Time Series Forecasting using Recurrent Neural Network

Recurrent Neural Network

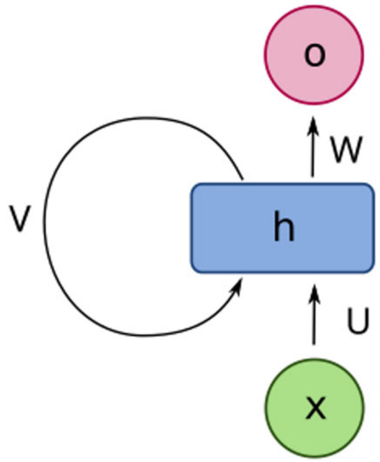
Recurrent Neural Network (RNN)

- a deep learning model designed for sequential data
  - time series analysis/prediction (e.g., weather, stock prices)
  - NLP (e.g., machine translation, sentiment analysis)
  - speech recognition (e.g., generate text from spoken word)
  - image captioning (e.g., generating text descriptions for images)
- make sequential predictions or conclusions based on sequence of inputs

Basic RNN architecture

Basic RNN unit contains/uses internal memory

- known as the hidden states, to remember past inputs
- allowing outputs to depend on previous elements/states in the sequence



Source: Wikipedia

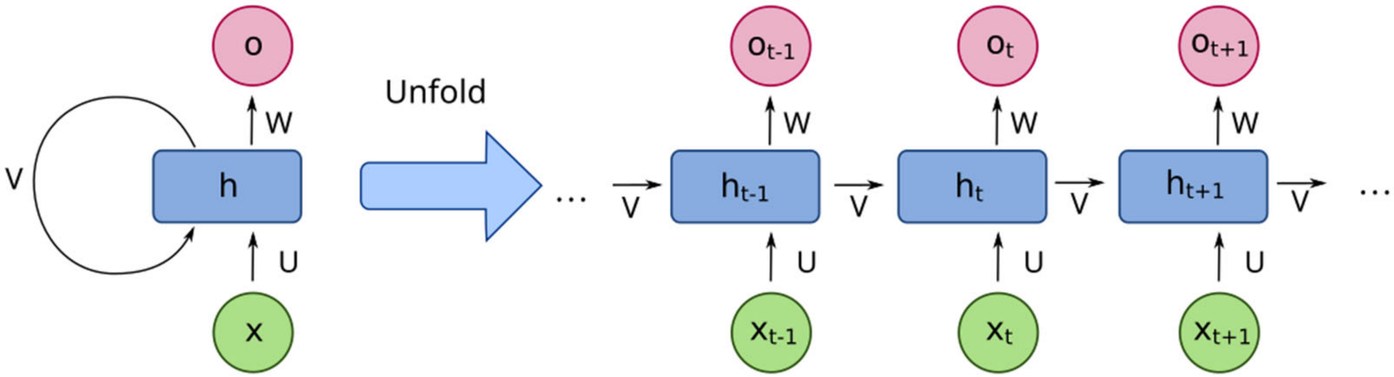
Basic RNN – Time Steps

RNN model can be "unfolded" or "unrolled" over time

- showing the number of <u>time steps</u> used to do the prediction

In the unfolded view

- each time step is represented as a separate, but <u>identical</u> (hidden) cell in a feedforward configuration

