

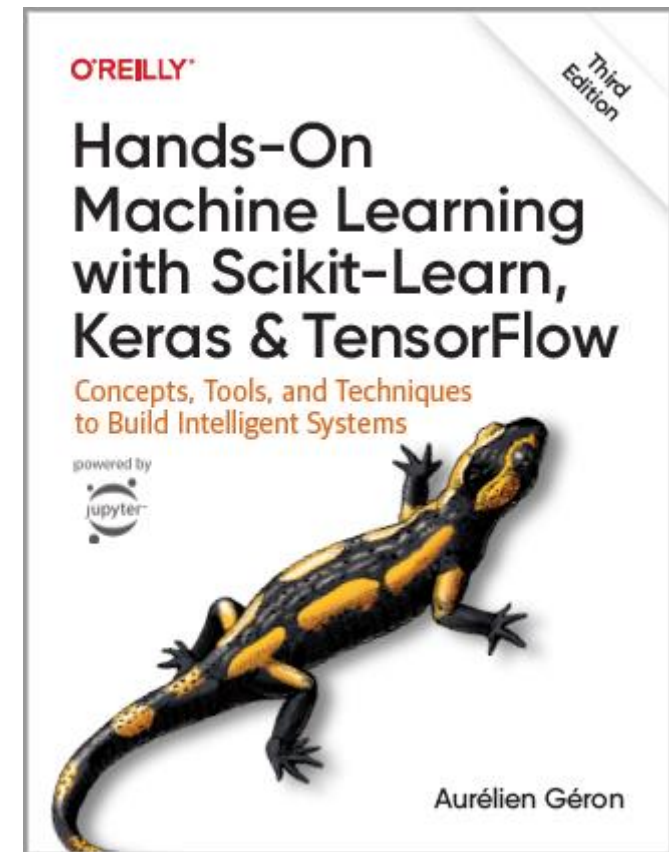
# **Processing Sequences Using RNNs and CNNs**

**Prof. Gheith Abandah**

# Reference 1

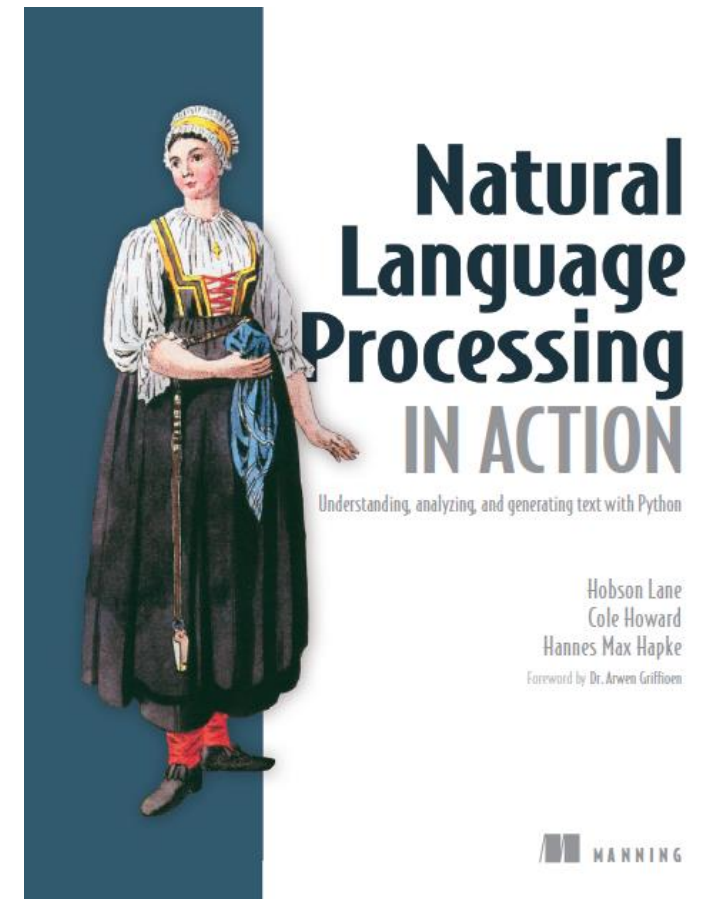
- Chapter 15: **Processing Sequences Using RNNs and CNNs**

- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 3rd Edition, 2022
  - Material: <https://github.com/ageron/handson-ml3>



## Reference 2

- Chapter 7: **Getting words in order with convolutional neural networks (CNNs)**
- H. Lane, C. Howard, and H. Hapke, **Natural Language Processing in Action**: Understanding, analyzing, and generating text with Python, Manning, 2019.



# Outline

1. Introduction
2. Recurrent neurons and layers
3. Forecasting a time series
4. Handling long sequences
5. 1D Convolutional Layers
6. Exercises
7. Summary

# Introduction

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

<https://youtu.be/C0xoB8L8ms0>

# 1. Introduction

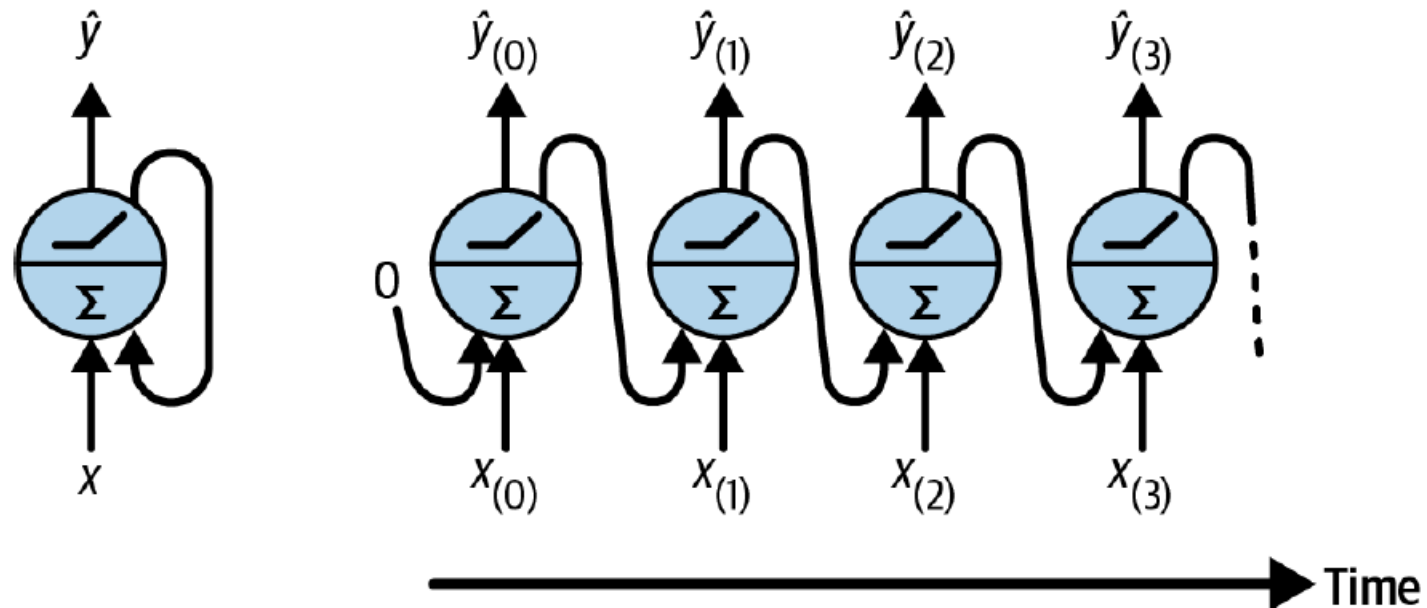
- **Recurrent neural networks (RNNs)** are used to handle time series data or sequences.
- **Applications:**
  - Predicting the future (stock prices)
  - Autonomous driving systems (predicting trajectories)
  - Natural language processing (automatic translation, speech-to-text, or sentiment analysis)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

