5-day Hands-on Workshop on:

Python for Scientific Computing and TensorFlow for Artificial Intelligence

By Dr Stephen Lynch FIMA SFHEA

Holder of Two Patents

Author of PYTHON, MATLAB®, MAPLE™ AND MATHEMATICA® BOOKS

STEM Ambassador and Speaker for Schools

s.lynch@mmu.ac.uk

https://www2.mmu.ac.uk/scmdt/staff/profile/index.php?id=2443

Schedule (Day 5): Start Session 1

Day 5			
Topics	**Hours**	**Topics**	**Hours**
AI: KERAS and TensorFlow	10am-11am	AI: Recurrent Neural Networks	1pm-2pm
AI: Convolutional Neural Networks	11am-12pm	AI: Introduction to TensorBoard	2pm-3pm

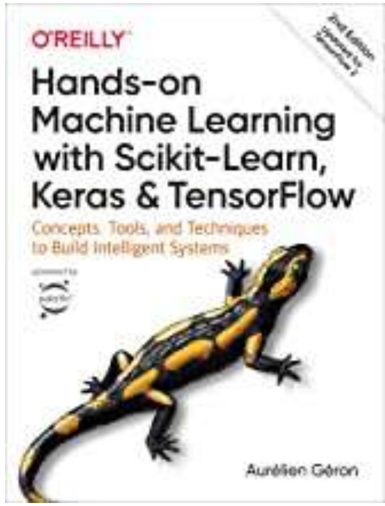
Download files from GitHub:

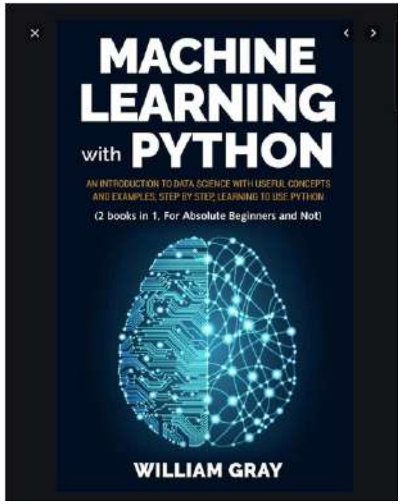
https://github.com/DrStephenLynch/Tekbac

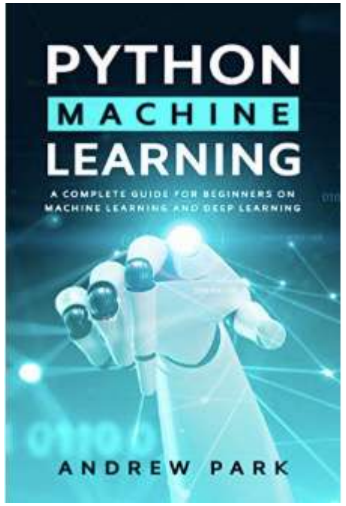
Application Programming Interface (API)

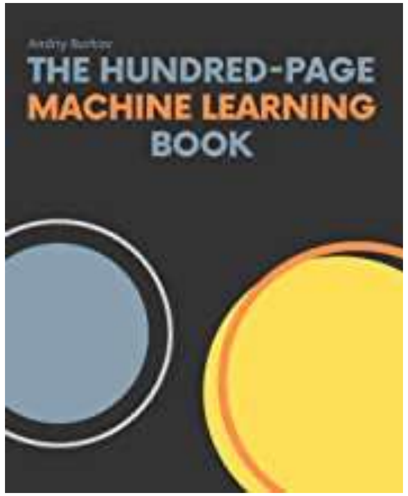
https://keras.io/api/applications/

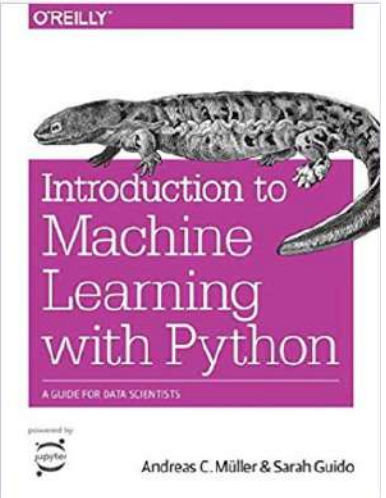
Machine and Deep Learning Books

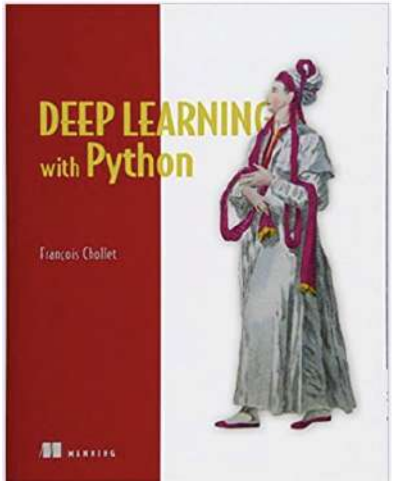


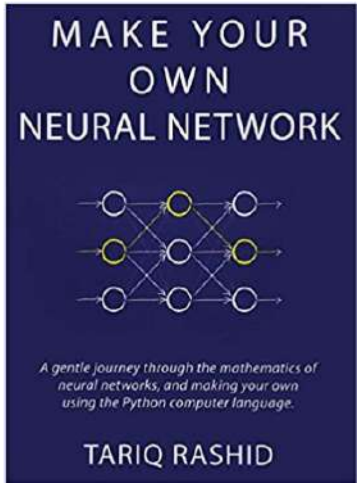


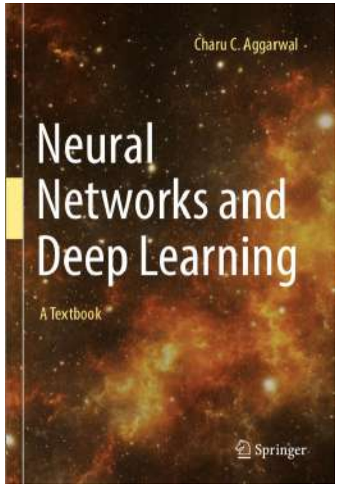




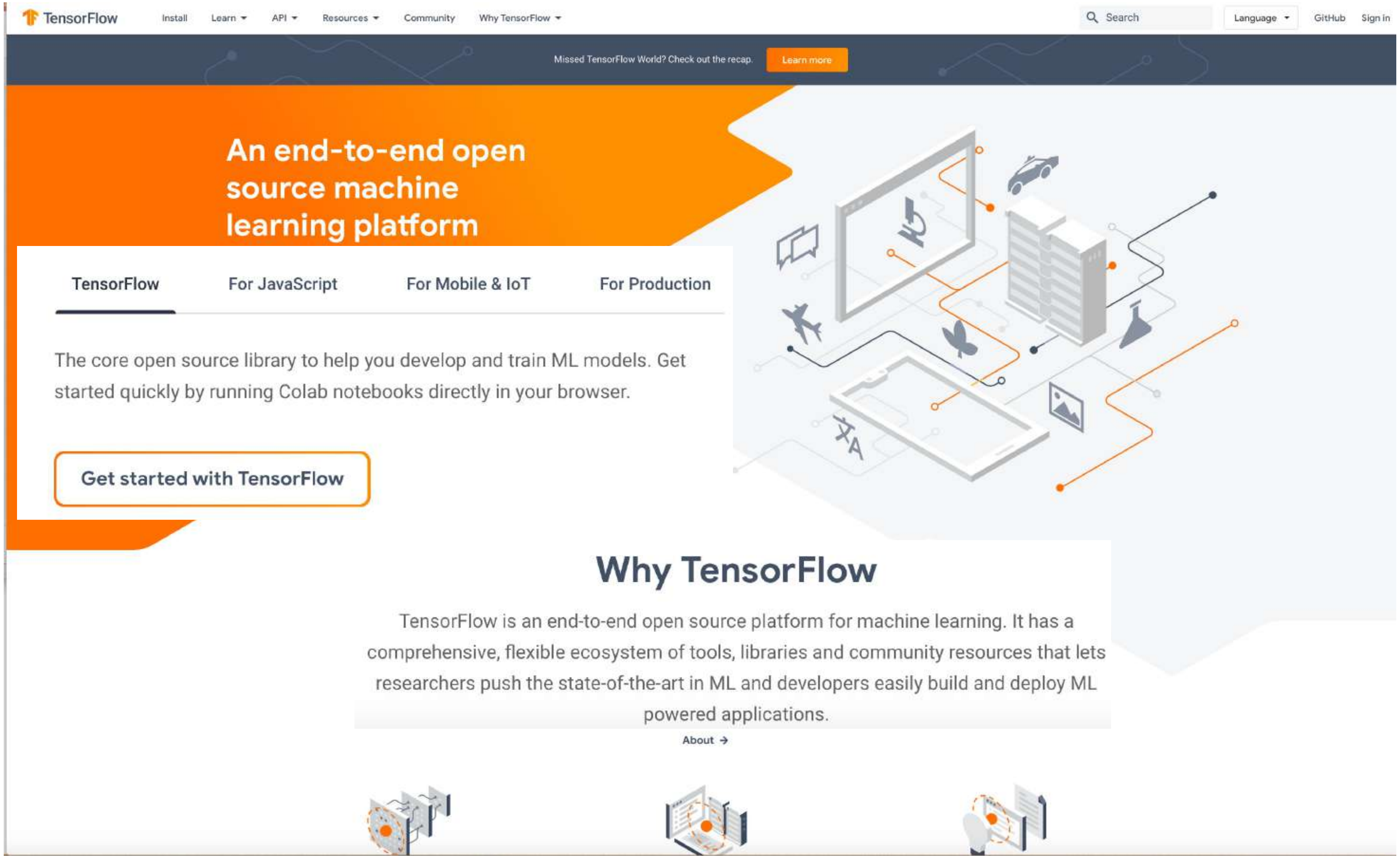




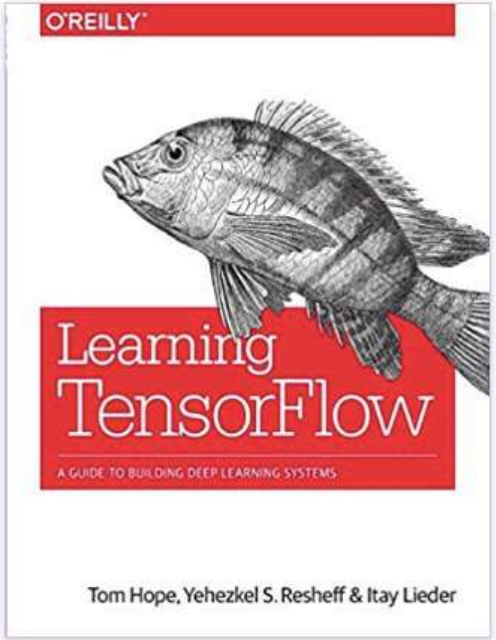


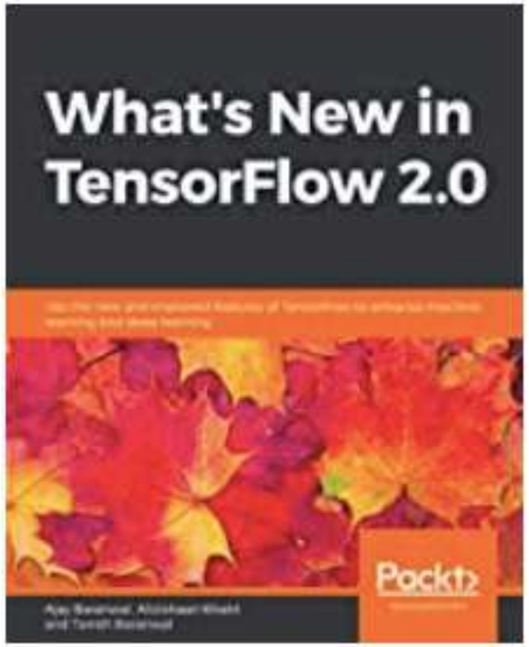


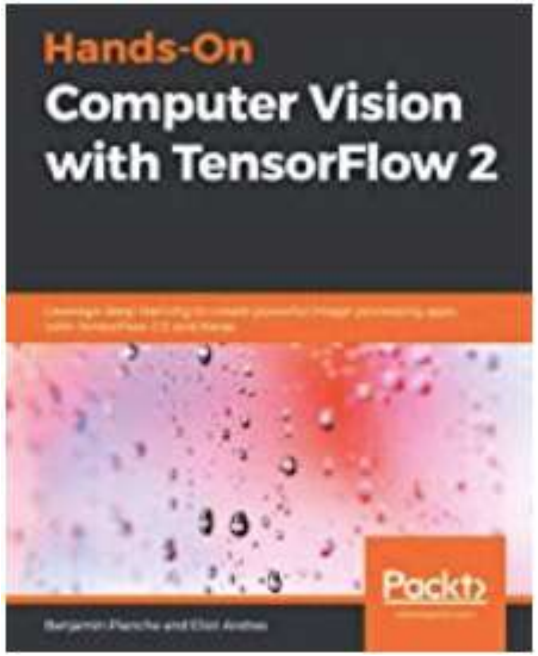
TensorFlow

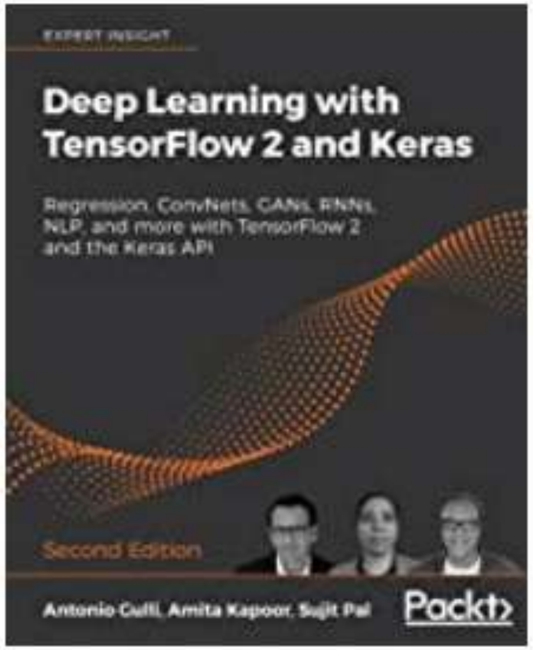

TensorFlow
Install
Learn
API
Resources
Community
Why TensorFlow
Search
Language
GitHub
Sign in
Missed TensorFlow World? Check out the recap.
Learn more
An end-to-end open source machine learning platform
TensorFlow
For JavaScript
For Mobile & IoT
For Production
The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.
Get started with TensorFlow
Why TensorFlow
TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.
About

TensorFlow Books



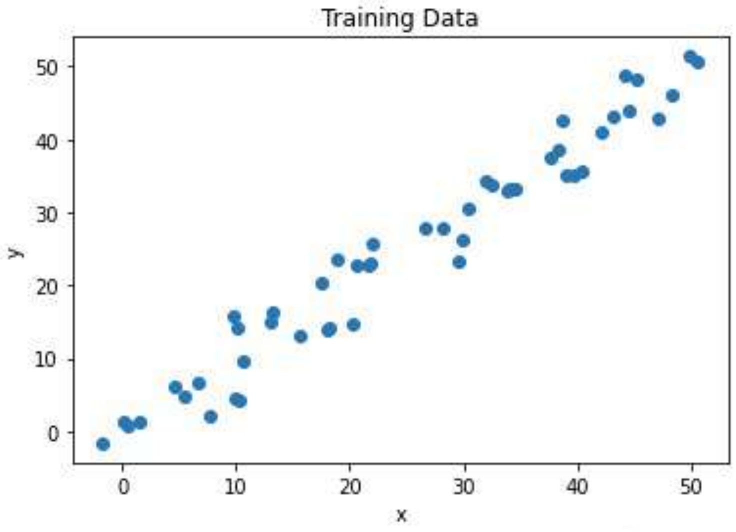


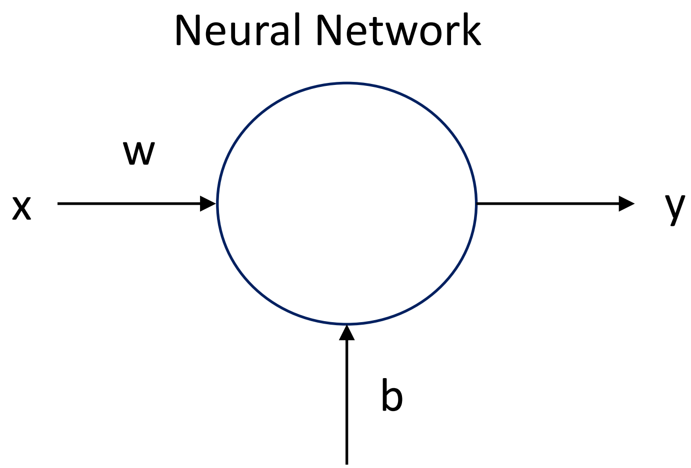




Linear Regression in TensorFlow 2

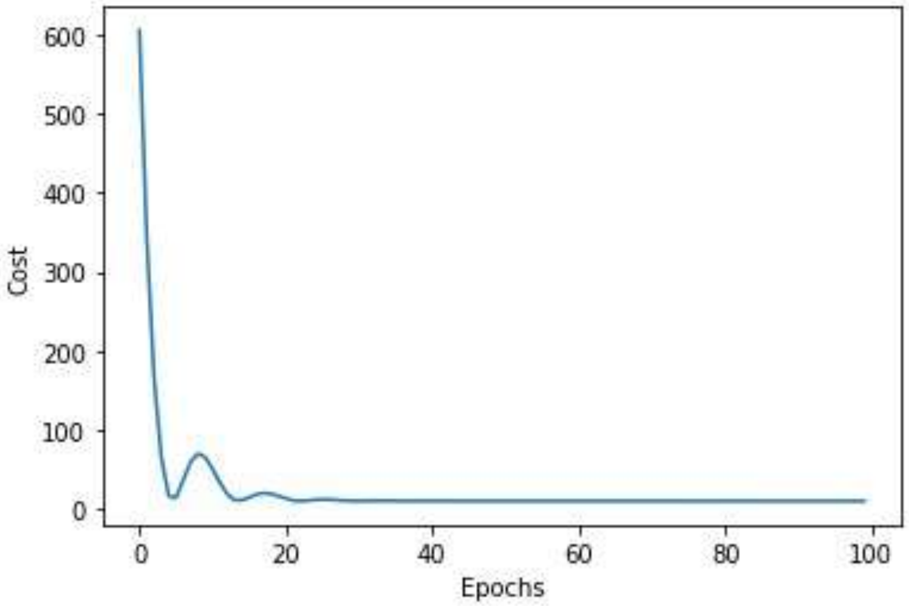
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

np.random.seed(101)
x_train = np.linspace(0, 50, 50)
y_train = np.linspace(0, 50, 50)          # There will be 50 data points ranging from 0 to 50.
x_train += np.random.uniform(-4, 4, 50)   # Adding noise to the random linear data.
y_train += np.random.uniform(-4, 4, 50)
n = len(x_train)                          # Number of data points.
plt.scatter(x_train, y_train)
plt.xlabel('x')
plt.ylabel('y')
plt.title("Training Data")
plt.show()
```





Linear Regression in TensorFlow 2

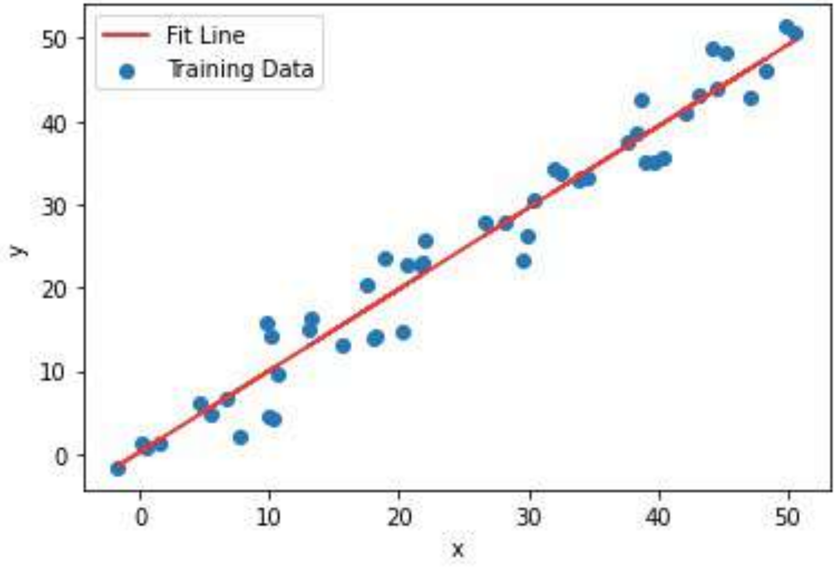
```
[6] layer0 = tf.keras.layers.Dense(units=1, input_shape=[1])
    model = tf.keras.Sequential([layer0])
    model.compile(loss='mean_squared_error',
                  optimizer=tf.keras.optimizers.Adam(0.1))
    history = model.fit(x_train, y_train, epochs=100, verbose=False)
    plt.xlabel('Epochs')
    plt.ylabel("Cost")
    plt.plot(history.history['loss'])
    plt.show()
```



Linear Regression in TensorFlow 2