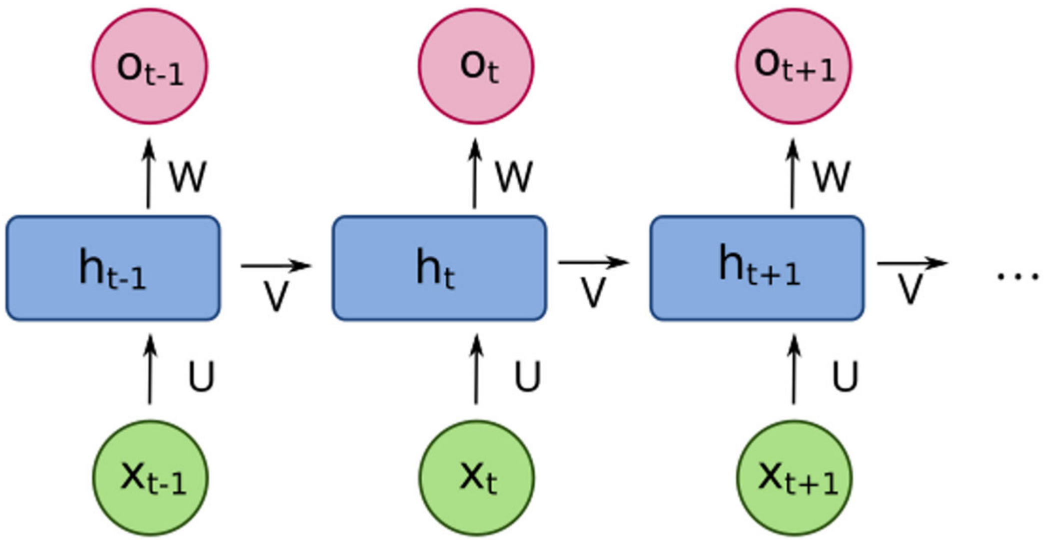


Source: Wikipedia

Time Series for Basic RNN

There are 2 inputs to each hidden cell

- x is the time series data applied to the RNN
- v is the computed value from previous state(s) - generated through the previous hidden cell

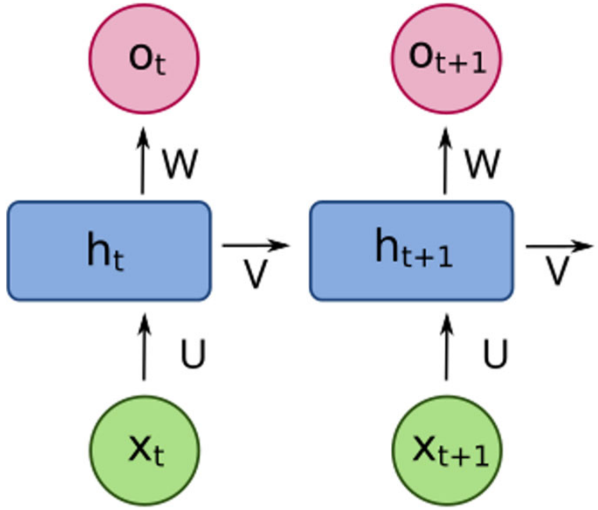


Time Series for RNN - Features

x is the time series data applied to the RNN

- each can be of single value, or a group of values for each timestep
  - depending on the features contain in x

  Examples of features:
  - Share Price Forecasting:
    Opening, Closing, Max/Min prices
  - Weather Forecasting:
    Temperature, humidity, wind speed



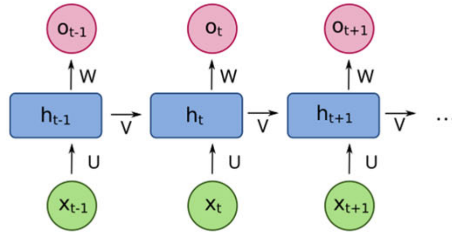
X_1 [f1, f2, f3]
X_2[f1, f2, f3]
X_3 [f1, f2, f3]
X_4 [f1, f2, f3]

X_2 [f1, f2, f3]
X_3[f1, f2, f3]
X_4 [f1, f2, f3]
X_5 [f1, f2, f3]

