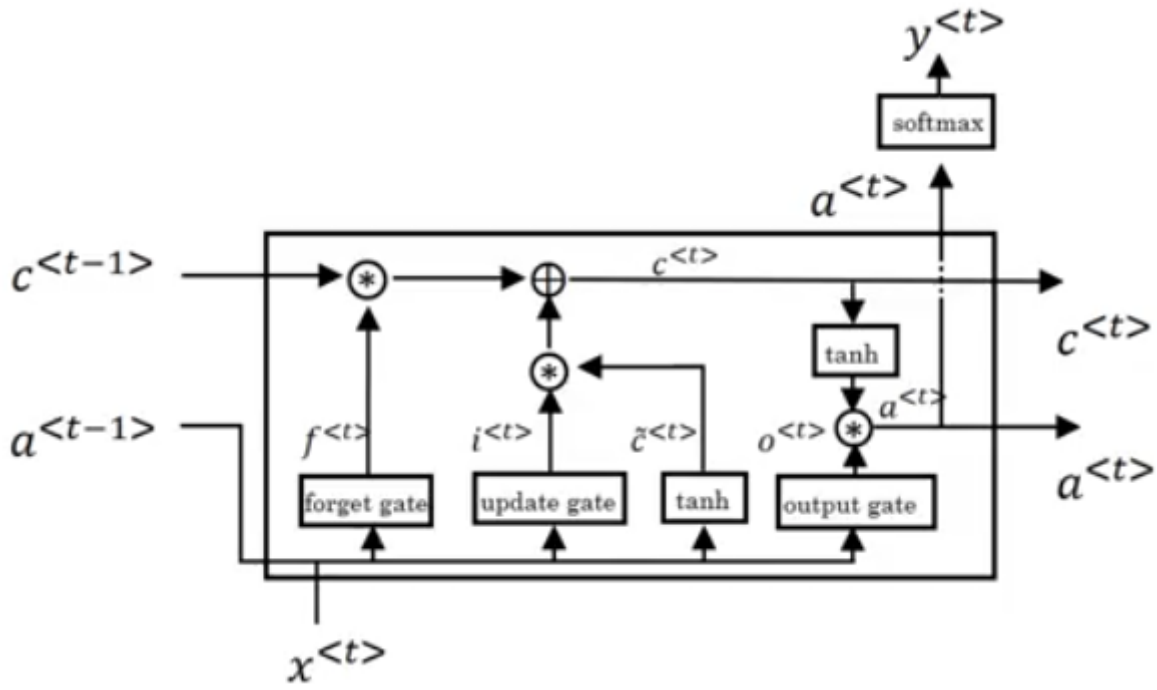


Figure 4: Schematic diagram of an LSTM.

learn when the value of a certain input or activation will have an effect on an output much later on in the input time series. This is the 'memory' referred to in the title of the LSTM.

10.1.2 Gated Recurrent Unit (GRU)[4, 5]

The GRU can be seen as a simplification of the LSTM because it avoids the vanishing gradient function in the same way as the LSTM, but using fewer equations and fewer parameters. Similar to the LSTM, the GRU uses an internal value or 'state' which might be updated slightly or updated a lot at every time step. But the GRU uses this state as its activation, so there is no need for an output gate. The GRU also disregards the forget gate, and uses a single update gate to decide by how much the internal state will be updated. The only addition to the LSTM in the GRU is a *relevance gate*, which is designed to capture the relevance of the previous state (combined with the input) to the candidate state.

$$\Gamma_r = \sigma(W_r[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_r) \tag{59}$$
$$\tilde{c}^{\langle t\rangle} = \tanh(W_c[\Gamma_r \odot c^{\langle t\rangle}, x^{\langle t\rangle}] + b_c) \tag{60}$$
$$\Gamma_u = \sigma(W_u[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u) \tag{61}$$
$$c^{\langle t\rangle} = \Gamma_u \odot \tilde{c}^{\langle t\rangle} + (1 - \Gamma_u) \odot c^{\langle t-1\rangle} \tag{62}$$

Equation 59 is the relevance gate, mentioned above. Equation 60 defines the candidate state. Equation 61 defines the update gate, similar to the LSTM. Equation 62 defines the the updated internal state. A schematic diagram of a GRU is shown in figure 5.

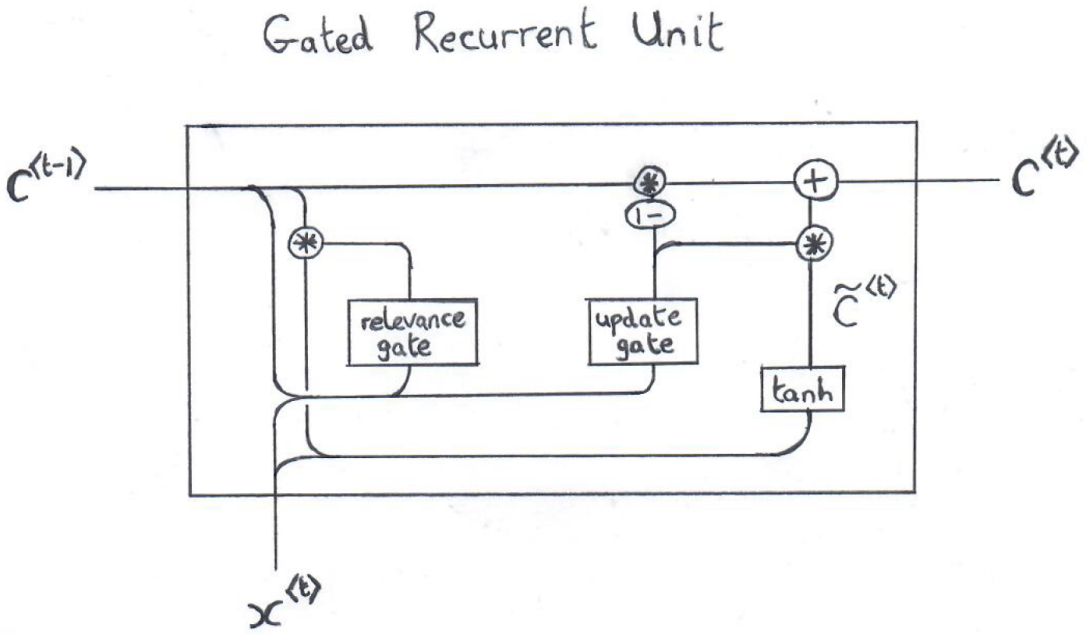


Figure 5: Schematic diagram of a GRU.