# Outline

1. Introduction
2. Recurrent neurons and layers
3. Forecasting a time series
4. Handling long sequences
5. 1D Convolutional Layers
6. Exercises
7. Summary

## 2.1 Recurrent Neurons

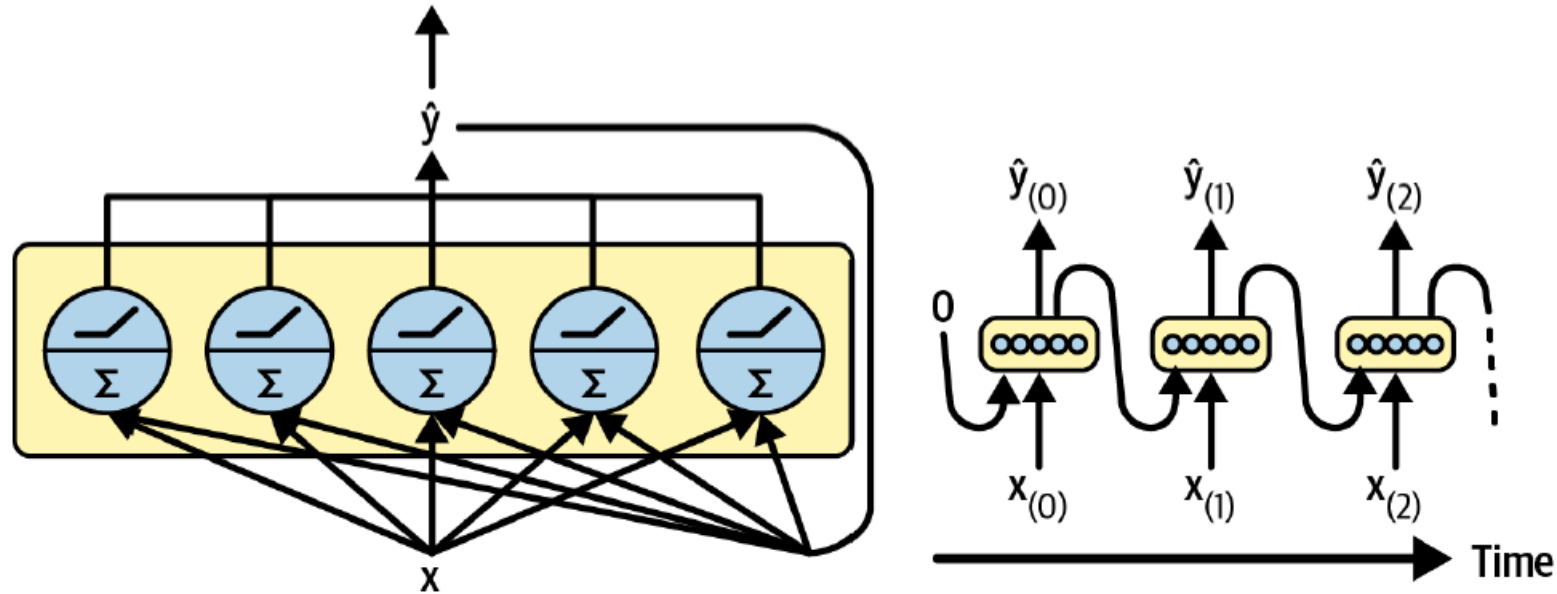
- The figure below shows a **recurrent neuron** (left), unrolled through time (right).





## 2.2 Recurrent Layers

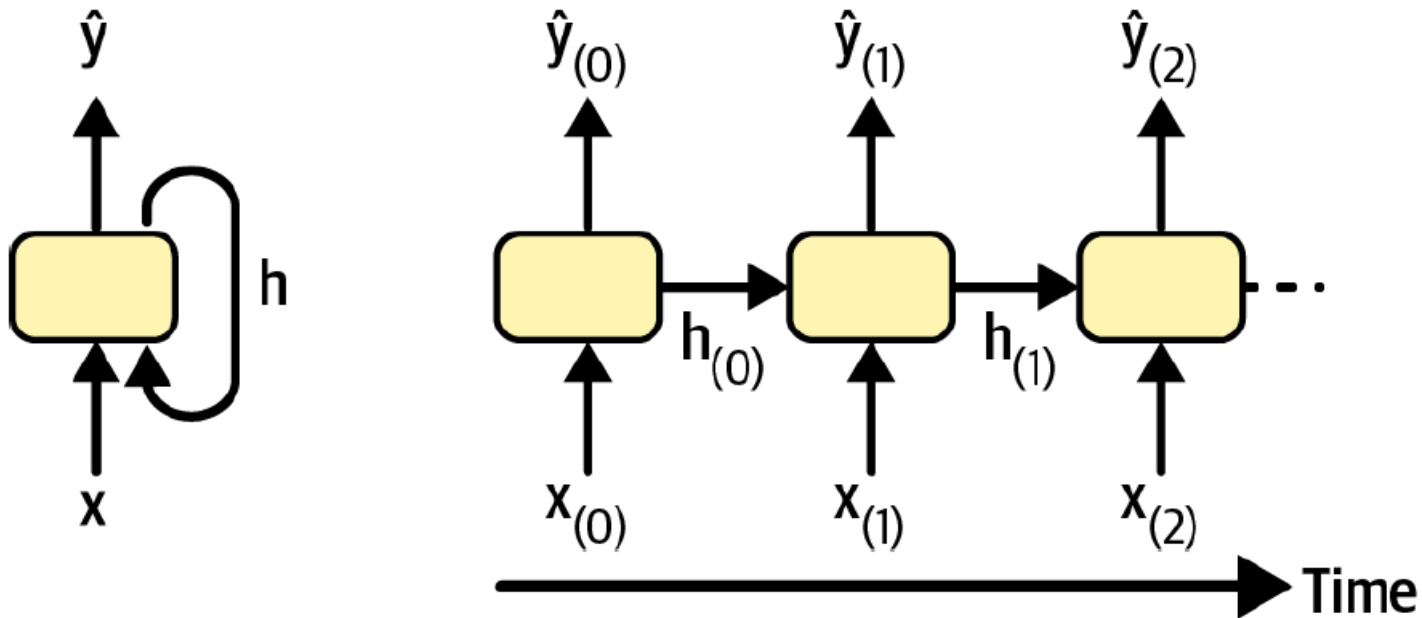
- Multiple recurrent neurons can be used in a **layer**.



