Recurrent Neural Networks

Prof. Gheith Abandah

References:

- *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly), 2017, 978-1-491-96229-9.
- François Chollet, *Deep Learning with Python*, Manning Pub. 2018

Introduction

- YouTube Video: *Deep Learning with Tensorflow - The Recurrent Neural Network Model* from Cognitive Class

https://youtu.be/C0xoB8L8ms0

Outline

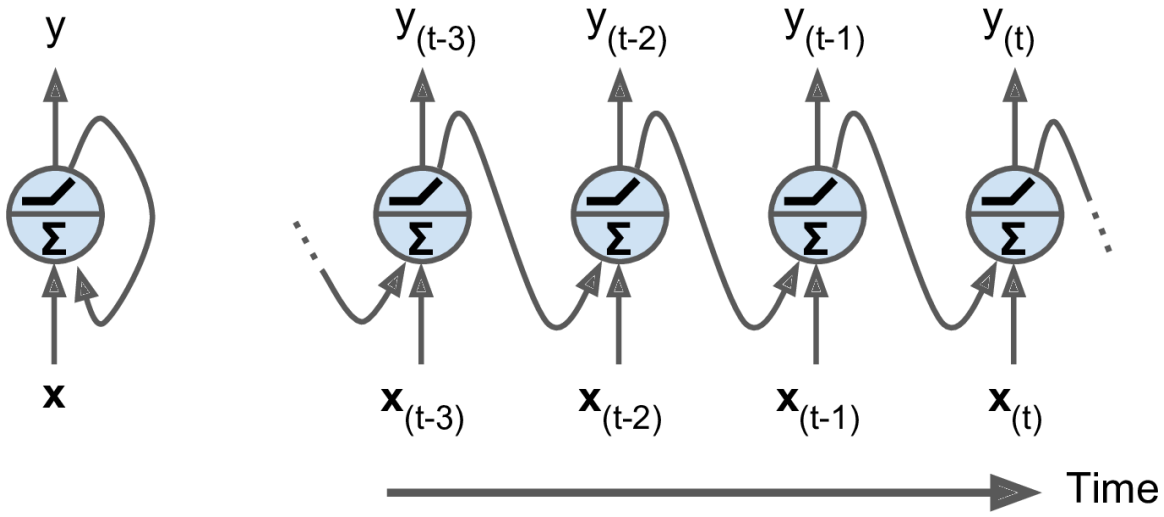
1. Introduction
2. Recurrent neurons
3. Deep RNNs
4. LSTM and GRU cells
5. Input and output sequences lengths
6. Training RNNs
    1. IMDB example
    2. Temperature-forecasting example
7. Exercises

1. Introduction

- *Recurrent neural networks (RNNs)* are used to handle time series data or sequences.
- Applications:
  - Predicting the future (stock prices)
  - Autonomous driving systems (predicting trajectories)
  - Natural language processing (automatic translation, speech-to-text, or sentiment analysis)
  - Creativity (music composition, handwriting, drawing)
  - Image analysis (image captions)

