QBUS 6840: Lecture 10

Neural Networks and Recurrent Neural Networks

Professor Junbin Gao

The University of Sydney Business School

Outlines

- Neural Networks Architecture
- Deep Structure in Neural Networks
- Recurrent Neural Networks (RNNs)
- Neural Network Autoregression
- Long Short-Term Memory (LSTM)
- Other Variants of Recurrent Neural Networks
- An Application Example

Readings:

Online textbook Section 11.3 https://otexts.com/fpp2/nnetar.html; and Slides http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Lecture slides are adopted from the above tutorial.

Neural Networks

- Artificial neural networks (or simply neural networks (NNs)) are forecasting methods that are based on simple mathematical models of the brain
- They allow complex nonlinear relationships between the response variable and its predictors
- A neural network can be thought of as a network of "neurons" organised in layers
- The predictors (or inputs) form the input layer, and the forecasts form the output layer
- There may be intermediate layers containing "hidden neurons"

Multilayer Feed-forward Neural Networks

- There are large variants of neural networks depending on the networks topology
- The most popular and simple one is the so-called multilayer feed-forward neural network
- In feed-forward way, each layer of nodes receives inputs from the previous layers. And the outputs of nodes in one layer are inputs to the next layer
- Thus the information is passed on along the layers from the input layer towards the output layer

Linear Regression Model

- Consider three features (regressors/predictors) x_1, x_2 and x_3, for one dependent/response variable y
- The linear regression model is

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Linear Regression Model

- Consider three features (regressors/predictors) x_1, x_2 and x_3, for one dependent/response variable y
- The linear regression model is

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

- Let add an intermediate variable z

$$y := z := w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Linear Regression Model

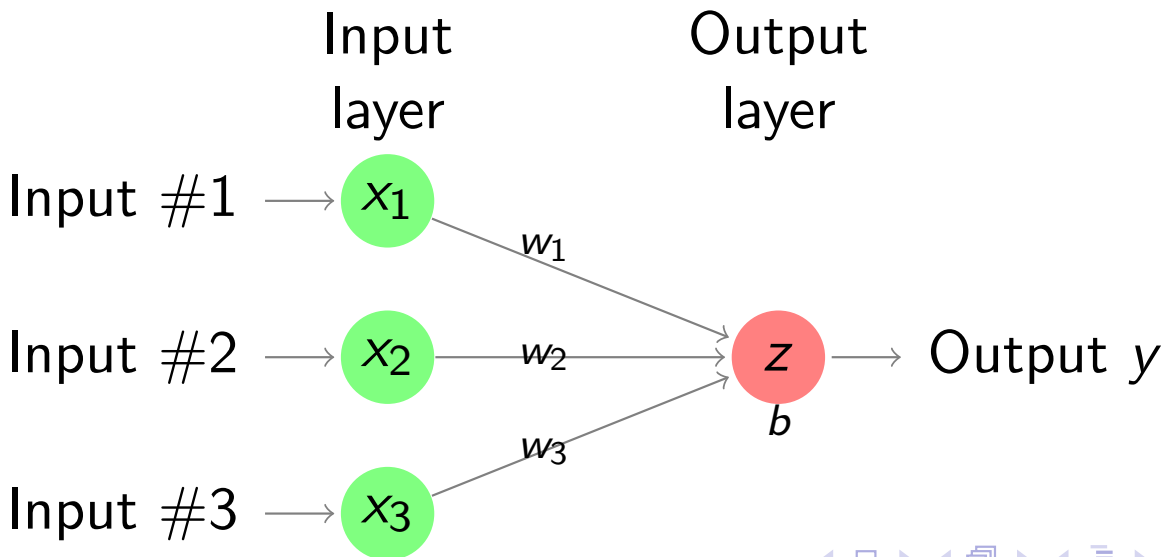
- Consider three features (regressors/predictors) x_1, x_2 and x_3, for one dependent/response variable y
- The linear regression model is

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

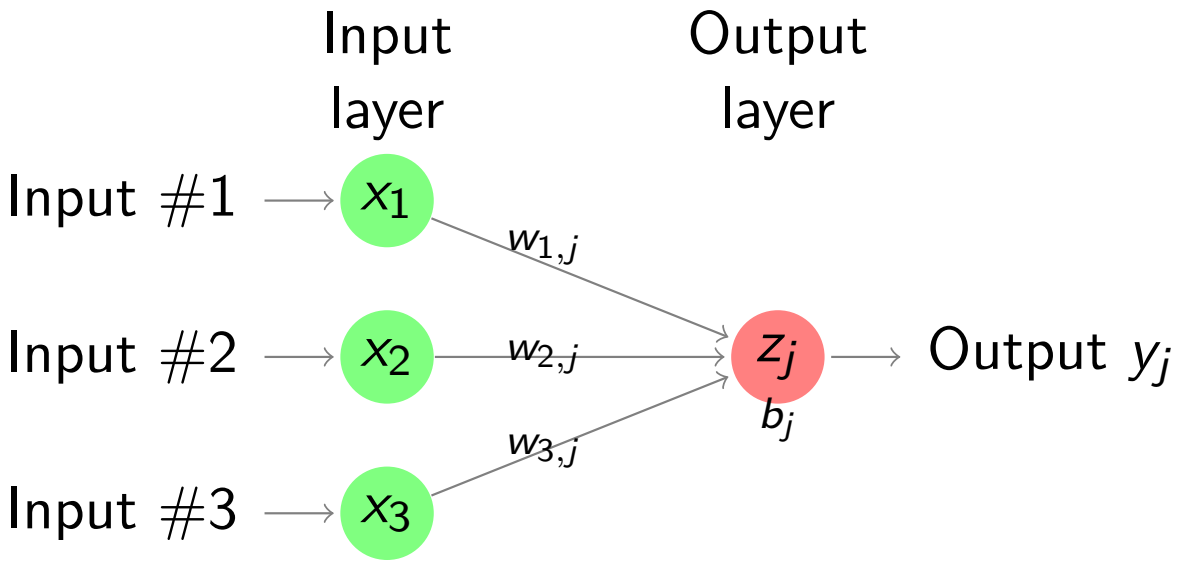
- Let add an intermediate variable z

$$y := z := w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

- Draw this model in the following diagram



A Simple Neural Networks (No hidden layer)

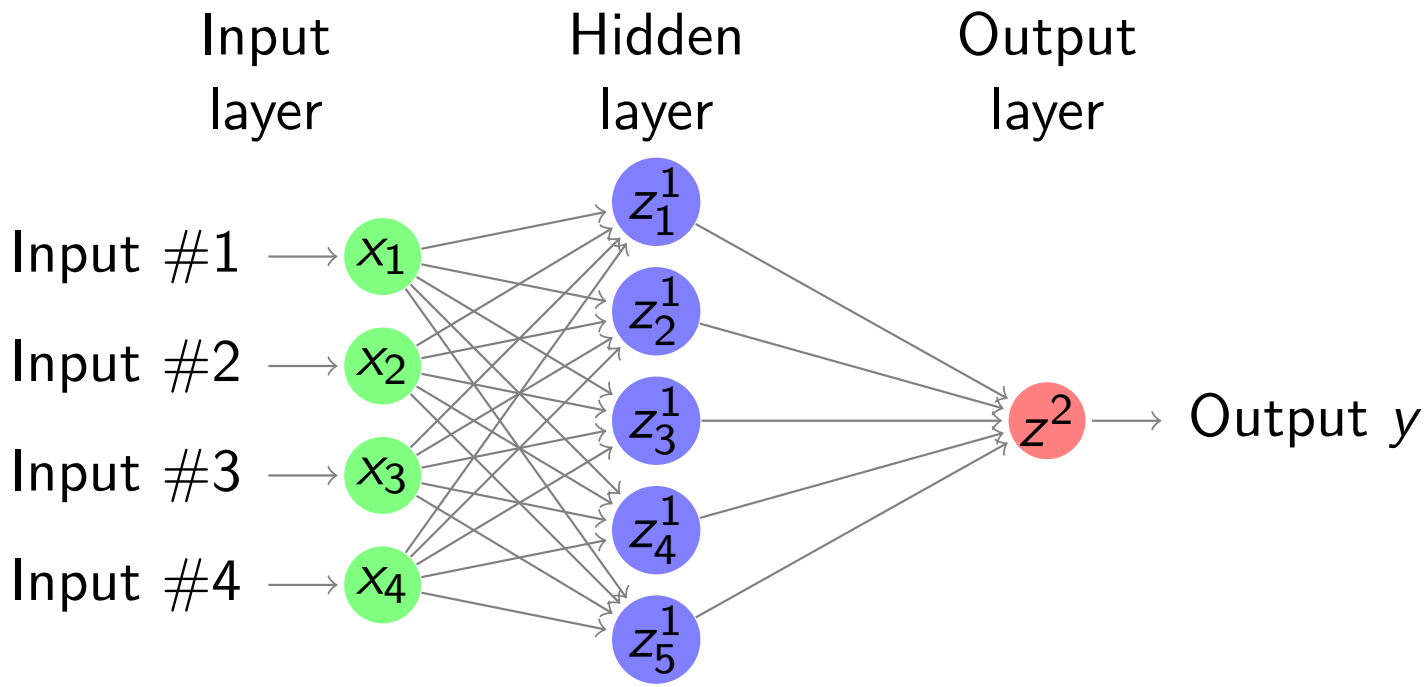


- Information flow through a linear mapping
  $z_j = \sum_{i=1}^{3} w_{i,j} x_i + b_j$
- At the output, the above value may be then modified using a nonlinear function such as a sigmoid,

$$y_j = \sigma(z_j) = \frac{1}{1 + \exp(-z_j)} = \frac{1}{1 + \exp(-w_{1,j}x_1 - w_{2,j}x_2 - w_{3,j}x_3 - b_j)}$$

- Without applying the so-called activation function, the network is equivalent to a linear regression.