x can also applied as a batch of data

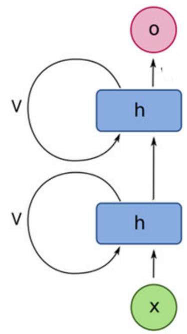
- consists of sequence of the data

Time Series for RNN – Hidden Layer and Cell



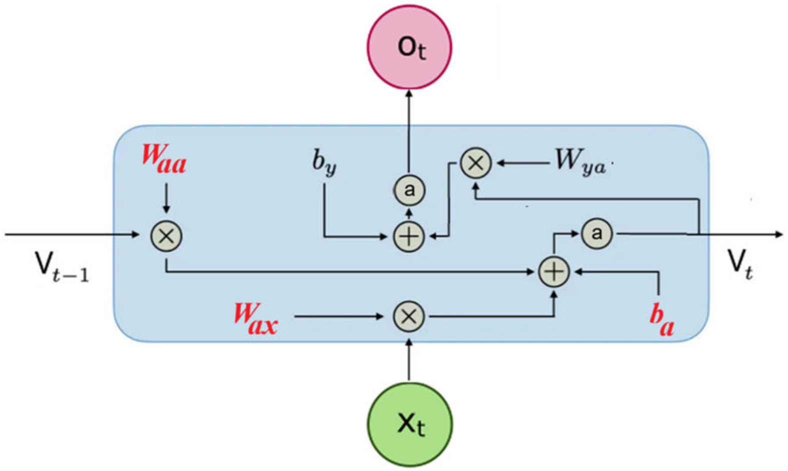
The hidden layer refers to the (horizontal) hidden cells

- number of hidden cells in a layer corresponds to the time steps
- we can also have multiple hidden layers per timestep



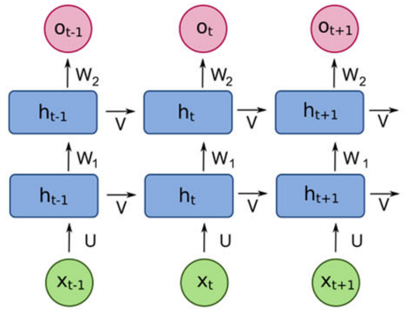
Each hidden cell consists of 'unit' that computes the value to be output to the next timestep V_t (and the current timestep O_t)

- $V_t = F_{Act1}(V_{t-1}*W_{aa}+X_t*W_{ax})+b_a)$
- $O_t = F_{Act2}(V_t*W_{ya}+b_y)$



There can be multiple 'unit' within each hidden cell

- which we can set empirically (e.g. 1, 4, 16. 32 etc)