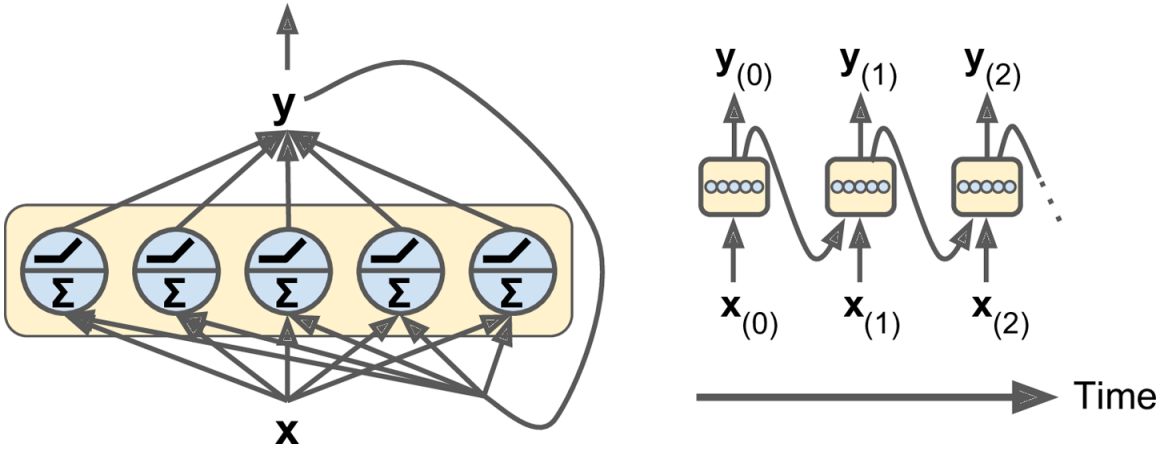
2. Recurrent Neurons

- The figure below shows a *recurrent neuron* (left), unrolled through time (right).



2. Recurrent Neurons

- Multiple recurrent neurons can be used in a layer.



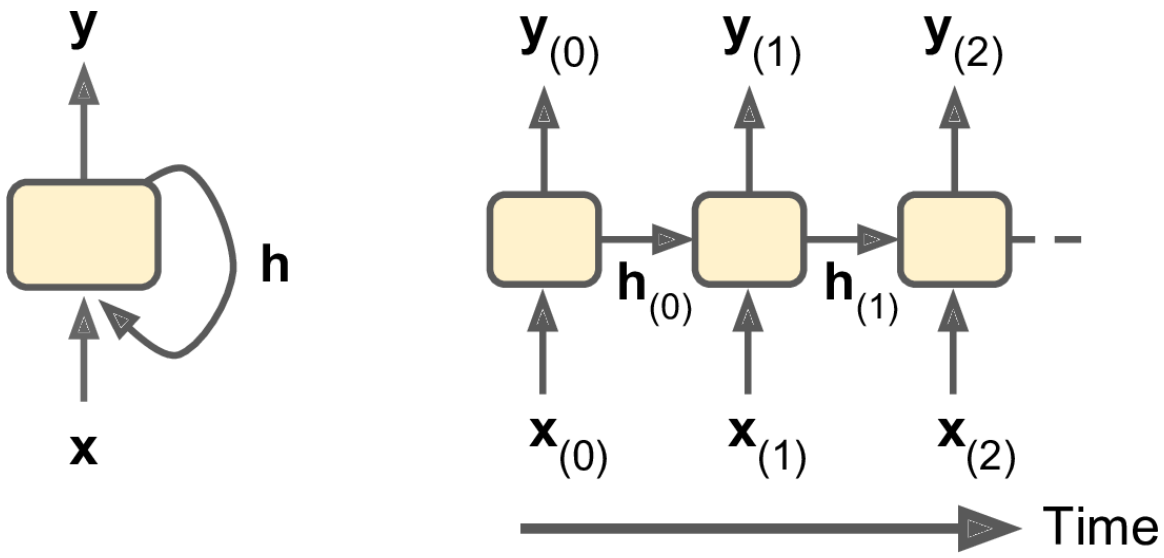
- The output of the layer is:

$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b}\right)$$

Keras Code: `model.add(SimpleRNN(32))`

2. Recurrent Neurons

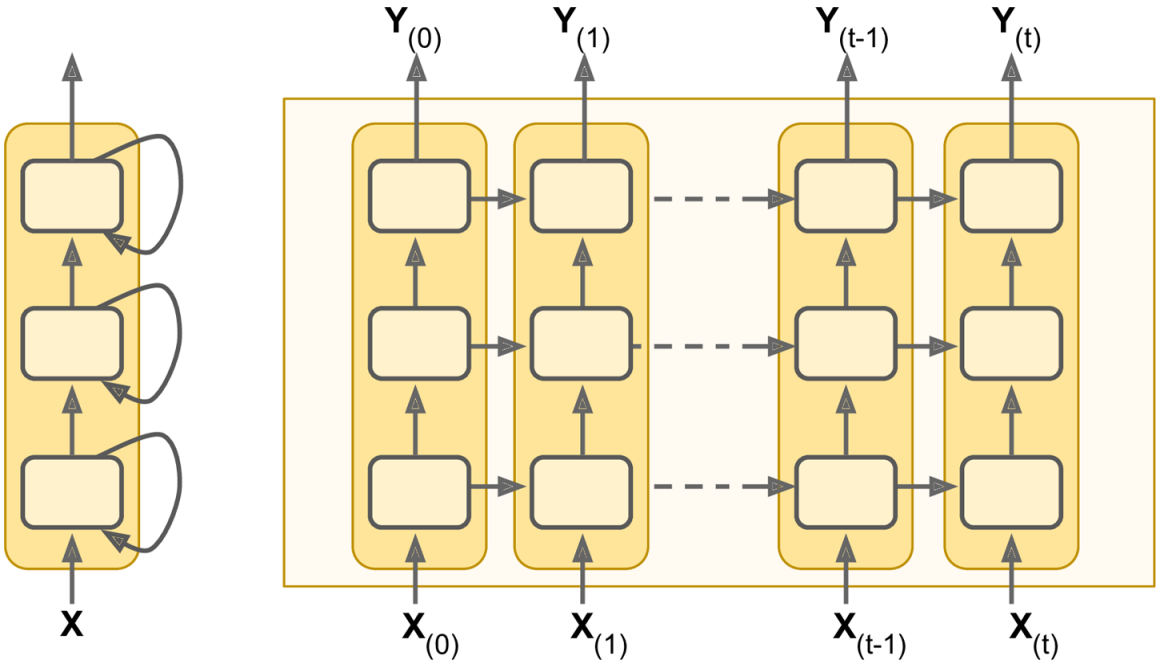
- Recurrent neurons have memory (hold state) and are called *memory cells*.
- The state $\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$, not always ≡ $\mathbf{y}_{(t)}$



Outline

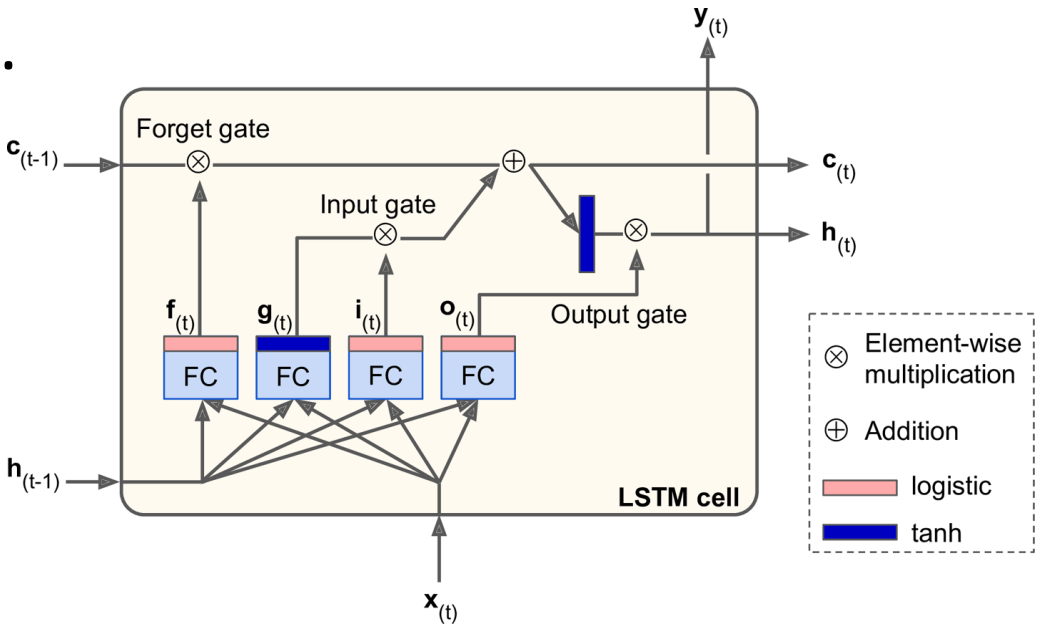
1. Introduction
2. Recurrent neurons
3. Deep RNNs
4. LSTM and GRU cells
5. Input and output sequences lengths
6. Training RNNs
    1. IMDB example
    2. Temperature-forecasting example
7. Exercises