Slightly Complex Neural Networks



- The function applied on hidden or output nodes is called activation function
- Except for the sigmoid function, other functions like `tanh` or the rectified linear unit (ReLU)

$$\text{ReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

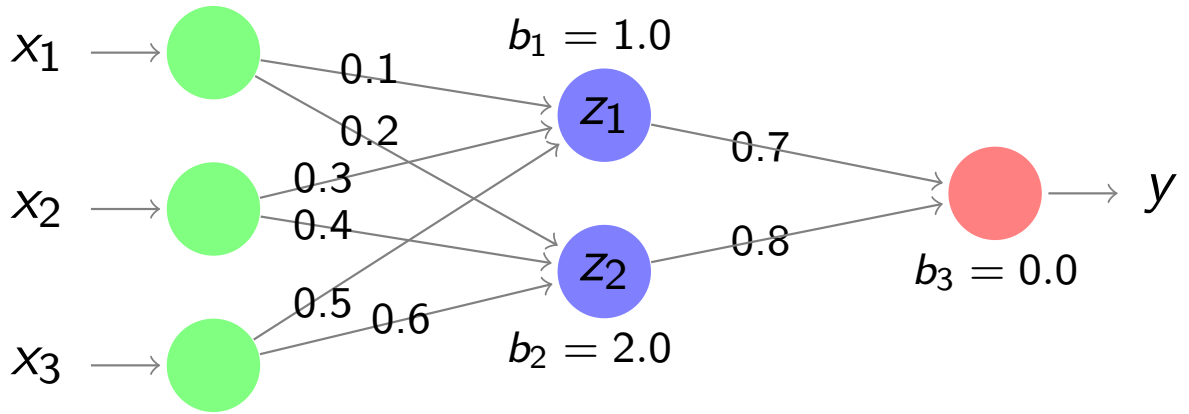
Models defined by Neural Networks and Training

- The previous network actually defines the following composition nonlinear function, (on the output node, we take no activation) , where f is the chosen activation function,

$$
\begin{aligned}
y =& z^2 \\
=& w_1 f(z_1^1) + w_2 f(z_2^1) + w_3 f(z_3^1) + w_4 f(z_4^1) + w_5 f(z_5^1) + b \\
=& \sum_{j=1}^{5} w_j f(z_j^1) + b \\
=& \sum_{j=1}^{5} w_j f(\sum_{k=1}^{4} w_{k,j} x_k + b_j) + b
\end{aligned}
$$

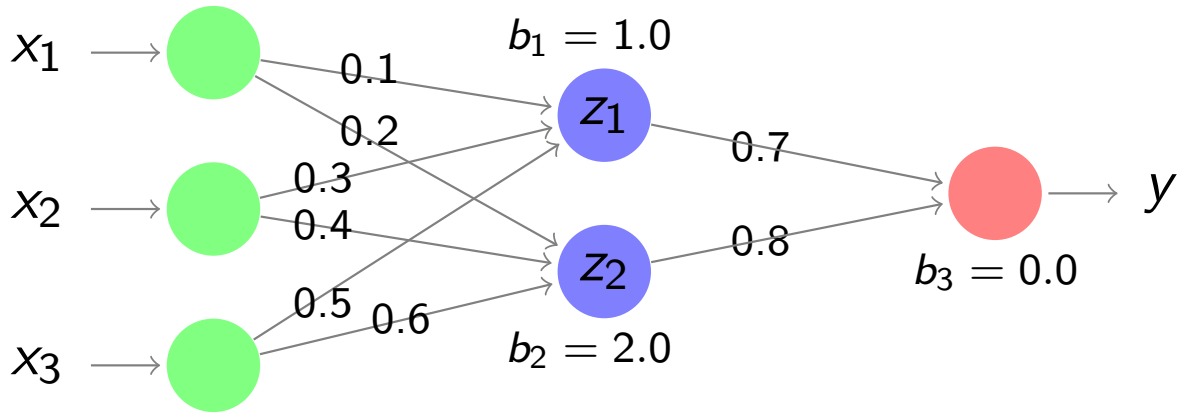
- Training means fitting this mathematical model with training data, i.e., finding the best weights $w_{1,1}, ..., w_{4,5}, b_1, ..., b_5, w_1, ..., w_5, b$.