- The **output** of the layer is:  $\hat{y}(t) = \varphi(\mathbf{W}_x^\top \mathbf{x}(t) + \mathbf{W}_y^\top \hat{y}(t-1) + \mathbf{b})$

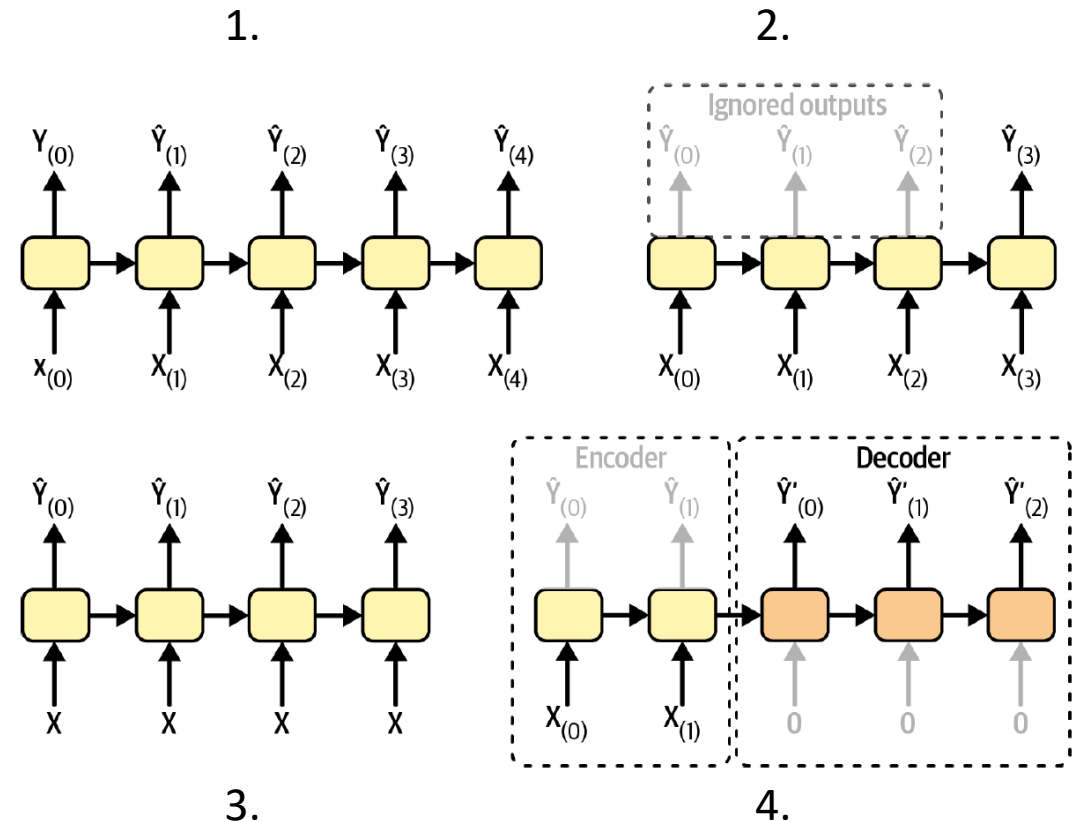
## 2.3 Memory Cells

- 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  $\equiv \mathbf{y}_{(t)}$