```
[8] weights = layer0.get_weights()
    weight = weights[0][0]
    bias = weights[1]
    print('weight: {} bias: {}'.format(weight, bias))
    y_learned = x_train * weight + bias
    plt.scatter(x_train, y_train, label='Training Data')
    plt.plot(x_train, y_learned, color='red', label='Fit Line')
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

```
weight: [0.9789486] bias: [0.23586343]
```



Equation of Line of Best Fit

y = w * x + b

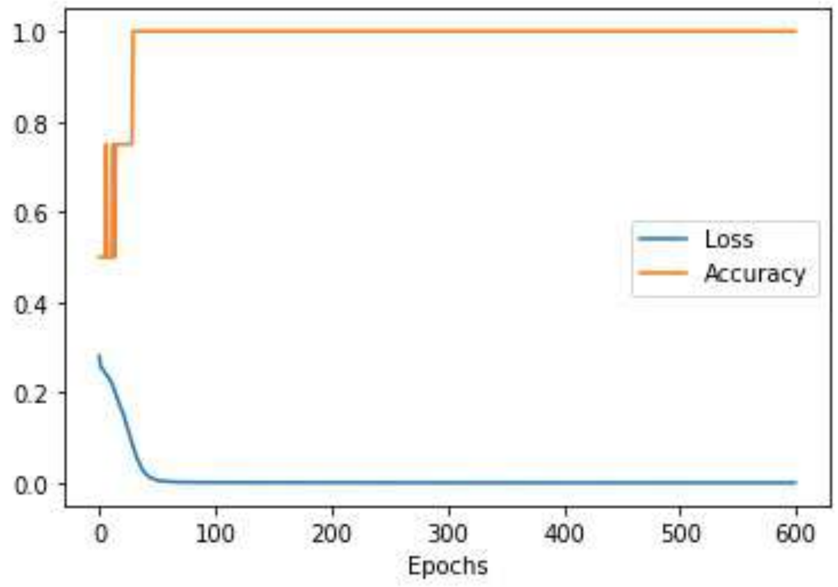
XOR Implementation in TensorFlow 2

```
[6] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import tensorflow as tf
    from tensorflow import keras
    import sys

    training_data = np.array([[0,0],[0, 1], [1, 0], [1, 1]], 'float32')
    target_data = np.array([[0], [1], [1], [0]], 'float32')
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Dense(4, input_dim = 2, activation = 'relu'))
    model.add(tf.keras.layers.Dense(1, activation = 'sigmoid'))

    model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.Adam(0.1),metrics=['accuracy'])
    hist = model.fit(training_data, target_data, epochs = 600, verbose = 0)
    print(model.predict(training_data).round())
    val_loss, val_acc = model.evaluate(training_data, target_data)
    print(val_loss, val_acc)
```

```
[[0.]
 [1.]
 [1.]
 [0.]]
1/1 [==============================] - 0s 1ms/step - loss: 7.1157e-05 - accuracy: 1.0000
7.115719927242026e-05 1.0
```

XOR Implementation in TensorFlow 2

```
loss_curve = hist.history["loss"]
acc_curve = hist.history["accuracy"]
plt.plot(loss_curve, label='Loss')
plt.plot(acc_curve, label='Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



Keras, TensorFlow and PyTorch

	Keras	TensorFlow	PyTorch
Level of API	high-level API[1]	Both high & low level APIs	Lower-level API[2]
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex[3]
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook[4]
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs[5]

Boston Housing Data in TensorFlow: Keras

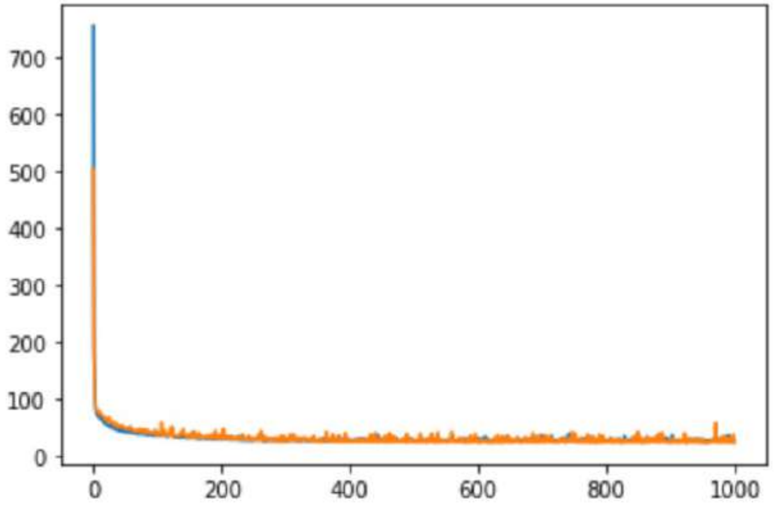
```
[1] import tensorflow as tf
    from tensorflow import keras
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[2] from keras.datasets import boston_housing
    (x_train, y_train), (x_test, y_test) = boston_housing.load_data(path='boston_housing.npz',test_split=0,seed=113)
```

```
[3] model = keras.Sequential([keras.layers.Dense(1, input_dim=13, kernel_initializer='normal'),])
```

```
[4] model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(0.01))
    hist=model.fit(x_train, y_train, epochs=1000, validation_split=0.2, verbose=0)
```

```
[5] plt.plot(range(1000), hist.history['loss'], range(1000), hist.history['val_loss'])
```



Boston Housing Data in TensorFlow: Hidden Layers and Overfitting

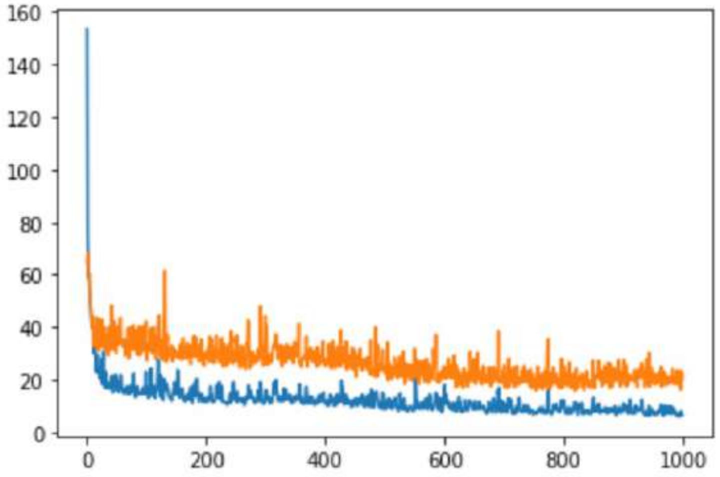
```
[1]  import tensorflow as tf
     from tensorflow import keras
     import numpy as np
     import matplotlib.pyplot as plt
```

```
[2]  from keras.datasets import boston_housing
     (x_train, y_train), (x_test, y_test) = boston_housing.load_data(path='boston_housing.npz',test_split=0,seed=113)
```

```
[3]  model = keras.Sequential([
     keras.layers.Dense(100, input_dim=13, kernel_initializer='normal', activation='relu'),
     keras.layers.Dense(100, kernel_initializer='normal', activation='relu'),
     keras.layers.Dense(1, kernel_initializer='normal'),
     ])
```

```
[4]  model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(0.01))
     hist=model.fit(x_train, y_train, epochs=1000, validation_split=0.2, verbose=0)
```

```
[5]  plt.plot(range(1000), hist.history['loss'], range(1000), hist.history['val_loss'])
```



Boston Housing Data in TensorFlow: Overfitting: End Session 1

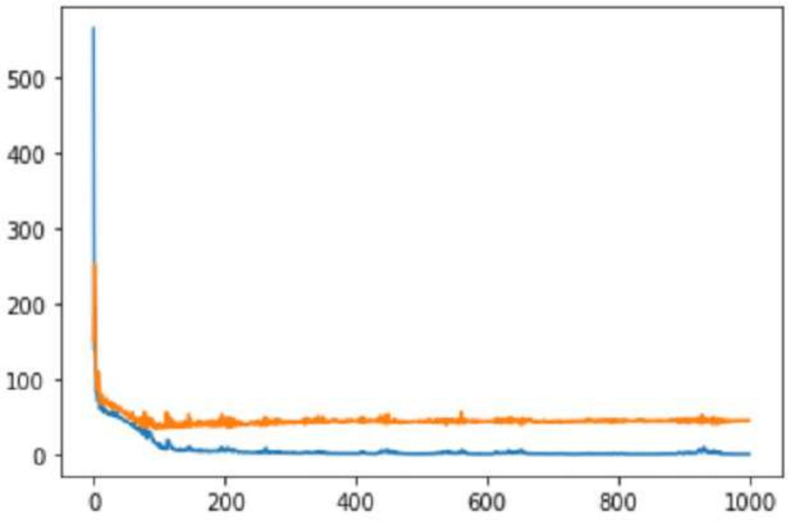
```
[1] import tensorflow as tf
    from tensorflow import keras
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[2] from keras.datasets import boston_housing
    (x_train, y_train), (x_test, y_test) = boston_housing.load_data(path='boston_housing.npz',test_split=0,seed=113)
```

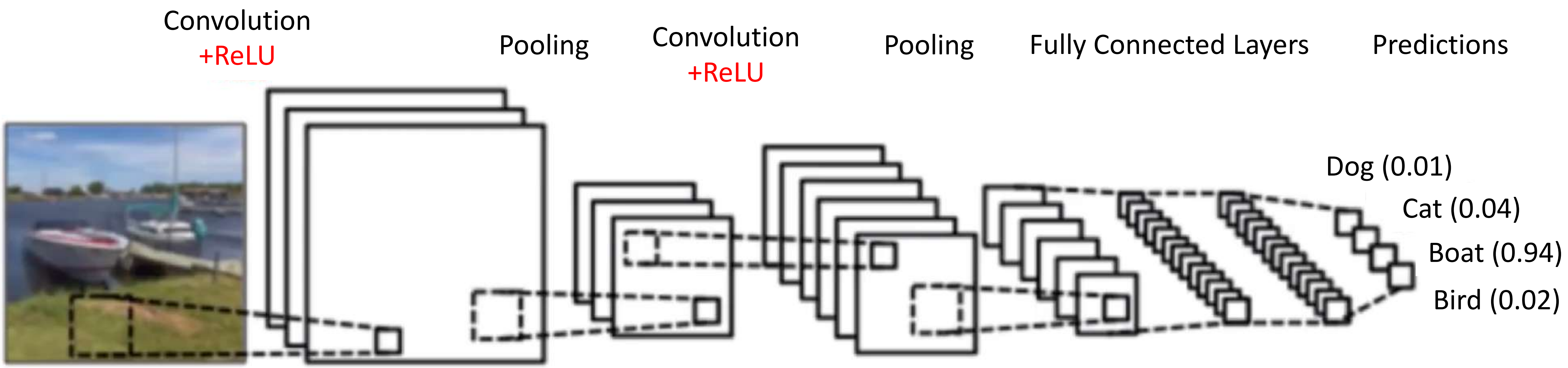
```
[3] model = keras.Sequential([
    keras.layers.Dense(100, input_dim=13, kernel_initializer='normal', activation='relu'),
    keras.layers.Dense(100, kernel_initializer='normal', activation='relu'),
    keras.layers.Dense(1, kernel_initializer='normal'),
    ])
```

```
[4] model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(0.01))
    hist=model.fit(x_train, y_train, epochs=1000, validation_split=0.9, verbose=0)
```