Example



- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

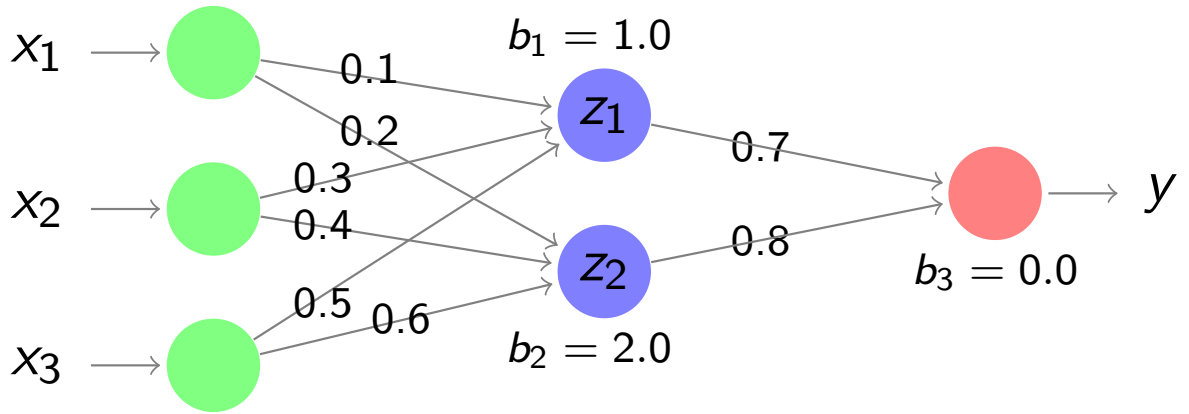
Example



- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

$$y =$$

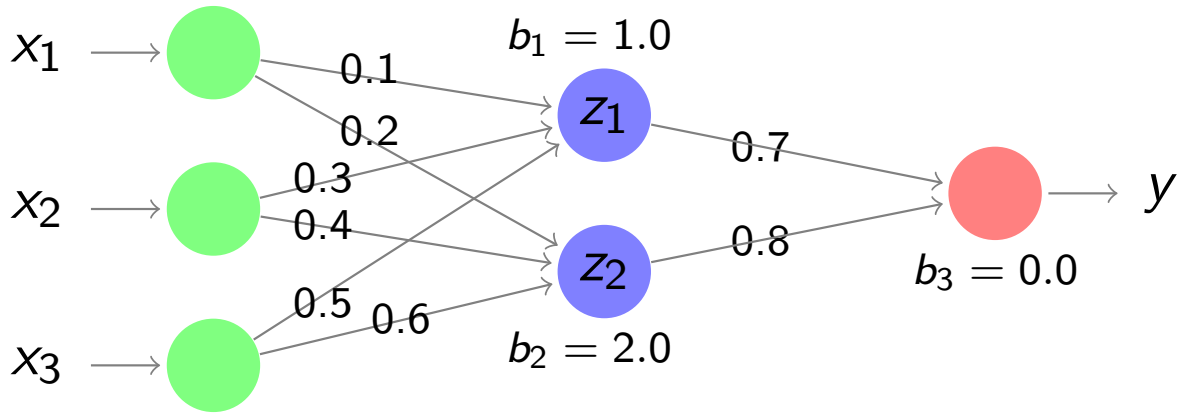
Example



- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

$$y = 0.7\sigma(z_1) + 0.8\sigma(z_2) + 0.0$$

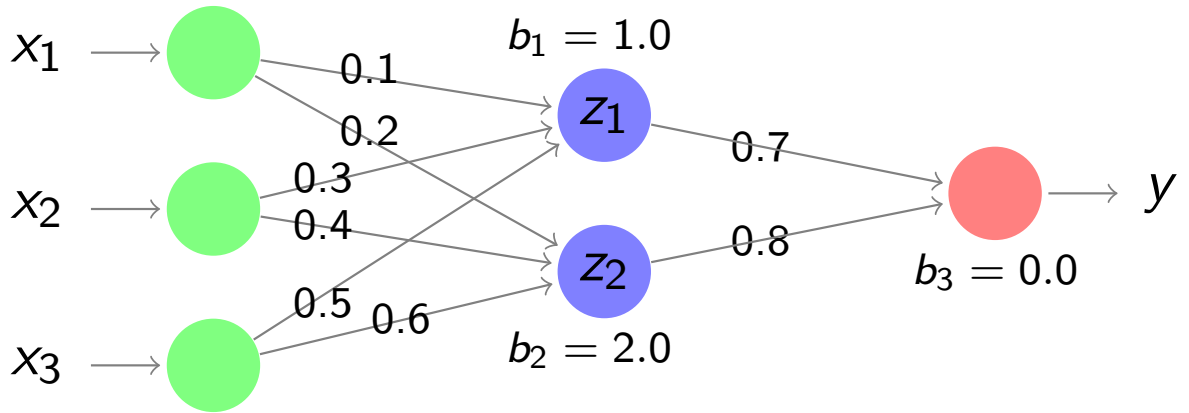
Example



- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

$$
\begin{aligned}
y &= 0.7\sigma(z_1) + 0.8\sigma(z_2) + 0.0 \\
&= 0.7\sigma(0.1\ x_1 + 0.3\ x_2 + 0.5\ x_3 + 1.0) \\
&\quad +
\end{aligned}
$$

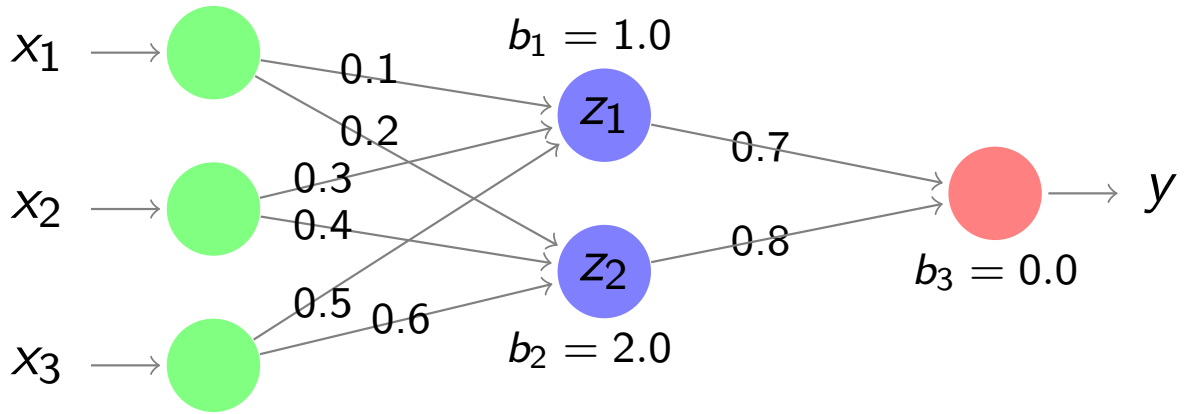
Example



- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

$$
\begin{aligned}
y &= 0.7\sigma(z_1) + 0.8\sigma(z_2) + 0.0 \\
&= 0.7\sigma(0.1\ x_1 + 0.3\ x_2 + 0.5\ x_3 + 1.0) \\
&\quad +
\end{aligned}
$$

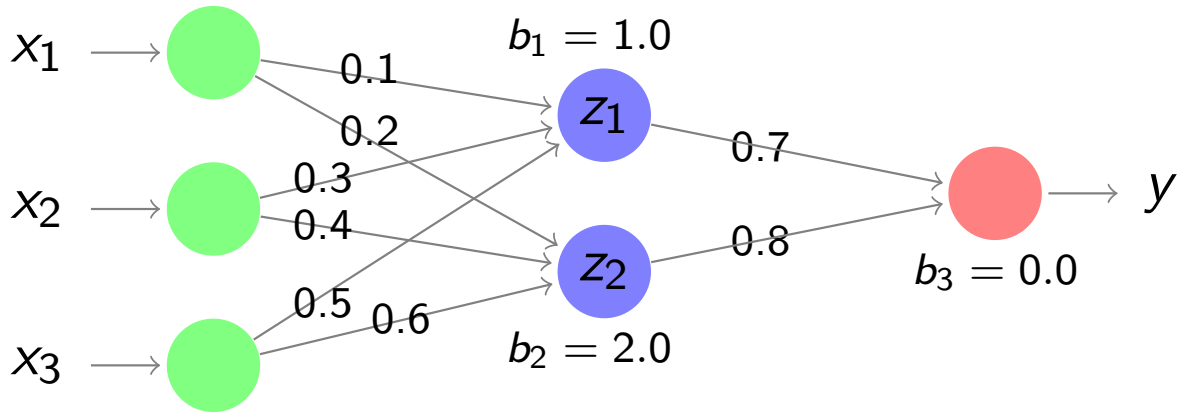
Example



- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

$$
\begin{aligned}
y &= 0.7\sigma(z_1) + 0.8\sigma(z_2) + 0.0 \\
&= 0.7\sigma(0.1\ x_1 + 0.3\ x_2 + 0.5\ x_3 + 1.0) \\
&\quad + 0.8\sigma(0.2\ x_1 + 0.4\ x_2 + 0.6\ x_3 + 2.0)
\end{aligned}
$$

Example

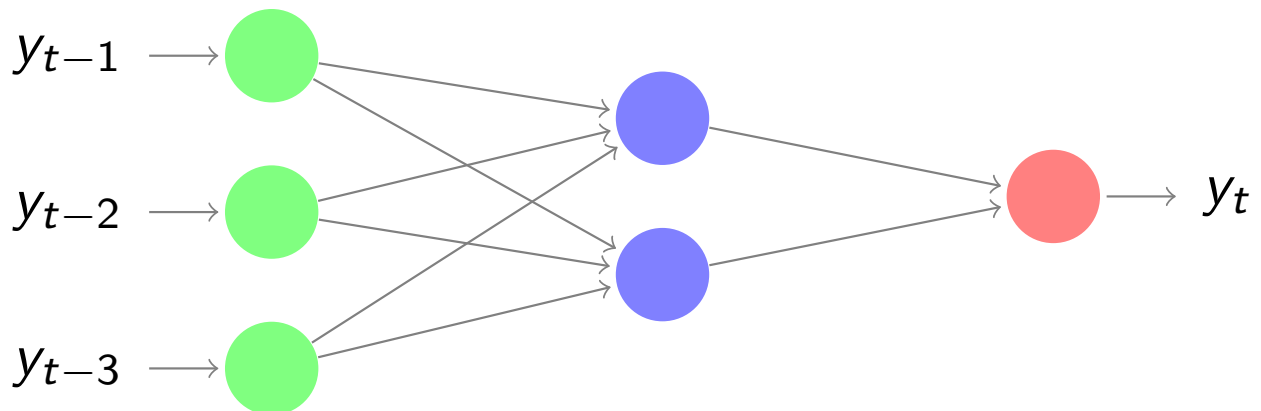


- The model (function) defined by this neural network is, suppose we use the sigmoid activation function σ,

$$\begin{aligned} y &= w_1\sigma(z_1) + w_2\sigma(z_2) + b \\ &= w_1\sigma(w_{1,1}x_1 + w_{2,1}x_2 + w_{3,1}x_3 + b_1) \\ &\quad + w_2\sigma(w_{1,2}x_1 + w_{2,2}x_2 + w_{3,2}x_3 + b_2) \end{aligned}$$

Neural Networks for Time Series: Autoregression

- For time series, we shall use lagged values of time series as inputs to a neural network and the output as the prediction
- This means, in general, the number of input neurons of a neural network is the number of the time lag and there is only one output neuron
- We will first consider feed-forward networks with one hidden layer. An example