GRUs have been found to be comparable to LSTMs in certain circumstances and to outperform LSTMs on smaller datasets [5].

10.1.3 Just Another Net (JANET)[6]

The JANET is another simplification of the standard LSTM. Van der Westhuizen et al (2018) removed the update and output gates from the LSTM, leaving only the forget gate. They also subtracted a constant from the argument for the forget gate when acting on the candidate state. The idea being that this would make it easier for information to accumulate in the cell's memory. The JANET is defined by

$$s^{\langle t \rangle} = W_f[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f \tag{63}$$

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c) \tag{64}$$

$$c^{\langle t \rangle} = \sigma(s^{\langle t \rangle}) \odot c^{\langle t-1 \rangle} + (1 - \sigma(s^{\langle t \rangle} - \beta)) \odot \tilde{c}^{\langle t \rangle} \tag{65}$$

Equation 63 defines the *input control component*. This would usually be the argument of the sigmoid function in the forget gate. But the forget gate takes a slightly different form here, so it is easier to define the cell this way. Equation 64 is the usual candidate definition. Equation 65 defines the updated internal state. The first term of this equation is standard and controls how much the remembered state contributes to the updated state. The second term is standard except for the subtraction of the β. This subtraction reduces the contribution of the candidate state to the updated state, which helps the JANET's 'memory' accumulate. In [6], the authors found that setting $\beta = 1$ gave the best results on the tests performed. Similar to the GRU, the JANET uses the updated state $c^{\langle t \rangle}$ as its activation. A schematic diagram of a JANET is shown in figure 6.

The process of *chrono initialization* detailed in [7] was used to improve the JANET's performance. This amounts to initialising the bias of the forget gate b_f to a *characteristic*

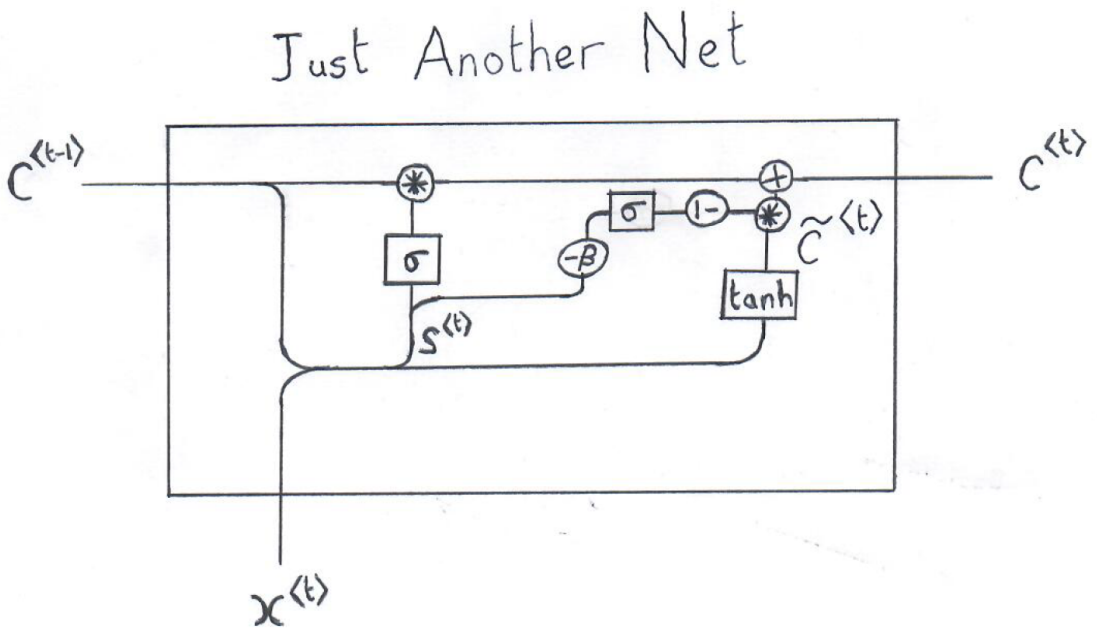


Figure 6: Schematic diagram of a JANET.

forgetting time, which is defined by the longest time dependency that the network will have to capture. This dependency is defined by the user when setting up the network.

Van der Westhuizen et al tested the JANET using some standard tests for RNNs such as testing the performance on the MNIST dataset, and the permuted MNIST dataset. They found that the JANET outperformed the standard LSTM on all the tests performed. A JANET was also used as part of the CFM competition to predict future volatility based on past volatility. The user reported better performance than a regualr LSTM, while using fewer parameters [8].

N.B.: The JANET was outperformed by something called the *tensorised LSTM (tLSTM)* [9]. May be worth looking into.

10.2 Architectural Innovations

10.2.1 Dilated Recurrent Neural Networks [10]

A skip connection in a recurrent neural network is an extra connection between two time steps a number of time steps apart. The normal connections in the RNN are still there. A dilated recurrent skip connection is a connection between steps in a recurrent neural network that skips one or more of the next steps without the normal one-step connections. A schematic diagram of a dilated skip connection can be seen in figure 7 as compared to a standard skip connection. This has the effect of reducing the number of layers through which an error must be back propogated, which addresses the vanishing gradient problem, reduces the number of parameters in the network, and forces the network to learn some historical dependencies. The dilated skip connections also allow calculations to be performed in parallel, thereby exploiting the parallelising power of a GPU.