```
[5] plt.plot(range(1000), hist.history['loss'], range(1000), hist.history['val_loss'])
```



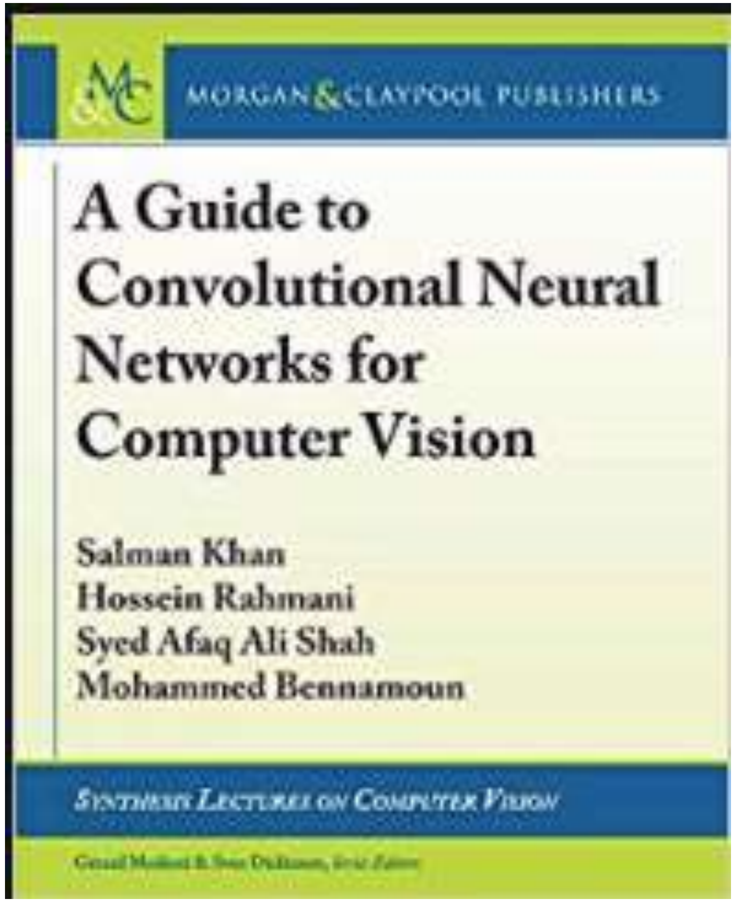
Convolutional Neural Network (CNN): Start Session 2

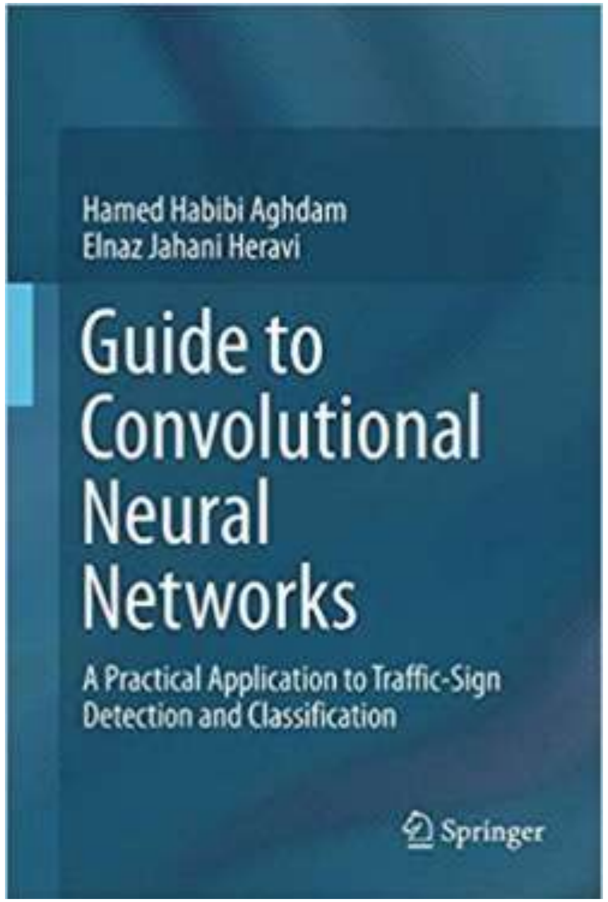


A **Convolutional Neural Network** (**CNN)** is a class of deep neural networks, most commonly applied to analyzing visual imagery. Inspired by biological processes in the animal visual cortex.

https://www.youtube.com/watch?v=2-Ol7ZB0MmU

Convolutional Neural Networks Books





Convolutional Neural Network

Identify the following features:

\

1	-1	-1
-1	1	-1
-1	-1	1

X

1	-1	1
-1	1	-1
1	-1	1

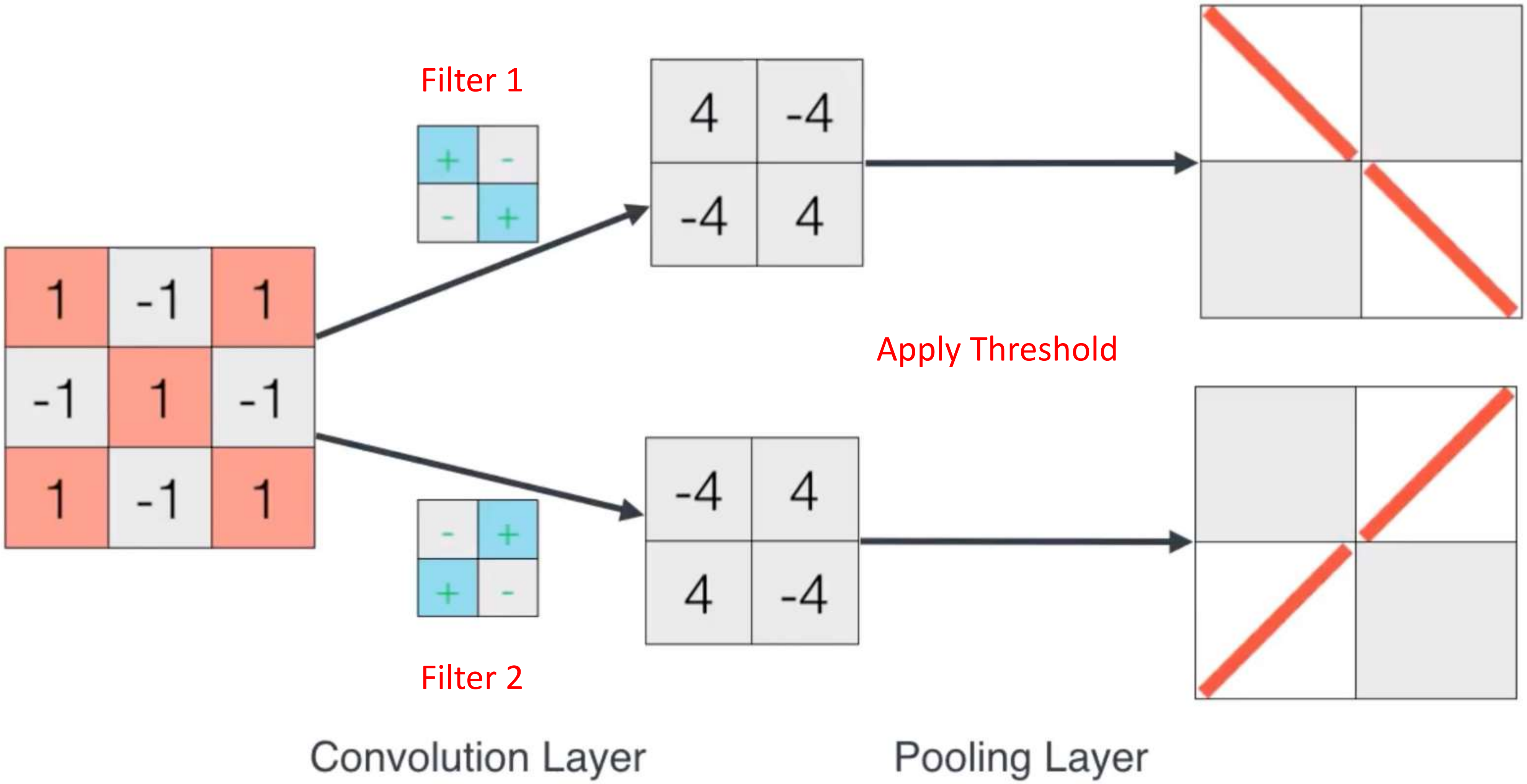
/

-1	-1	1
-1	1	-1
1	-1	-1

O

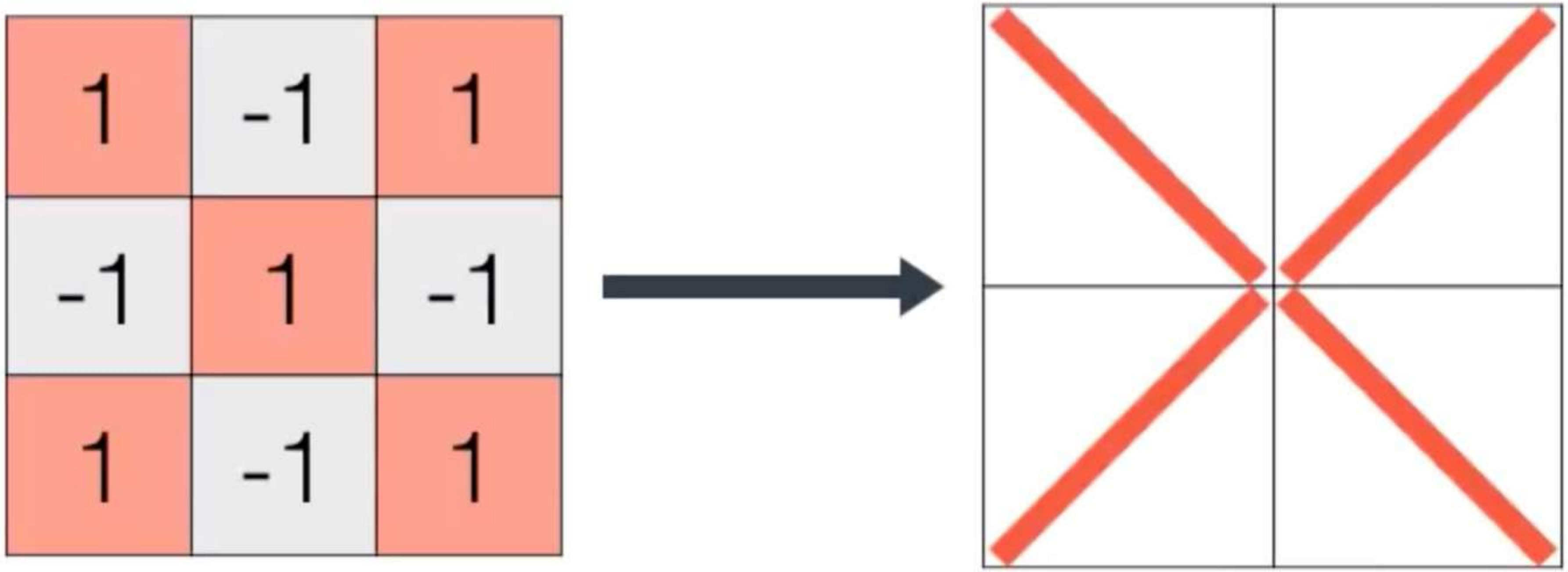
-1	1	-1
1	-1	1
-1	1	-1

Convolutional Neural Network



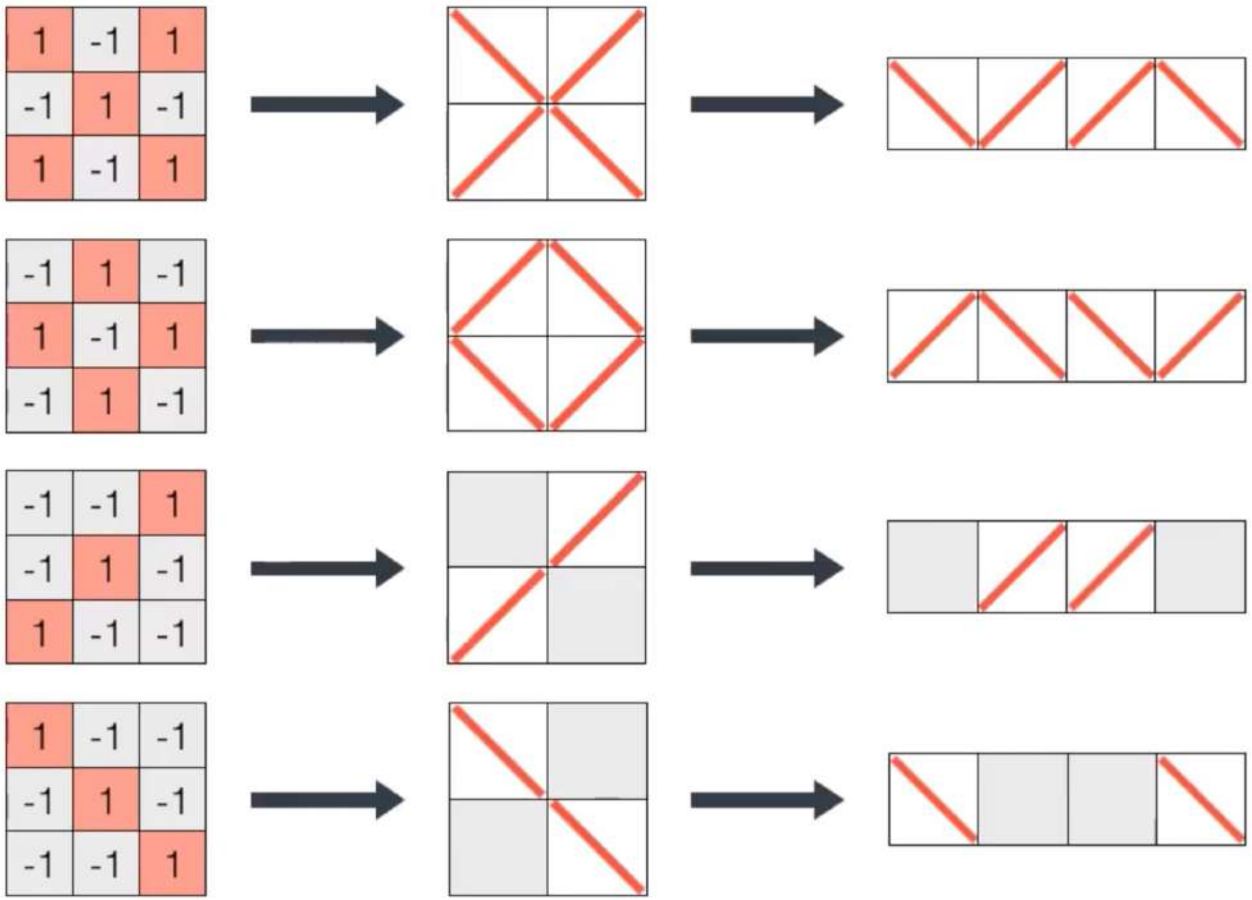
Convolutional Neural Network

Merge the two images to give:



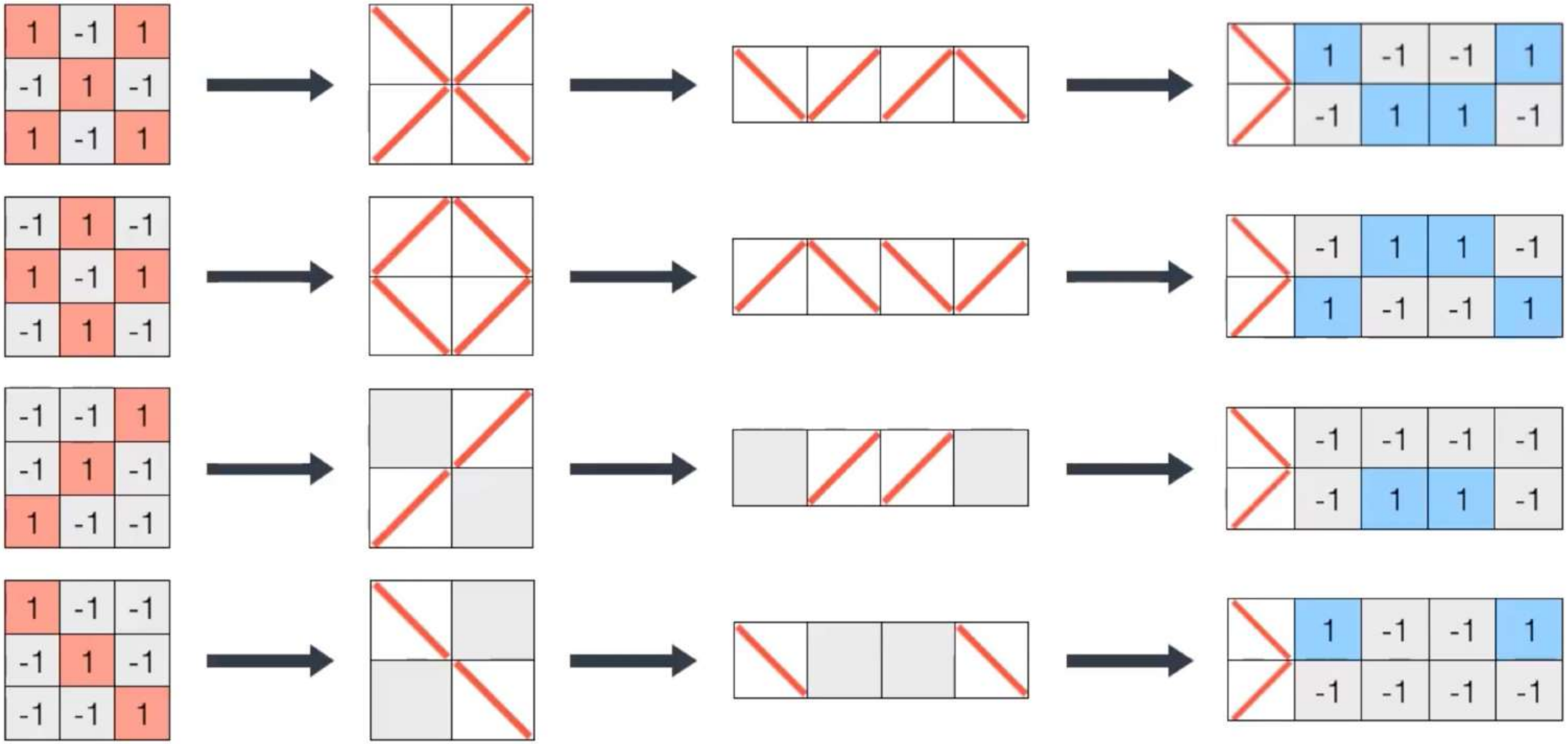
Convolutional Neural Network

Rewrite image as a sequence:



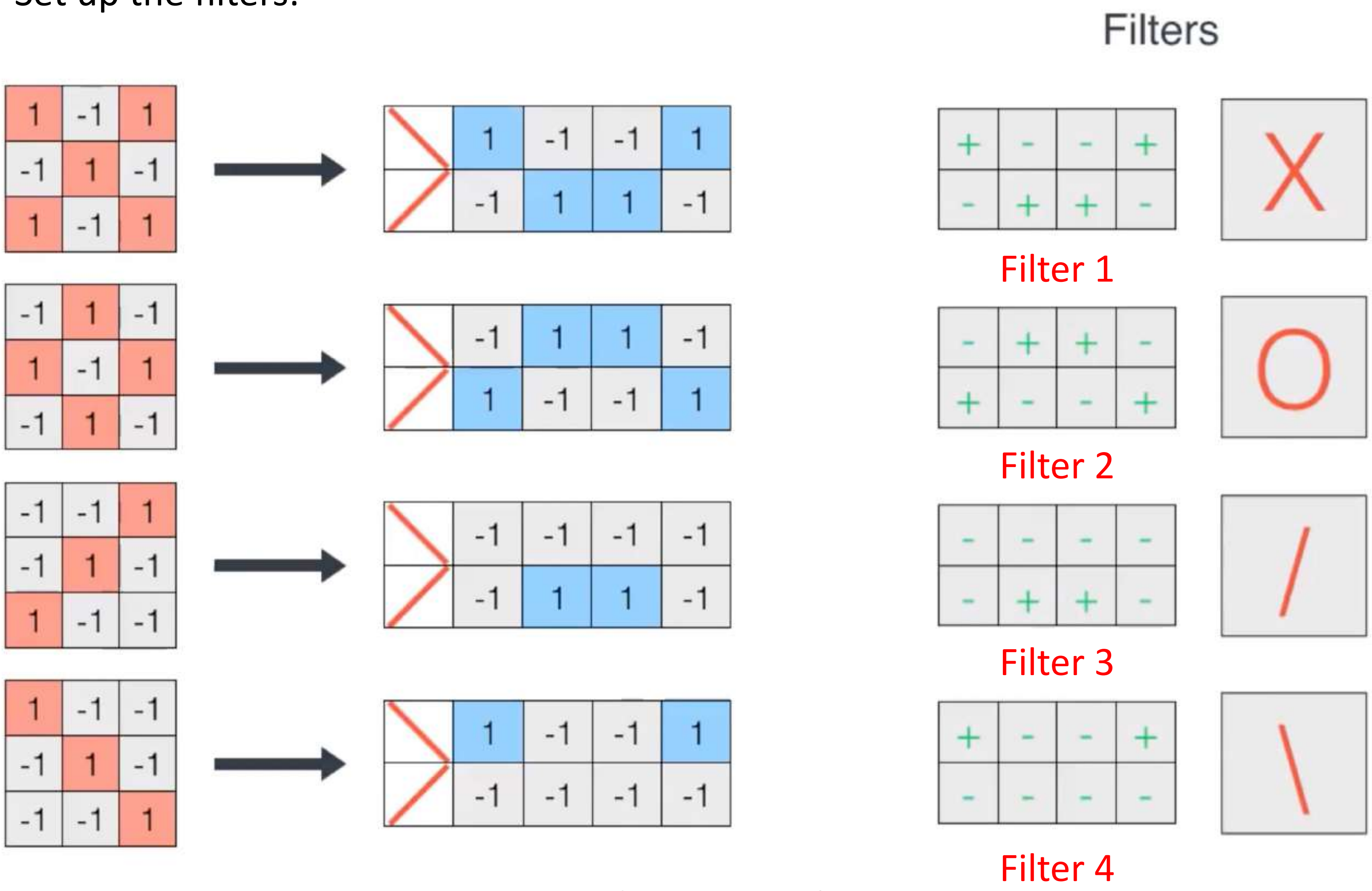
Convolutional Neural Network

Rewrite the sequence as a matrix:



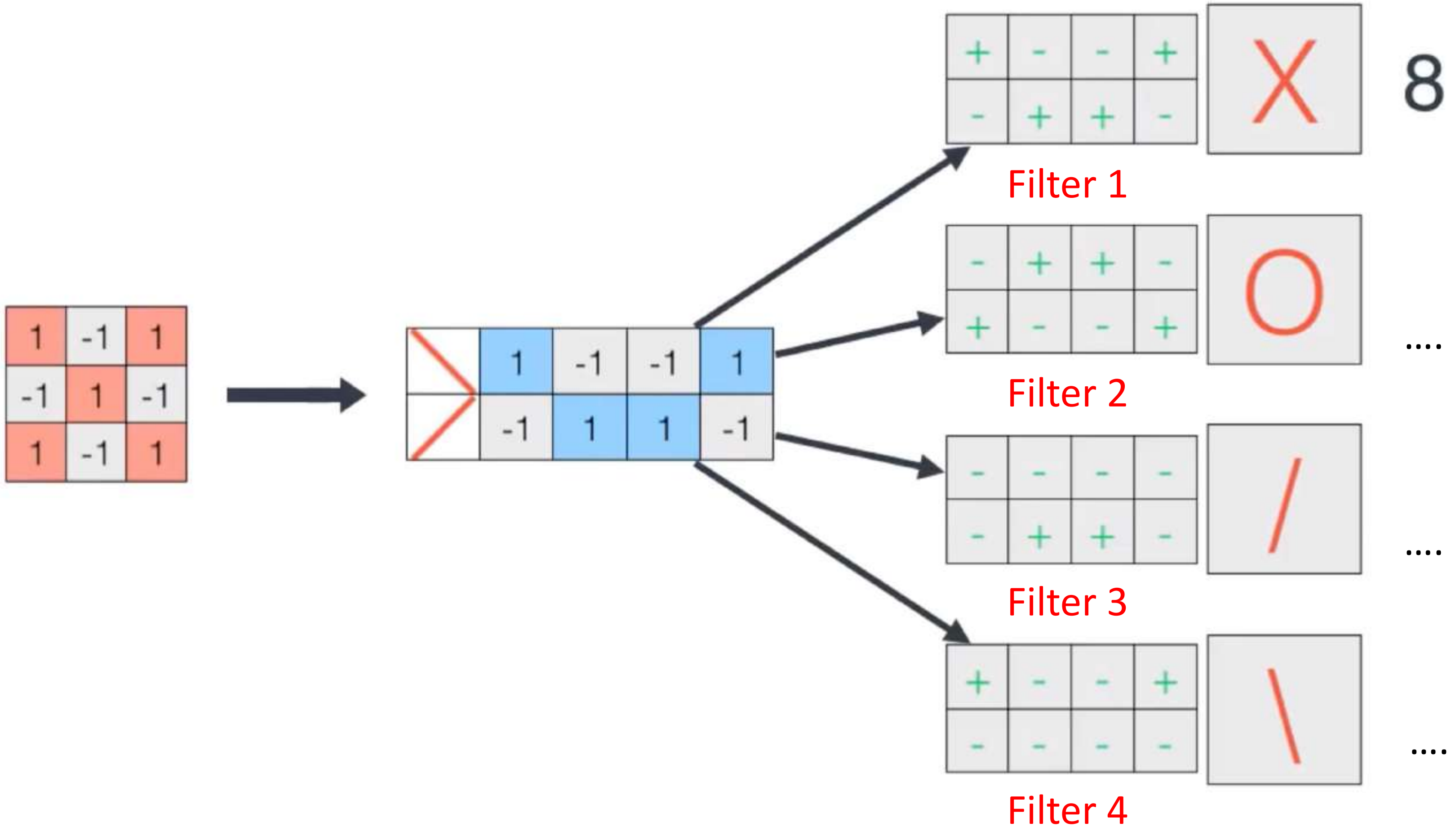
Convolutional Neural Network

Set up the filters:



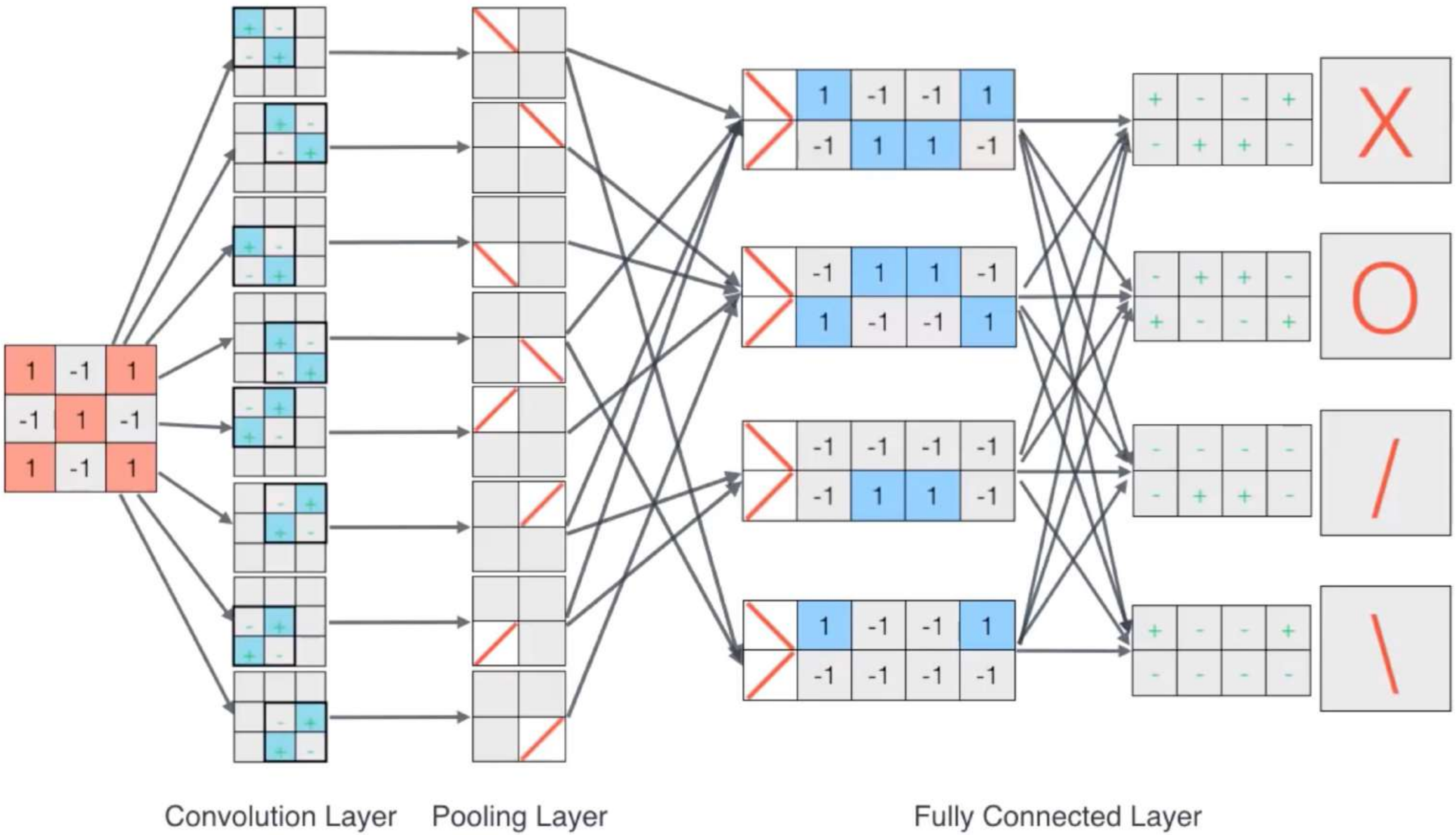
Convolutional Neural Network

Apply the filters:



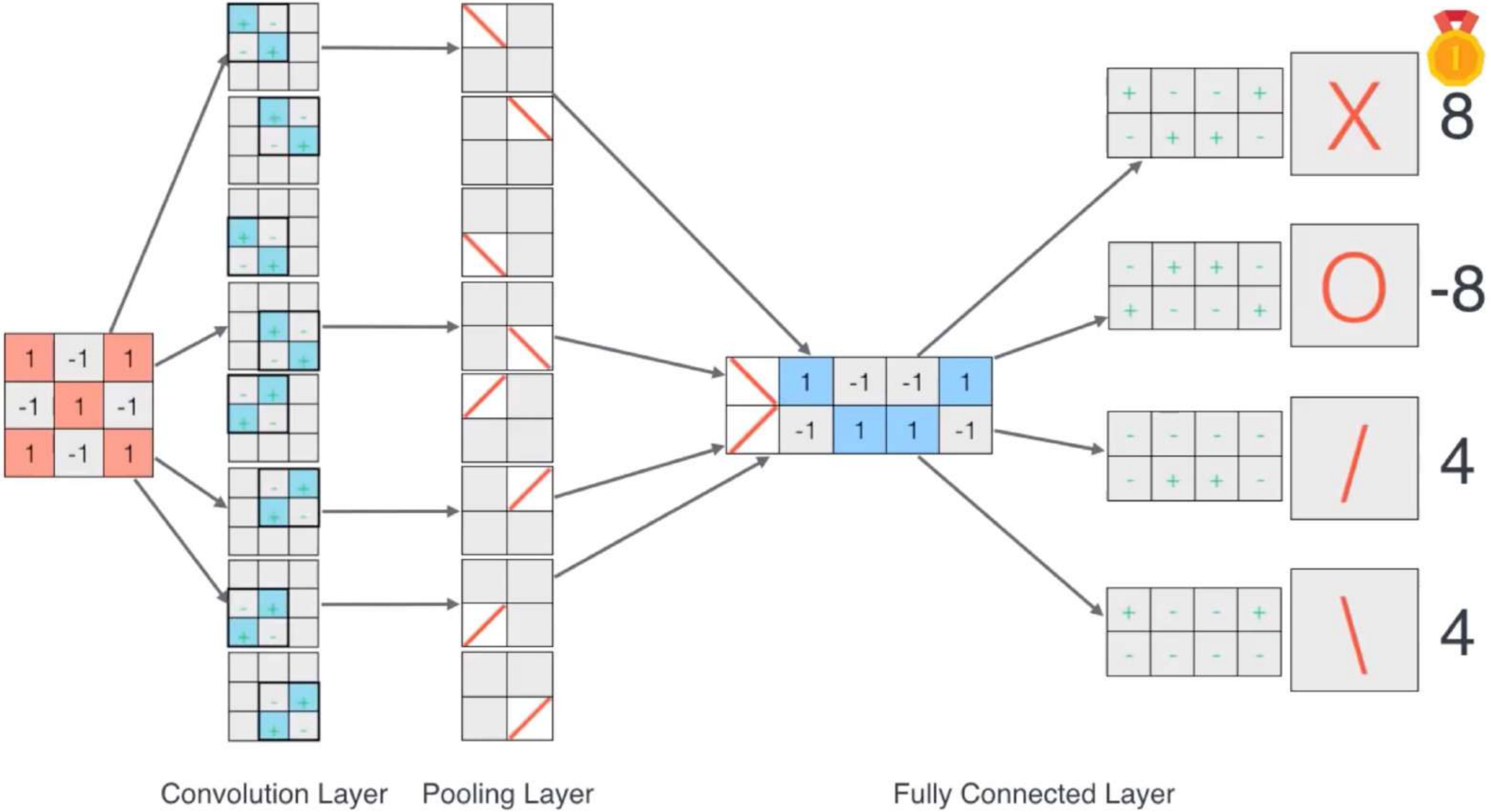
Work out these numbers

Convolutional Neural Network



Manchester Metropolitan University

Convolutional Neural Network



Convolutional Neural Network (MNIST Dataset)

MNIST database of handwritten digits

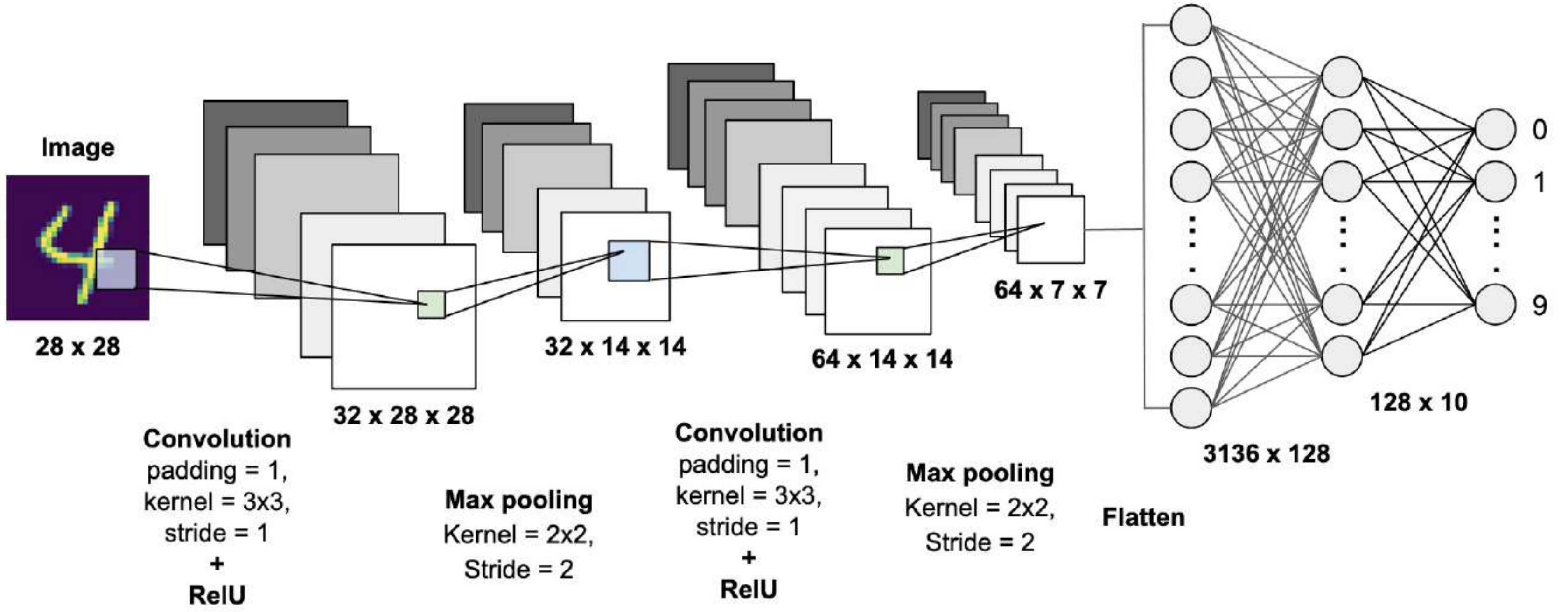
Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

Usage:

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- **Returns:**
  - 2 tuples:
    - **x_train, x_test**: uint8 array of grayscale image data with shape (num_samples, 28, 28).
    - **y_train, y_test**: uint8 array of digit labels (integers in range 0-9) with shape (num_samples,).
- **Arguments:**
  - **path**: if you do not have the index file locally (at `'~/.keras/datasets/' + path`), it will be downloaded to this location.

Convolutional Neural Network (MNIST Dataset)



Google Colab (MNIST Dataset)

MNIST Hidden Layers.ipynb

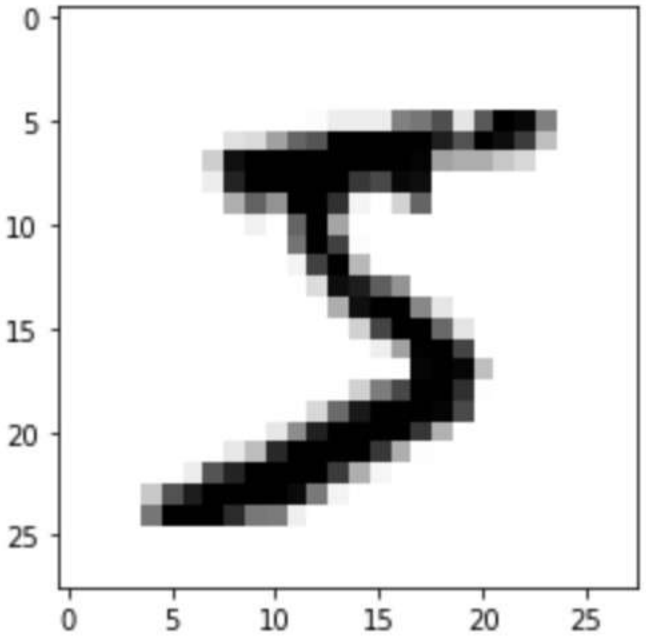
File Edit View Insert Runtime Tools Help

+ Code + Text

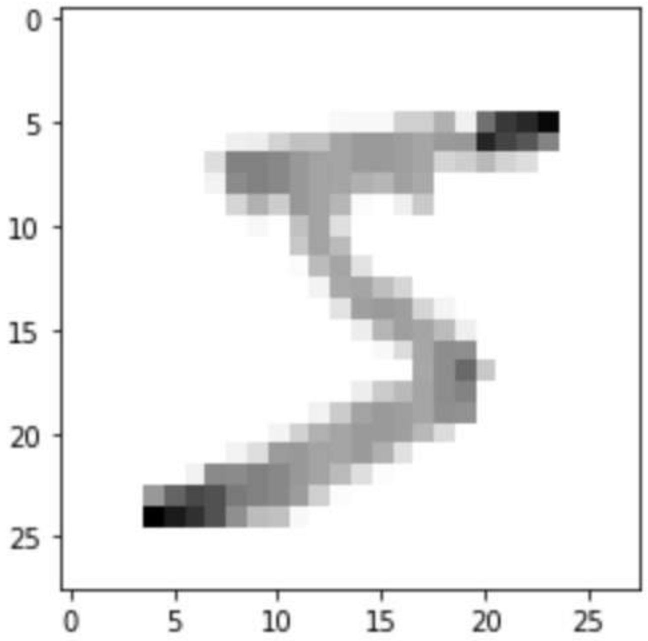
```
import tensorflow as tf
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist # Digits 0-9, 28x28= pixels
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print('Dimensions of first image=',x_train[0].shape)
print(x_train[0])
#plt.imshow(x_train[0])                              # Plots the colour image.
plt.imshow(x_train[0], cmap = plt.cm.binary)   # Plots a grey scale image.
```

Google Colab (MNIST Dataset)