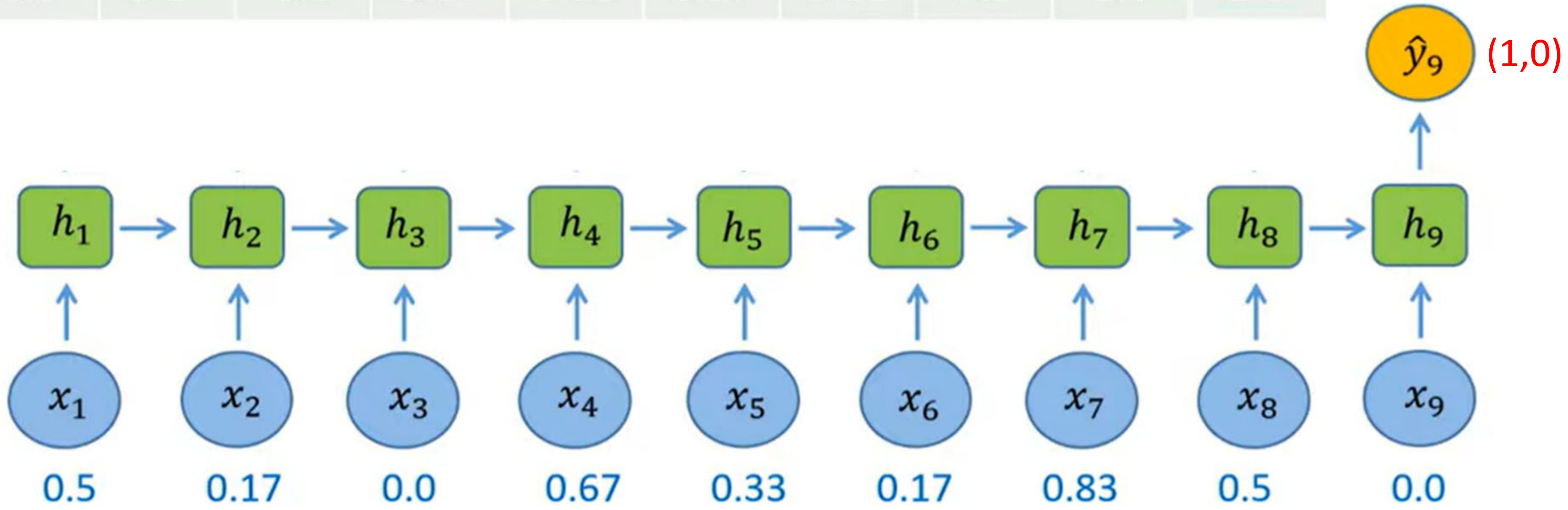


Example: A Simple RNN for Timer Series Prediction

Given a 10 data series

- we can train a simple RNN many-to-one model to predict the next data output

Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
9	7	6	10	8	7	11	9	6	12
0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0



Source: Youtube-TileStats

Example: A Simple RNN for Timer Series Prediction

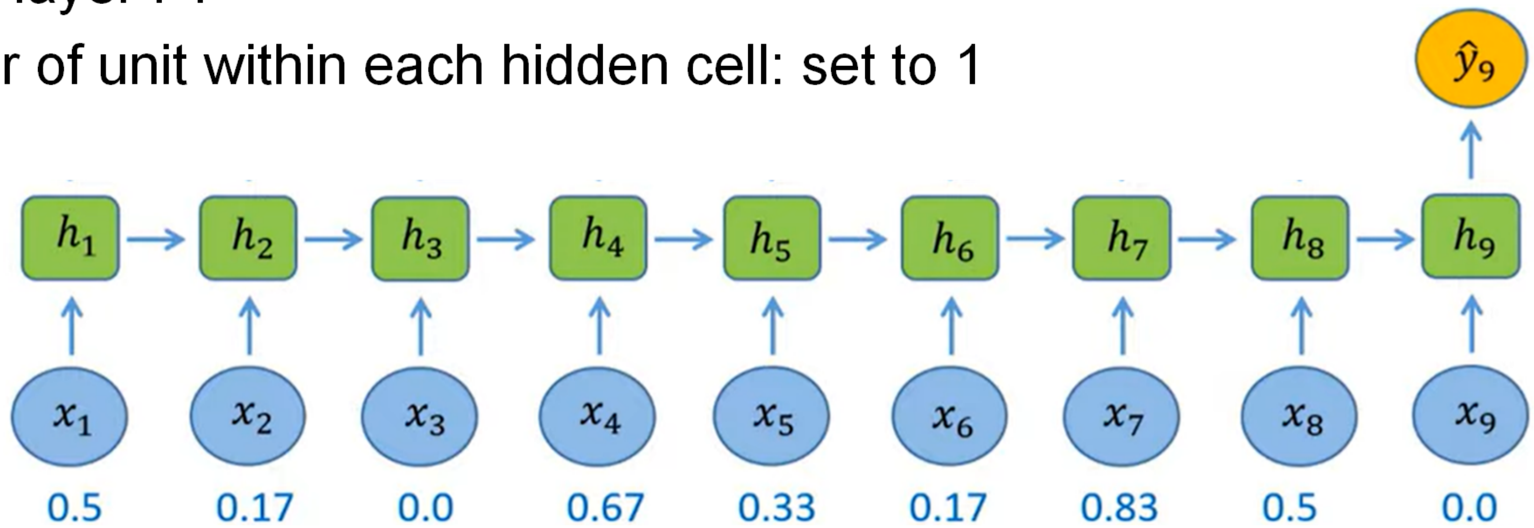
Time step : 9

Sample size : 1 (1 data for each input)