If the activation of a cell in layer l and time t of an RNN with standard skip connections

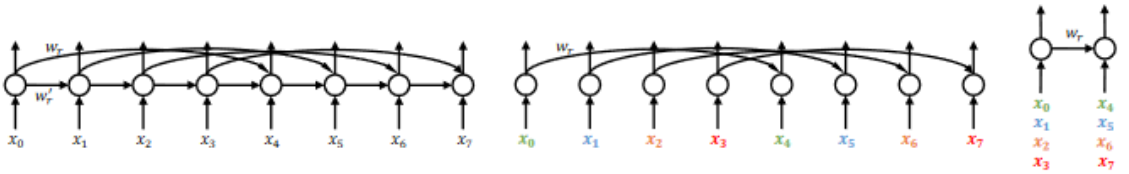


Figure 7: (left) A one layer recurrent neural network with a skip connections. (middle) A one layer recurrent neural network with dilated recurrent skip connections. (right) A structure equivalent to the middle network. The input sequence length is reduced by four, and each of the four sequences can be processed in parallel.

is denoted by $c^{\langle t \rangle (l)}$, then

$$a^{\langle t \rangle (l)} = f\left(x^{\langle t \rangle (l)}, a^{\langle t-1 \rangle (l)}, a^{\langle t-s^{(l)} \rangle (l)}\right) \tag{66}$$

The activation of a similar cell in an RNN with dilated skip connections is

$$a^{\langle t \rangle (l)} = f\left(x^{\langle t \rangle (l)}, a^{\langle t-s^{(l)} \rangle (l)}\right) \tag{67}$$

where $s^{(l)}$ is the length of the skip connection in layer l, $x^{(l)}$ is the input to layer l, and f is a function that could represent any linear or non-linear function or collection of functions commonly used in an RNN, including LSTM, GRU, etc.

A dilated recurrent neural network is one where dilated layers are stacked together in order to form a complete network. The dilation in each layer should increase exponentially. This makes different layers focus on different temporal resolutions, and reduces the average length of paths between nodes at different time points (proof in [10]). An example dilated RNN with three layers and a dilation of two can be seen in figure

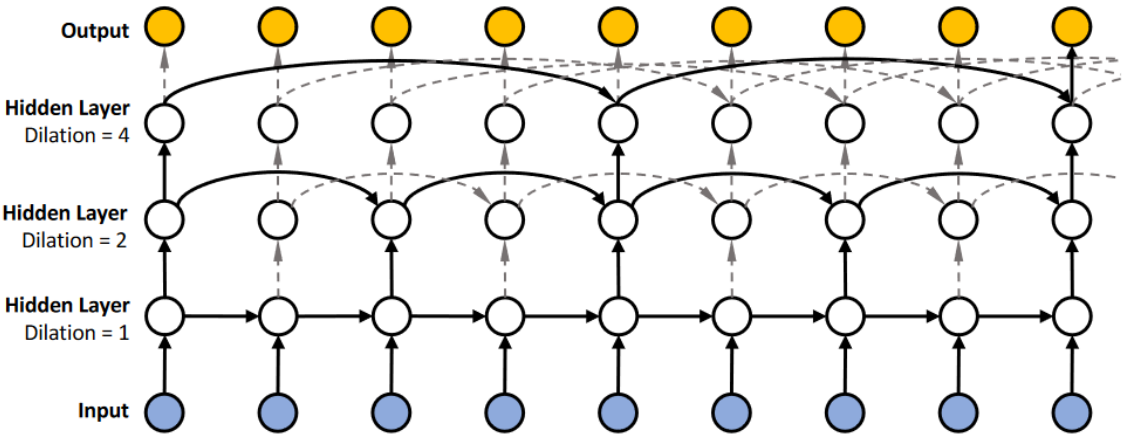


Figure 8: An example of a dilated RNN with three layers with dilations of 1, 2, and 4, respectively.

The dilations in a dilated RNN can cause missing dependencies in the output layer. These dependencies can be reintroduced by including convolutional connections from the final hidden layer to the output layer.

In order to test the performance of the dilated RNN, Chang et al uses the *copy memory problem*. This task tests the ability of recurrent models to memorise long-term information. Each input sequence is of length $T + 20$. The first ten values are randomly generated from integers 0 to 7, the next $T - 1$ values are all 8, the last 11 values are all 9. The first occurance of a 9 tells the model that it must reproduce the first 10 digits. The random guess yields an expected average cross entropy of $\ln(8) \approx 2.079$. The results of the copy memory problem can be seen in figure 9.

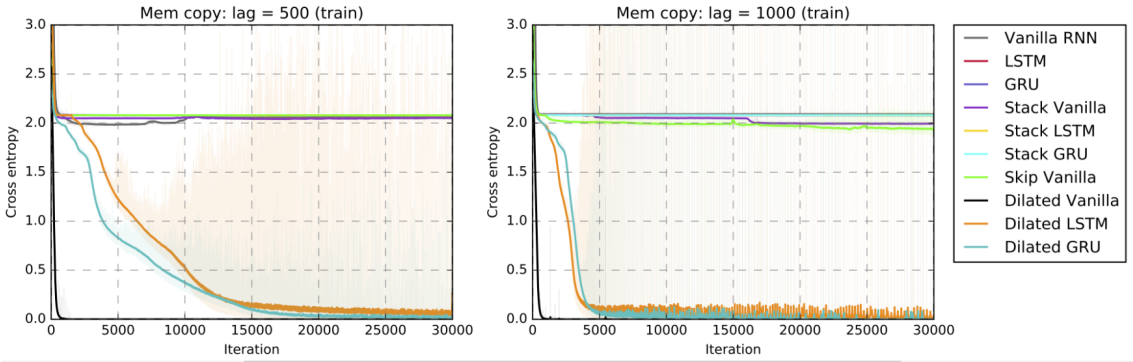


Figure 9: Results of the copy memory problem with $T = 500$ (left) and $T = 1000$ (right). Only RNNs with dilated skip connections converge to the perfect solution, all other methods are unable to improve over random guesses.