3. Deep RNNs



4. LSTM Cell

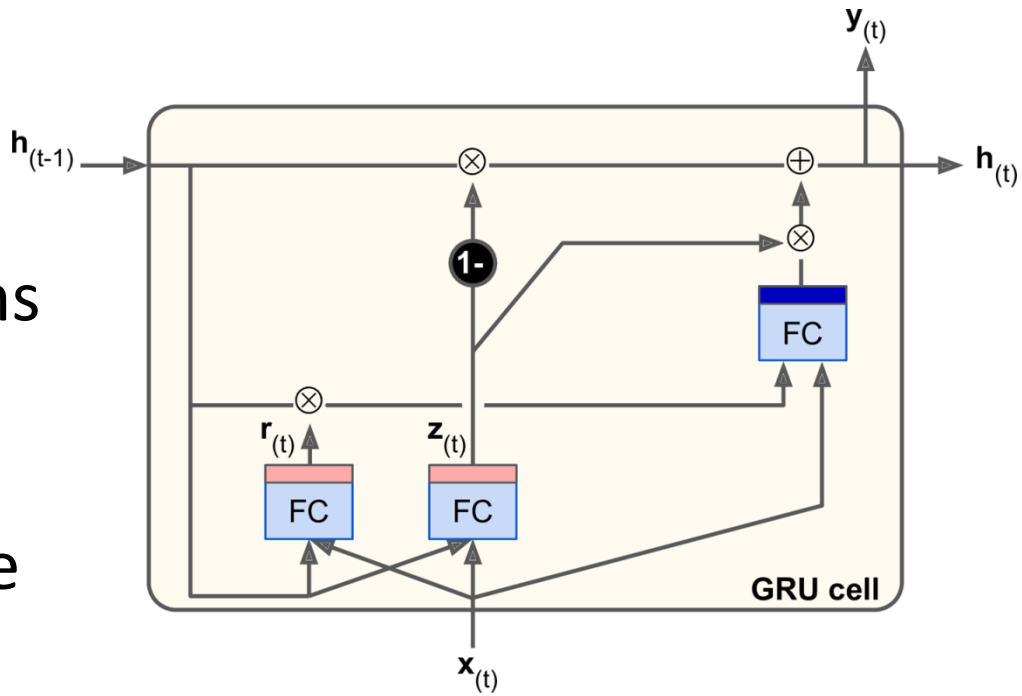
- The *Long Short-Term Memory* (LSTM) cell was proposed in 1997.
- Training converges faster and it detects long-term dependencies in the data.
- $\mathbf{h}_{(t)}$ as the short-term state and $\mathbf{c}_{(t)}$ as the long-term state.