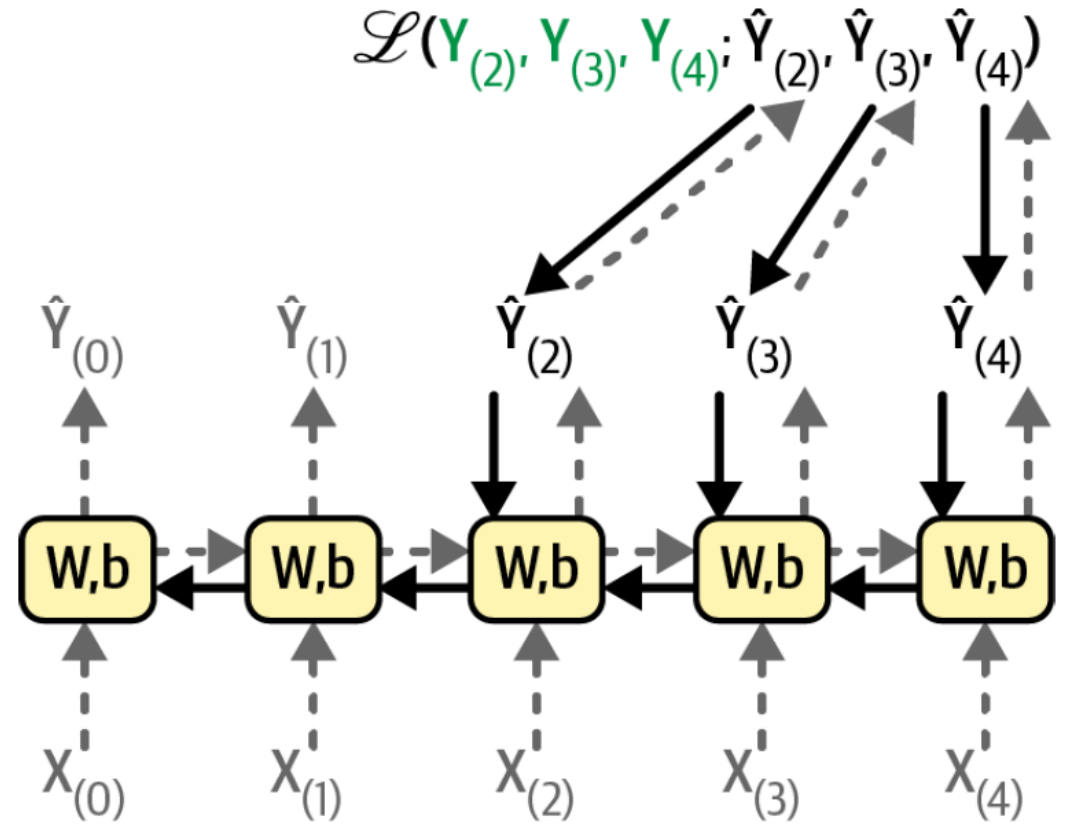
## 2.4 Input and Output Sequences

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



## 2.5 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.

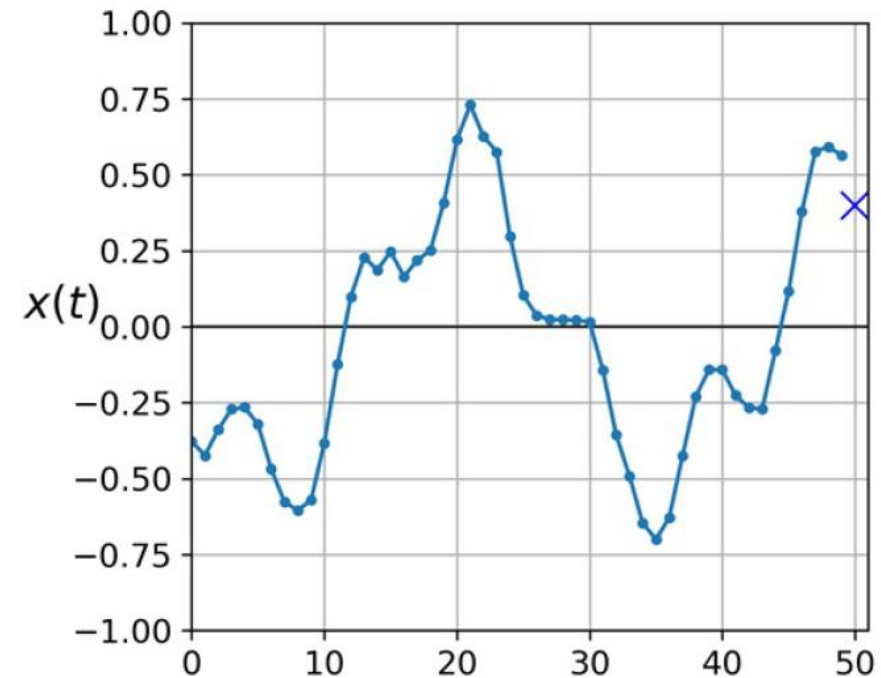


# Outline

1. Introduction
2. Recurrent neurons and layers
3. Forecasting a time series
4. Handling long sequences
5. 1D Convolutional Layers
6. Exercises
7. Summary

# 3. Forecasting a Time Series

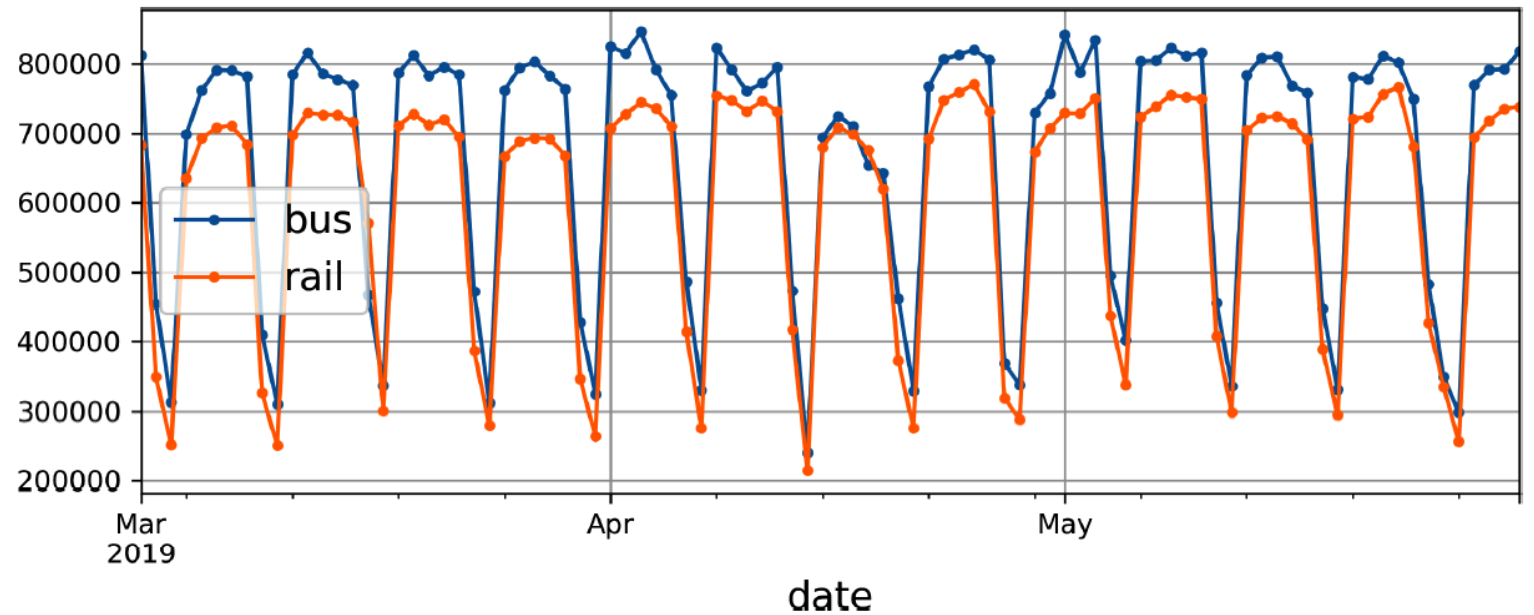
- The data is a sequence of one or more values per **time step**.
  - **Univariate** time series
  - **Multivariate** time series
- **Forecasting**: predicting future values
  - Forecast the **next** value
  - Forecast **N next** values



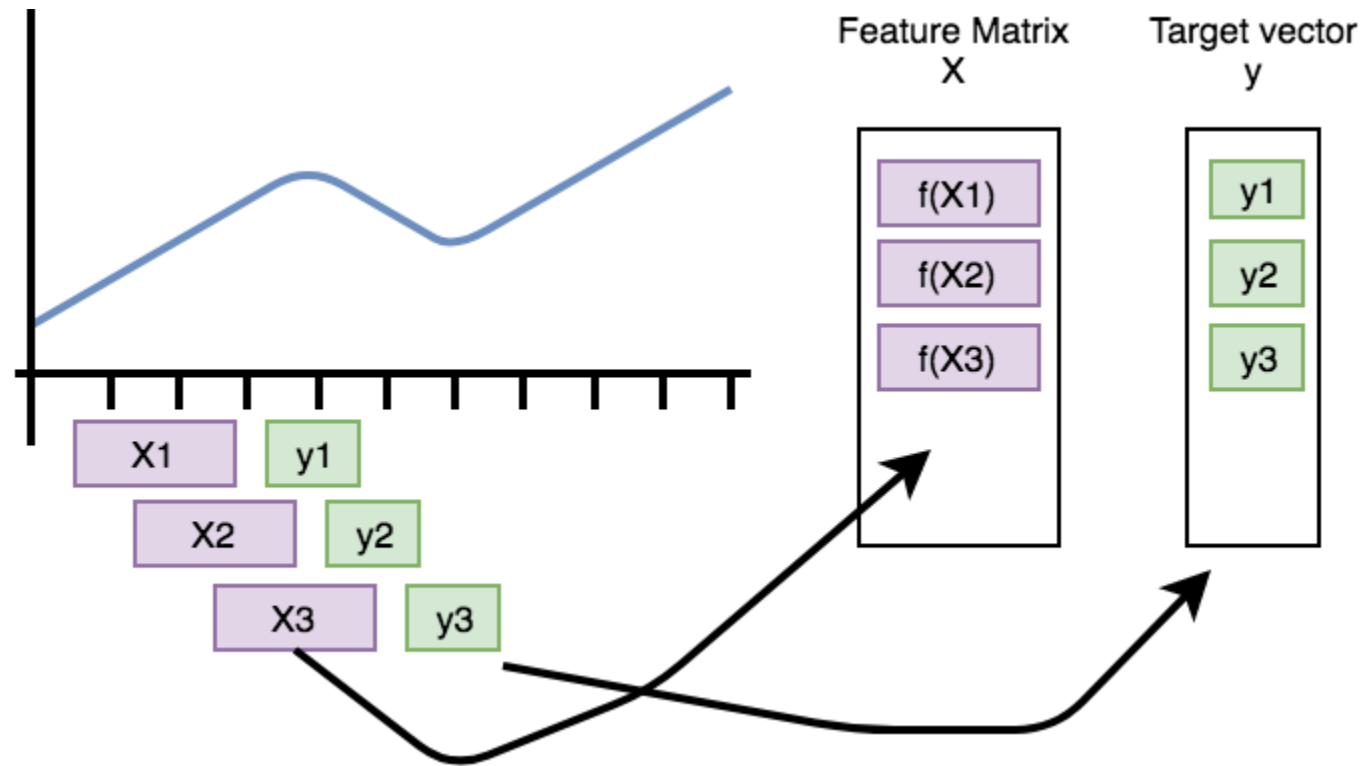
# Example Problem: Chicago's Transit Data

```
>>> df.head()
```

