機器學習研究應用
Study for Machine Learning and Its Applications

Getting Started with Neural Networks I

孫士韋

Shih-Wei Sun

swsun@newmedia.tnua.edu.tw

<ENGRAM : DATA SCULPTURE FOR MELTING MEMORIES> (2018)

- Artist: Refik ANADOL, Turkey
- 《消融記憶》，雷菲克．安納多爾 (土耳其)
  - https://vimeo.com/264369157
- Webpage: http://festival.dac.taipei/2020/artist-18.html
  - 台北數位藝術節 2020, 松山文創園區

Outline

- Basic concepts of neural network in deep learning
  - 3 key steps: add, compile, fit
- What is Keras?
- Binary classifier
  - Movie reviews as positive/negative

Basic concepts of neural network in deep learning

Keras vs. Tensorflow

- TensorFlow: machine learning platform
  - Python-based, free, open-source
- Keras: deep-learning API for Python,
  - built on top of TensorFlow

Both released on 2015

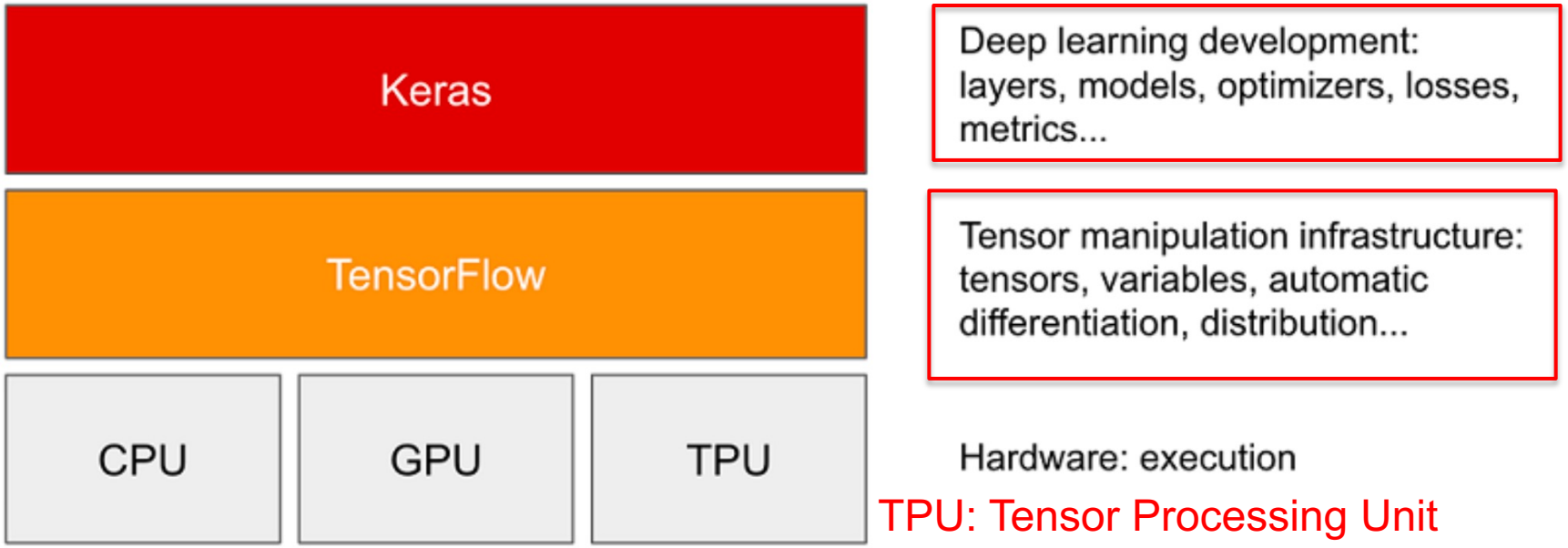


Figure 3.1 Keras and TensorFlow: TensorFlow is a low-level tensor computing platform, Keras is a high-level deep learning API