```python
# Normalize the data.
x_train = tf.keras.utils.normalize(x_train, axis = 1)
x_test = tf.keras.utils.normalize(x_test, axis = 1)
print(x_train[0])
plt.imshow(x_train[0], cmap = plt.cm.binary)
```



Grey Scale Image (x_train[0])



Normalized Image (x_train[0])

Manchester Metropolitan University

Google Colab (MNIST Dataset)

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())    # The input layer.
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))   # The 1st hidden layer with RELU activation.
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))   # The 2nd hidden layer with RELU activation.
model.add(tf.keras.layers.Dense(10, activation = tf.nn.softmax)) # The number of classifications with softmax activation.

model.compile(optimizer='adam',
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3)
```

```
Train on 60000 samples
Epoch 1/3
60000/60000 [==============================] - 5s 88us/sample - loss: 0.2625 - acc: 0.9244
Epoch 2/3
60000/60000 [==============================] - 5s 81us/sample - loss: 0.1072 - acc: 0.9664
Epoch 3/3
60000/60000 [==============================] - 5s 81us/sample - loss: 0.0732 - acc: 0.9767
<tensorflow.python.keras.callbacks.History at 0x7f70142ad860>
```

```
val_loss, val_acc = model.evaluate(x_test, y_test)
print(val_loss, val_acc)
```

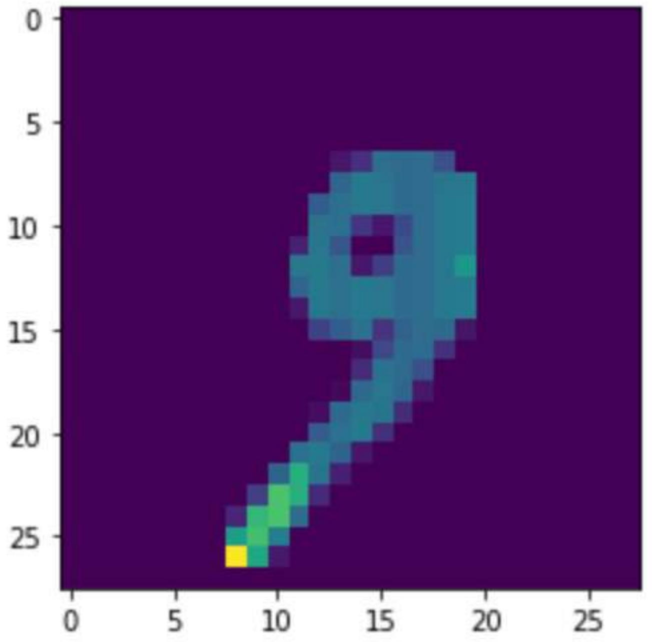
```
10000/10000 [==============================] - 0s 34us/sample - loss: 0.1104 - acc: 0.9657
0.1104153299085796 0.9657
```

Google Colab (MNIST Dataset)

```python
predictions = model.predict([x_test])
print(predictions)
```

```python
import numpy as np
index = 1000
print(np.argmax(predictions[index]))
plt.imshow(x_test[index])
plt.show()
```

9



Google Colab (CNN MNIST Dataset)

CNN MINIST.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] import tensorflow as tf
    import matplotlib.pyplot as plt
```

```
[ ] mnist = tf.keras.datasets.mnist # Digits 0-9, 28x28= pixels
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
[ ] # Normalize the data.
    x_train = tf.keras.utils.normalize(x_train, axis = 1)
    x_test = tf.keras.utils.normalize(x_test, axis = 1)
    x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], x_train.shape[2], 1))
    x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], x_test.shape[2], 1))
```

Google Colab (CNN MNIST Dataset)

```
[ ]  # Add convolution layers
     input_shape=(28,28,1)
     inputs = tf.keras.layers.Input(shape=input_shape)    # The input layer.
     layer = tf.keras.layers.Conv2D(filters=64, kernel_size=(5,5), strides=(2,2), activation=tf.nn.relu)(inputs)
     layer = tf.keras.layers.Conv2D(filters=64, kernel_size=(5,5), strides=(2,2), activation=tf.nn.relu)(layer)
     layer = tf.keras.layers.Flatten()(layer)
     layer = tf.keras.layers.Dense(128, activation = tf.nn.relu)(layer)   # The 1st hidden layer with RELU activation.
     layer = tf.keras.layers.Dense(128, activation = tf.nn.relu)(layer)  # The 2nd hidden layer with RELU activation.
     outputs = tf.keras.layers.Dense(10, activation = tf.nn.softmax)(layer) # The number of classifications with softmax activation.
```