date	day_type	bus	rail
2001-01-01	U	297192	126455
2001-01-02	W	780827	501952
2001-01-03	W	824923	536432
2001-01-04	W	870021	550011
2001-01-05	W	890426	557917



# 3.1 Preparing the Data for ML





# 3.1 Preparing the Data for ML

```
seq_length = 56
train_ds = tf.keras.utils.timeseries_dataset_from_array(
    rail_train.to_numpy(),
    targets=rail_train[seq_length:],
    sequence_length=seq_length,
    batch_size=32,
    shuffle=True,
    seed=42
)
valid_ds = tf.keras.utils.timeseries_dataset_from_array(
    rail_valid.to_numpy(),
    targets=rail_valid[seq_length:],
    sequence_length=seq_length,
    batch_size=32
)
```

## 3.2 Forecasting Using a Linear Model

```
model = tf.keras.Sequential([  
    tf.keras.layers.Dense(1, input_shape=[seq_length])  
])
```

```
early_stopping_cb = tf.keras.callbacks.EarlyStopping(  
    monitor="val_mae", patience=50,  
    restore_best_weights=True)
```

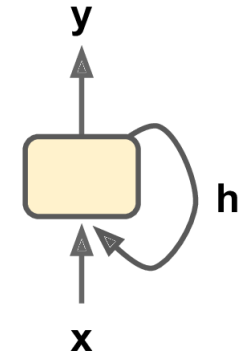
```
opt = tf.keras.optimizers.SGD(learning_rate=0.02,  
    momentum=0.9)
```

```
model.compile(loss=tf.keras.losses.Huber(),  
    optimizer=opt, metrics=["mae"])
```

```
history = model.fit(train_ds, validation_data=valid_ds,  
    epochs=500, callbacks=[early_stopping_cb])
```

MAE = 37,866

# 3.3 Forecasting Using a Simple RNN

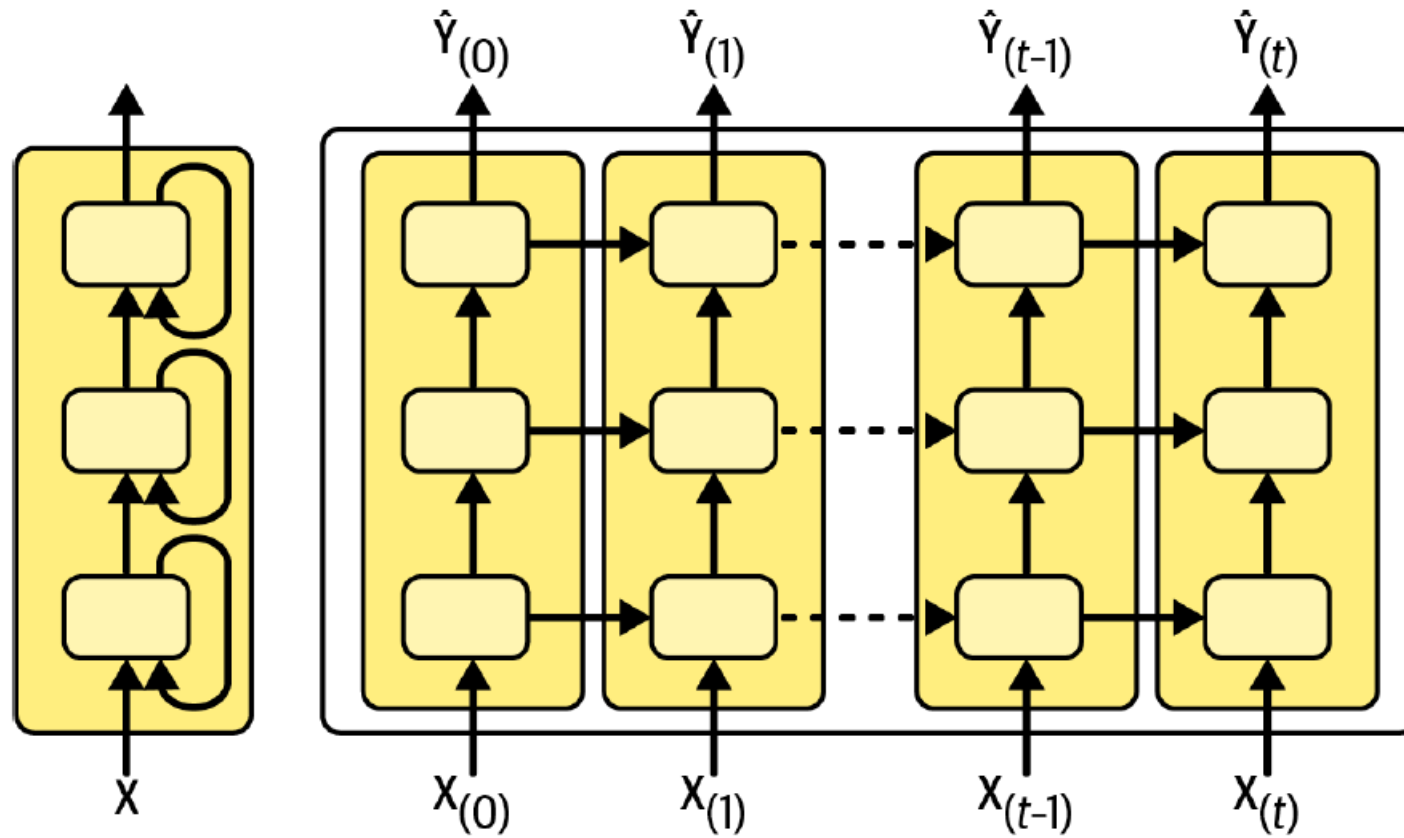


```
univar_model = tf.keras.Sequential([  
    tf.keras.layers.SimpleRNN(32,   
                                input_shape=[None, 1]),  
    tf.keras.layers.Dense(1) # no activation function  
])
```