Anatomy of a Neural Network

- Layers: model of a neural network
- Input data
- Loss function
- Optimizer

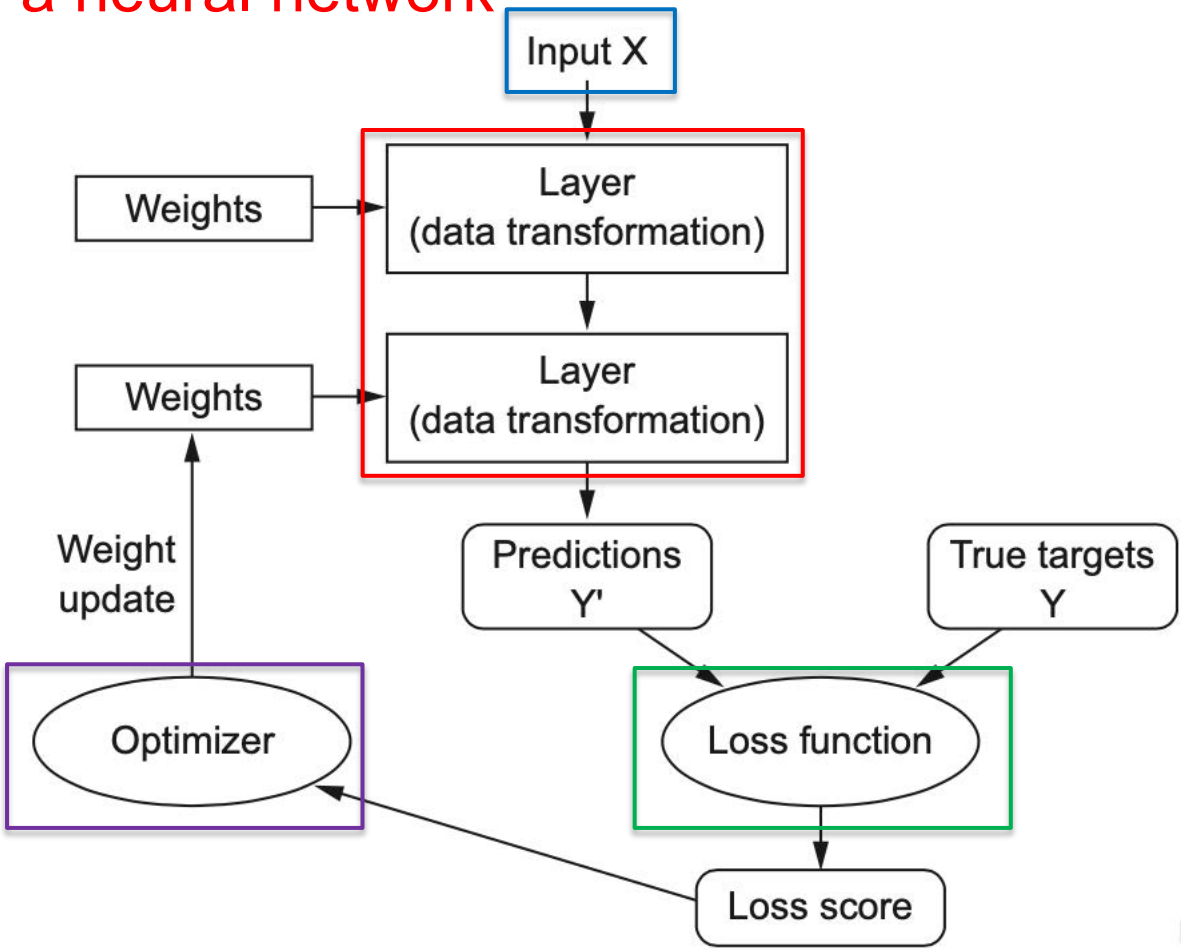


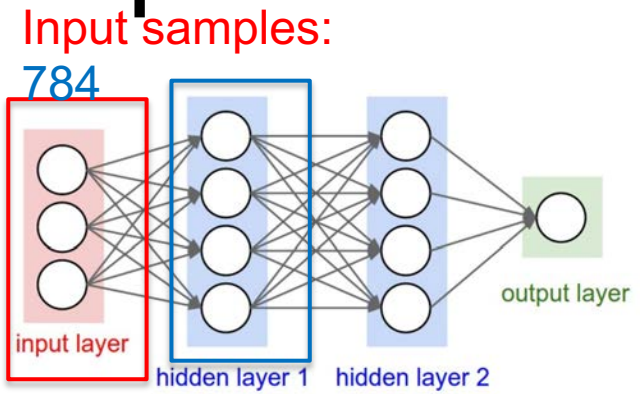
Figure 3.1 Relationship between the network, layers, loss function, and optimizer

Layers: the NN building blocks

- Layer: fundamental data structure
  - Data processing module
  - Input/output: tensors
- Vector data, 2D tensors (samples, features)
  - *fully connected* / *dense* layers
- Sequence data, 3D tensors (samples, timesteps, features),
  - *recurrent* layers / LSTM layer
- Image data, 4D tensors
  - 2D convolution layers / Conv2D

A simple layer example

- Layers: think as the LEGO bricks
- Layer compatibility
  - Every layer
    - Accept input tensor: a certain shape
    - Return output tensor: a certain shape



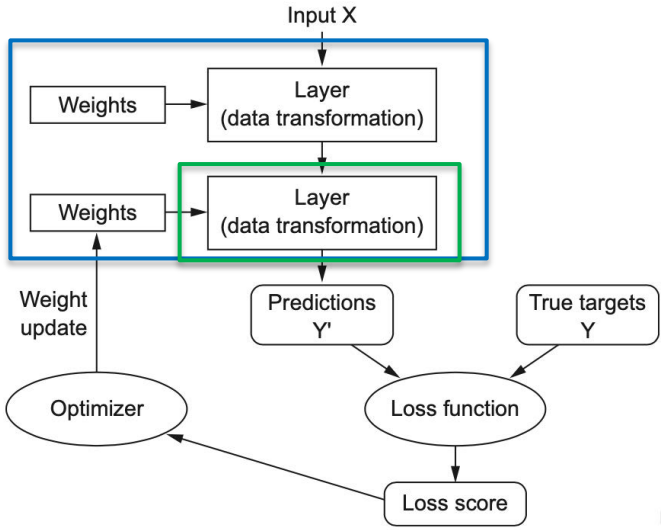

```
2  from tensorflow.keras import layers

5  model.add(layers.Dense(32, input_shape(784,)))
```

- Input of the layer: 784 (samples of data)
- Output of the layer: 32

Step 1: Define the model, Two layers

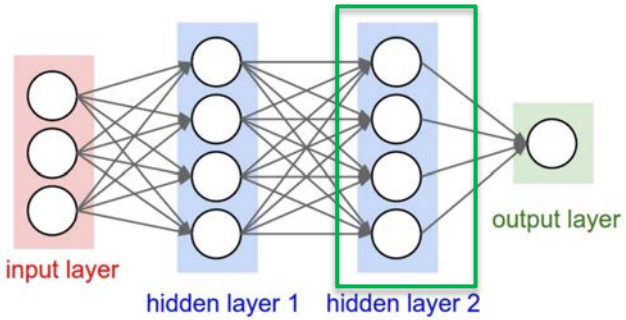
- Add two layers

```
1  from tensorflow.keras import models
2  from tensorflow.keras import layers
3
4  model = models.Sequential()
5  model.add(layers.Dense(32, input_shape(784,)
6  model.add(layers.Dense(32))
```