Neural Networks for Time Series: Autoregression

- We denote by NNAR(p, k) the NNs with p lagged inputs and k neurons in the hidden layer and with one output as the forecast
- For example NNAR$(12, 50)$ model is a neural network with the last 12 observations $(y_{t-1}, y_{t-2}, ..., y_{t-12})$ to fit y_t at any time step t, with 50 neurons in the hidden layer
- A NNAR$(p, 0)$ model is equivalent to ARIMA$(p, 0, 0)$ without the restrictions on the parameters to ensure stationary.
- In terms of NNAR$(p, 0)$ model, we may use $\phi_1 Y_{t-1} + \phi_2 Y_{t-2}$ to predict Y_t for any ϕ_1 and ϕ_2, however, for an AR(2) model $Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \varepsilon_t$ to be stationary:

$$-1 < \phi_1 < 1, \phi_1 + \phi_2 < 1, \phi_2 - \phi_1 < 1.$$

Neural Networks for Time Series: Training

- Recall how we learn an ARIMA$(p, 0, 0)$ model

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

- Suppose the given time series is

$$\mathcal{T} = \{y_1, y_2, y_3, \cdots, y_{T-1}, y_T\}$$

- We form the training data as

$$y_1, y_2, ..., y_p \rightarrow y_{p+1} : \ \varepsilon_{p+1} = (y_{p+1} - c - \phi_1 y_p - \phi_2 y_{p-1} - \cdots - \phi_p y_1)$$
$$y_2, y_3, ..., y_{p+1} \rightarrow y_{p+2} : \ \varepsilon_{p+2} = (y_{p+2} - c - \phi_1 y_{p+1} - \phi_2 y_p - \cdots - \phi_p y_2)$$
$$y_3, y_4, ..., y_{p+2} \rightarrow y_{p+3} : \ \varepsilon_{p+3} = (y_{p+3} - c - \phi_1 y_{p+2} - \phi_2 y_{p+1} - \cdots - \phi_p y_3)$$
$$\vdots$$
$$y_{T-p}, y_{T-p+1}, ..., y_{T-1} \rightarrow y_T : \ \varepsilon_T = (y_T - c - \phi_1 y_{T-1} - \phi_2 y_{T-2} - \cdots - \phi_p y_{T-p})$$

- To find out all the coefficients c, ϕ_1, ..., ϕ_p, we minimise

$$\min_{c, \phi_1, ..., \phi_p} E(c, \phi_1, ..., \phi_p) = \sum_{i=p+1}^{T} \varepsilon_i^2 = \varepsilon_{p+1}^2 + \varepsilon_{p+2}^2 + \varepsilon_{p+3}^2 + \cdots + \varepsilon_T^2$$

Neural Networks for Time Series: Training

- Consider the neural network NNAR(p, k). For a given section of time series $y_{t-p+1}, y_{t-p+2}, ..., y_t$ denote the output from NNAR(p, k) by $\widehat{y}_{t+1} = F(y_{t-p+1}, y_{t-p+2}, ..., y_t; W)$ where W collects all the weights in the network.
- We can also form the following

$$
\begin{aligned}
&y_1, y_2, ..., y_p \to y_{p+1} : \ \varepsilon_{p+1} = (y_{p+1} - F(y_1, y_2, ..., y_p; W)) \\
&y_2, y_3, ..., y_{p+1} \to y_{p+2} : \ \varepsilon_{p+2} = (y_{p+2} - F(y_2, y_3, ..., y_{p+1}; W)) \\
&y_3, y_4, ..., y_{p+2} \to y_{p+3} : \ \varepsilon_{p+3} = (y_{p+3} - F(y_3, y_4, ..., y_{p+2}; W)) \\
&\qquad \vdots \\
&y_{T-p}, y_{T-p+1}, ..., y_{T-1} \to y_T : \ \varepsilon_T = (y_T - F(y_{T-p}, y_{T-p+1}, ..., y_{T-1}; W))
\end{aligned}
$$

- To find out all the coefficients W, we minimise

$$\min_W E(W) = \sum_{i=p+1}^{T} \varepsilon_i^2 = \varepsilon_{p+1}^2 + \varepsilon_{p+2}^2 + \varepsilon_{p+3}^2 + \cdots + \varepsilon_T^2$$

- BP (backpropagation) algorithm can be applied to minimise the above objective.

Neural Networks for Seasonal Time Series

- Except for the lagged data as inputs, it is useful to add the last observed values from the same seasons as inputs, for seasonal time series
- The notation NNAR$(p, P, k)_m$ means a model with inputs

$$(y_{t-1}, y_{t-2}, \ldots, y_{t-p}, y_{t-m}, y_{t-2m}, \ldots, y_{t-Pm})$$

  and k neurons in the hidden layer
- What is the input for NNAR$(3, 2, 10)_6$?
- A NNAR$(p, P, 0)_m$ model is similar to a special ARIMA$(Pm, 0, 0)$ model without the restrictions on the parameters to ensure stationarity
- This time, How do you organise your training data?

Modeling Time Series with Neural Networks

- Python package `keras` implements most neural networks in a user-friendly way.
- Please note `keras` relies on backend either `Theano` or `TensorFlow` for heavy symbolic computation, e.g, automatically creating the BP algorithm.
- Normally we can simply use theano which can be easily installed on Anaconda. In fact, we dont need to understand how theano works behind the scene
- If there is no C++ compiler or compiled binary library on your desktop/laptop, the speed will be very slow.
- I hope you are ready

A simple Recipe

1. Exploratory data analysis: Apply some of the traditional time series analysis methods to estimate the lag dependence in the data (e.g. auto-correlation and partial auto-correlation plots, transformations, differencing). For example the lag may be $p = 4$
2. Split your data into two main sections: training section $\{Y_1, Y_2, ..., Y_n\}$ and validation section $\{Y_{n+1}, Y_{n+2}, ..., Y_T\}$. For example, you may use data in the first two years for training, the data in the third year as validation for a time series of three years data
3. Define the neural network architecture: Although you have freedom to decide how many hidden layers (depth) and how many neurons on each hidden layer, picking appropriate numbers for them is not something very easy. This is actually part of model selection

A simple Recipe

4. Create the training patterns: Each training pattern will be $p+1$ values, with the first four p corresponding to the input neurons and the last one defining the prediction as the output node. Here is all the data pattern for the training data $\{Y_1, Y_2, ..., Y_n\}$ for a NNAR$(4, k)$.

$$
\begin{aligned}
\text{Training Pattern 1:} \quad & Y_1, Y_2, Y_3, Y_4 \rightarrow Y_5 \\
\text{Training Pattern 2:} \quad & Y_2, Y_3, Y_4, Y_5 \rightarrow Y_6 \\
\text{Training Pattern 3:} \quad & Y_3, Y_4, Y_5, Y_6 \rightarrow Y_7 \\
& \vdots \qquad\qquad\qquad\qquad \vdots \\
\text{Training Pattern } n-4: \quad & Y_{n-4}, Y_{n-3}, Y_{n-2}, Y_{n-1} \rightarrow Y_n
\end{aligned}
$$

Forecasts are highlighted.

A simple Recipe

5. Train the neural network on these patterns
6. Test/predict the network on the validation set $\{Y_{n+1}, Y_{n+2}, ..., Y_T\}$: Here you will pass in four values as the input layer and see what the output node gets.

$$
\begin{aligned}
&\text{Test Patt 1:} && Y_{n+1}, Y_{n+2}, Y_{n+3}, Y_{n+4} \rightarrow \widehat{Y}_{n+5} \\
&\text{Test Patt 2:} && Y_{n+2}, Y_{n+3}, Y_{n+4}, Y_{n+5} \rightarrow \widehat{Y}_{n+6} \\
&\text{Test Patt 3:} && Y_{n+3}, Y_{n+4}, Y_{n+5}, Y_{n+6} \rightarrow \widehat{Y}_{n+7} \\
&\vdots && \vdots
\end{aligned}
$$

A Sample Recipe: Training Pattern for NNAR$(3, 2, k)_4$

1. This is a network of 5 inputs for a seasonal time series of period $M = 4$. Three inputs from normal lagged observations and Two from seasonal lags
2. As we will use the seasonal lagged for prediction, we must look back at least $P \times M = 2 \times 4 = 8$ time units. That means the first training pattern is to predict Y_9: that is

$$\text{Training Patt 1:} \qquad Y_1, Y_5, Y_6, Y_7, Y_8 \rightarrow Y_9$$

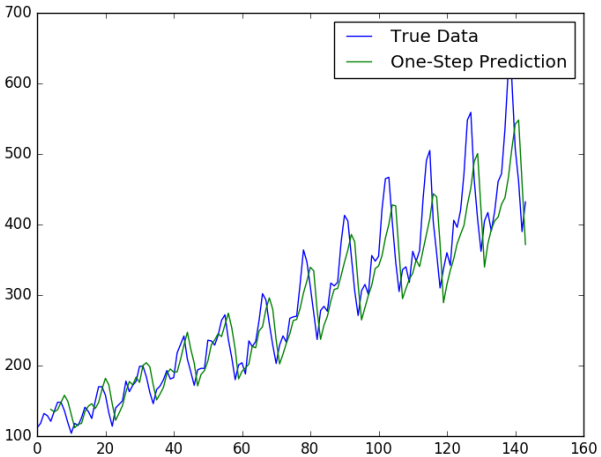
3. $t = 9$, so the inputs $(Y_{t-1}, Y_{t-2}, Y_{t-3}, Y_{t-M}, Y_{t-2M})$ becomes $(Y_8, Y_7, Y_6, Y_5, Y_1)$.
4. Other patterns are

$$\begin{aligned}
\text{Training Patt 2:} &\qquad Y_2, Y_6, Y_7, Y_8, Y_9 \rightarrow Y_{10} \\
\text{Training Patt 3:} &\qquad Y_3, Y_7, Y_8, Y_9, Y_{10} \rightarrow Y_{11} \\
\text{Training Patt 4:} &\qquad Y_4, Y_8, Y_9, Y_{10}, Y_{11} \rightarrow Y_{12} \\
&\qquad \vdots \\
\text{Last Traing Patt:} &\qquad Y_{T-8}, Y_{T-4}, Y_{T-3}, Y_{T-2}, Y_{T-1} \rightarrow Y_T
\end{aligned}$$