Keras Code: `model.add(LSTM(32))`

4. GRU Cell

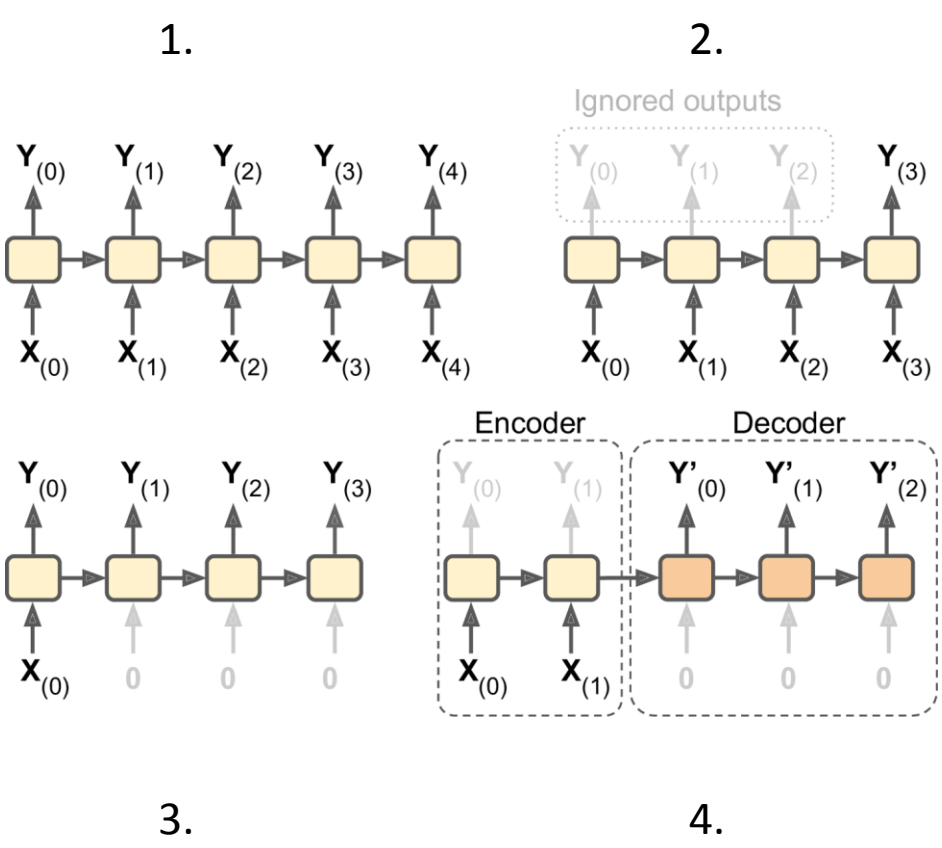
- The *Gated Recurrent Unit* (GRU) cell was proposed in 2014.
- Simplified version of the LSTM cell, performs just as well.
- A single gate controls the forget gate and the input gate.



Keras Code: `model.add(GRU(32))`

5. Input and Output Sequences Lengths

1. **Seq to seq**: For predicting the future.
2. **Seq to vector**: For analysis, e.g., sentiment score.
3. **Vector to seq**: For image captioning.
4. **Delayed seq to seq**: For sequence transcription.