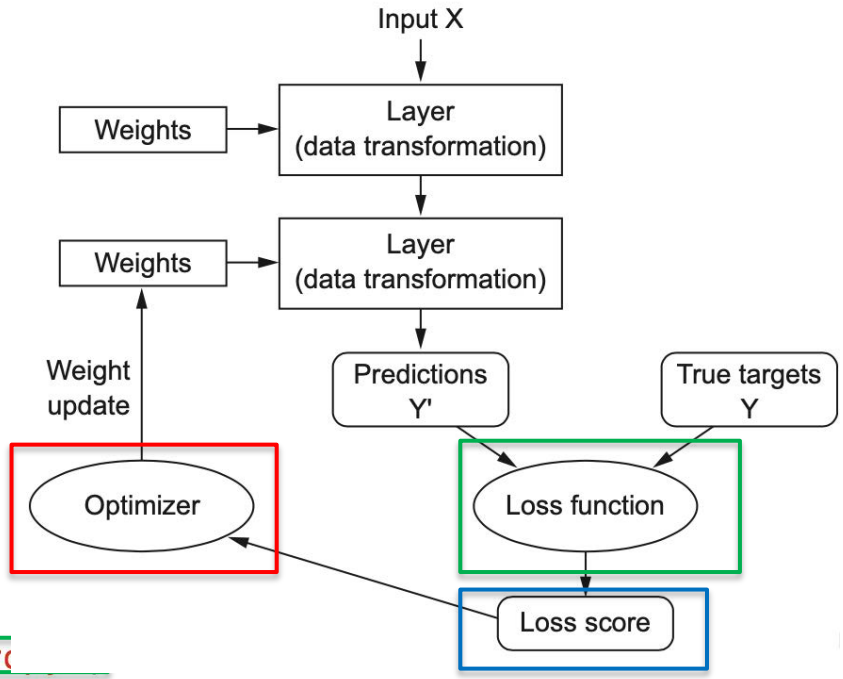
- Import models
- Declare the model
- Add another layer:
  - Automatically inferred
    - its input shape

Layer output: 32



Step 2: Configuring the Compilation Step

- Specify the
  - Optimizer
  - Loss function
  - Loss score

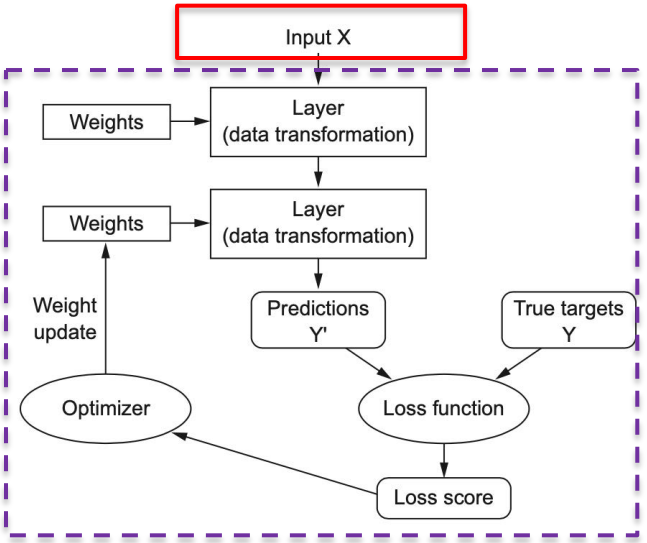



```
8  model.compile(optimizer="rmsprop",
9                loss="sparse_categorical_crossentro
10               metrics=["accuracy"])
```

Step 3: Fit the model

Wait!
We don't have input data yet…

- Before:
  - Built the neural network
- Here:
  - Execute the training process



Code:

```
1  from tensorflow.keras import models
2  from tensorflow.keras import layers
3
4  model = models.Sequential()
5  model.add(layers.Dense(32, input_shape(784,)))
6  model.add(layers.Dense(32))
7
8  model.compile(optimizer="rmsprop",
9                loss="sparse_categorical_crossentropy",
10               metrics=["accuracy"])
11
12 model.fit(input_tensor,
13           target_tensor,
14           epochs=5,
15           batch_size=128)
```

Result:

```
Traceback (most recent call last):
  File "/Users/sunshih-wei/Documents/python/03
    model.fit(input_tensor, target_tensor, bat
NameError: name 'input_tensor' is not defined
```

Summary of Neural networks in Keras

```
1  from tensorflow.keras import models
2  from tensorflow.keras import layers
3
4  model = models.Sequential()
5  model.add(layers.Dense(32, input_shape(784,)))
6  model.add(layers.Dense(32))
7
8  model.compile(optimizer="rmsprop",
9                loss="sparse_categorical_crossentropy",
10               metrics=["accuracy"])
11
12 model.fit(input_tensor,
13           target_tensor,
14           epochs=5,
15           batch_size=128)
```

Step 1: Define the model (add)

Step 2: Configure the Compilation (compile)

Step 3: Fit the model (fit)

3 steps: add, compile, fit

Ready to go ….