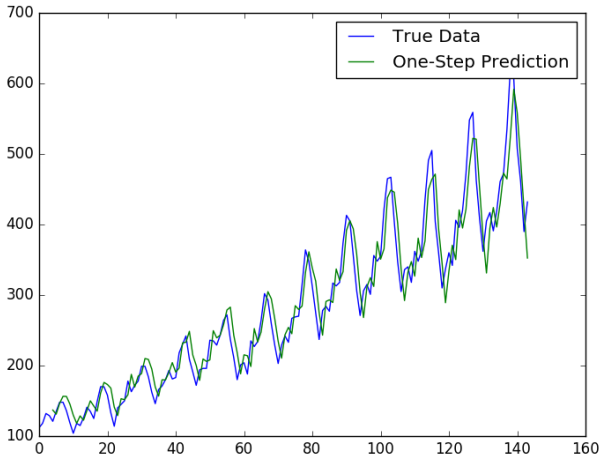
The Example

- Dataset: International Airport Arriving Passengers: in csv format
- Prepare Data: `Xtrain, Ytrain, Xtest` and `Ytest` with time lag 4
- Define the NN architecture:
  - `model = Sequential()`
  - `model.add(Dense(30, input_dim=time_lag, activation='relu'))`
  - `model.add(Dense(1))`
- This defines a network with 30 neurons on the hidden layer
- Test on `Lecture10_Example01.py`

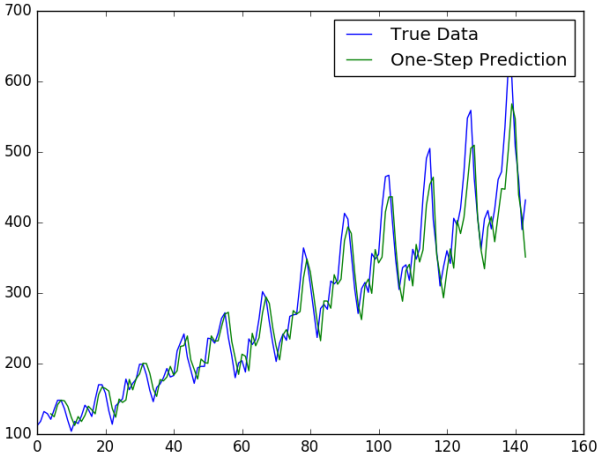
The example: Forecasting



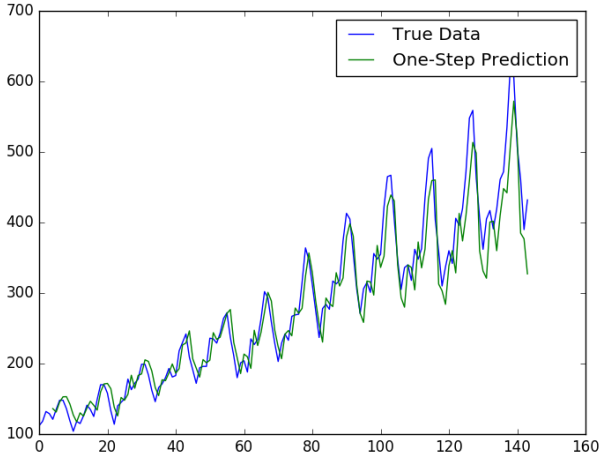
Hidden = 3



Hidden = 30



Hidden = 100

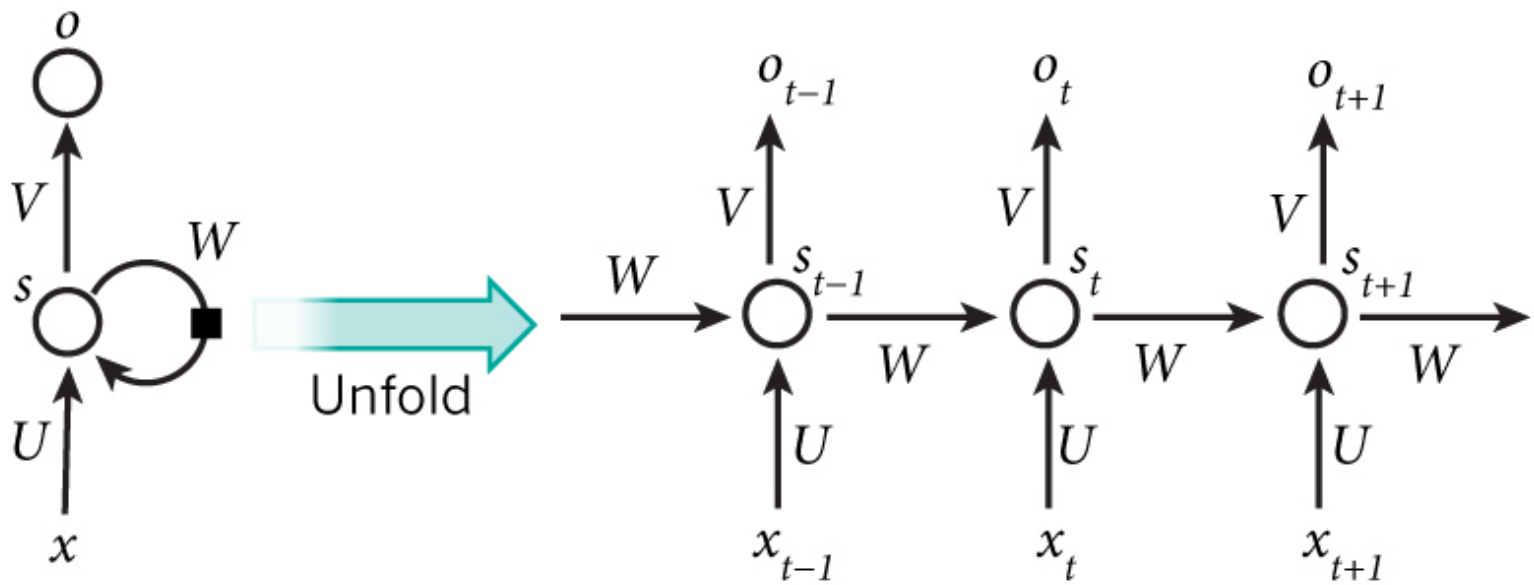


Hidden = 500

What are RNNs?

- The ordinary NNs can be trained for time series forecasting, but the dependence information among the time series was totally ingored, resulting in less accurate predictions or forecasts
- The idea behind recurrent NNs (RNNs) is to make use of sequential information
- To predict the next month beer consumption, we better know what are the previous consumptions before even last year
- RNNs are called recurrent because they perform the same task for every amount in a time series, with the prediction being dependent of the previous computations
- ARIMA can be "regarded" as kind of RNNs without considering nonlinear information or nonlinear relations.

A Simple RNN



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: *Nature*

Read x, x_{t-1}, x_t and x_{t+1} as our time series notation y, y_{t-1}, y_t and y_{t+1} respectively.

Notations in RNNs

- y_t is the input at time step t. It could be a vector or a scalar. For our time series, it should be a scalar
- s_t is the hidden state at time step t. This is the information to be memorised by the networks and passed on to the next time step.
- In most case, it is a vector state for keeping as much information as possible. Of course, you can set it as a scalar state in which case the accuracy could be compromised. It is computed as a recurrent relation

$$s_t = f(Uy_t + Ws_{t-1})$$

where f is the chosen activation function such as sigmoid, tanh or the rectified linear unit function. Usually $s_{-1} = 0$

Recall: Exponential Smoothing

- Simple Exponential Smoothing (SES): We update the level according to

$$l_t = \alpha y_t + (1-\alpha) l_{t-1}, \quad \text{and} \quad \widehat{y}_{t+1} = l_t$$

- $s_t = l_t$ with $U = \alpha$ and $W = (1-\alpha)$ and $f(z) = z$.