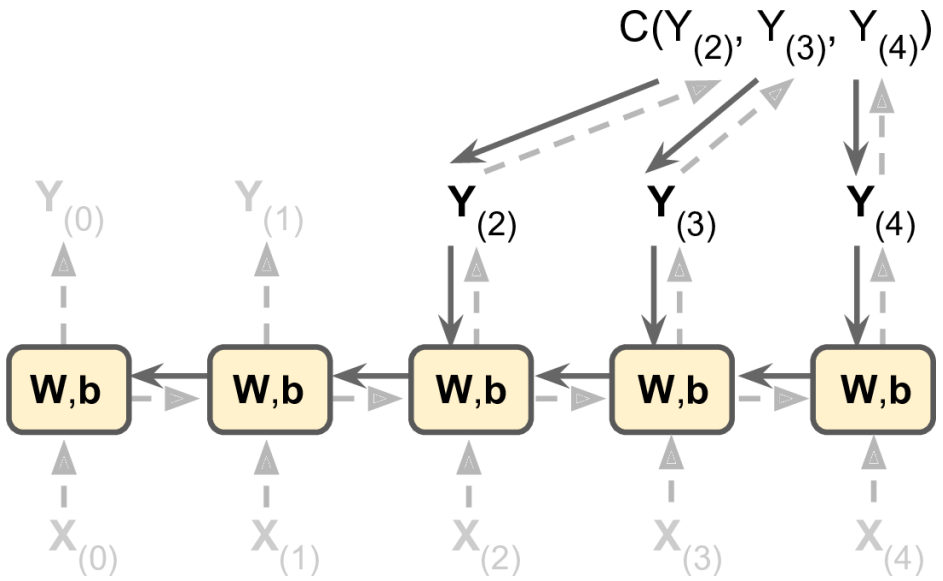


Outline

1. Introduction
2. Recurrent neurons
3. Deep RNNs
4. LSTM and GRU cells
5. Input and output sequences lengths
6. Training RNNs
    1. IMDB example
    2. Temperature-forecasting example
7. Exercises

6. Training RNNs

- Training using strategy called *backpropagation through time* (BPTT).
- Forward pass (dashed)
- Cost function of the not-ignored outputs.
- Cost gradients are propagated backward through the unrolled network.



6.1 IMDB Example

- IMDB Dataset
- A set of 50,000 highly polarized reviews from the Internet Movie Database
- Split into 25,000 reviews for training and 25,000 reviews for testing
- Each set consisting of 50% negative and 50% positive reviews

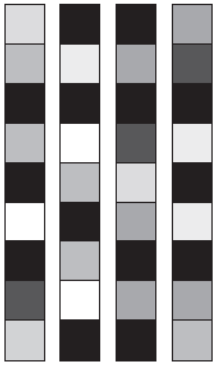
6.1 IMDB Example – Build Model

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN


model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32))
model.add(Dense(1,
    activation='sigmoid'))
```

Embed sparse vectors into dense vectors

One-hot word vectors:
- Sparse
- High-dimensional
- Hardcoded



Word embeddings:
- Dense
- Lower-dimensional
- Learned from data

6.1 IMDB Example

```
>>> Model.summary()

Layer (type) Output Shape Param #
=================================================================
embedding_23 (Embedding)         (None, None, 32)         320000
_________________________________________________________________
simplernn_11 (SimpleRNN)         (None, None, 32)         2080
_________________________________________________________________
dense_1 (Dense)                  (None, 1)                33
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
```

6.1 IMDB Example – Prepare the data

```
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 10000
maxlen = 500

(input_train, y_train), (input_test, y_test) =
      imdb.load_data(num_words=max_features)
input_train = sequence.pad_sequences(input_train,
      maxlen=maxlen)
input_test = sequence.pad_sequences(input_test,
      maxlen=maxlen)
```

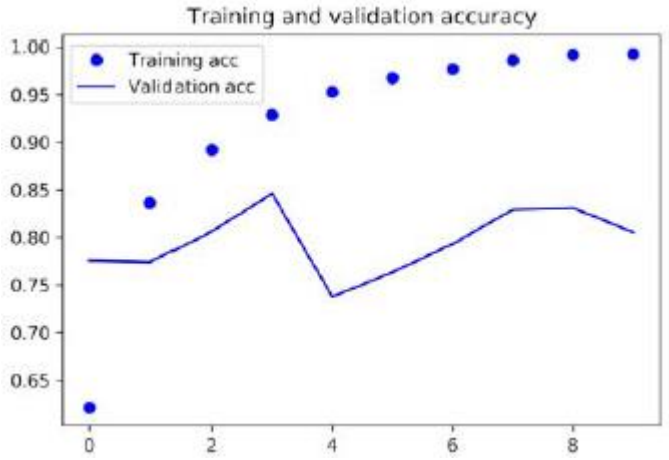
Pads sequences to the same length

6.1 IMDB Example