Summary

- Step1: add
- Step2: compile
- Step3: fit

```
4  model = models.Sequential()
5  model.add(layers.Dense(32, input_shape(784,)))
6  model.add(layers.Dense(32))
```

```
8  model.compile(optimizer="rmsprop",
9                loss="sparse_categorical_crossentropy",
10               metrics=["accuracy"])
```

```
12 model.fit(input_tensor,
13           target_tensor,
14           epochs=5,
15           batch_size=128)
```

Binary Classifier

Classifying movie reviews: a binary classification example

- The IMDB dataset: Internet Movie Database
  - A set of 50,000 highly polarized reviews
  - Split into
    - 25,000 for training
    - 25,000 for testing
  - Each set: 50% negative, 50% positive reviews
- Never test a machine learning model
  - On the same data
- Performance on new data
  - You should care about!

IMDB reviews

- Reviews turned into sequences of integers
  - Specific word in a dictionary
- Load the IMDB dataset

Num_words=10,000:
Only keep the top 10,000
Most frequently occurring words
in the training data

Code:

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels)=
    imdb.load_data(num_words=10000)

print(train_data[0])
print(train_labels[0])
```

Result:

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 2
56, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112,
167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16,
6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38
, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 3
16, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5,
 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407,
 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 53
0, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2
071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 3
0, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16,
 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
1
```

train_data, test_data:
- lists of reviews of word indices
- encoding a sequence of words

train_labels, test_labels:
- 0, negative
- 1, positive

A sequence of values:
representing review texts

Practice 1

- Print the data
  - train_data[1], [2], …[10]
  - train_labels[1], [2], …[10]
- Using a for-in loop

```
[1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 1
18, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 18
8, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35,
8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952
, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11,
 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 12
3, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299,
 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 23
50, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625
, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462
, 33, 89, 78, 285, 16, 145, 95]
0
```

IMDB reviews, showing the words

- Word_index: dictionary mapping words -> integer index
  - Reverse it, mapping integer indices to words
  - Decode reviews

Listing 3.13 Decoding newswires back to text

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in
    train_data[0]])
```

Code:

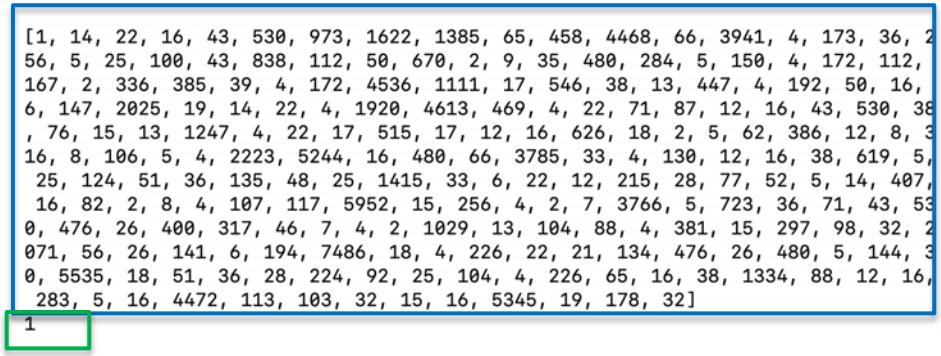
```
7  word_index = imdb.get_word_index()
8  reverse_word_index = dict([(value, key) for
       (key, value) in word_index.items()])
9  decoded_review = '
       '.join([reverse_word_index.get(i - 3, '?')
       for i in train_data[0]])
10
11 print(decoded_review)
```

Result: a positive review

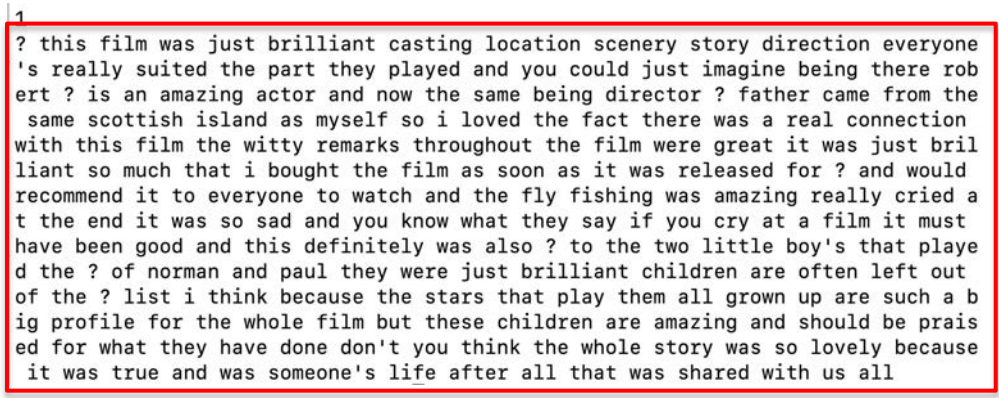
```
? this film was just brilliant casting location scenery story direction everyone
's really suited the part they played and you could just imagine being there rob
ert ? is an amazing actor and now the same being director ? father came from the
 same scottish island as myself so i loved the fact there was a real connection
with this film the witty remarks throughout the film were great it was just bril
liant so much that i bought the film as soon as it was released for ? and would
recommend it to everyone to watch and the fly fishing was amazing really cried a
t the end it was so sad and you know what they say if you cry at a film it must
have been good and this definitely was also ? to the two little boy's that playe
d the ? of norman and paul they were just brilliant children are often left out
of the ? list i think because the stars that play them all grown up are such a b
ig profile for the whole film but these children are amazing and should be prais
ed for what they have done don't you think the whole story was so lovely because
 it was true and was someone's life after all that was shared with us all
```

IMDB dataset

- You should keep in mind:
  - A sequence of values
  - Review texts



=



Preparing the data

- Changing the integer sequences -> binary vectors
  - One-hot encode: *sequence*
    - categorical variables as binary vectors (matrix)

Code:

```
14 import numpy as np
15 def vectorize_sequence(sequences, dimension=10000):
16     results= np.zeros((len(sequences), dimension))
17     for i, sequence in enumerate(sequences):
18         results[i, sequence] = 1.
19     return results
20
21 x_train = vectorize_sequence(train_data)
22 x_test = vectorize_sequence(test_data)
23
24 print(x_train[0])
```

Crates an all-zero matrix
of shape (len(sequences, dimension))

Sets specific indices
of results[i] to 1s
-> one-hot encode

Result:

```
[0. 1. 1. ... 0. 0. 0.]
```

Vectorized training / test data

Vectorize the labels

- Convert the values to ‘float32’
  - Used for the first ‘Dense’ layer
  - Straightforward

Code:

```
25
26 y_train = np.asarray(train_labels).astype('float32')
27 y_test = np.asarray(test_labels).astype('float32')
28 print(y_train)
29
```

Result:

```
[1. 0. 0. ... 0. 1. 0.]
```

Ready to be fed into a neural network

Step 1: Define the model, Building the network

- The model definition
  - Activation function

relu:
rectified linear unit
zero-out negative values

Code:

- relu, sigmoid

Input shape = (10000,)

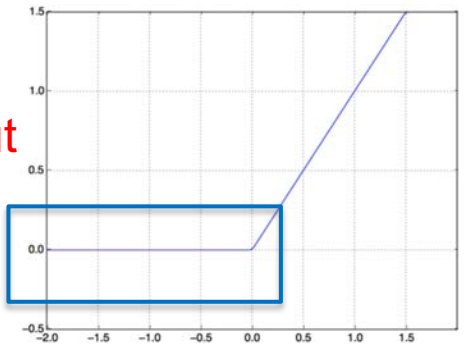
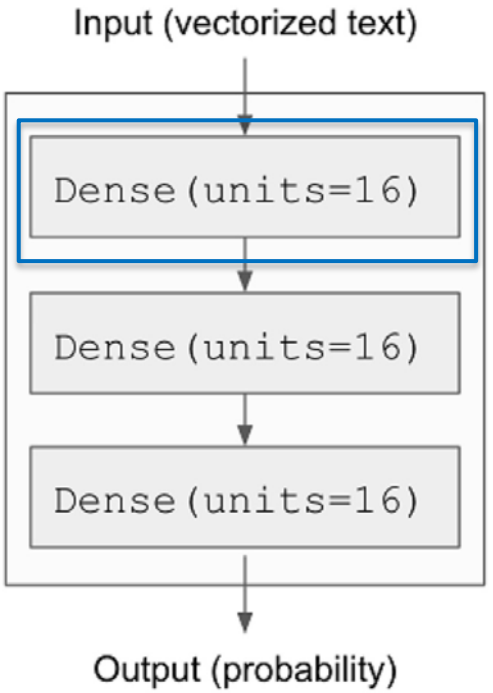


Figure 3.4 The rectified linear unit function

```
30 from tensorflow.keras import models
31 from tensorflow.keras import layers
32
33 model = models.Sequential()
34 model.add(layers.Dense(16, activation='relu',
       input_shape=(10000,)))
35 model.add(layers.Dense(16, activation = 'relu'))
36 model.add(layers.Dense(1, activation = 'sigmoid'))
```



sigmoid:
'squash' arbitrary values
into [0,1]

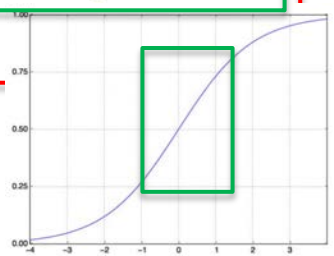


Figure 3.5 The sigmoid function

Step 2: Compiling the model

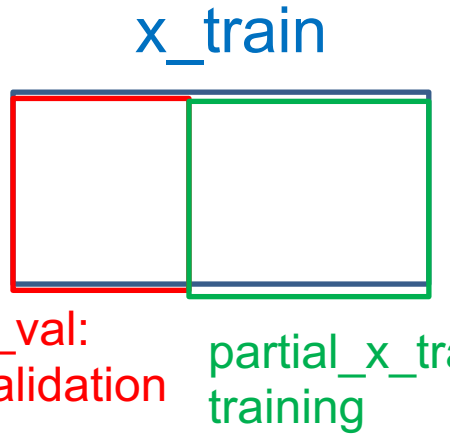
- Passing the
  - Optimizer: ‘rmsprop’
  - Loss function: ‘binary_crossentropy’
  - Metrics: ‘acc’

Code:

```
38 model.compile(optimizer='rmsprop',
39                loss = 'binary_crossentropy',
40                metrics=['accuracy'])
41
```

Validating the approach

- Create a validation set
  - Setting apart 10,000 samples
    - From the original training data



Code:

```
42 x_val=x_train[:10000]
43 partial_x_train = x_train[10000:]
44
45 y_val = y_train[:10000]
46 partial_y_train = y_train[10000:]
47
```

```
48 print(len(x_train))
49 print(len(x_val))
50 print(len(partial_x_train))
```

Result:

```
25000
10000
15000
```

Step 3: Fit the model

- It takes less than 2 seconds per epoch
  - on CPU
- Training is over in 20 seconds

Code:

```
52 history = model.fit(partial_x_train,
53                     partial_y_train,
54                     epochs=20,
55                     batch_size=512,
56                     validation_data=(x_val, y_val))
```

Returned history object

Ready for displaying the classification results

Result:

```
Epoch 17/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0104 - accuracy:
 0.9989 - val_loss: 0.6162 - val_accuracy: 0.8683
Epoch 18/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0053 - accuracy:
 0.9999 - val_loss: 0.6631 - val_accuracy: 0.8644
Epoch 19/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0066 - accuracy:
 0.9990 - val_loss: 0.6709 - val_accuracy: 0.8627
Epoch 20/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0027 - accuracy:
 0.9999 - val_loss: 0.7072 - val_accuracy: 0.8628
```

Practice 2

- *Show the training results*

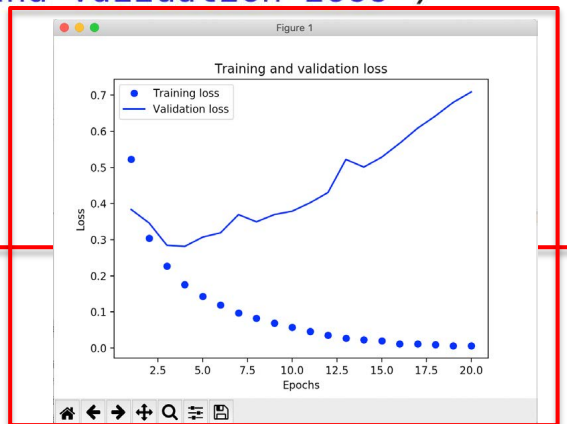
```
Epoch 17/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0104 - accuracy:
 0.9989 - val_loss: 0.6162 - val_accuracy: 0.8683
Epoch 18/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0053 - accuracy:
 0.9999 - val_loss: 0.6631 - val_accuracy: 0.8644
Epoch 19/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0066 - accuracy:
 0.9990 - val_loss: 0.6709 - val_accuracy: 0.8627
Epoch 20/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0027 - accuracy:
 0.9999 - val_loss: 0.7072 - val_accuracy: 0.8628
```

Plot the Training / Validation Loss

- Use pyplot and draw the curves

Code:

```
59  import matplotlib.pyplot as plt
60
61  acc= history.history['accuracy']
62  val_acc = history.history['val_accuracy']
63  loss=history.history['loss']
64  val_loss=history.history['val_loss']
65
66  epochs = range(1, len(acc)+1)
67  plt.plot(epochs, loss, 'bo', label='Training loss')
68  plt.plot(epochs, val_loss, 'b', label='Validation loss')
69
70  plt.title('Training and validation loss')
71  plt.xlabel('Epochs')
72  plt.ylabel('Loss')
73  plt.legend()
74
75  plt.show()
```



Training loss/
Validation loss

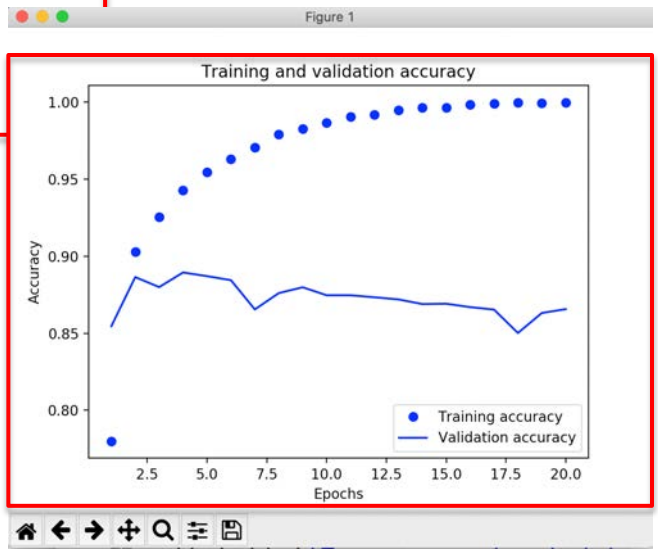
Plot the Training / Validation Accuracy

- Use pyplot and draw the curves

Code:

```
77  plt.clf()
78  plt.plot(epochs, acc, 'bo', label='Training accuracy')
79  plt.plot(epochs, val_acc, 'b', label='Validation
        accuracy')
80
81  plt.title('Training and validation accuracy')
82  plt.xlabel('Epochs')
83  plt.ylabel('Accuracy')
84  plt.legend()
85  plt.show()
```

Training accuracy/
Validation accuracy



The full code (1/2)

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels)=
    imdb.load_data(num_words=10000)

print(train_data[0])
print(train_labels[0])

word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key ,value)
    in word_index.items()])

decoded_review = ' '.join( [reverse_word_index.get(i-3,
    '?') for i in train_data[0]] )

print(decoded_review)

import numpy as np
def vectorize_sequence(sequences, dimension=10000):
    results= np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequence(train_data)
x_test = vectorize_sequence(test_data)

print(x_train[0])

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
print(y_train)
```