Recall: Exponential Smoothing

- Simple Exponential Smoothing (SES): We update the level according to

$$l_t = \alpha y_t + (1-\alpha) l_{t-1}, \quad \text{and} \quad \widehat{y}_{t+1} = l_t$$

- $s_t = l_t$ with $U = \alpha$ and $W = (1-\alpha)$ and $f(z) = z$.
- Holt's Linear Smoothing (Trend Corrected Exponential Smoothing)

$$\begin{aligned} l_t &= \alpha y_t + (1-\alpha) l_{t-1} + (1-\alpha) b_{t-1} \\ b_t &= \gamma\alpha y_t - \alpha\gamma l_{t-1} + (1-\gamma) b_{t-1} \end{aligned}$$

or

$$s_t = \begin{bmatrix} l_t \\ b_t \end{bmatrix} = \begin{bmatrix} \alpha \\ \alpha\gamma \end{bmatrix} y_t + \begin{bmatrix} (1-\alpha) & (1-\alpha) \\ -\alpha\gamma & (1-\gamma) \end{bmatrix} \begin{bmatrix} l_{t-1} \\ b_{t-1} \end{bmatrix} = U y_t + W s_{t-1}$$

with $s_t = \begin{bmatrix} l_t \\ b_t \end{bmatrix}$, $U \begin{bmatrix} \alpha \\ \alpha\gamma \end{bmatrix}$, $W = \begin{bmatrix} (1-\alpha) & (1-\alpha) \\ -\alpha\gamma & (1-\gamma) \end{bmatrix}$ and $f(z) = z$.

Notations in RNNs

- o_t is the output at step t. Depending on tasks, it could be a vector or a scalar. We set it as a scalar for time series. In our case, it may be computed as

$$o_t = Vs_t$$

  as a normal linear mapping.
  - For SES, we have $V = 1$.
  - For Holt's linear smoothing, we have

$$\widehat{y}_{t+1} = l_t + b_t = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} l_t \\ b_t \end{bmatrix} = Vs_t$$

- It seems that the output o_t at the time step t is calculated solely based on the memory s_t at time t, but it is implicitly determined by all the past values in the time series.

The Structure

- The overall network looks like a forward neural network along the time, however the use of information is recurrently.
- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W above) across all nodes
- The total of number of parameters to be learned has greatly been reduced comparing with the traditional deep neural networks
- The previous diagram has outputs at each time step, but depending on the task this may not be necessary. For time series forecasting, we only care about the last output o_t as the forecast for the time step $t+1$. When we know the new observation y_{t+1}, the error is $e_{t+1} = y_{t+1} - \widehat{y}_{t+1} = y_{t+1} - o_t$.

An Example

- Consider the case in which we use two hidden states for each time step, then we shall have $s_t = \begin{pmatrix} s_{t1} \\ s_{t2} \end{pmatrix}$.
- The input x_t (a scalar) is from a time series. Then the parameters will be in matrix form

$$U = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \quad W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \quad V = [v_1, v_2]$$

  Please note the dimension of each parameter matrix.
- The total number of parameters is $2 + 4 + 2$ (without considering bias parameters).
- The training becomes to estimate 8 parameters by minimising e.g. the mean squared errors.

An Example

- Mathematically this defines a model of

$$\begin{pmatrix} s_{t1} \\ s_{t2} \end{pmatrix} = f\left(\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} y_t + \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} s_{(t-1)1} \\ s_{(t-1)2} \end{pmatrix} \right)$$

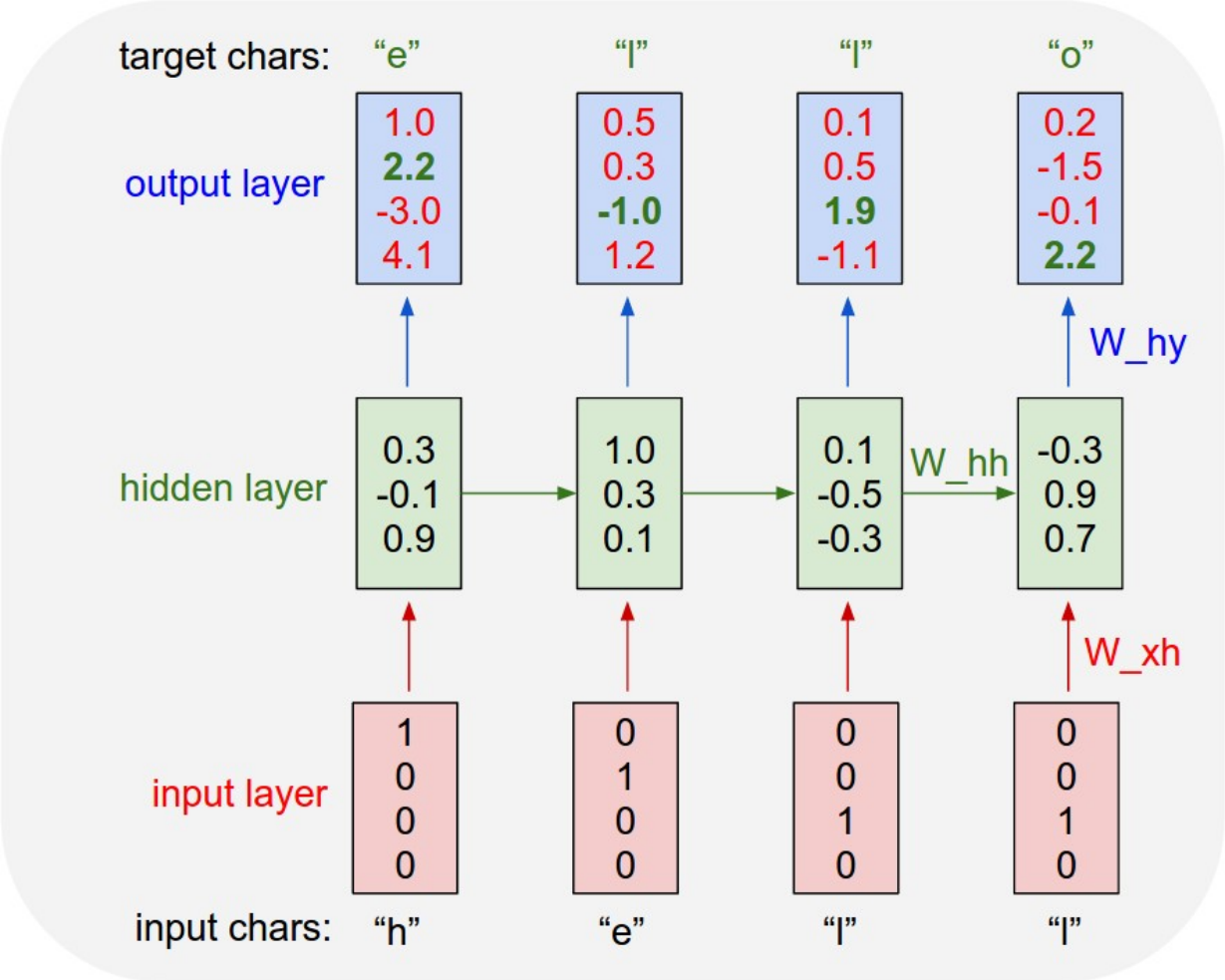
$$o_t = [v_1, v_2] \times \begin{pmatrix} s_{t1} \\ s_{t2} \end{pmatrix};$$

- The activation function f introduces nonlinearity
- The training is to minimise the following objective function with respect to parameters U, W, V

$$L(U, W, V) = \sum_{t=1}^{T-1} (y_{t+1} - o_t)^2 = \sum_{t=1}^{T-1} \left(y_{t+1} - [v_1, v_2] \times \begin{pmatrix} s_{t1} \\ s_{t2} \end{pmatrix} \right)^2$$

$$= \sum_{t=1}^{T-1} \left(y_{t+1} - [v_1, v_2] \times f\left(\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} y_t + \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} s_{(t-1)1} \\ s_{(t-1)2} \end{pmatrix} \right) \right)^2$$