Chang et al also used the MNIST dataset of handwritten digits to test the performance of various RNNs. The RNN was tasked with classifying the digit after reading in each of the pixels one-by-one as a 784×1 sequence. The results of the task can be seen in figure 10. All the RNNs without skip connections failed at the task. The 'Vanilla' RNN with standard skip connections showed some success. This is consistent with the finding that RNNs with skip connections are better at learning long term dependencies. The RNNs with dilated skip connections performed the best, with the dilated GRU showing the best performance after training.

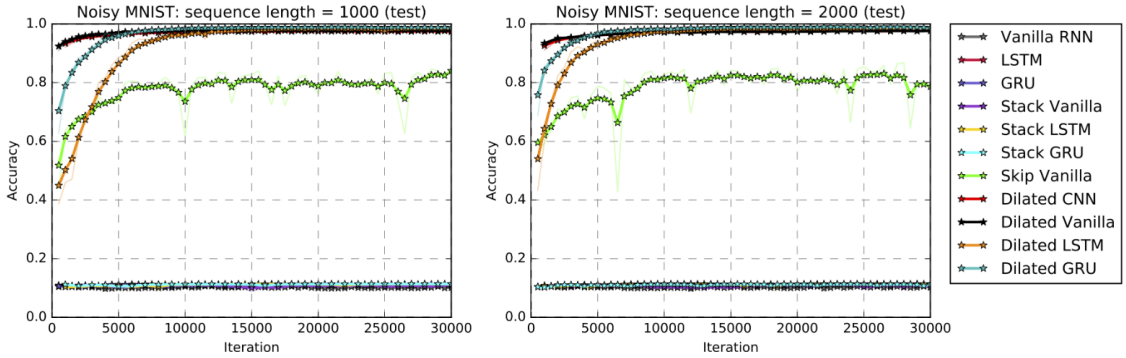


Figure 10: Results of the MNIST classification task used by Chang et al to assess the performance of the dilated RNNs against other RNNs. The dilated RNNs perform the best, with the dilated GRU showing the best performance after training.

10.2.2 The Attention Mechanism [11]

The attention mechanism deals with the case where there exists a target time series $y_1, \dots, y_{t-1} \in \mathbb{R}$ and n driving (exogeneous) series $\mathbf{x}_1, \dots, \mathbf{x}_t \in \mathbb{R}^n$, and the objective is to learn a function F such that $\hat{y}_t = F(y_1, \dots, y_{t-1}, \mathbf{x}_1, \dots, \mathbf{x}_t)$.

The first step is encoding the driving series $(\mathbf{x}_1, \dots, \mathbf{x}_t) = \mathbf{X} \in \mathbb{R}^{n \times T}$. This is done using a bi-directional RNN. If $\mathbf{X}^\top \in \mathbb{R}^{T \times n}$ is a time series of vectors containing one element from each of the n driving series, then a bi-directional RNN will read forward from $\mathbf{x}_1, \dots, \mathbf{x}_{T_x}$ calculating foward hidden states $\overrightarrow{h}_1, \dots, \overrightarrow{h}_{T_x}$, then read backward from $\mathbf{x}_{T_x}, \dots, \mathbf{x}_1$ calculating backward hidden states $\overleftarrow{h}_{T_x}, \dots, \overleftarrow{h}_1$. Concatenating these forward and backward hidden states together gives an encoding, also known as an annotation, h_t which contains information from the whole time series of vectors, but is focussed around timestep t. It is common to use LSTMs or GRUs for these encoders.

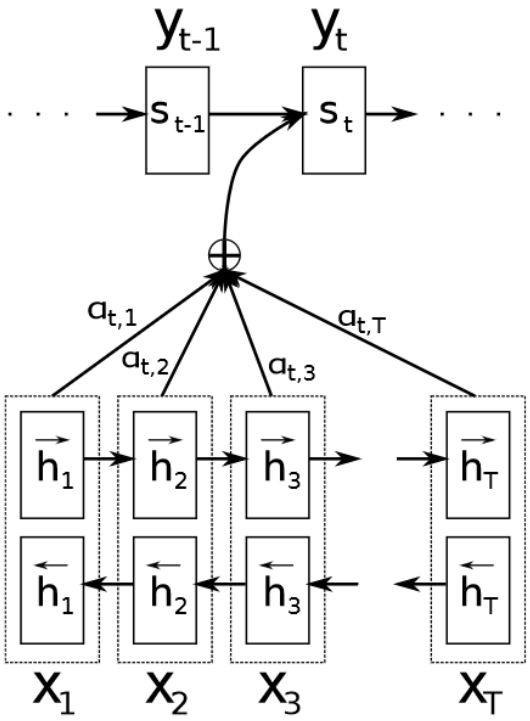


Figure 11: A schematic diagram of an RNN with an attention mechanism. The inputs $\mathbf{X}_t$ are encoded by a bi-directional RNN into hidden states h_t. Those hidden states are combined with the decoder RNN hidden state s_t in the attention model to create a context vector. The context vector is combined with the previous output, and the previous hidden state to create the next output.