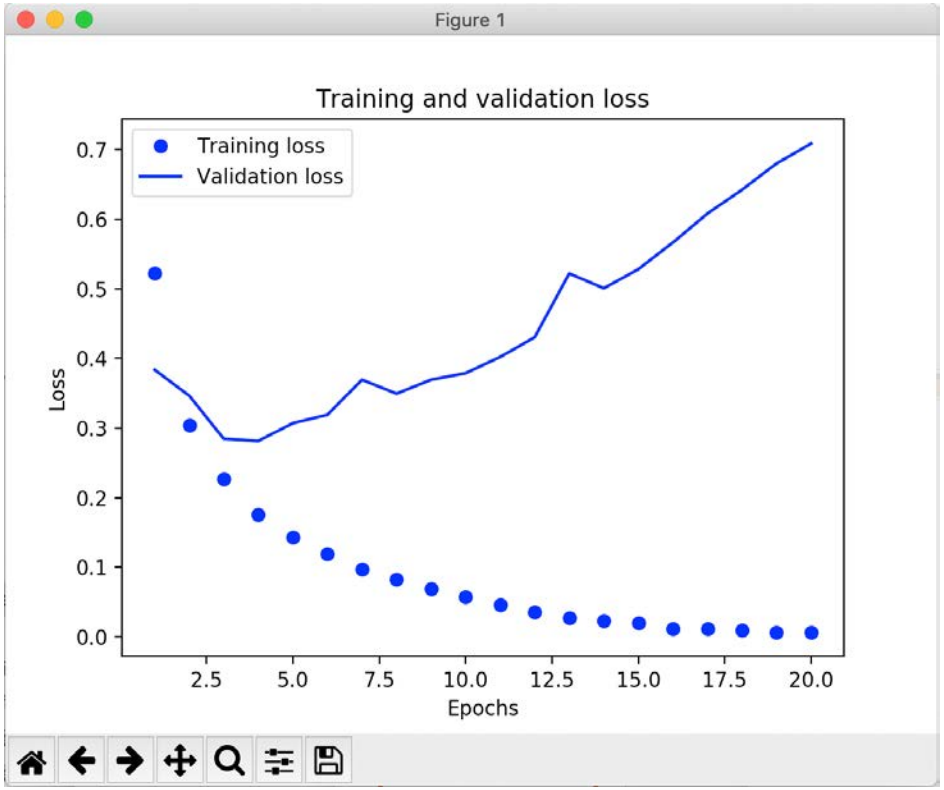
```
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
    input_shape=(10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))

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

x_val=x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

print(len(x_train))
print(len(x_val))
print(len(partial_x_train))

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

The full code (2/2)

```
59 import matplotlib.pyplot as plt
60
61 acc= history.history['accuracy']
62 val_acc = history.history['val_accuracy']
63 loss=history.history['loss']
64 val_loss=history.history['val_loss']
65
66 epochs = range(1, len(acc)+1)
67 plt.plot(epochs, loss, 'bo', label='Training loss')
68 plt.plot(epochs, val_loss, 'b', label='Validation loss')
69
70 plt.title('Training and validation loss')
71 plt.xlabel('Epochs')
72 plt.ylabel('Loss')
73 plt.legend()
74
75 plt.show()
76
77 plt.clf()
78 plt.plot(epochs, acc, 'bo', label='Training accuracy')
79 plt.plot(epochs, val_acc, 'b', label='Validation
       accuracy')
80
81 plt.title('Training and validation accuracy')
82 plt.xlabel('Epochs')
83 plt.ylabel('Accuracy')
84 plt.legend()
85 plt.show()
```

Practice 3

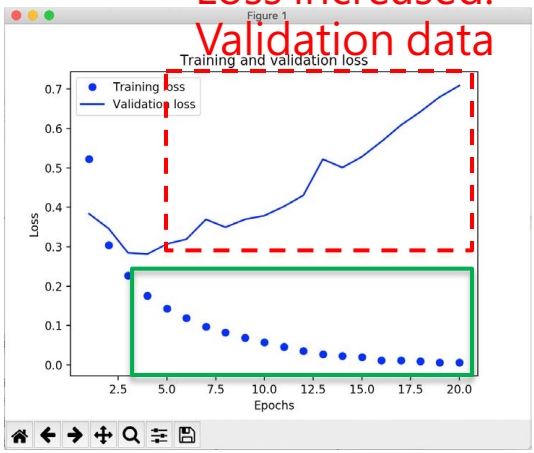
- *Show the training and validation accuracy*

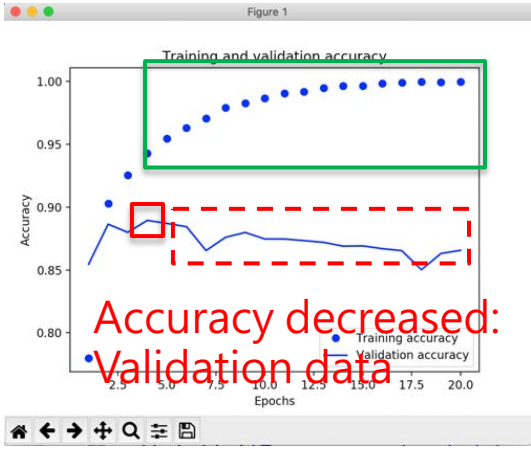


Overfitting Problem

- Training loss:
  - Decreases every epoch
- Training accuracy:
  - Increases every epoch
- Expected: SGD
  - Gradient decent optimization
- But peak accuracy: validation
  - At the 4th epoch
- Over optimizing, training data
  - **Overfitting!**

Loss increased:
Validation data





Accuracy decreased:
Validation data

Prevent Overfitting: Stop Training at the PEAK epoch

- Accuracy results: better!

Code:

```
52 history = model.fit(partial_x_train,
53                     partial_y_train,
54                     epochs=20,
55                     batch_size=512,
56                     validation_data=(x_val, y_val))
57
58
```

20 ➔ 4

add ➔

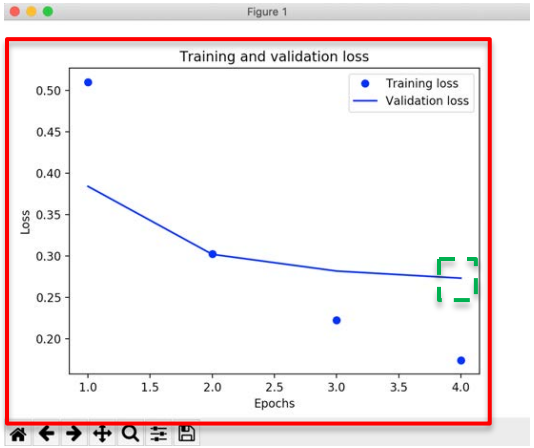
```
results = model.evaluate(x_test, y_test)
print(results)
```

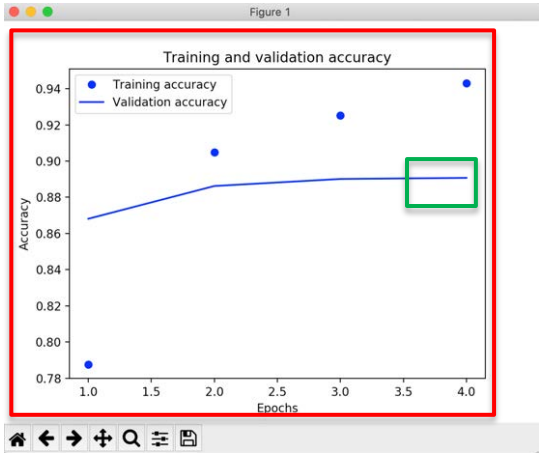
Evaluate on the validation (test) set…

```
[0.2914773111104965, 0.881879985332489]
```

Loss: 29%

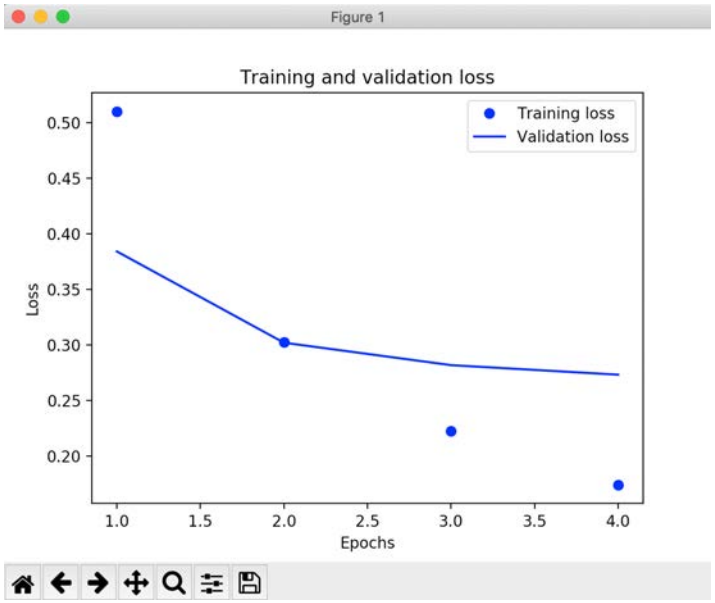
Accuracy: 88.19%

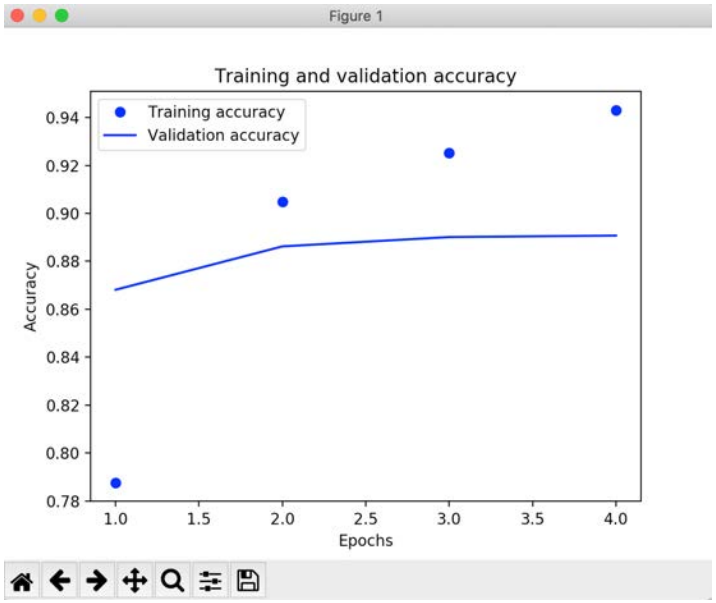




Practice 4

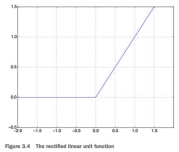
- *Show the training and validation*
  - *Loss / accuracy*





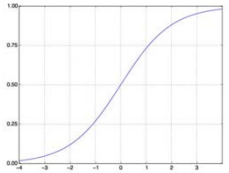
Summary of Binary Classification

- Preprocessing on your raw data
  - To feed into a neural network: Tensors
- Stacks of *Dense* layer
  - With relu activation



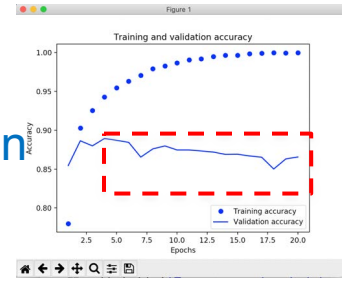
Step 1: Define the model (add)

- Binary classification:
  - A dense layer, end with a sigmoid activation
  - Output: between the range [0, 1]



- Loss function: binary_crossentropy
- Optimizer: rmsprop

Step 2: Configure the Compilation (compile)



Step 3: Fit the model (fit)

- Overfitting: always monitor the performance

Textbook Reading

- *Deep Learning with Python*
  - Ch 3, Getting started with neural networks
    - 3.1, Anatomy of a neural network
    - 3.2, Introduction to Keras
    - 3.4, Classifying movie review: a binary classification example
      - p. 56 - p. 77