```
# Run the model.
model = tf.keras.Model(inputs, outputs)
model.summary()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3)
```

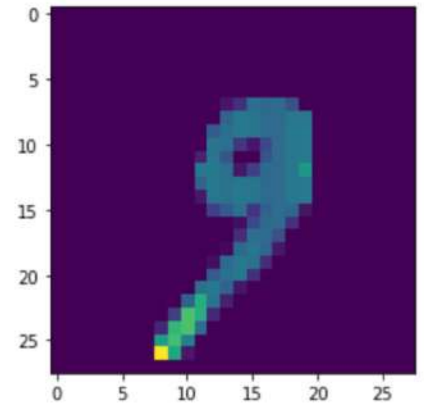
Google Colab (CNN MNIST Dataset)

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        1664
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 64)          102464
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               131200
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 253,130
Trainable params: 253,130
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/3
60000/60000 [==============================] - 33s 558us/sample - loss: 0.1717 - acc: 0.9465
Epoch 2/3
60000/60000 [==============================] - 33s 547us/sample - loss: 0.0608 - acc: 0.9814
Epoch 3/3
60000/60000 [==============================] - 32s 537us/sample - loss: 0.0412 - acc: 0.9872
<tensorflow.python.keras.callbacks.History at 0x7efbbac3fc50>
```

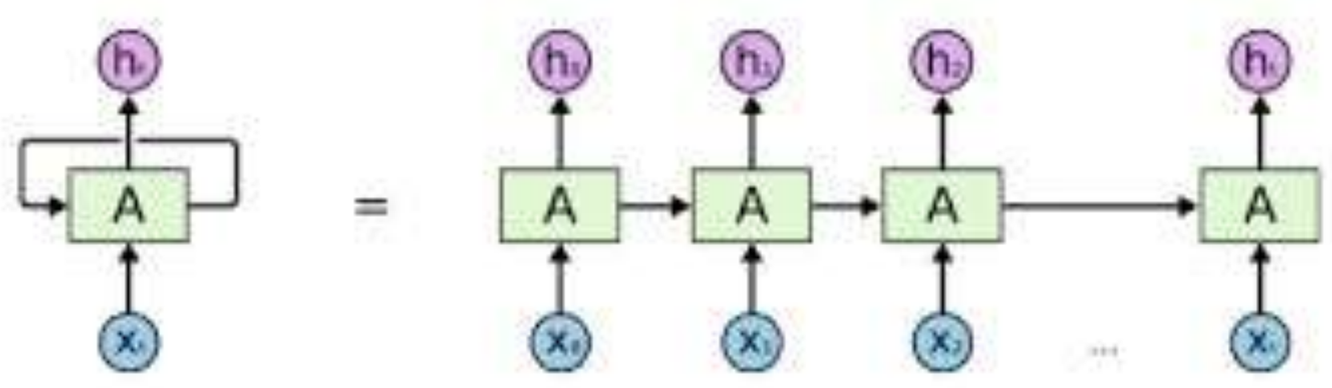
Google Colab (CNN MNIST Dataset) : End Session 2

```python
x = x_test.reshape((x_test.shape[0], x_test.shape[1], x_test.shape[2], 1))
predictions = model.predict([x_test])
print(predictions)
import numpy as np
index = 1000
print(np.argmax(predictions[index]))
plt.imshow(x_test[index].reshape((28,28)))
plt.show()
```

```
[[3.91767618e-10 3.75979825e-09 4.56916780e-08 ... 9.99998450e-01
  4.04364231e-09 1.31759430e-06]
 [7.06394231e-12 1.56008383e-07 9.99999881e-01 ... 5.07569098e-10
  4.52388370e-11 3.62123864e-14]
 [1.08492145e-07 9.99971986e-01 7.53841096e-06 ... 5.89023466e-06
  8.70545534e-07 8.48745231e-07]
 ...
 [4.42824692e-08 3.24178254e-05 1.39420010e-07 ... 8.49580756e-05
  4.52493041e-05 6.18437247e-04]
 [4.69727501e-09 1.78956447e-10 3.33151770e-11 ... 1.50632440e-09
  3.29844079e-05 2.30408276e-10]
 [3.05831890e-07 4.55387106e-09 1.44481149e-07 ... 1.76121354e-10
  8.38539762e-08 5.32932765e-09]]
9
```



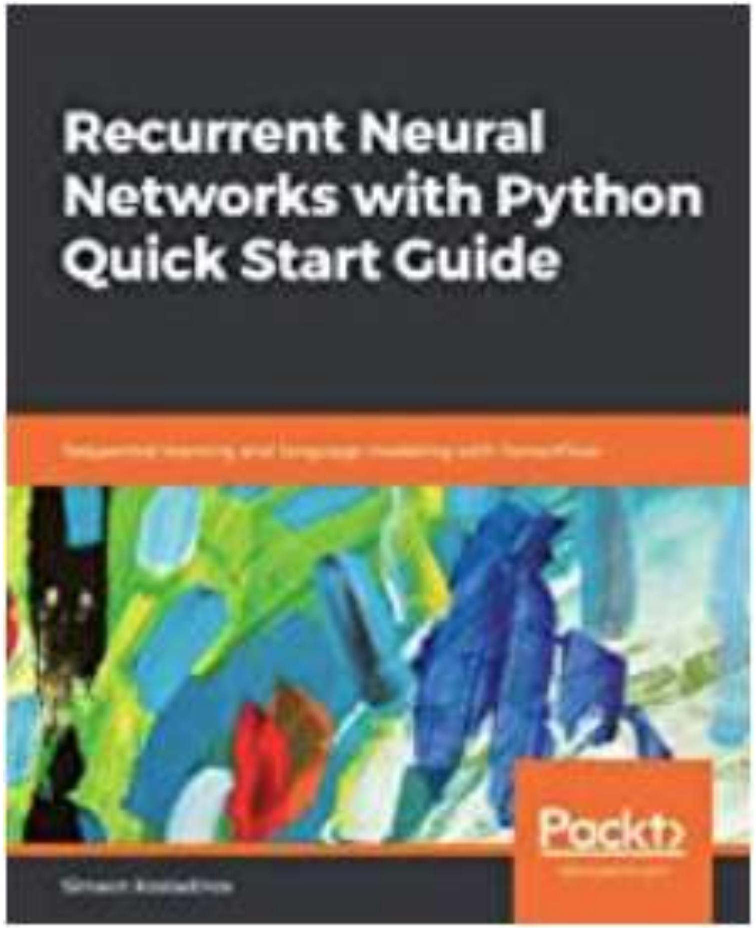
Recurrent Neural Network (RNN): Start Session 3

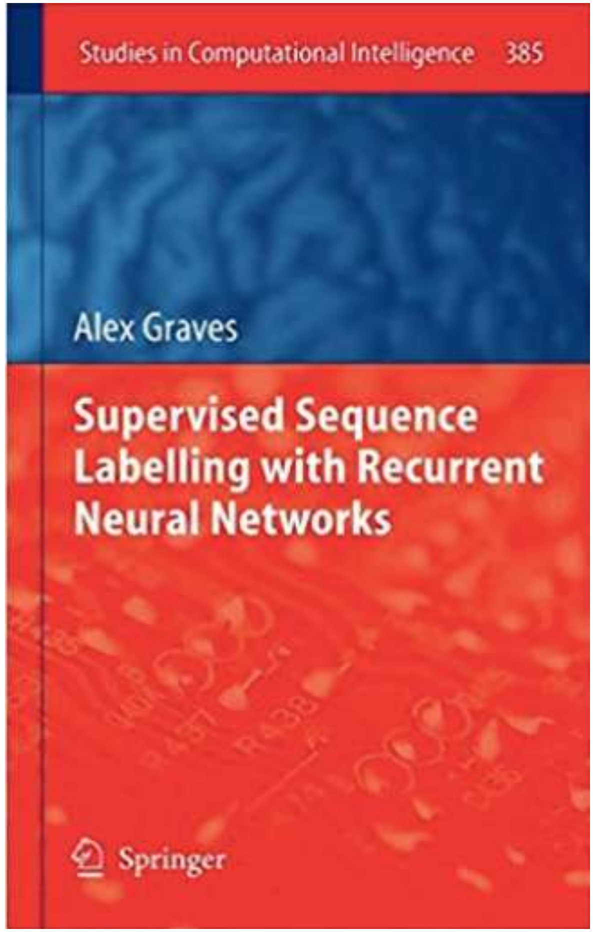


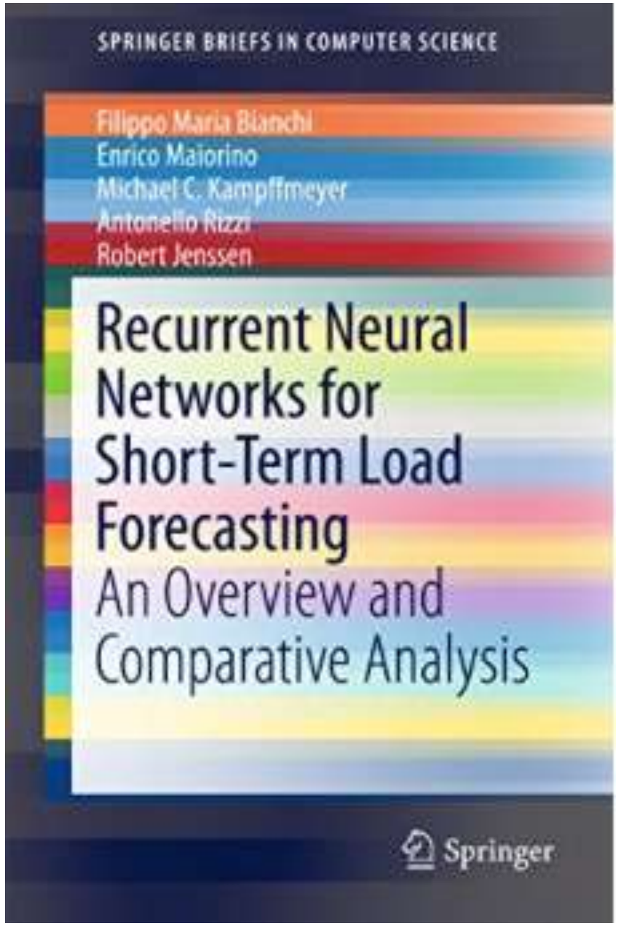
An unrolled recurrent neural network.

A **Recurrent Neural Network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour.

Recurrent Neural Networks Books

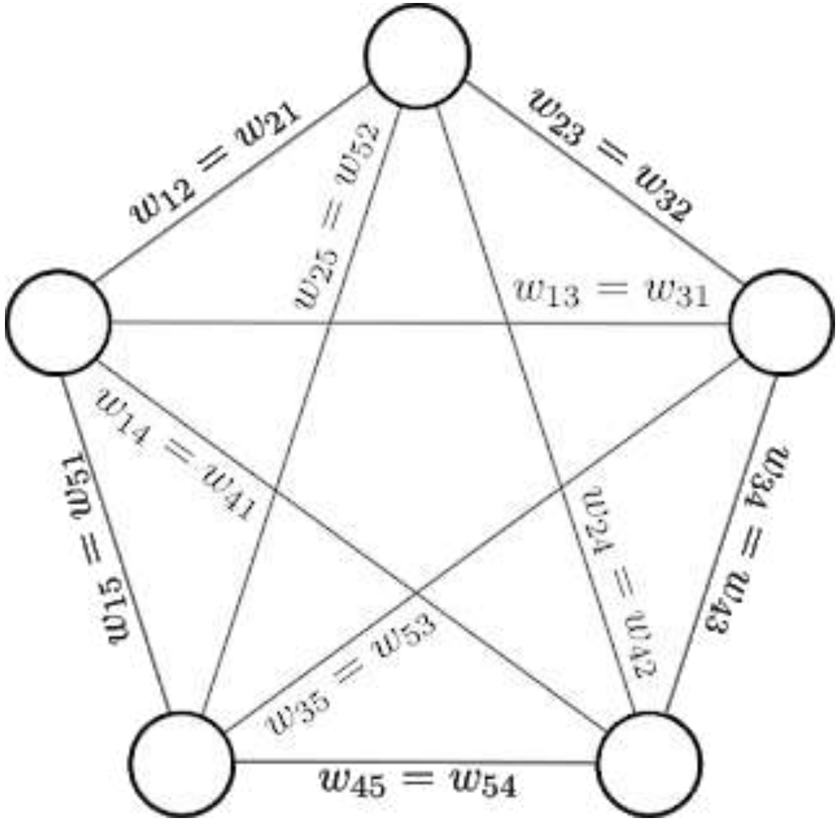






Recurrent Neural Network: The Hopfield Neural Network

John J Hopfield



RNN: The Discrete Hopfield Model

1. **Hebb's Postulate of Learning.** Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$ denote a set of N-dimensional fundamental memories. The synaptic weights of the network are determined using the formula

$$\mathbf{W} = \frac{1}{N} \sum_{r=1}^{M} \mathbf{x}_r \mathbf{x}_r^T - \frac{M}{N} \mathbf{I}_n$$

where $\mathbf{I}_n$ is the $N \times N$ identity matrix. Once computed, the synaptic weights remain fixed.

2. **Initialization.** Let $\mathbf{x}_p$ denote the unknown probe vector to be tested. The algorithm is initialized by setting

$$x_i(0) = x_{ip}, \quad i = 1, 2, \ldots, N,$$

where $x_i(0)$ is the state of neuron i at time $n = 0$, x_{ip} is the ith element of vector $\mathbf{x}_p$, and N is the number of neurons.

RNN: The Discrete Hopfield Model

3. **Iteration.** The elements are updated asynchronously (i.e., one at a time in a random order) according to the rule

$$x_i(n+1) = \text{hsgn}\left(\sum_{j=1}^{N} w_{ij}x_j(n)\right), i = 1, 2, \ldots, N,$$

where

$$\text{hsgn}(v_i(n+1)) = \begin{cases} 1, & v_i(n+1) > 0 \\ x_i(n), & v_i(n+1) = 0 \\ -1, & v_i(n+1) < 0 \end{cases}$$

and $v_i(n+1) = \sum_{j=1}^{N} w_{ij}x_j(n)$. The iterations are repeated until the vector converges to a stable value. Note that at least N iterations are carried out to guarantee convergence.

4. **Result.** The stable vector, say, $\mathbf{x}_{\text{fixed}}$, is the result.

RNN: The Discrete Hopfield Model

Example 5. A five-neuron discrete Hopfield network is required to store the following fundamental memories:

$$\mathbf{x}_1 = (1, 1, 1, 1, 1)^T, \quad \mathbf{x}_2 = (1, -1, -1, 1, -1)^T, \quad \mathbf{x}_3 = (-1, 1, -1, 1, 1)^T.$$

(a) Compute the synaptic weight matrix $\mathbf{W}$.

(b) Use asynchronous updating to show that the three fundamental memories are stable.

(c) Test the following vectors on the Hopfield network (the random orders affect the outcome):

$$\mathbf{x}_4 = (1, -1, 1, 1, 1)^T, \quad \mathbf{x}_5 = (0, 1, -1, 1, 1)^T, \quad \mathbf{x}_6 = (-1, 1, 1, 1, -1)^T.$$

RNN: The Discrete Hopfield Model

Solution. (a) The synaptic weight matrix is given by

$$\mathbf{W} = \frac{1}{5}\left(\mathbf{x}_1\mathbf{x}_1^T + \mathbf{x}_2\mathbf{x}_2^T + \mathbf{x}_3\mathbf{x}_3^T\right) - \frac{3}{5}\mathbf{I}_5,$$

so

$$\mathbf{W} = \frac{1}{5}\begin{pmatrix} 0 & -1 & 1 & 1 & -1 \\ -1 & 0 & 1 & 1 & 3 \\ 1 & 1 & 0 & -1 & 1 \\ 1 & 1 & -1 & 0 & 1 \\ -1 & 3 & 1 & 1 & 0 \end{pmatrix}.$$

(b) Step 1. First input vector, $\mathbf{x}_1 = \mathbf{x}(0) = (1,1,1,1,1)^T$.

Step 2. Initialize $x_1(0) = 1, x_2(0) = 1, x_3(0) = 1, x_4(0) = 1, x_5(0) = 1$.

Step 3. Update in random order $x_3(1), x_4(1), x_1(1), x_5(1), x_2(1)$, one at a time.

$$\begin{aligned} x_3(1) &= \text{hsgn}(0.4) = 1, \\ x_4(1) &= \text{hsgn}(0.4) = 1, \\ x_1(1) &= \text{hsgn}(0) = x_1(0) = 1, \\ x_5(1) &= \text{hsgn}(0.8) = 1, \\ x_2(1) &= \text{hsgn}(0.8) = 1. \end{aligned}$$

Thus $\mathbf{x}(1) = \mathbf{x}(0)$ and the net has converged.

Step 4. The net has converged to the steady state $\mathbf{x}_1$.

RNN: The Discrete Hopfield Model

Step 1. Sixth input vector, $\mathbf{x}_6 = \mathbf{x}(0) = (-1, 1, 1, 1, -1)^T$.

Step 2. Initialize $x_1(0) = -1, x_2(0) = 1, x_3(0) = 1, x_4(0) = 1, x_5(0) = -1$.

Step 3. Update in random order $x_3(1), x_2(1), x_5(1), x_4(1), x_1(1)$, one at a time.

$$\begin{aligned}
x_3(1) &= \text{hsgn}(-0.4) = -1, \\
x_2(1) &= \text{hsgn}(-0.4) = -1, \\
x_5(1) &= \text{hsgn}(-0.4) = -1, \\
x_4(1) &= \text{hsgn}(-0.4) = -1, \\
x_1(1) &= \text{hsgn}(0) = x_1(0) = -1.
\end{aligned}$$

Step 3 (again). Update in random order $x_2(1), x_1(1), x_5(1), x_4(1), x_3(1)$, one at a time.

$$\begin{aligned}
x_2(2) &= \text{hsgn}(-0.8) = -1, \\
x_1(2) &= \text{hsgn}(0) = x_1(1) = -1, \\
x_5(2) &= \text{hsgn}(-0.8) = -1, \\
x_4(2) &= \text{hsgn}(-0.4) = -1, \\
x_3(2) &= \text{hsgn}(-0.4) = -1.
\end{aligned}$$

Thus $\mathbf{x}(2) = \mathbf{x}(1)$ and the net has converged.

Step 4. The net has converged to the spurious steady state $-\mathbf{x}_1$.

RNN: The Discrete Hopfield Model

Example 6. Write a Python program that illustrates the behavior of the discrete Hopfield network as a content-addressable memory using $N = 81$ neurons and the set of handcrafted patterns displayed in Figure 20.12.