The output of the network will come from a forward reading RNN, that reads in the last output y_{t-1}, the current hidden state s_t, and the current *context vector* c_t. To calculate the context vector, the last hidden state of the output RNN is combined with the encoder hidden state corresponding to the current timestep $h_{t'}$ in an *alignment model*,

$$e_{tt'} = a(s_{t-1}, h_{t'}) \tag{68}$$

In [11], the alignment model a takes the form of a feed-forward neural network that is trained along with the rest of the network. The $e_{tt'}$ are combined to create attention coefficients, or attention weightings using the softmax function

$$\alpha_{tt'} = \frac{\exp(e_{tt'})}{\sum_{v=1}^{T_x} \exp(e_{tv})} \tag{69}$$

In the machine translation context $\alpha_{tt'}$ is a probability that the target word y_t is aligned to, or translated from, a source word $x_{t'}$. These attention weightings are used to combine the hidden states linearly to make the context vector,

$$c_t = \sum_{v=1}^{T_x} \alpha_{tv} h_v \tag{70}$$

The context vector, the previous hidden state, and the previous output is combined to create the current hidden state of the output RNN

$$s_t = f(s_{t-1}, y_{t-1}, c_t) \tag{71}$$

and this hidden state is combined with the previous output and the context vector to create the conditional probability of the next output

$$P(y_t | y_1, \dots, t_{t-1}, \mathbf{X}) = g(y_{t-1}, s_t, c_t) \tag{72}$$

The next output y_t is sampled from this distribution. A schematic diagram of the attention mechanism, aka the attention model is shown in figure 11.

The point here is to encode the amount of attention that should be paid to input $\mathbf{x}_{t'}$ when calculating output y_t. This information is held in the attention weightings $\alpha_{tt'}$ where $t, t' \in 1, \dots, T$. The attention weightings can be displayed as a kind of cross correlation matrix in order to discover where dependencies lie. The encoder-decoder architecture also allows the output length to vary regardless of the input length.

In [11], Bahdanau et al tested an RNN encoder-decoder with attention mechanism on English to French translation. A corpus of 348 million words was used. The BLEU score was used to asses the quality of the translation. The BLEU score assesses how well machine translation matches human translation, the closer the match, the higher the score. [2] The RNN encoder-decoder with the attention mechanism (referred to as RNN-search) performance was compared to the performance of an RNN encoder-decoder without the attention mechanism (referred to as RNN-enc). Each type of model was trained using sentences of length up to 30 words, and then sentences of length up to 50 words. The results of the translation task are shown in figure 12. The network with the attention mechanism significantly outperformed the network without. Even the attention network trained with sentences up to 30 words in length outperformed the attention-less network trained with sentences up to 50 words in length when translating sentences up to 60 words in length.

[2] For more information on the BLEU score see: `https://en.wikipedia.org/wiki/BLEU`

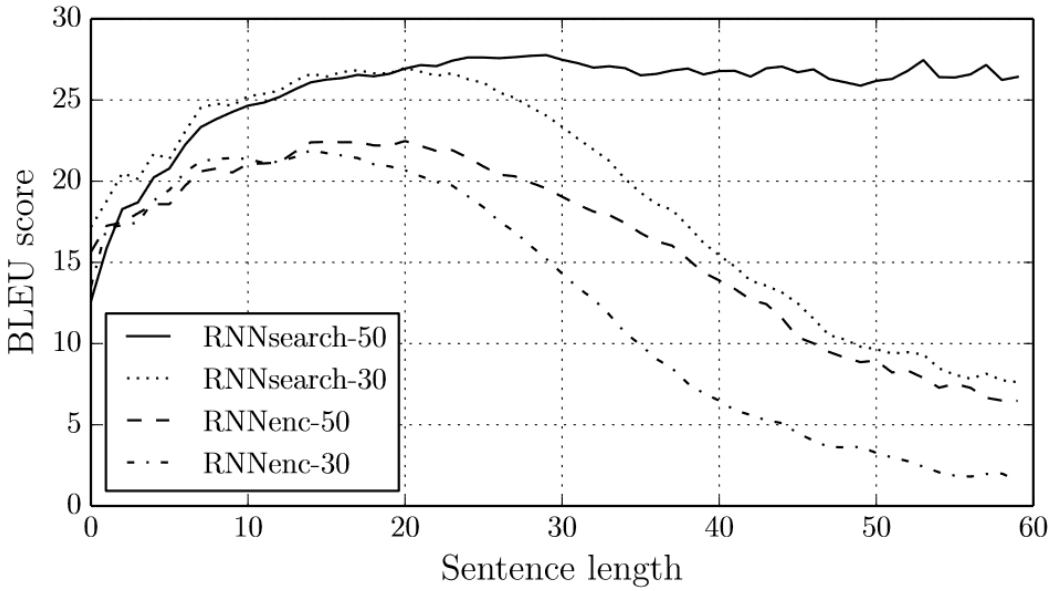


Figure 12: The RNN encoder-decoder with attention (RNN-search) and the RNN encoder-decoder without attention (RNN-enc) were used to translate English into French, and then assessed using the BLEU score system. Each type of model was trained with sentences up to 30 words in length, then up to 50 words in length. The RNN with the attention mechanism significantly outperformed the RNN without the attention mechanism, particularly the 50 word model on longer sentences.

10.2.3 Residual Network [12]

A residual network learns a function for the difference between the inputs and the outputs rather than a function to express the outputs given the inputs. Instead of learning the function F,

$$\mathbf{y} = F(\mathbf{x}) \tag{73}$$

the residual network learns the function H

$$\mathbf{y} = H(\mathbf{x}) - \mathbf{x} \tag{74}$$

The residual network is motivated by the *degradation problem*, where an increase in the number of layers results in a disimprovement in both training and test error. Theoretically, if the added layers just learn the identity function there should be no degradation. This shows that some systems are more diffcult to optimise than others. The idea of the residual network is that it might be easier to optimise a network that learns residual functions instead of the usual output functions.