Number of features per data sample: 1

Hidden layer : 1

Number of unit within each hidden cell: set to 1

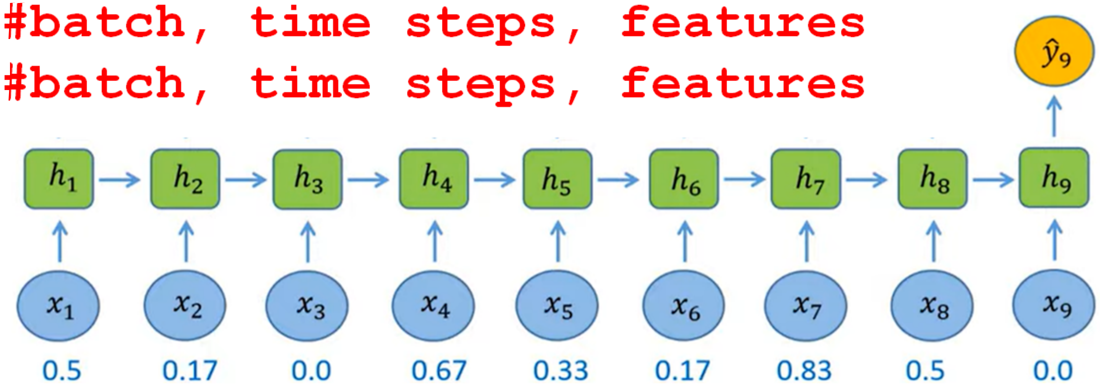


Simple RNN (TF-Keras)

TF-Keras based code snippet

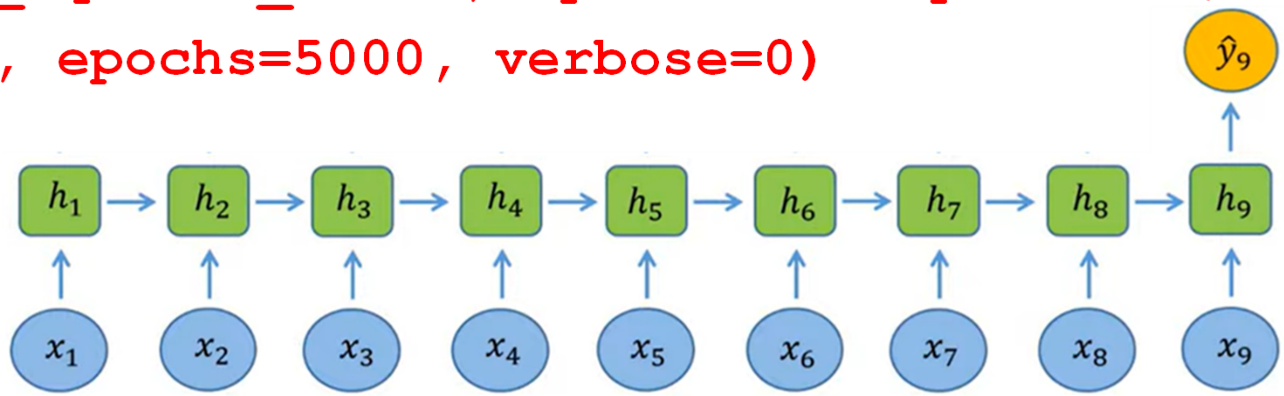
Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
9	7	6	10	8	7	11	9	6	12
0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0

```
# Original time series
X = np.array([9, 7, 6, 10, 8, 7, 11, 9, 6, 12])
Xn = (X-np.min(X))/(np.max(X)-np.min(X)) # Scaled

X = Xn[:-1] # 9 time steps input (remove the last value in series)
y = Xn[9]   # Labelled output (use the last value in series)

#Reshape X and y to suit RNN model input requirement
XM = X.reshape(1, 9, 1) #batch, time steps, features
yM = y.reshape(1, 1, 1) #batch, time steps, features
```



Simple RNN Model Definintion

```
Num_unit = 1 @ number of unit per hidden cell
model = Sequential()
model.add(Input(shape(9,1) # 9 timesteps, 1 feature
model.add(SimpleRNN(Num_unit, activation='tanh'))
model.add(Dense(1,activation='tanh'))

optimizer = Adam(learning_rate=0.01)
model.compile(loss='mean_squared_error, optimizer= optimizer)
history=model.fit(XM, yM, epochs=5000, verbose=0)
```