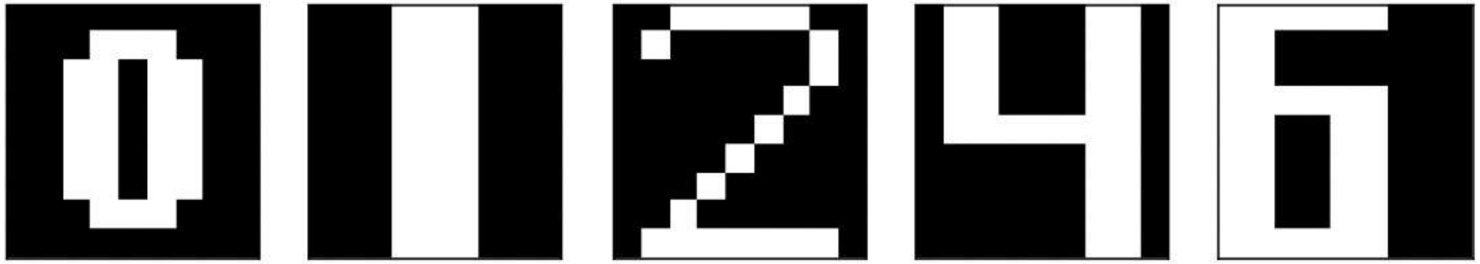


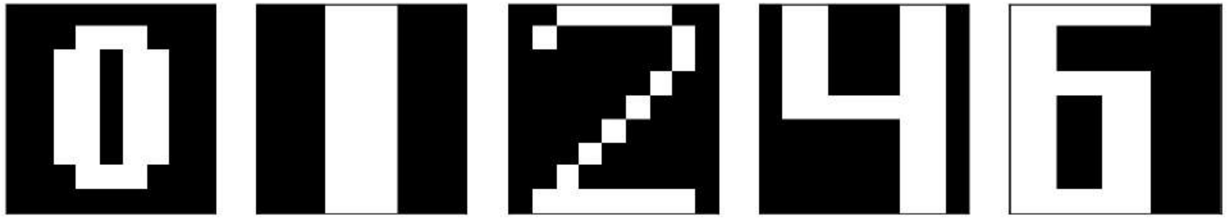
Figure 20.12: Five handcrafted patterns.

RNN: The Discrete Hopfield Model

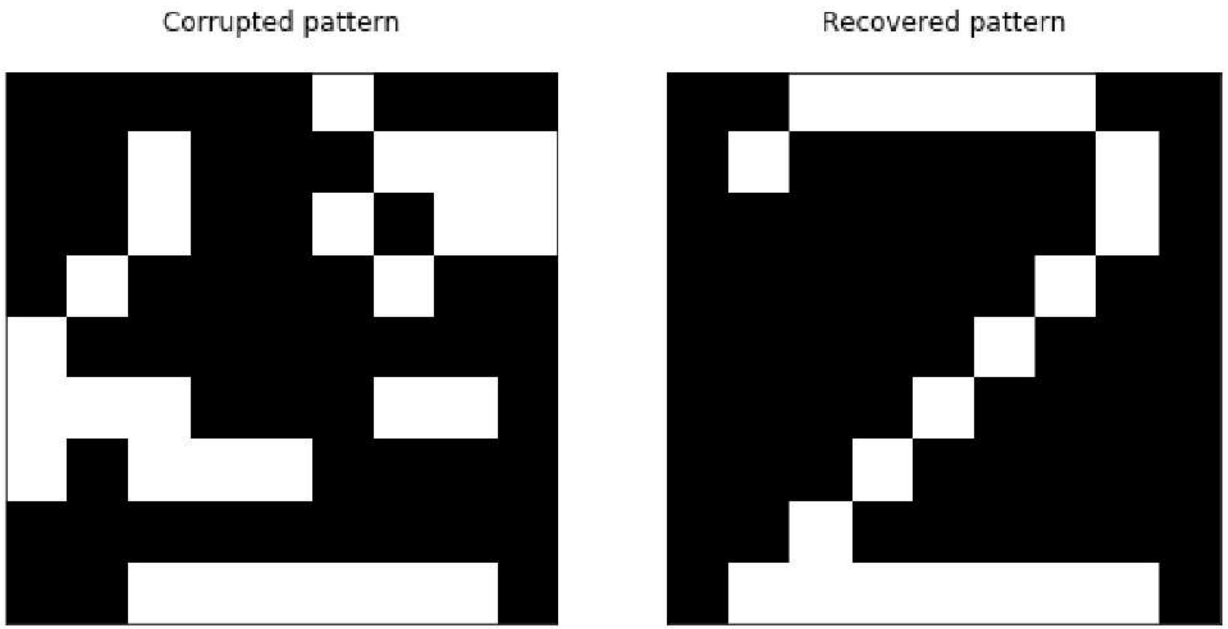
```python
# Hopfield Model

import matplotlib.pyplot as plt
import numpy as np
import random


nb_patterns = 5
pattern_width = 9
pattern_height = 9
max_iterations = 81

# Initialize the patterns
X = np.zeros((nb_patterns, pattern_width * pattern_height))

X[0] = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1, 1, -1,
 1, 1, -1, -1, -1, -1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1, 1, -1, 1, 1, -1, -1, -1, -1, -1, 1, 1,
 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
X[1] = [-1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1,
 -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1,
 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1]
X[2] = [-1, -1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1,   -1, -1, -1, -1, 1, -1,  -1, -1, -1, -1, -1, -1, -1, 1, -1,  -1, -1, -1, -1,
 -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, 1, -1,  -1, -1,   -1, -1, -1, -1, 1, -1, -1, -1,-1,   -1, -1, -1, 1, -1, -1, -1,-1, -1,
 -1, -1,1, -1, -1,  -1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, 1, -1]
X[3] = [-1, 1, 1, -1, -1, -1, 1, 1, -1,  -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, 1, -1, -1, -1, 1,
 1, -1, -1,  1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1,  -1, -1, -1, -1, -1,
 -1, 1, 1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1]
X[4] = [1, 1, 1, 1, 1, 1, -1, -1, -1,  1, 1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1, -1, -1, -1, -1, -1, -1,  1, 1, 1, 1, 1, 1, -1,
 -1, -1, 1, 1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, 1, -1, -1, -1,  1, 1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, 1, -1,
 -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1]

# Show the patterns
fig, ax = plt.subplots(1, nb_patterns, figsize=(10, 5))

for i in range(nb_patterns):
    ax[i].matshow(X[i].reshape((pattern_height, pattern_width)), cmap='gray')
    ax[i].set_xticks([])
    ax[i].set_yticks([])

plt.show()
```

RNN: The Discrete Hopfield Model

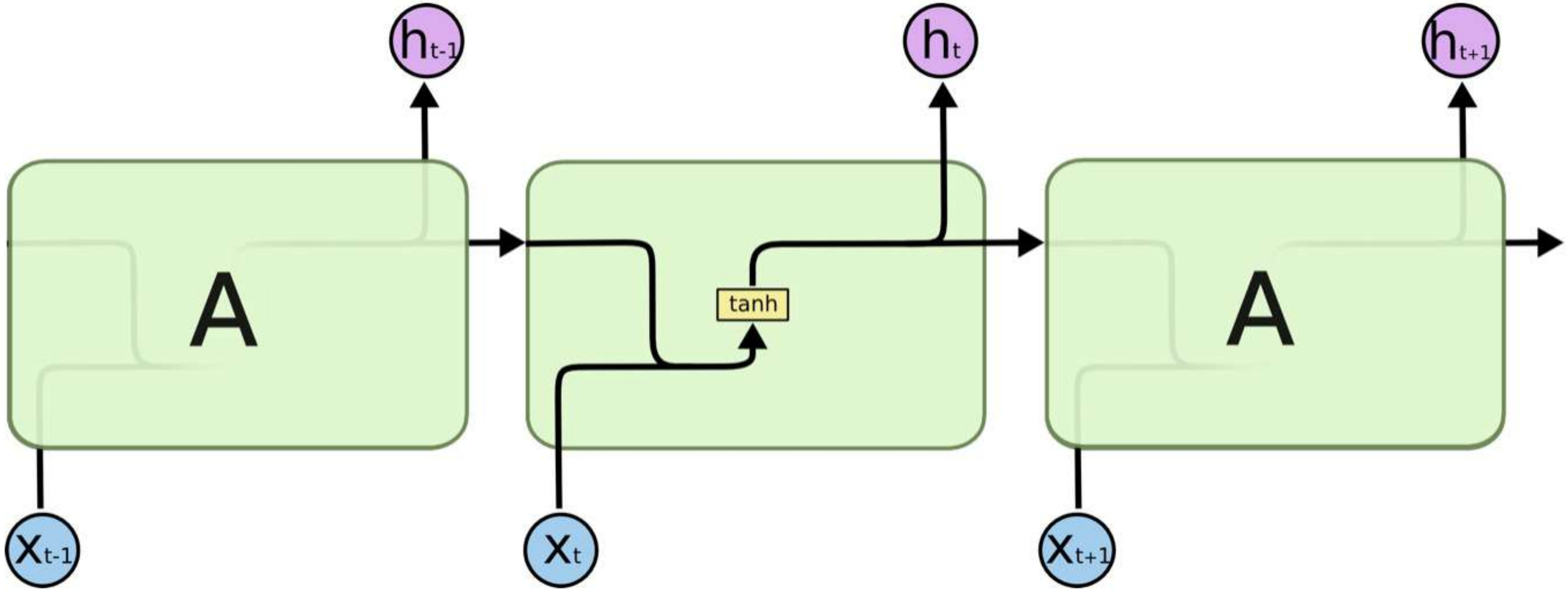
```python
W = ((np.outer(X[0],X[0])+np.outer(X[1],X[1])+np.outer(X[2],X[2])+np.outer(X[3],X[3])+np.outer(X[4],X[4]))-5*np.identity(81))/81

def hsgn(v, x):
    if v>0:
        return 1
    elif v == 0:
        return x
    else:
        return -1

# Create a corrupted test pattern

noiselevel = 1/3
values = list(range(nb_patterns))
patInd = random.choice(values)
Y = np.array(X[patInd])
x_test = np.array((2*(np.random.rand(81, 1).flatten() > noiselevel)-1)*Y)
x_test.flatten()
print('Pattern index=',patInd)

# Recover the original patterns
A = x_test.copy()
A.flatten()

n=np.random.permutation(81)

for _ in range(max_iterations):
    for j in range(81):
        A[n[j]]=hsgn(np.dot(W[n[j]],A), A[n[j]])


# Show corrupted and recovered patterns
fig, ax = plt.subplots(1, 2, figsize=(10, 5))

ax[0].matshow(x_test.reshape(pattern_height, pattern_width), cmap='gray')
ax[0].set_title('Corrupted pattern')
ax[0].set_xticks([])
ax[0].set_yticks([])

ax[1].matshow(A.reshape(pattern_height, pattern_width), cmap='gray')
ax[1].set_title('Recovered pattern')
ax[1].set_xticks([])
ax[1].set_yticks([])

plt.show()
```



```
Pattern index= 2
```

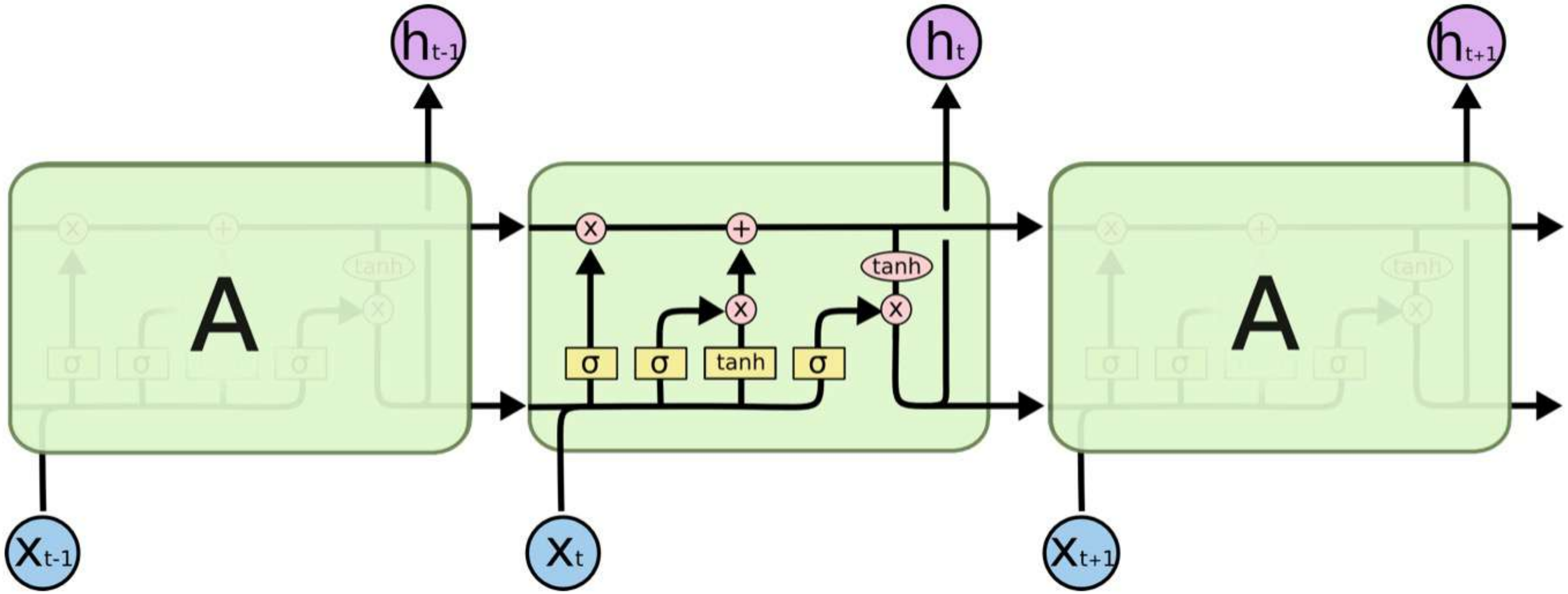


RNN: Long Short Term Memory Networks

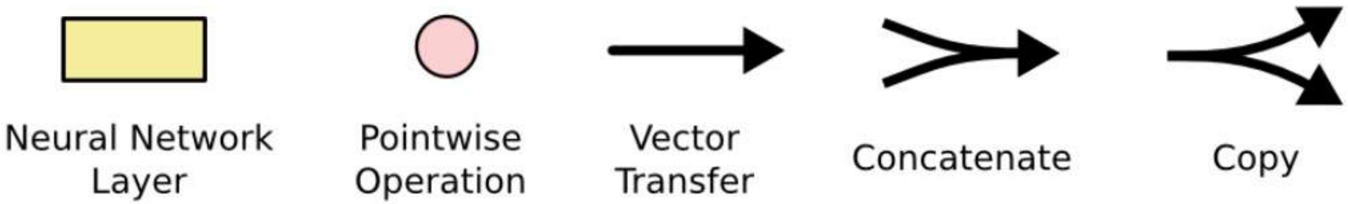


The repeating module in a standard RNN contains a single layer.

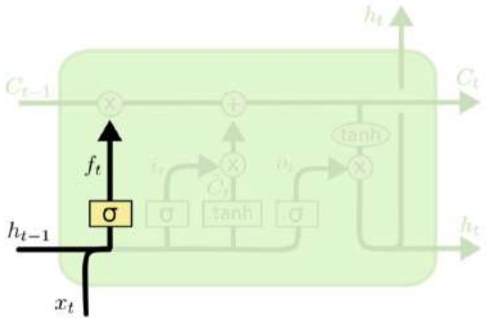
RNN: Long Short Term Memory Networks



The repeating module in an LSTM contains four interacting layers.

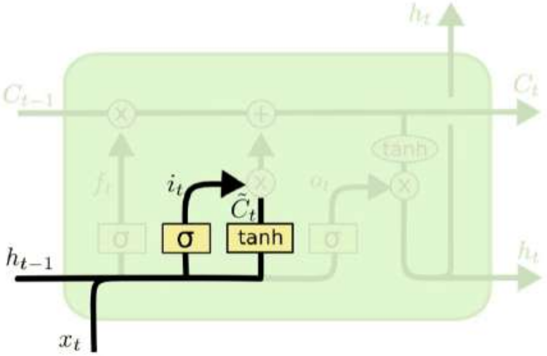


RNN: Long Short Term Memory Networks



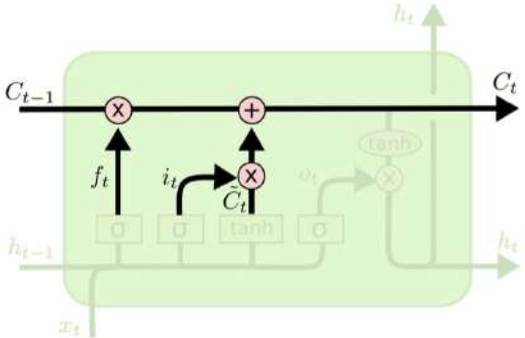
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure (a): Forget gate layer.



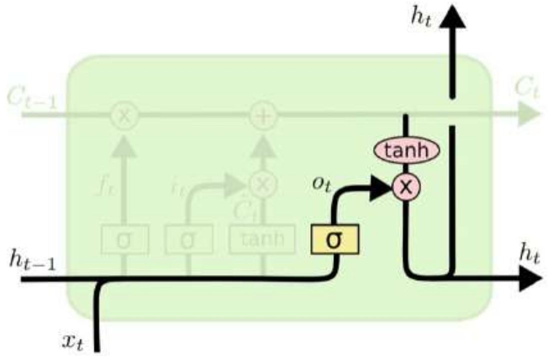
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure (b): Update the state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure (c): The new cell state.



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure (d): Decide on output.