```
# Compile and train

model.compile(optimizer='rmsprop',
        loss='binary_crossentropy', metrics=['acc'])


model.fit(input_train, y_train, epochs=10,
        batch_size=128, validation_split=0.2)
```



Outline

1. Introduction
2. Recurrent neurons
3. Deep RNNs
4. LSTM and GRU cells
5. Input and output sequences lengths
6. Training RNNs
    1. IMDB example
    2. Temperature-forecasting example
7. Exercises

6.2 Temperature Forecasting

- Weather time series dataset recorded at the Weather Station at the Max Planck Institute for Biogeochemistry in Jena, Germany
- 14 different quantities (such air temperature, atmospheric pressure, humidity, wind direction, and so on) were recorded every 10 minutes
- The example is limited to data from 2009–2016
- Build a model that takes as input some data from the recent past (a few days' worth of data points) and predicts the air temperature 24 hours in the future.

6.2 Get the Data

```
import os

data_dir = '/home/ubuntu/data/'
fname = os.path.join(data_dir,
        'jena_climate_2009_2016.csv')
f = open(fname)
data = f.read()
f.close()
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[1:]
print(header)
print(len(lines))
```

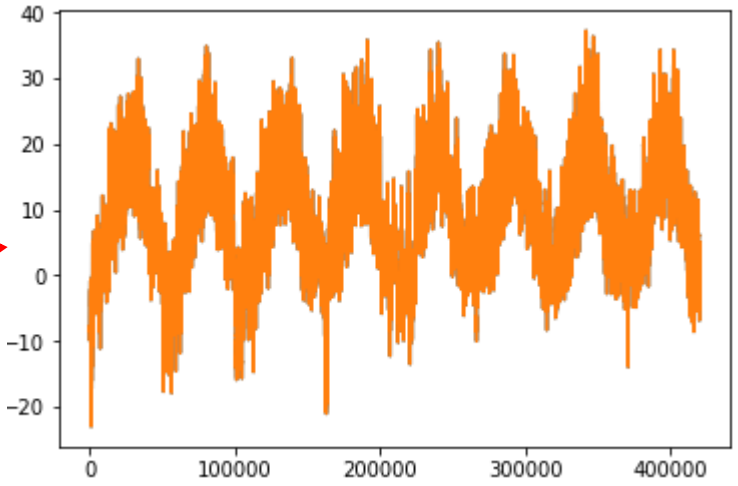
```
["Date Time",
"p (mbar)",
"T (degC)",
"Tpot (K)",
"Tdew (degC)",
"rh (%)",
"VPmax (mbar)",
"VPact (mbar)",
"VPdef (mbar)",
"sh (g/kg)",
"H2OC
(mmol/mol)",
"rho (g/m**3)",
"wv (m/s)",
"max. wv (m/s)",
"wd (deg)"]
420551
```

6.2 Prepare the Data

```
import numpy as np

float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[1:]]
    float_data[i, :] = values
```

Temperature
`float_data[:, 1]`



6.2 Prepare the Data

```
# normalize the data
# 420551 samples: the training from the first 200,000
# timesteps, the validation from the following 100,000,
# and the test from the remainder

mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std
```

6.2 Prepare the Data

- Given data going as far back as lookback time steps (a time step is 10 minutes) and sampled every steps time steps, can we predict the temperature in delay time steps?
  - lookback = 1440, i.e. our observations will go back 10 days.
  - steps = 6, i.e. our observations will be sampled at one data point per hour.
  - delay = 144, i.e. our targets will be 24 hours in the future.

6.2 Prepare the Data

```
def generator(data, lookback, delay, min_index, max_index,
              shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(min_index + lookback, max_index,
                size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)
        samples = np.zeros((len(rows), lookback//step, data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples, targets
```

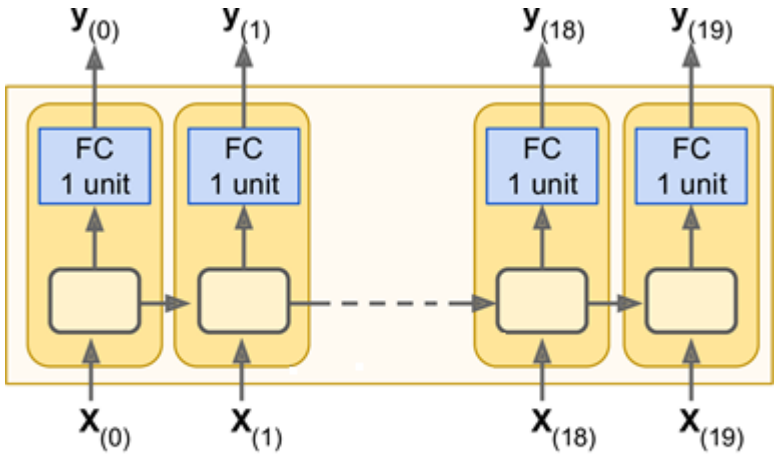
6.2 Prepare the Data