In a residual network, the inputs for some given layer are fed back into the network two or three layers down. A block of a residual network can be seen in figure 13. A residual network was used to perform classification on the ImageNet database, and the inclusaion of the residual innovation on top of all the other tricks resulted in a classification error lower than human error [12]. An example 34 layer neural network used on the ImageNet database, with and without residual connections is shown in figure 14.

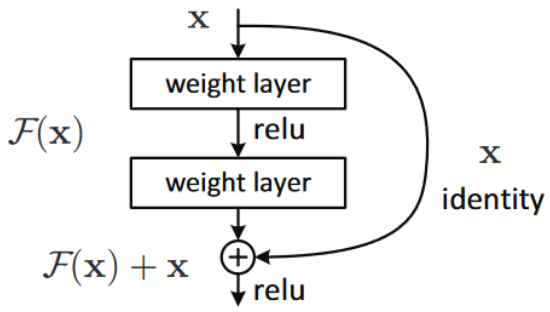


Figure 13: A building block for a residual network. The input for the top layer is projected to the bottom layer, so the block learns a function for the difference between the input and output, i.e. the residual.

10.2.4 Residual LSTM [13]

A residual LSTM is a standard LSTM with a residual innovation where the output of an LSTM in a higher layer is passed in additively before the output gate is applied. Equations 53, 54, 55, 56, and 57 are all the same. A *projection output* is defined by

$$m^{\langle t \rangle (l)} = W_m \tanh(c^{\langle t \rangle (l)}) \tag{75}$$

the output from the previous layer's LSTM is added to this projection, then the output gate is applied

$$a^{\langle t \rangle (l)} = \Gamma_o \odot \left(m^{\langle t \rangle (l)} + W_{a'} a^{\langle t \rangle (l-1)} \right) = \Gamma_o \odot \left(m^{\langle t \rangle (l)} + W_{a'} x^{\langle t \rangle (l)} \right) \tag{76}$$

Essentially, the input to the LSTM is carried forward and reinserted just before last step of the LSTM processing. Each LSTM unit is a residual building block, similar but more complicated to that shown in figure 13.

in [13], Kim et al used the AMI meeting corpus to evaluate residual LSTMs compared to standard LSTMs and highway LSTMs (LSTMs with a parametrised spacial skip connection). The AMI meeting corpus is a dataset of recordings of people meeting up and speaking English. The RNNs were trained to predict the next word for each speaker speaking at any given time. Both 3-layer and 10-layer networks were trained. The results of the AMI assessment for highway and residual LSTMs are shown in figure 15. Both the training and cross-validation error for the 10-layer highway LSTM network are higher than that for the 3-layer. This indicates that the highway LSTM suffers from degredation. For the residual LSTM, although the training error is higher for the 10-layer network, the cross-validation error is lower for the deeper network. So the residual LTSM does not suffer from degredation. The higher training error for the deeper network could be due to better generalisation for data outside of the training set.

The *word error rate* (WER) for overlapping and non-overlapping speech was also assessed for standard LSTMs, highway LSTMs, and residual LSTMs. 3-layer, 5-layer, and 10-layer versions of each network were used. The results are shown in figure 16. Both the standard and highway LSTM networks have higher word error rates in general than the residual LSTM, and increasing word error rates for deeper networks. The residual LSTM network shows improving performance for deeper networks for non-overlapping speech.

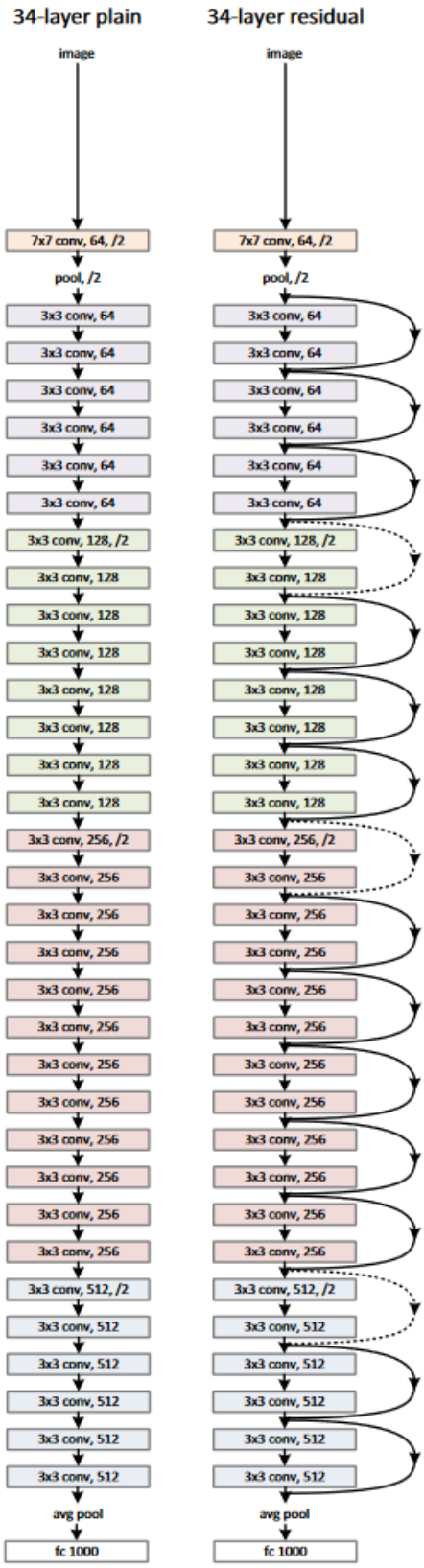
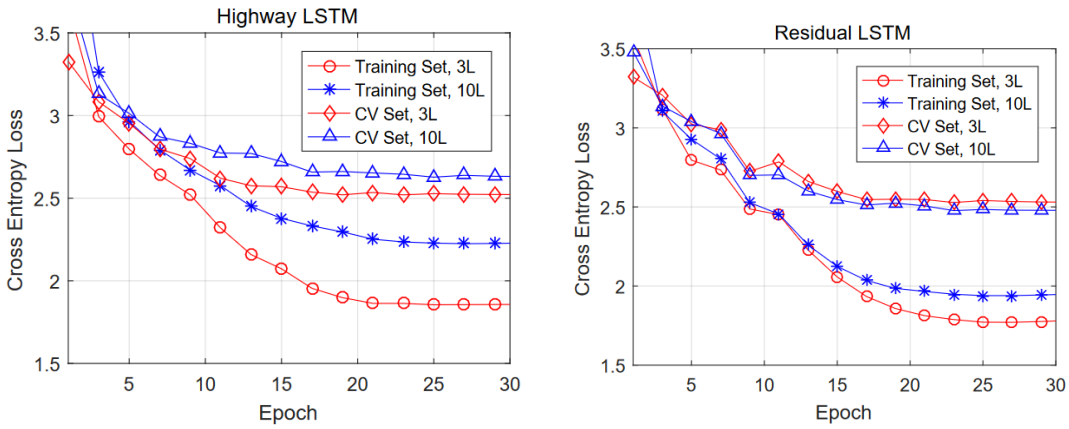