Uses tanh activation  
 $h_t = y_t$

MAE = 27,703

## 3.4 Forecasting Using Deep RNNs



## 3.4 Forecasting Using Deep RNNs

```
deep_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(32, return_sequences=True,
                               input_shape=[None, 1]),
    tf.keras.layers.SimpleRNN(32, return_sequences=True),
    tf.keras.layers.SimpleRNN(32),
    tf.keras.layers.Dense(1)
])
```

MAE = 31,211

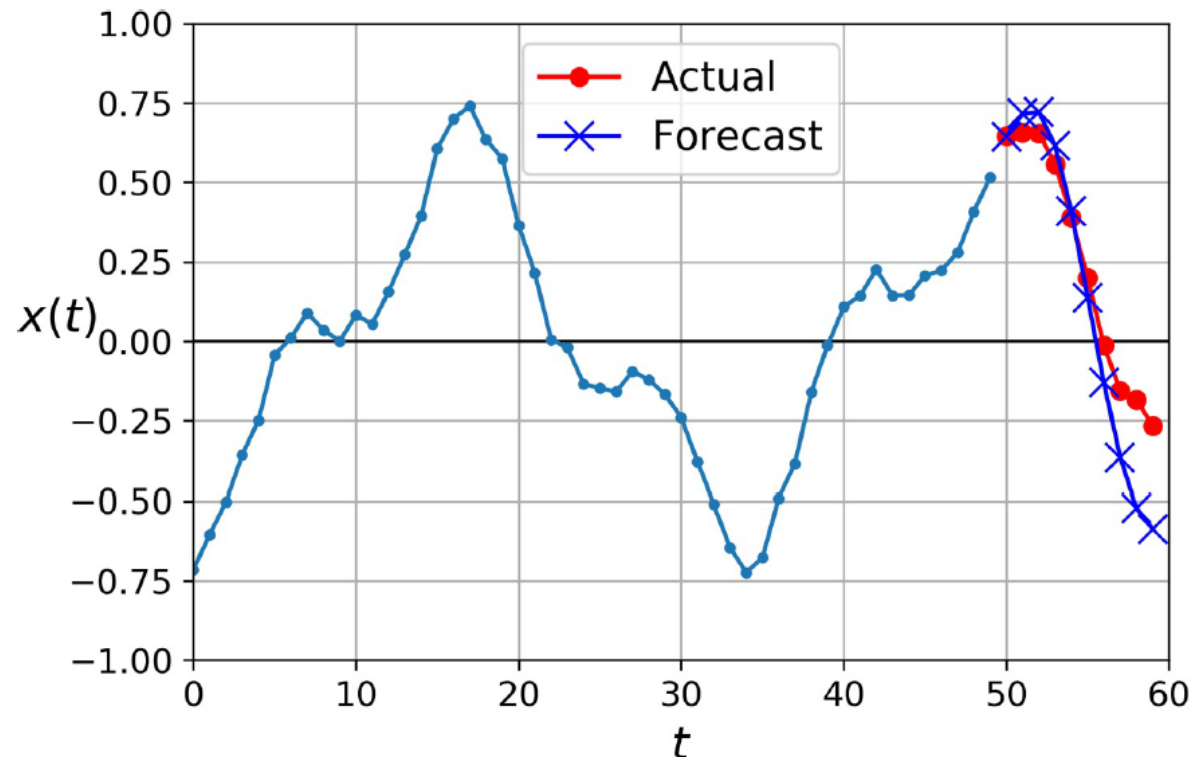
## 3.5 Forecasting Multivariate Time Series

```
# Five input values for each time step  
# Predict two variats  
mulvar_model = tf.keras.Sequential([  
    tf.keras.layers.SimpleRNN(32, input_shape=[None, 5]),  
    tf.keras.layers.Dense(2)  
])
```

MAE = 25,330 for rail  
MAE = 26,369 for bus

# 3.6 Forecasting Several Time Steps Ahead

- Can train an RNN to predict all **N next** values at once (sequence-to-vector model).
- The output layer should have N neurons.



## 3.6 Forecasting Several Time Steps Ahead

```
# Five input values for each time step
seq2seq_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(32,
                               input_shape=[None, 5]),
    tf.keras.layers.Dense(14)
])
```



## 3.7 Forecasting Using a Sequence-to-Sequence Model

```
# Loss is evaluated at every time step
seq2seq_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(32,
                               input_shape=[None, 5],
                               return_sequences=True),
    tf.keras.layers.Dense(14)
])
```

Forecasts for  $t + 1$ , has MAE of 25,519  
Forecasts for  $t + 2$ , has MAE of 26,274  
Forecasts for  $t + 14$ , has MAE is 34,322

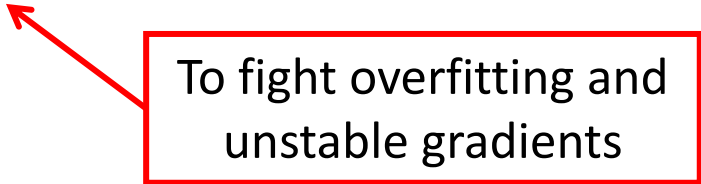
# Outline

1. Introduction
2. Recurrent neurons and layers
3. Forecasting a time series
4. Handling long sequences
5. 1D Convolutional Layers
6. Exercises
7. Summary

# 4. Handling Long Sequences

- Training long sequences has two major challenges:
  - Unstable gradients
  - Forgetting the first inputs in the sequence
- For the **unstable gradients**:
  - **Does not help**: ReLU activation, batch normalization
  - **Helps**: good parameter initialization, faster optimizers, dropout

```
model = Sequential()  
model.add(layers.SimpleRNN(32, dropout=0.2, recurrent_dropout=0.2,  
                           input_shape=[None, 5]))  
model.add(layers.Dense(14))
```