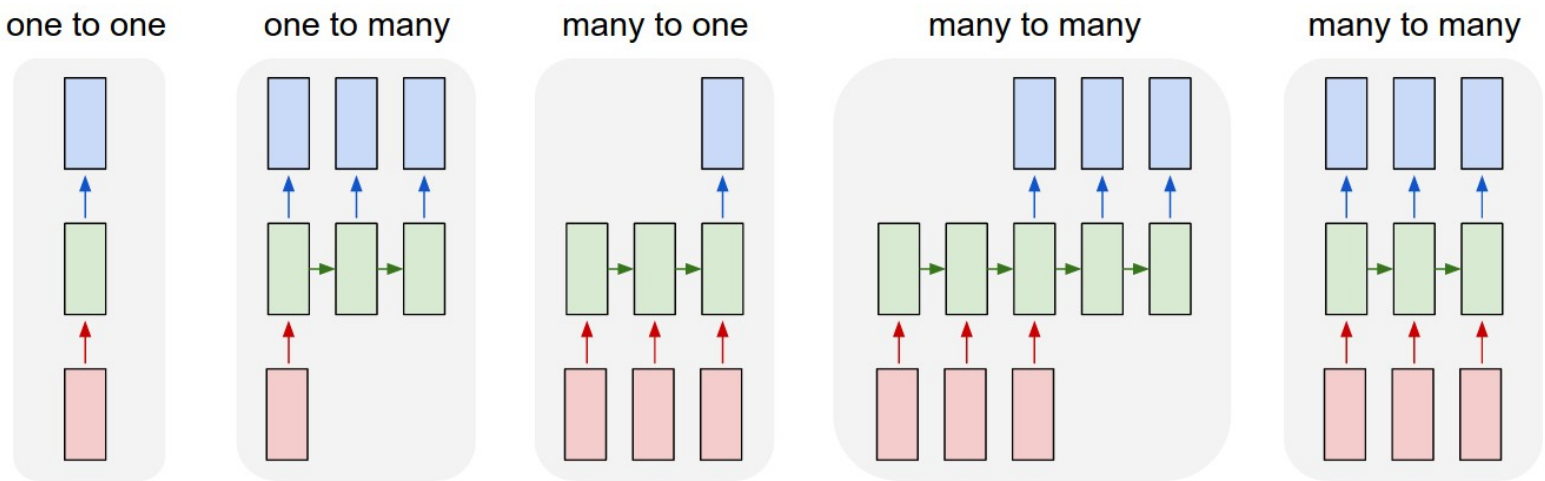
Another Example



Source:

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Basic Architectures



The "many-to-one" is the most suitable architecture for the time series.

Source:

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Successful Application in NLP (Natural Language Processing)

- Given a sequence of words we want to predict the probability of each word given the previous words. You can produce "new" Shakespeare dramas, after training RNN with Shakespeare's
- Google is doing Machine Translation with RNN

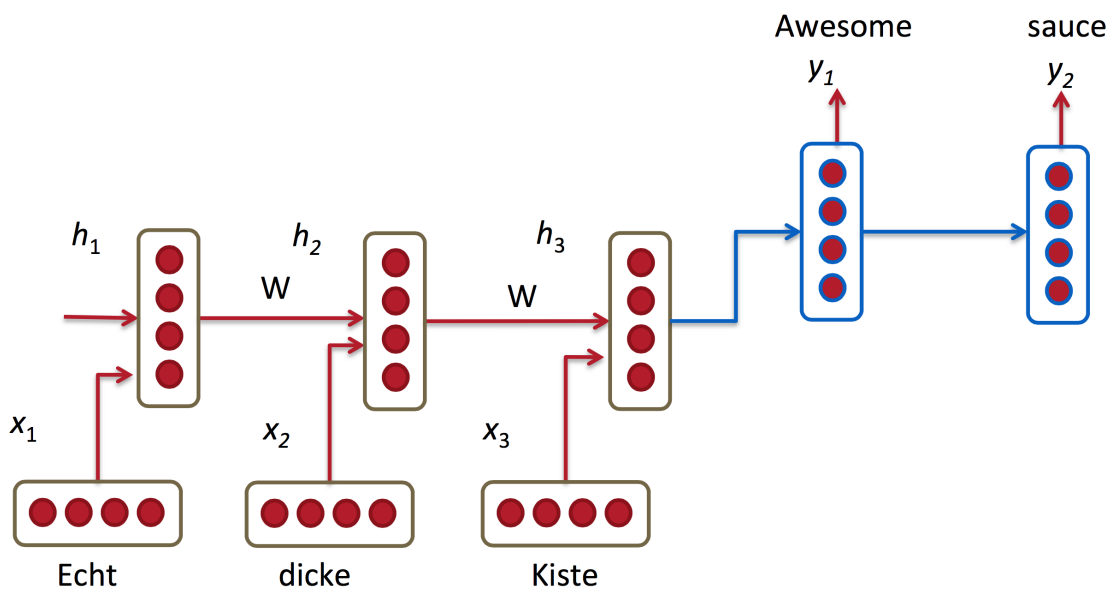


Image Source: ImageSource:http://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Generating Image Description

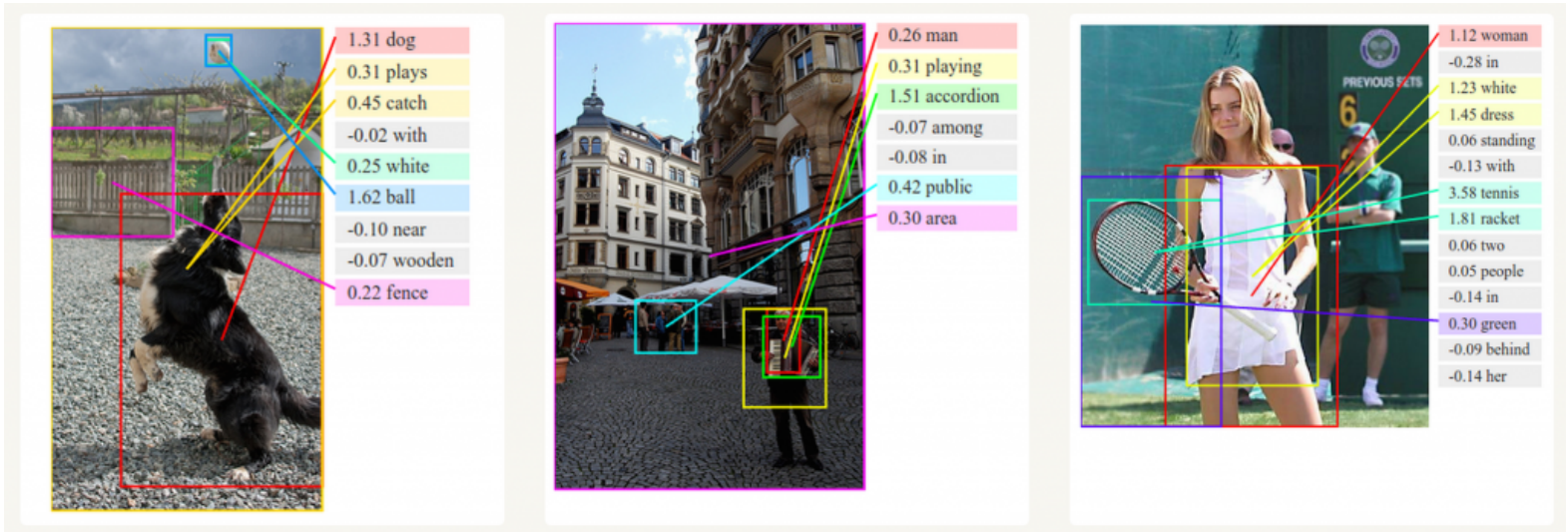


Image Source: `ImageSource:http://cs.stanford.edu/people/karpathy/deepimagesent/`

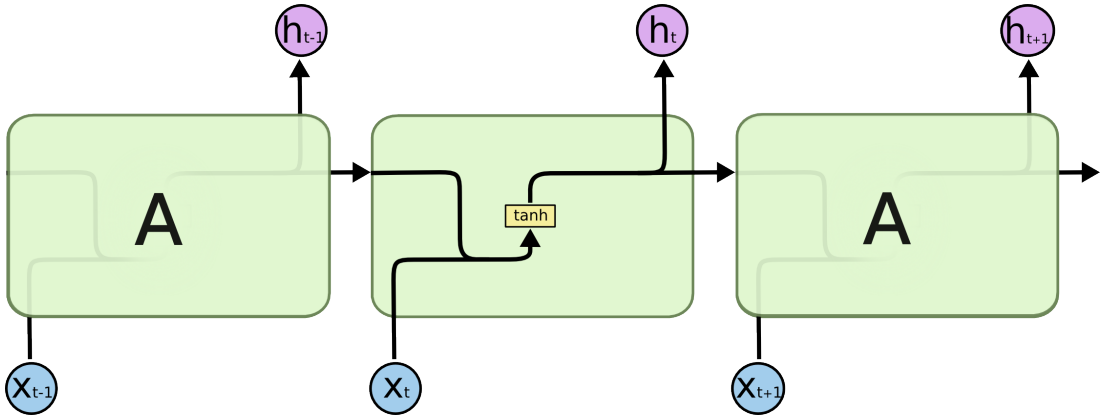
Training RNNs

- Training an RNN is similar to training a traditional Neural Network, by using the so-called backpropagation algorithm, but in a revised way
- The architecture shows that the parameters are shared in all time steps, thus the gradient should be computed by taking into account all the previous time steps.
- For example, the gradient at the time step $t = 4$ should be backpropagated 3 steps to calculate other gradients at outputs for the time steps 3, 2 and 1. In many case, the calculation involves recurrent multiplication of parameter matrices.
- This is called Backpropagation Through Time (BPTT)