```
lookback = 1440
step = 6
delay = 144
batch_size = 128

train_gen = generator(float_data, lookback=lookback,
      delay=delay, min_index=0, max_index=200000,
      shuffle=True, step=step, batch_size=batch_size)
val_gen = generator(float_data, lookback=lookback,
      delay=delay, min_index=200001, max_index=300000,
      step=step, batch_size=batch_size)
test_gen = generator(float_data, lookback=lookback,
      delay=delay, min_index=300001, max_index=None,
      step=step, batch_size=batch_size)
```

6.2 Build the Model

- Use 32 LSTM cells and one fully-connected output neuron.

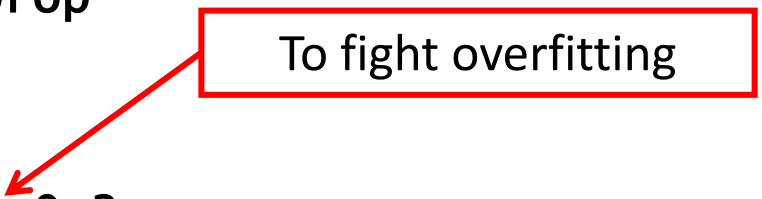


6.1 Build the Model

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.LSTM(32, dropout=0.2,
        recurrent_dropout=0.2, input_shape=(None,
        float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
val_steps = (300000 - 200001 - lookback)
test_steps = (len(float_data) - 300001 - lookback)
history = model.fit_generator(train_gen,
        steps_per_epoch=500, epochs=40,
        validation_data=val_gen,
        validation_steps=val_steps)
```



6.1 Build the Model

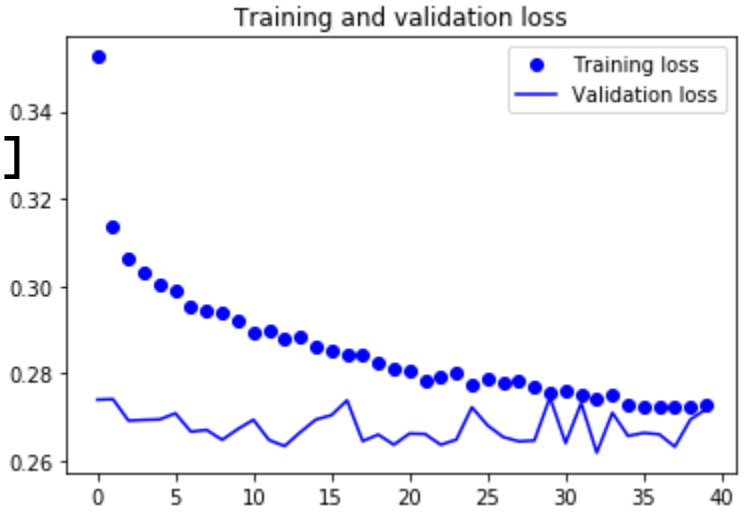
```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Summary

1. Introduction
2. Recurrent neurons
3. Deep RNNs
4. LSTM and GRU cells
5. Input and output sequences lengths
6. Training RNNs
    1. IMDB example
    2. Temperature-forecasting example
7. Exercises

7. Exercises

From Chapter 14, solve exercises:

- 2
- 3
- 8