Figure 14: (left) A network architecture with 34 layers that was applied to the ImageNet database. (right) The same architecture with residual connections, making it a residual network.



(a) (b)

Figure 15: (a) The results of the AMI assessment on the highway LSTM network. Note that for both training and cross-validation, the cross-entropy error is higher for the deeper network. (b) The results of the AMI assessment on the residual LSTM network. Although the training error is higher for the deeper network, the cross validation error is lower for the deeper network.

Acoustic Model	Layer	WER (over)	WER (non-over)
Plain LSTM	3	51.1%	42.4%
	5	51.4%	42.5%
	10	56.3%	48.2%
Highway LSTM	3	50.8%	42.2%
	5	51.0%	42.2%
	10	53.5%	44.8%
Residual LSTM	3	50.8%	41.9%
	5	50.0%	41.4%
	10	50.0%	41.0%

Figure 16: Word error rates for overlapped and non-overlapped speech for standard LSTMs, highway LSTMS, and residual LSTMs, of depth 3, 5, or 10.

10.3 Example: Exponential Smooting RNN - M4 Competition Winner [14]

The winner of the M4 competition, a competition for time series forecasting, is a hybrid model combining exponential smoothing and RNNs. The model was implemented using DyNet and `C++`.

10.3.1 Holt-Winter's Decomposition Preprocessing

RNNs require a large amount of data in order to be trained. This usually means that they are trained with time series from different contexts. As a result of this, RNN models tend to over-generalise and their forecasts are not time-series specific. One particular problem is that RNN models struggle with seasonality. Smyl et al (2018) deal with this problem by using an Holt-Winter's seasonal model decomposition (see section 5.3) as a preprocessing step. For each time series in the M4 dataset, the level and seasonality are calculated as

$$\ell_t = \alpha \frac{y_t}{s_t} + (1 - \alpha)\ell_{t-1} \tag{77}$$

$$s_{t+m} = \gamma \frac{y_t}{l_t} + (1 - \gamma)s_t \tag{78}$$

then the forecasting equation is

$$\hat{y}_{t+1,\dots,t+h} = RNN(X_t) \odot \ell_t \odot s_{t+1,\dots,t+h} \tag{79}$$

where X_t is a vector of a normalised and deseasonlised time series features, usually the length of one season, and includes the origin of the series in a binary vector the length of the number of categories.

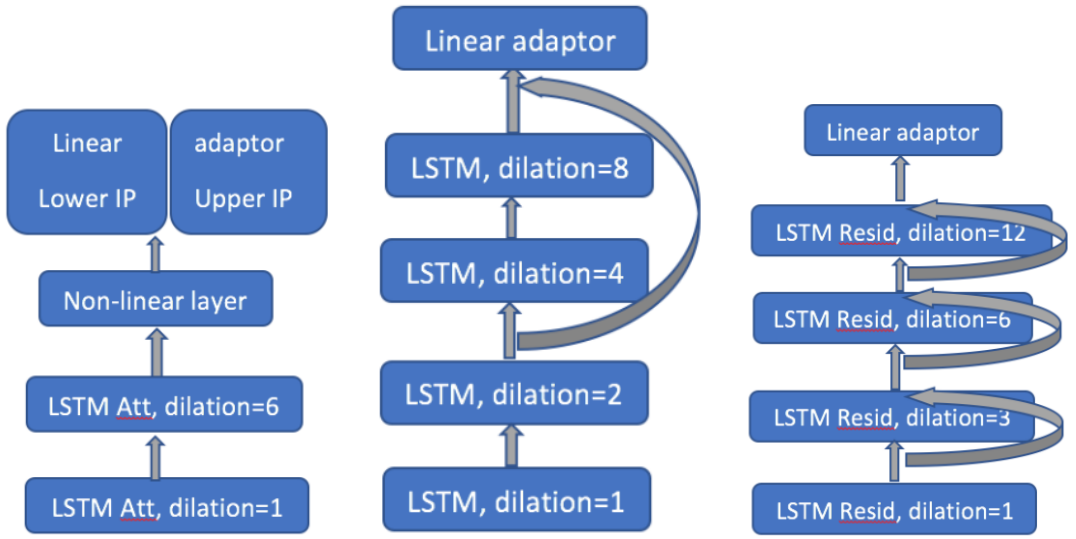
Bear in mind that the parameters for the Holt-Winter's decomposition α, γ, are learned for each time series separately. The parameters in the RNN are learned for all of the time series together. Therefore the mode is hierarchical in nature, part of the model deals with the time series individually, and part of the model deals with all of the dataset as one.