RNN: Long Short Term Memory Time Series Prediction

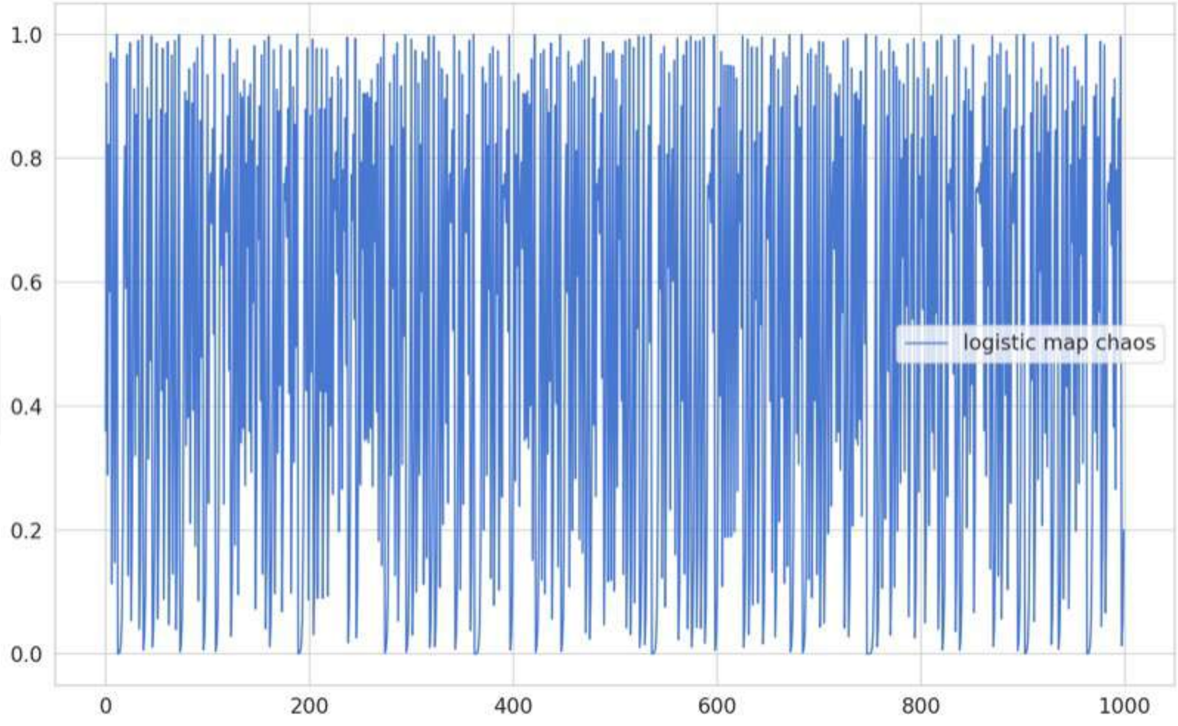
```
[ ] import tensorflow as tf
    from tensorflow import keras
    import pandas as pd
    import numpy as np
    import seaborn as sns
    from pylab import rcParams
    import matplotlib.pyplot as plt
    from matplotlib import rc

    %matplotlib inline
    %config InlineBackend.figure_format='retina'
    sns.set(style='whitegrid', palette='muted', font_scale=1.5)
    rcParams['figure.figsize'] = 16, 10
    RANDOM_SEED = 42
    np.random.seed(RANDOM_SEED)
```

RNN: Long Short Term Memory Time Series Prediction

```
[2] x = 0.1
    chaos = []
    for t in range(1000):
      x = 4 * x * (1 - x)
      chaos = np.append(chaos,x)

    time = np.arange(0, 100, 0.1)
    plt.plot(chaos, label='logistic map chaos')
    plt.legend();
```

```
[3] df = pd.DataFrame(dict(chaos=chaos), index=time, columns=['chaos'])
    df.head()
```

	chaos
0.0	0.360000
0.1	0.921600
0.2	0.289014
0.3	0.821939
0.4	0.585421



RNN: Long Short Term Memory Time Series Prediction

```
[4] train_size = int(len(df) * 0.8)
    test_size = len(df) - train_size
    train, test = df.iloc[0:train_size], df.iloc[train_size:len(df)]
    print(len(train), len(test))
```

```
800 200
```

```
[5] def create_dataset(X, y, time_steps=1):
        Xs, ys = [], []
        for i in range(len(X) - time_steps):
            v = X.iloc[i:(i + time_steps)].values
            Xs.append(v)
            ys.append(y.iloc[i + time_steps])
        return np.array(Xs), np.array(ys)
```

RNN: Long Short Term Memory Time Series Prediction

```
[6] time_steps = 10

    # reshape to [samples, time_steps, n_features]

    X_train, y_train = create_dataset(train, train.chaos, time_steps)
    X_test, y_test = create_dataset(test, test.chaos, time_steps)

    print(X_train.shape, y_train.shape)

    (790, 10, 1) (790,)
```

```
[7] model = keras.Sequential()
    model.add(keras.layers.LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(keras.layers.Dense(1))
    model.compile(loss='mean_squared_error', optimizer=keras.optimizers.Adam(0.001))
```

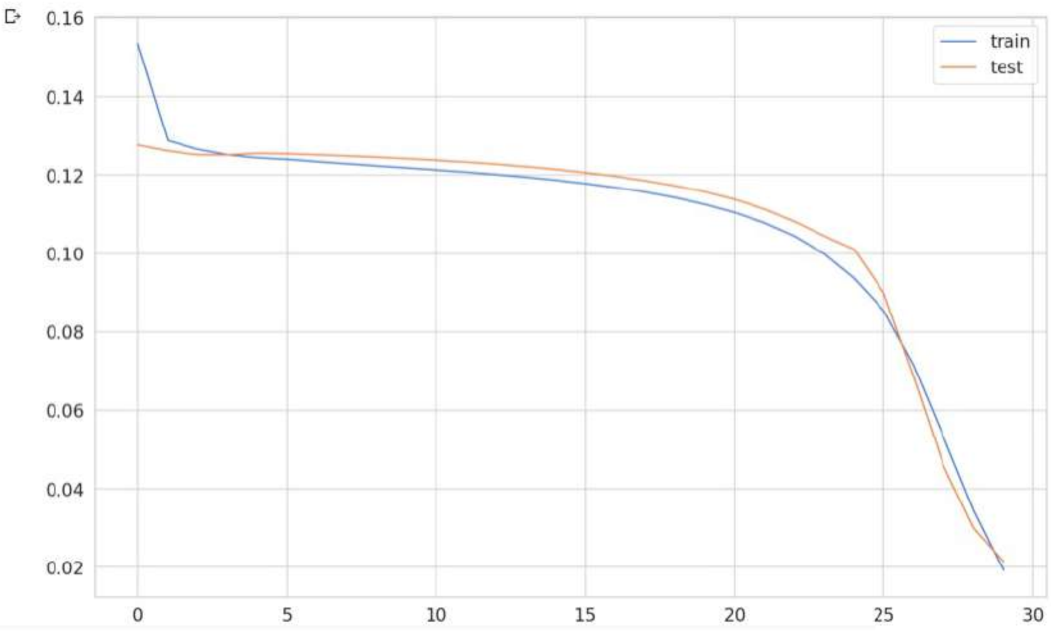
RNN: Long Short Term Memory Time Series Prediction

```
[8] history = model.fit(X_train, y_train, epochs=30, batch_size=16,
                        validation_split=0.1, verbose=1, shuffle=False)
```

```
Epoch 1/30
711/711 [==============================] - 1s 2ms/sample - loss: 0.1535 - val_loss: 0.1277
Epoch 2/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1288 - val_loss: 0.1261
Epoch 3/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1266 - val_loss: 0.1252
Epoch 4/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1251 - val_loss: 0.1251
Epoch 5/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1244 - val_loss: 0.1255
Epoch 6/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1240 - val_loss: 0.1254
Epoch 7/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1234 - val_loss: 0.1252
Epoch 8/30
711/711 [==============================] - 1s 1ms/sample - loss: 0.1229 - val_loss: 0.1249
```

RNN: Long Short Term Memory Time Series Prediction

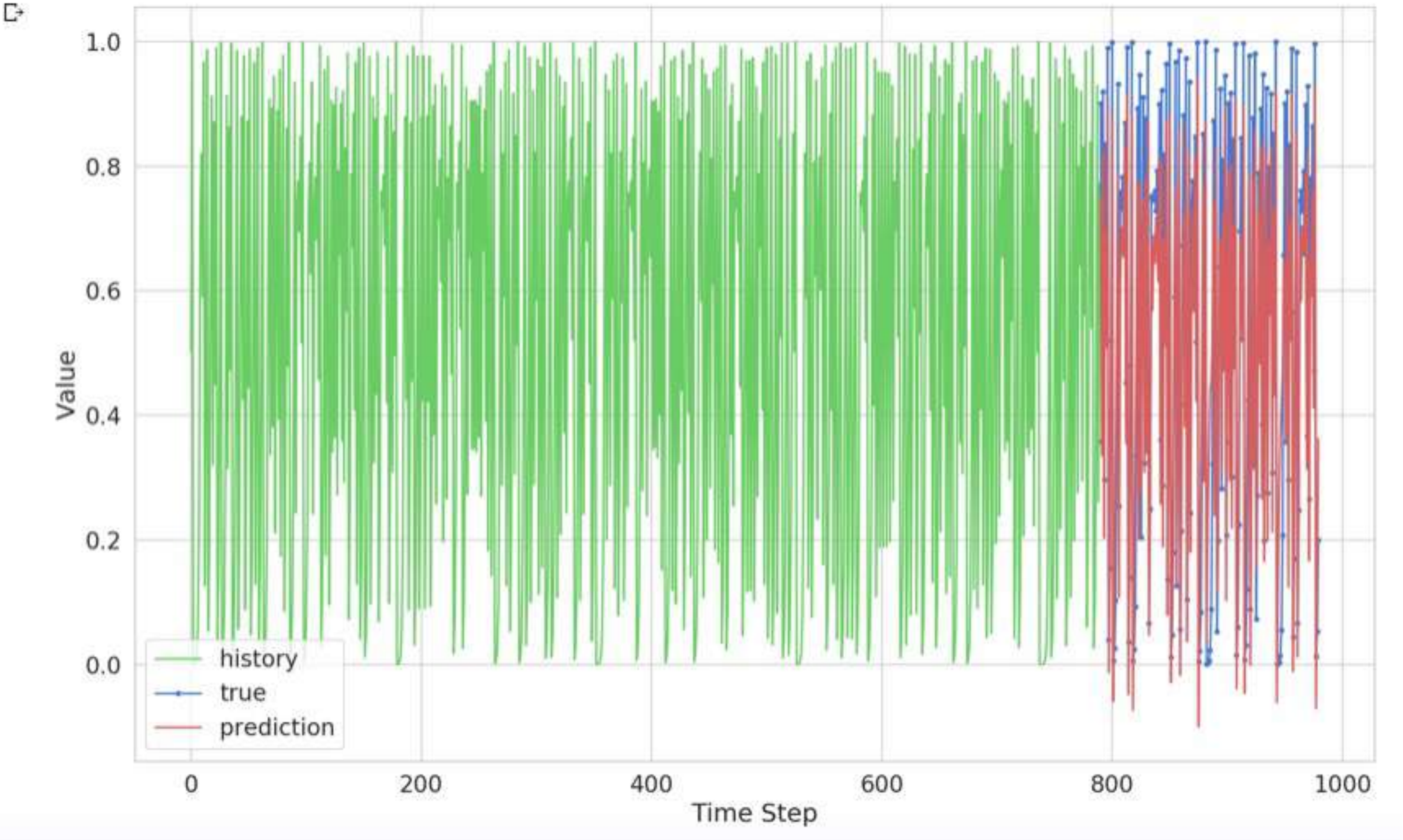
```
[9]  plt.plot(history.history['loss'], label='train')
     plt.plot(history.history['val_loss'], label='test')
     plt.legend();
```




```
[10] y_pred = model.predict(X_test)
```

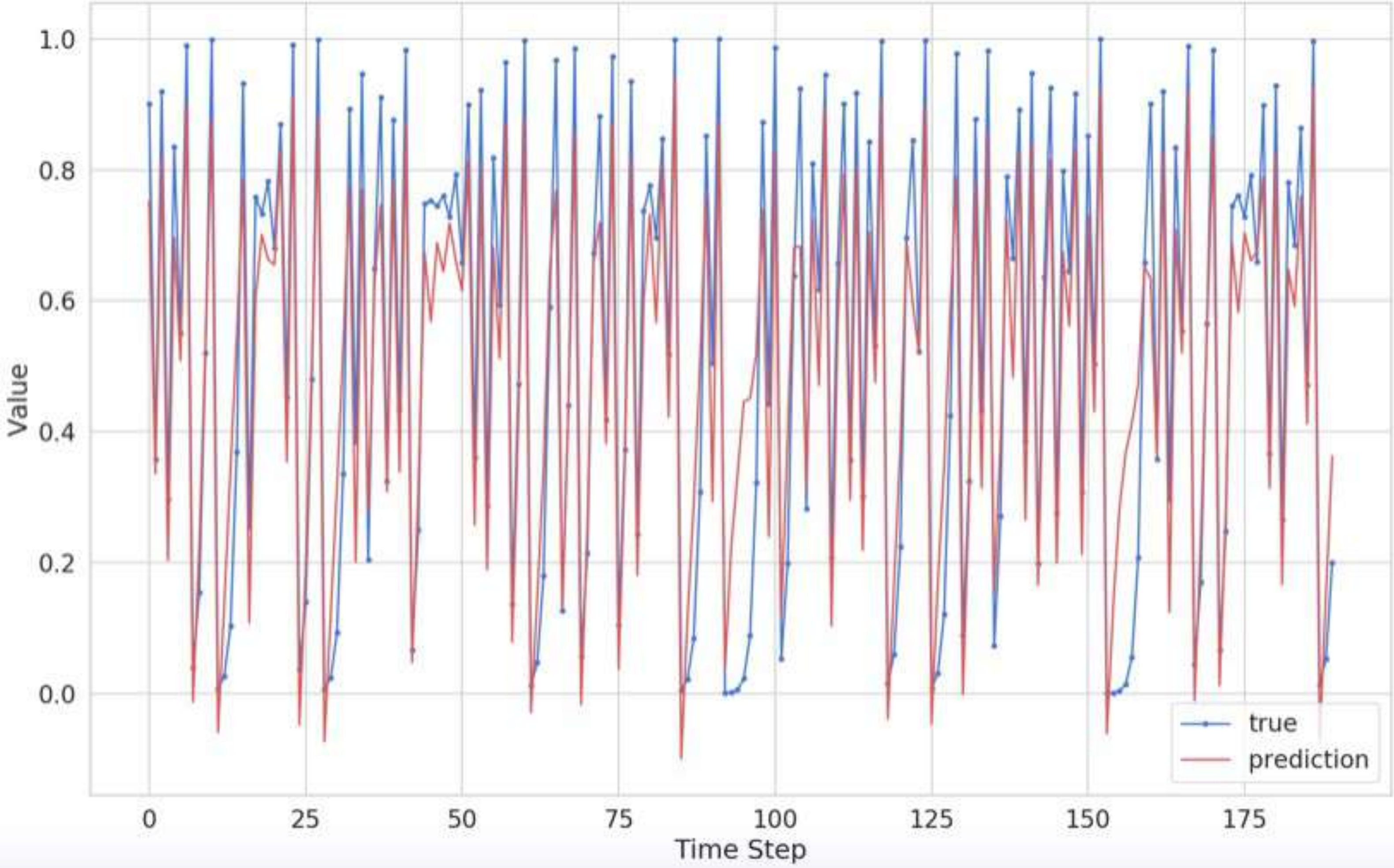
```
[11] plt.plot(np.arange(0, len(y_train)), y_train, 'g', label="history")
     plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_test, marker='.', label="true")
     plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_pred, 'r', label="prediction")
     plt.ylabel('Value')
     plt.xlabel('Time Step')
     plt.legend()
     plt.show();
```

RNN: Long Short Term Memory Time Series Prediction



RNN: Long Short Term Memory Time Series Prediction

```
[12] plt.plot(y_test, marker='.', label="true")
     plt.plot(y_pred, 'r', label="prediction")
     plt.ylabel('Value')
     plt.xlabel('Time Step')
     plt.legend()
     plt.show()
```



RNN: LSTM and Financial Mathematics: End Session 3

Run the Python notebook LSTM_TS_Forecast_US_EUR_Exchange_Rate.ipynb through GitHub.

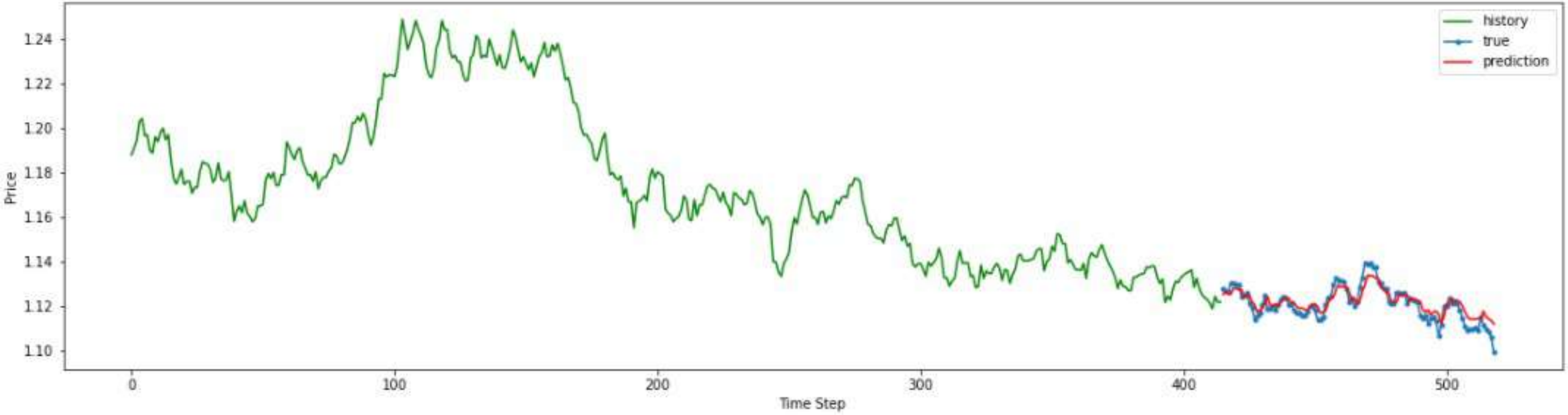


Figure: Using LSTM to predict the US/EUR exchange rate.

An Introduction to TensorBoard: MNIST Dataset: Start Session 4

```
[1] try:
      %tensorflow_version 2.x
    except Exception:
      pass
    # Load the TensorBoard notebook extension
    %load_ext tensorboard
```

```
TensorFlow 2.x selected.
```

```
[2] import tensorflow as tf
    import datetime
```

```
[3] # Clear any logs from previous runs
    !rm -rf ./logs/
```

```
[4] mnist = tf.keras.datasets.mnist

    (x_train, y_train),(x_test, y_test) = mnist.load_data()
    x_train, x_test = x_train / 255.0, x_test / 255.0

    def create_model():
      return tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
      ])
```

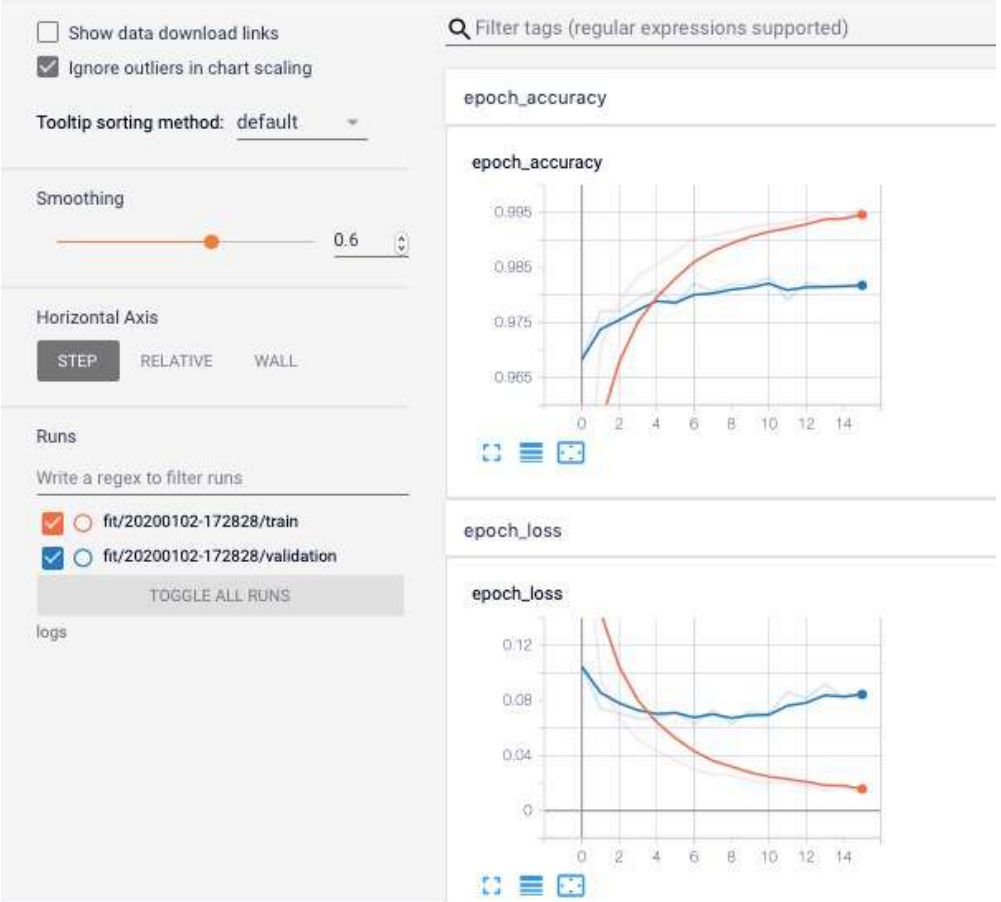
An Introduction to TensorBoard

```
[5] model = create_model()
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

    model.fit(x=x_train, y=y_train, epochs=16, validation_data=(x_test, y_test), callbacks=[tensorboard_callback])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/16
60000/60000 [==============================] - 11s 180us/sample - loss: 0.2224 - accuracy: 0.9337 - val_loss: 0.1046 - val_accuracy: 0.9683
Epoch 2/16
60000/60000 [==============================] - 11s 177us/sample - loss: 0.0969 - accuracy: 0.9704 - val_loss: 0.0742 - val_accuracy: 0.9771
Epoch 3/16
60000/60000 [==============================] - 11s 182us/sample - loss: 0.0665 - accuracy: 0.9787 - val_loss: 0.0708 - val_accuracy: 0.9771

                    . . .                 . . .                 . . .

Epoch 14/16
60000/60000 [==============================] - 11s 176us/sample - loss: 0.0147 - accuracy: 0.9951 - val_loss: 0.0919 - val_accuracy: 0.9816
Epoch 15/16
60000/60000 [==============================] - 10s 175us/sample - loss: 0.0177 - accuracy: 0.9940 - val_loss: 0.0815 - val_accuracy: 0.9817
Epoch 16/16
60000/60000 [==============================] - 11s 176us/sample - loss: 0.0124 - accuracy: 0.9957 - val_loss: 0.0867 - val_accuracy: 0.9820
<tensorflow.python.keras.callbacks.History at 0x7fa77ac7a5f8>
```