Minibatch Many-to-one model

Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
9	7	6	10	8	7	11	9	6	12
0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0

Time step : 5

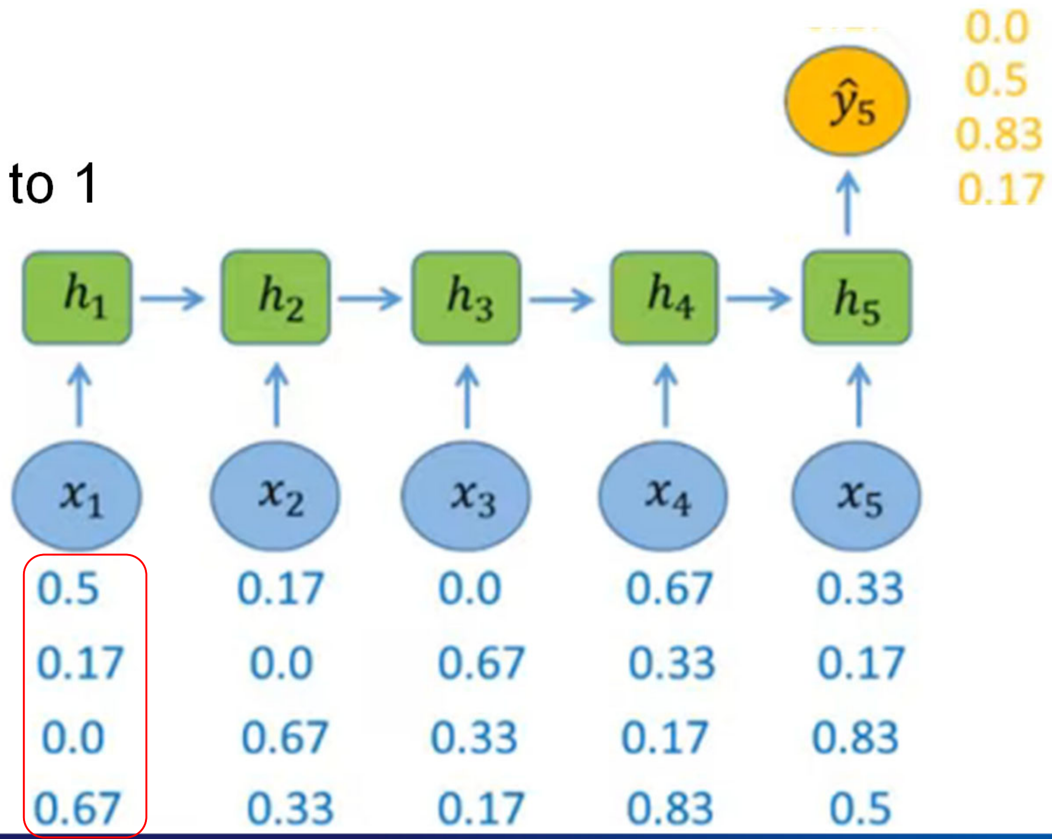
Sample size : 4 (group of 4 data)

Number of features per data sample: 1

Hidden layer : 1

Number of unit within each hidden cell: set to 1

```
for i in range(5, 9):
  X.append(Xn[i-5:i])
  y.append(Xn[i])

#samples, time steps, features
XM = X.reshape(4, 5, 1)
yM = y.reshape(4, 1, 1)
```



Simple RNN Model Definition

```
Num_unit = 1 @ number of unit per hidden cell
model = Sequential()
model.add(Input(shape(5,1) # 5 timesteps, 1 feature
model.add(SimpleRNN(Num_unit, activation='tanh'))
model.add(Dense(1,activation='tanh'))

optimizer = Adam(learning_rate=0.01)
model.compile(loss='mean_squared_error, optimizer= optimizer)

# use batch of 4 -> update parameters after 4 data
history=model.fit(XM, yM, epochs=5000, batch_size=4, verbose=1)

Yp=model.predict(XM, verbose=0) #predict the next output value
```

Many-to-Many RNN Model

Time step : 8

Epoch: 2500