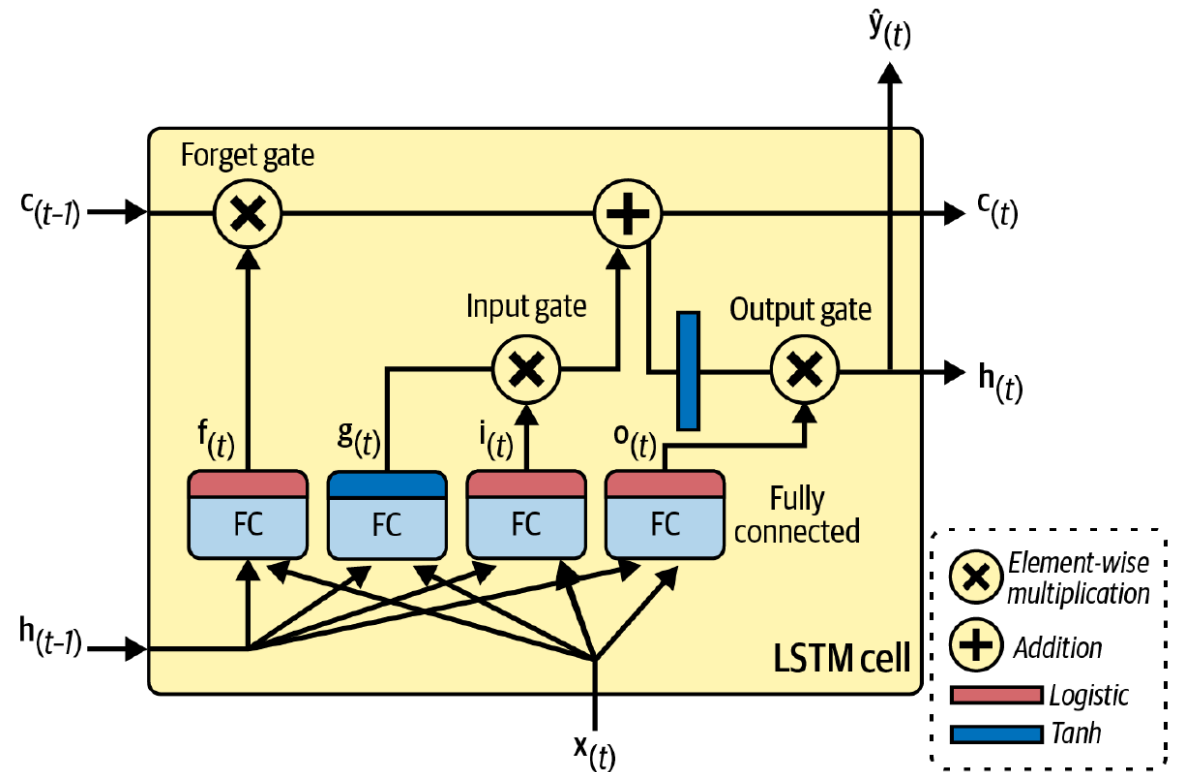
To fight overfitting and  
unstable gradients

# 4. Handling Long Sequences

- To solve the **short-term memory problem**, use
  - **LSTM cell**
  - **GRU cell**
- These cells can be used in place of **SimpleRNN**

# 4.1 LSTM Cell

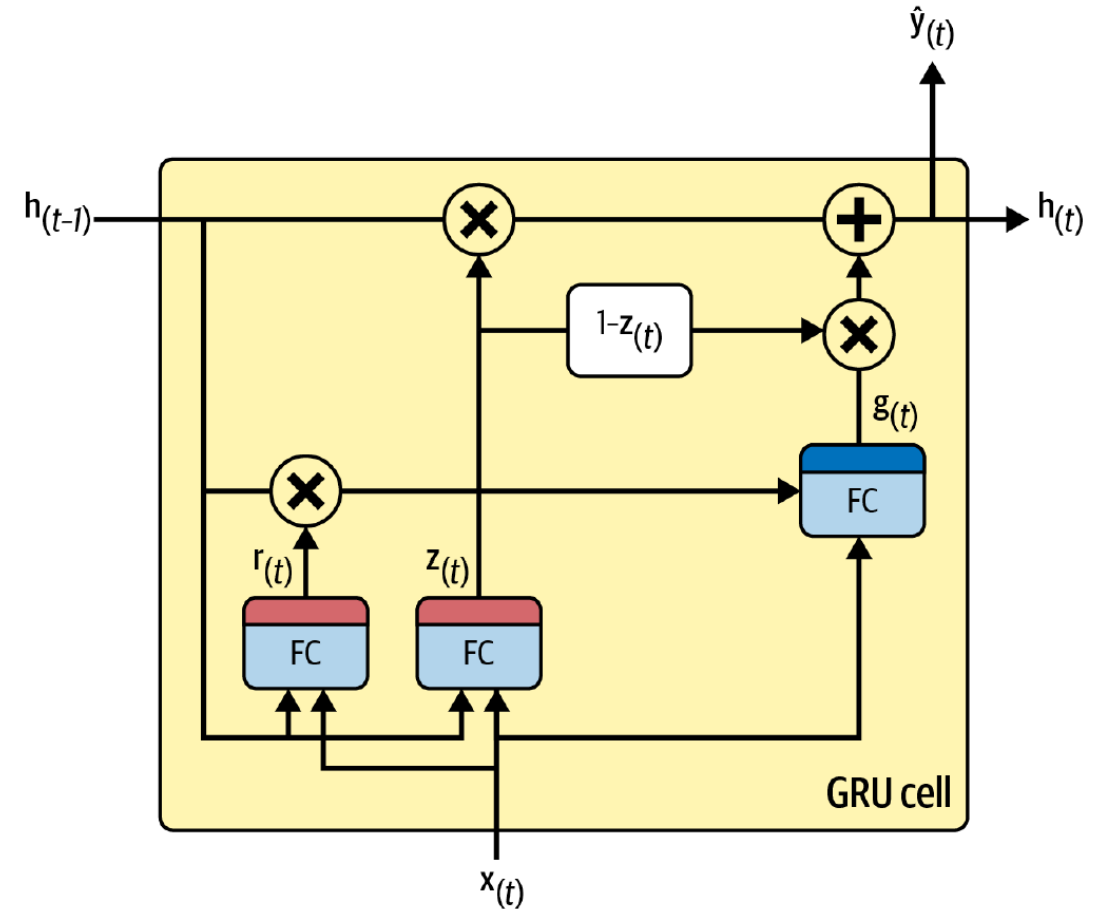
- The **Long Short-Term Memory** (LSTM) cell was proposed in 1997.
- Training converges faster and it **detects long-term dependencies** in the data.
- $\mathbf{h}_{(t)}$  as the short-term state and  $\mathbf{c}_{(t)}$  as the long-term state.



```
model.add(LSTM(20))
```