An Introduction to TensorBoard

```
%tensorboard --logdir=logs
```

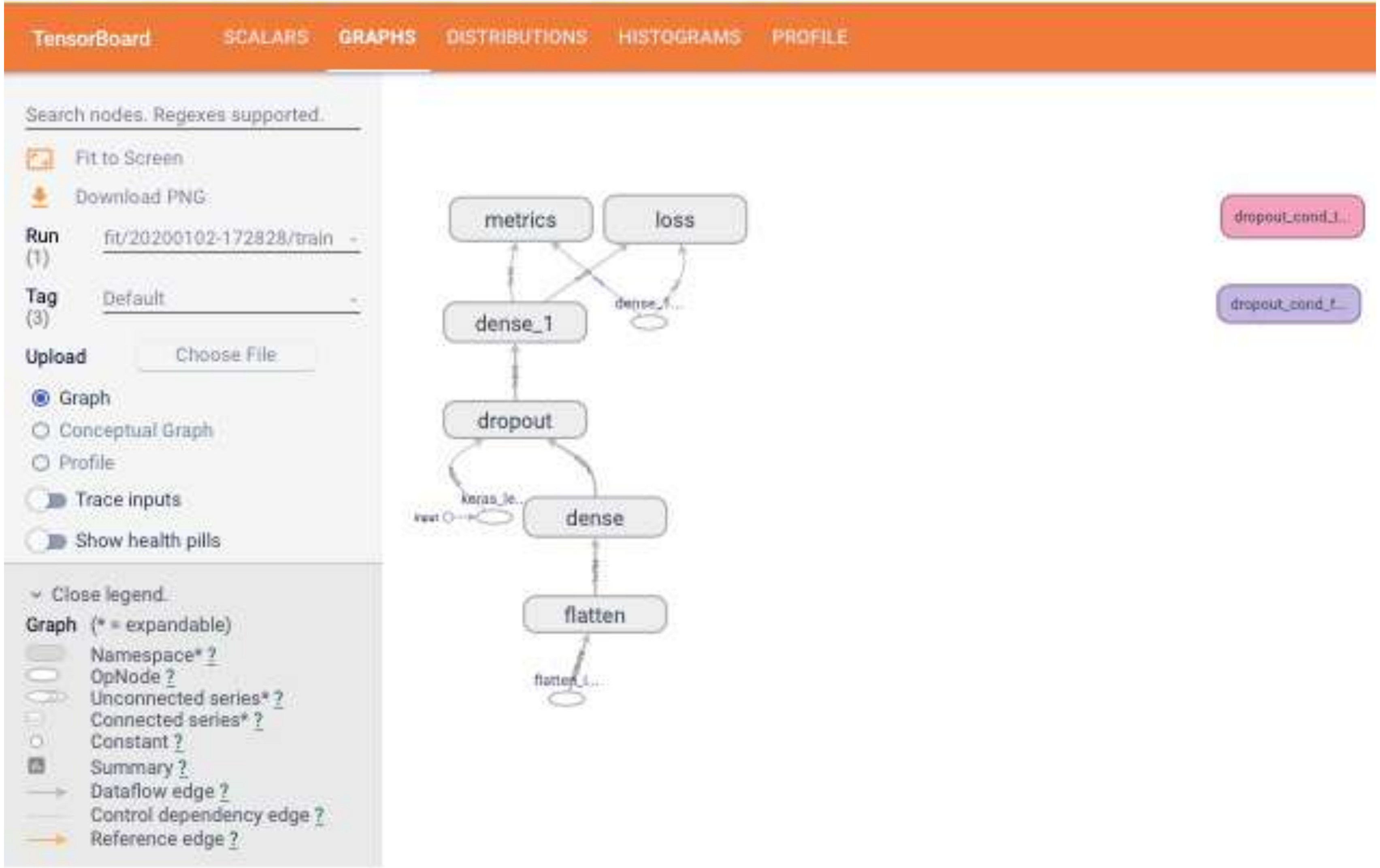


Run this command to get both curves!

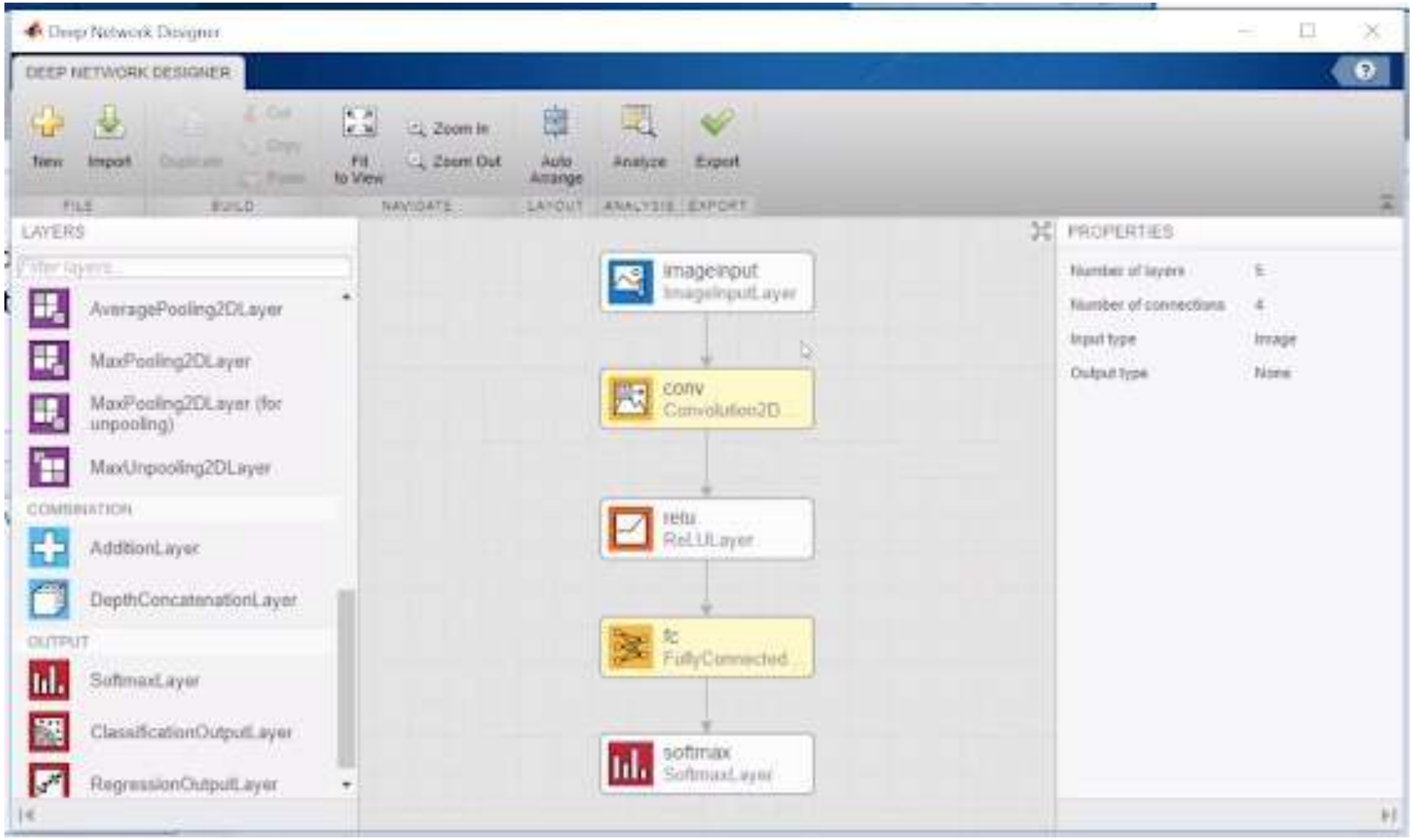
```
[7] !kill 4696
```

Manchester Metropolitan University

TensorBoard Graphs



MATLAB® Deep Learning Toolbox

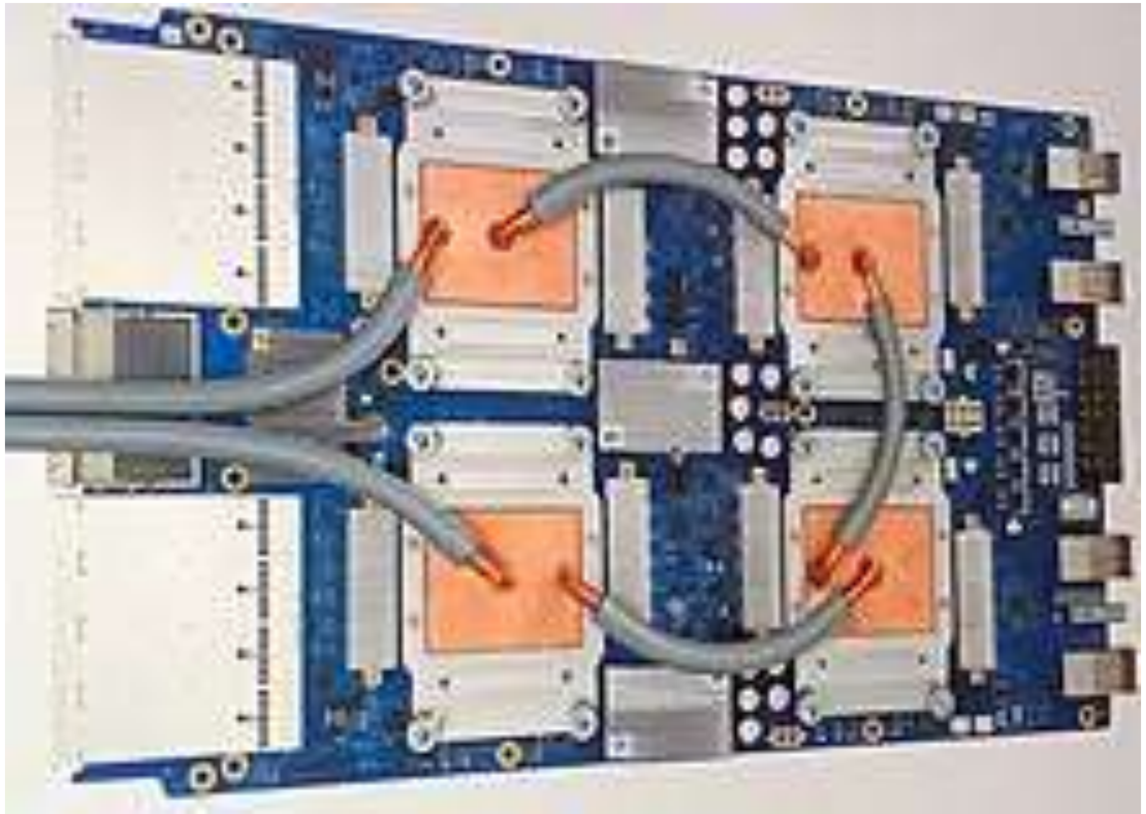


https://uk.mathworks.com/videos/what-is-deep-learning-toolbox--1535667599631.html

Google Colab and the Tensor Processing Unit (TPU)

A Tensor Processing Unit (TPU) is an AI Accelerator application-Specific Integrated Circuit (ASIC) developed By Google for Deep Learning using TensorFlow.

TensorFlow
2.0



Google Colab and the Tensor Processing Unit

TPUs in Colab

In this example, we'll work through training a model to classify images of flowers on Google's lightning-fast Cloud TPUs. Our model will take as input a photo of a flower and return whether it is a daisy, dandelion, rose, sunflower, or tulip.

We use the Keras framework, new to TPUs in TF 2.1.0. Adapted from [this notebook] by [Martin Gorner].

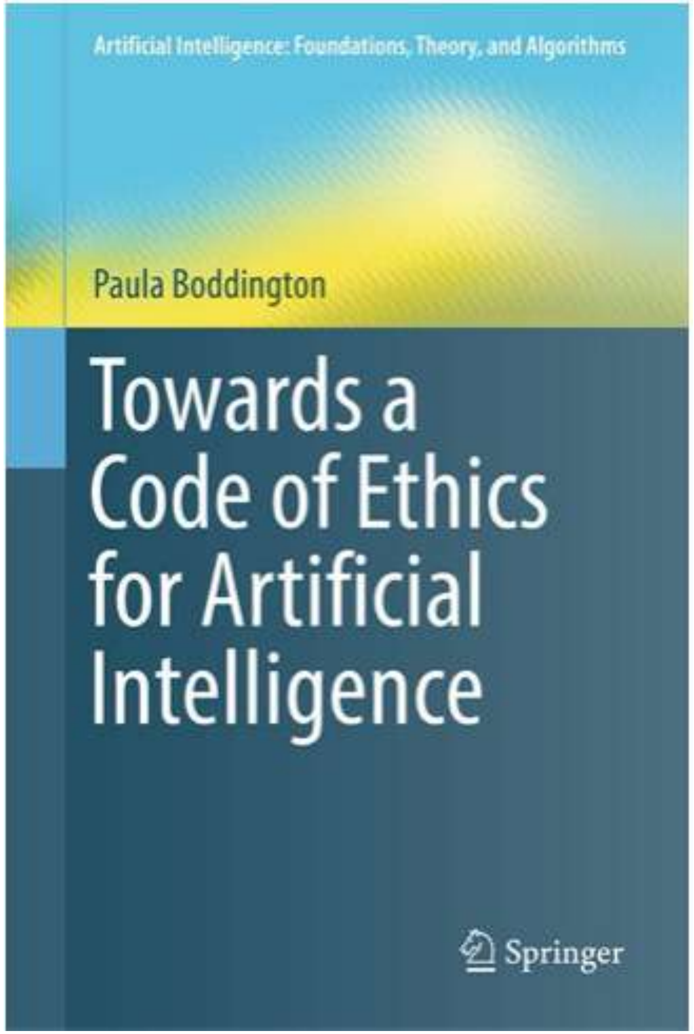
Manchester Metropolitan University

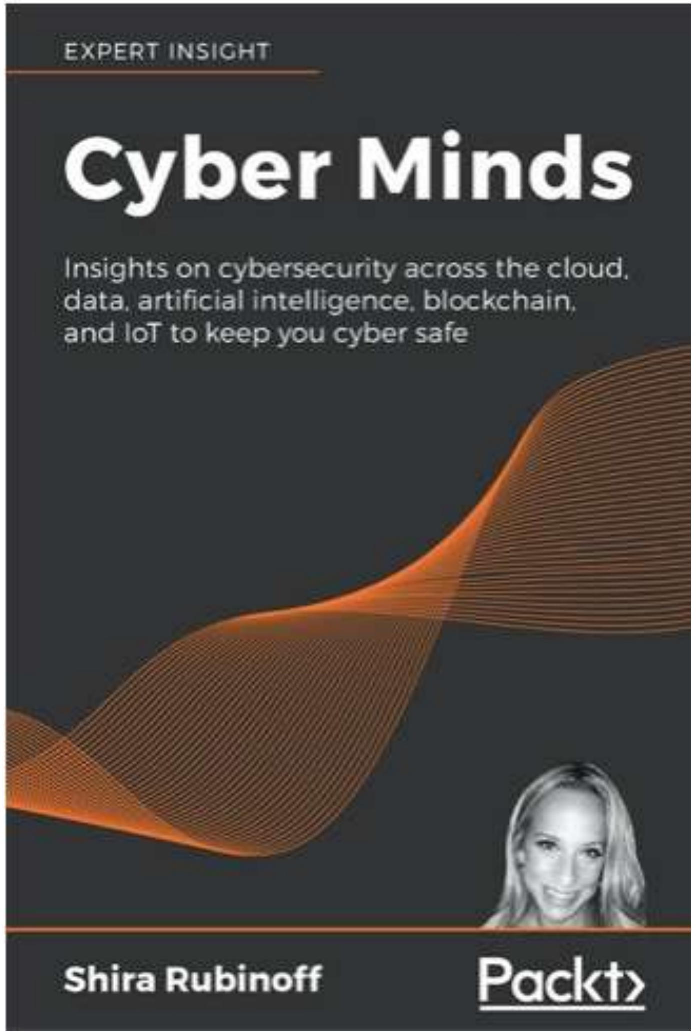
Google Colab and the Tensor Processing Unit

Using AI to write Shakespeare!

```
ake thee of thy sweet self dost see,
From heaven thee, as the beauty of thy didge?
Then were thou art my love whose soor coll, and she vounes,
That in my stars in his praise the ever wor,
Whose whould his spiret the deser thee is bart,
  And thou thy self dost thou mayst live in thee
  Then do I not the wrose to deepile lease.

The worthous shalt be bland nor my seas,
With pentter than the owness doth bear,
Where that beauty like of many a forming.
Thou art as find in that which the thing thee,
```

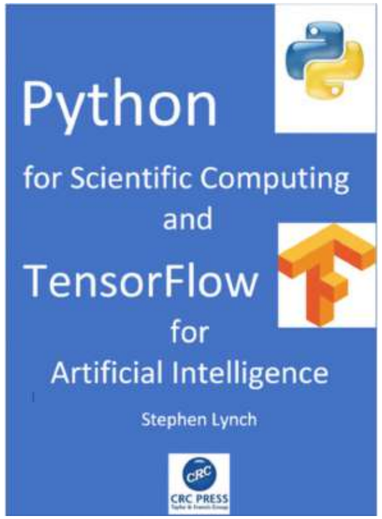
AI Ethics and Cyber Security





End Day 5 Summary

Day 5			
Topics	**Hours**	**Topics**	**Hours**
AI: KERAS and TensorFlow	10am-11am	AI: Recurrent Neural Networks	1pm-2pm
AI: Convolutional Neural Networks	11am-12pm	AI: Introduction to TensorBoard	2pm-3pm



Application Programming Interface (API)

https://github.com/DrStephenLynch/Tekbac

https://keras.io/api/applications/