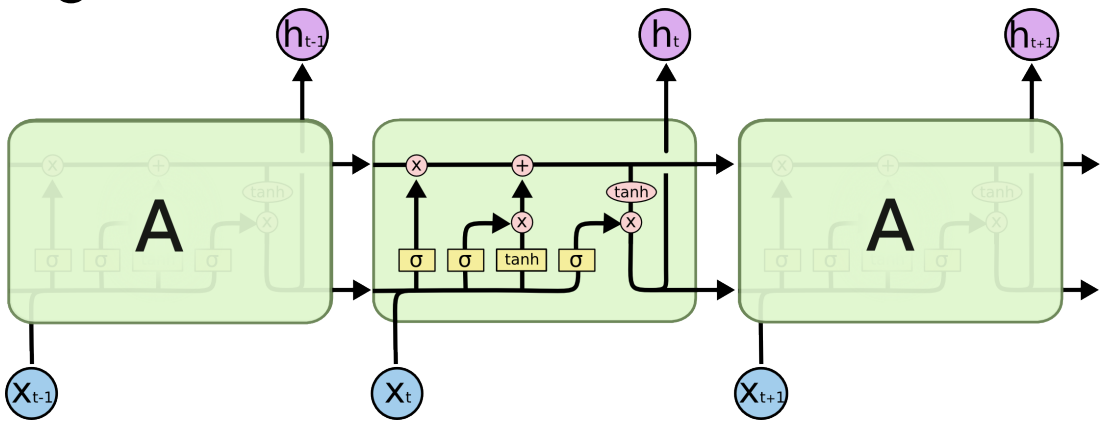
The Problem of Long-Term Dependencies

- There exist great difficulties to train an RNN with BPTT to learn long-term dependencies (e.g. dependencies between steps that are far apart)
- The reason is due to what is called vanishing/exploding gradient problem discovered in the time of commencing RNNs
- In BPTT, the gradient information is passed on back accordingly multiplication of networks parameters U and V
- Check a simple example to see what will happen. $0.5^n \to 0$; while $1.0015^n \to \infty$ when n grows
- There exists some machinery to deal with these problems, and certain types of RNNs were specifically designed to get around them.
- The most successful is the so-called Long Short Term Memory (LSTM) networks

LSTM Networks



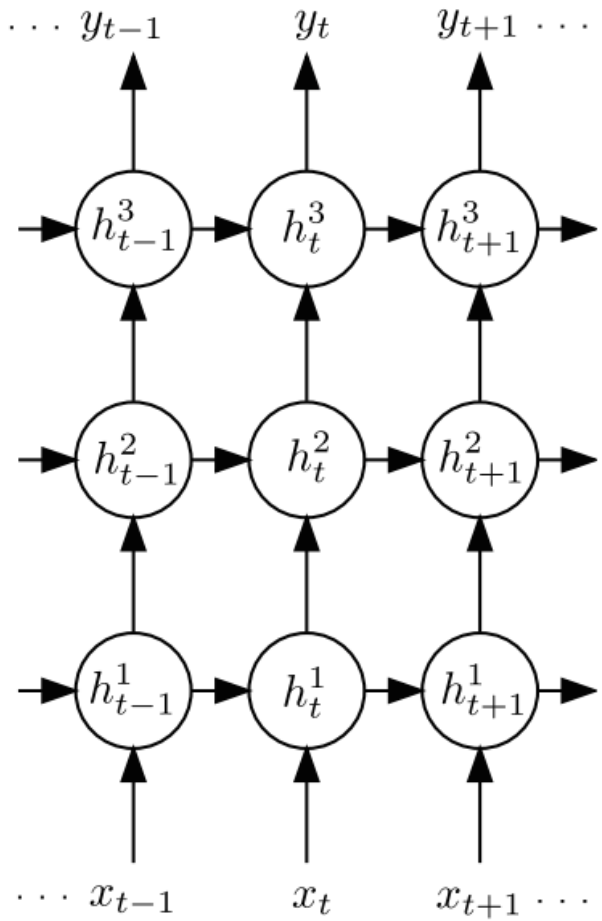
The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

Image Source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Deep Recurrent Neural Networks



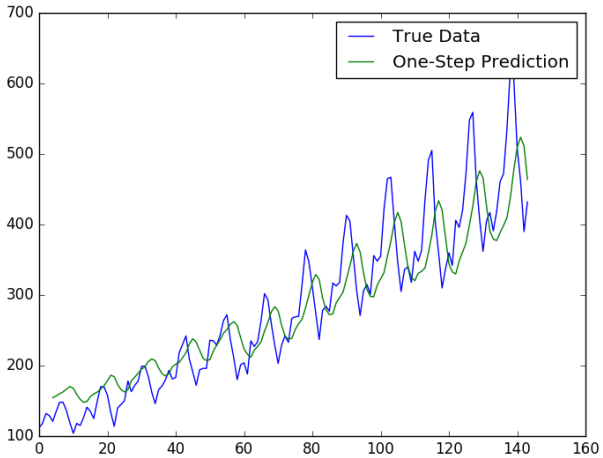
An example

- Dataset: International Airport Arriving Passengers: in csv format
- Prepare Data: `Xtrain, Ytrain, Xtest` and `Ytest` as we have done for traditional neural networks
- Define the RNN architecture:
  - We use one layer of LSTM. For time series, both the input and output dimension is 1.
  - Choose the size of time-window $k = 4$: This should be determined by exploring ACF or PACF
  - Choose the dimension of hidden states $d = 100$
  - ```
    model = Sequential()
    model.add(LSTM(input_dim=1, output_dim=100,
    return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(output_dim=1)
    model.add(Activation("linear"))
    ```
    ```

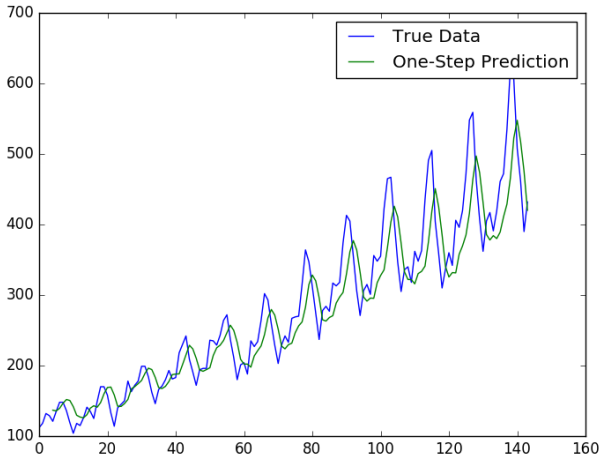
An example: Training

- One statement to train the model
  `model.fit(Xtrain, Ytrain, batch_size=100, nb_epoch=10, validation_split=0.05)`
  - `batch_size` defines the batch size in stochastic optimisation process. You may use default value
  - `nb_epoch` defines the number of epoches of optimisation. Start with a smaller value for testing
  - `validation_split=0.05` means in the training process, about 5% data from `Xtrain` will be used for validation
- Please read the program `Lecture10_Example02.py`

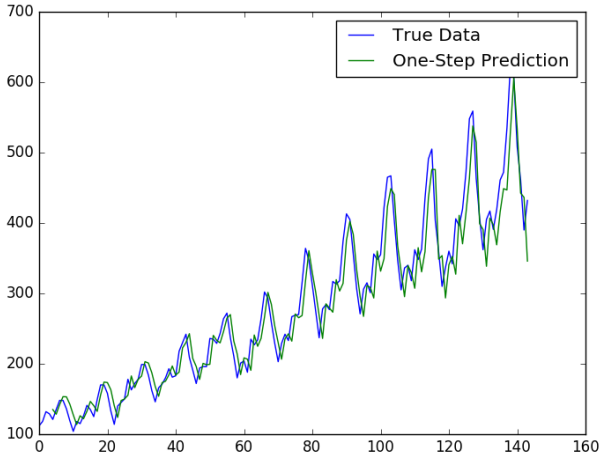
An example: Forecasting



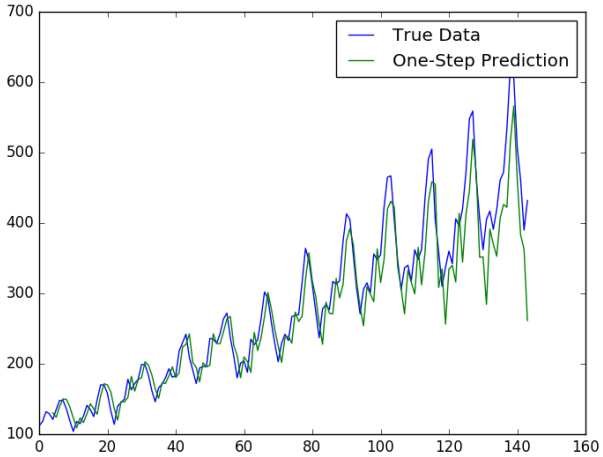
Epoch = 1



Epoch = 20



Epoch = 100



Epoch = 200

An example: Share Price

- Please read the program `Lecture10_Example03.py` where we use two layers of LSTMs