## 4.2 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.



```
model.add(GRU(20))
```

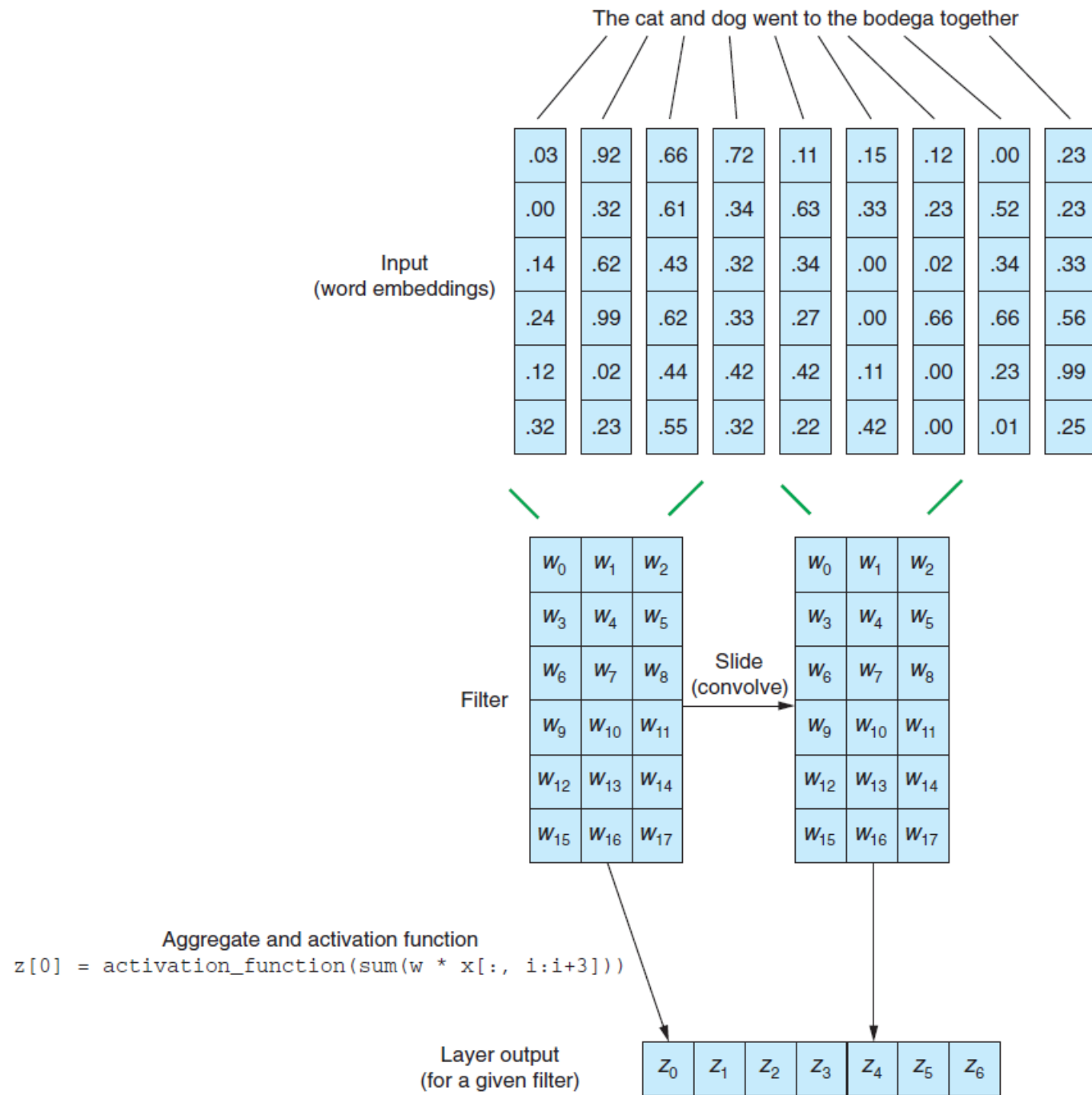
# Outline

1. Introduction
2. Recurrent neurons and layers
3. Forecasting a time series
4. Handling long sequences
5. 1D Convolutional Layers
6. Exercises
7. Summary

# 5.1 1D Convolutional Layers

- Unlike 2D convolution for images, 1D convolution operates over a single dimension, making it ideal for analyzing sequences such as sentences, time series data, or audio signals.
- **Application in NLP:** 1D convolutions are used to slide over sequences of words or characters, capturing local patterns within the text. This can be beneficial for tasks like sentiment analysis, text classification, and entity recognition.
- **How It Works:** A 1D convolutional layer applies filters to a sequence, producing a new set of outputs. For instance, with a filter size of 3, each output element is computed from one word/token and its two neighbors.
- **Benefits:**
  - Captures local dependencies within the sequence.
  - Reduces the dimensionality of the sequence, highlighting important features.





# 5.1 Keras Conv1D

```
# Kernel size is 4 with down sampling by 2 (stride)

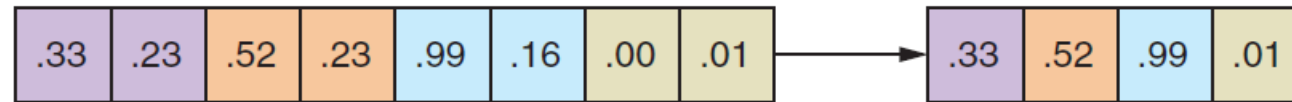
conv_rnn_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=4,
                            strides=2, activation="relu",
                            input_shape=[None, 5]),
    tf.keras.layers.GRU(32),
    tf.keras.layers.Dense(14)
])
```

# 5.2 1D Pooling

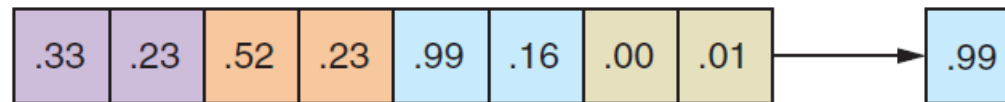
- **Purpose of Pooling:** 1D pooling layers are used to downsample sequence data. This reduces the computational load, memory usage, and helps to prevent overfitting by abstracting higher-level features.
- **Types of 1D Pooling:**
  - **Max Pooling:** Selects the maximum value from each segment of the sequence.
  - **Average Pooling:** Calculates the average value over each segment, providing a smoother representation.
- **Application:** Often follows a 1D convolutional layer to reduce the length and complexity of the sequence while retaining critical information.
- **Benefits:**
  - Helps in capturing the most prominent feature in a local patch of the sequence.
  - Reduces the sensitivity to the exact location of features in the sequence.

# 5.2 1D Pooling

1D max pooling (1 x 2 window)



1D global max pooling



## 5.3 Example: Movie Reviews Sentiment Analysis

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import Conv1D, GlobalMaxPooling1D

from nltk.tokenize import TreebankWordTokenizer
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format(
    'GoogleNews-vectors-negative300.bin.gz',
    binary=True, limit=200000)
```

```
def tokenize_and_vectorize(dataset):
    tokenizer = TreebankWordTokenizer()
    vectorized_data = []
    for sample in dataset:
        tokens = tokenizer.tokenize(sample[1])
        sample_vecs = []
        for token in tokens:
            try:
                sample_vecs.append(word_vectors[token])
            except KeyError:
                pass # No matching
        vectorized_data.append(sample_vecs)
    return vectorized_data
```

```
vectorized_data = tokenize_and_vectorize(dataset)
```

```
# Then split to 20% test, 80% train ...
```

```
filters = 250
kernel_size = 3
maxlen = 400
embedding_dims = 300

model = Sequential()

model.add(Conv1D(filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1,
                 input_shape=(maxlen, embedding_dims)))
```

```
model.add(GlobalMaxPooling1D())
model.add(Dense(250))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=32,
          epochs=2,
          validation_data=(x_test, y_test))
```



# 6. Exercises

- 15.1. Can you think of a few applications for a sequence-to-sequence RNN? What about a sequence-to-vector RNN, and a vector-to-sequence RNN?
- 15.2. How many dimensions must the inputs of an RNN layer have? What does each dimension represent? What about its outputs?
- 15.3. If you want to build a deep sequence-to-sequence RNN, which RNN layers should have `return_sequences=True`? What about a sequence-to-vector RNN?
- 15.4. Suppose you have a daily univariate time series, and you want to forecast the next seven days. Which RNN architecture should you use?
- 15.5. What are the main difficulties when training RNNs? How can you handle them?
- 15.6. Can you sketch the LSTM cell's architecture?

# Summary

1. Introduction
2. Recurrent neurons and layers
3. Forecasting a time series
4. Handling long sequences
5. 1D Convolutional Layers