10.3.2 RNN Architectures

Three different RNN architectures were used to model yearly, quarterly, and monthly time series. A schematic diagram of the RNN for yearly data is shown in figure 17a. The network consists of two layers of LSTM units using the attention mechanism and dilations, followed by a fully connected (aka dense) layer with a $\tanh$ activation function, followed by a linear adaptor layer. The linear adaptor layer transforms the output of the non-linear layer into the shape of the forecast horizon and produces upper and lower prediction intervals.

A schematic diagram of the RNN used for quarterly data is shown in figure 17b. The network consists of four layers of LSTM units with exponentially increasing dilations. The top two layers of LSTMs make up a 'classical' residual block. The output layer is a linear adaptor that converts the size of the LSTM output to the size of the forecast horizon.

A schematic diagram of the RNN used for the monthly data is shown in figure 17c. The network consists of four layers of residual LTSM units. Note that these are residual LSTMs, not residual connections around standard LSTMs. The output layer is a linear adaptor similar to that in the RNN for quarterly data.



(a) (b) (c)

Figure 17: (a) A schematic diagram of the RNN architecture used by Smyl et al (2018) for forecasting **yearly** time series. (b) A schematic diagram of the RNN architecture used by Smyl et al (2018) for forecasting **quarterly** time series. (c) A schematic diagram of the RNN architecture used by Smyl et al (2018) for forecasting **monthly** time series.

10.3.3 Loss Function

The ES-RNN model uses a *Pinball loss function*.

A Pinball loss function is a loss function that measures the difference between the actual and forecasted values and introduces some bias to penalise positive errors more than negative errors, or negative errors more than positive errors. A Pinball loss function takes the form of

$$L_Q(y_t - \hat{y}_t) = \begin{cases} Q(y_t - \hat{y}_t) & \text{if } y_t \geq \hat{y}_t \\ (Q-1)(y_t - \hat{y}_t) & \text{if } y_t < \hat{y}_t \end{cases} \tag{80}$$

where $0 \leq Q \leq 1$. If $Q = 0.5$ then $L_Q(y_t - \hat{y}_t)$ is equivalent to the ℓ_1 loss function. If $Q > 0.5$ then errors where $y_t \geq \hat{y}_t$ will be penalised more heavily than errors where $y_t \leq \hat{y}_t$.

For the ES-RNN model, $Q = 0.48$. This value was chosen to counteract positive bias that the authors noticed while developing the model.

For assessing the quality of prediction intervals, the M4 competition used the *Mean Scaled Interval Score*, or MSIS,

$$MSIS(U_t, L_t, Y_t, h, a) = \frac{1}{h} \frac{\sum_{t=1}^{h} (U_t - L_t) + \frac{2}{a}(L_t - Y_t)\mathbf{1}\{Y_t < L_t\} + \frac{2}{a}(Y_t - U_t)\mathbf{1}\{Y_t > U_t\}}{\frac{1}{n-m}\sum_{t=m+1}^{n} |Y_t - Y_{t-m}|} \tag{81}$$

where U_t is the estimate for the upper prediction interval, L_t is the estimate for the lower

prediction interval, h is the horizon, a is a significance level ,and

$$\mathbf{1}\{\textbf{conditional}\} = \begin{cases} 1 & \text{if } \textbf{conditional} \text{ is true} \\ 0 & \text{if } \textbf{conditional} \text{ is false} \end{cases} \tag{82}$$

is called the *indicator function*. The ES-RNN model used the numerator of the MSIS function to train the model when forecasting prediction intervals.

References

[1] Rob J Hyndman, George Athanasopoulos, *Forecasting: Principles and Practice*. Monash University, Australia, (2018)

[2] Thiyanga S Talagala, Rob J Hyndman, George Athanasopoulos, *Meta-learning how to forecast time series*. Monash Econometrics and Business Statistics Working Papers, Monash University, Department of Econometrics and Business Statistics, (2018)

[3] Pablo Montero-Manso, Thiyanga Talagala, Rob J Hyndman, George Athanasopoulos, *https://github.com/robjhyndman/M4metalearning*. M4 Competition (2018)

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. EMNLP, 1724–1734, (2014)

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. NIPS, (2014)

[6] Jos van der Westhuizen, Joan Lasenby, *The Unreasonable Effectiveness of the Forget Gate*. https://arxiv.org/pdf/1804.04849.pdf, (2018)

[7] Corentin Tallec, Yann Ollivier, *Can Recurrent Neural Networks Warp Time?* International Conference on Learning Representations, (2018)

[8] Cyrille Delabre, https://github.com/cyrilledelabre/cfm-challenge

[9] Zhen He, Shaobing Gao, Liang Xiao, Daxue Liu, Hangen He, David Barber, *Wider and Deeper, Cheaper and Faster: Tensorized LSTMs for Sequence Learning*. NIPS, (2017)

[10] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Kiaodong Cui, Michael Witbrock, Mark Hasegawa-Johnson, Thomas S. Huang, *Dilated Recurrent Neural Networks*. NIPS (2017)

[11] Dzmitry Bahdanau, Kyung Hyun Cho, Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*. International Conference on Learning Representations, (2015)

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Su, *Deep Residual Learning for Image Recognition*. Conference on Computer Vision and Pattern Recognition (CVPR), (2016)

[13] Jaeyoung Kim, Mostafa El-Khamy, Jungwon Lee, *Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition*. arXiv:1701.03360 (2017)

[14] Slawek Smyl, Jai Ranganathan, Andrea Pasqua, *M4 Forecasting Competition: Introducing a New Hybrid ES-RNN Model*. https://eng.uber.com/m4-forecasting-competition/, (2018)