Assignment 1

September 24, 2023

This is a companion notebook for the book Deep Learning with Python, Second Edition. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

1 Getting started with neural networks: Classification and regression

1.1 Classifying movie reviews: A binary classification example

1.1.1 The IMDB dataset

Loading the IMDB dataset

[1]:
```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

[2]:
```
train_data[0]
```

[2]:
```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
```

```
173,
36,
256,
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,
316,
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,
530,
476,
26,
400,
317,
46,
7,

```
4,
2,
1029,
13,
104,
88,
4,
381,
15,
297,
98,
32,
2071,
56,
26,
141,
6,
194,
7486,
18,
4,
226,
22,
21,
134,
476,
26,
480,
5,
144,
30,
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]
```

[3]:
```python
train_labels[0]
```

[3]: 1

[4]:
```python
max([max(sequence) for sequence in train_data])
```

[4]: 9999

Decoding reviews back to text

[5]:
```python
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]])
```

1.1.2 Preparing the data

Encoding the integer sequences via multi-hot encoding

[6]:
```python
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

[7]:
```python
x_train[0]
```

[7]: array([0., 1., 1., …, 0., 0., 0.])

[8]:
```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

1.1.3 Building your model

Model definition

[9]:
```
from tensorflow import keras
from tensorflow.keras import layers

#Tried different combinations of layers (3,2,1) and dense units (16,32,64)

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
# using tanh as activation function
    #layers.Dense(16, activation="tanh"),
    #layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])
```

[10]:
```
###  Including regularization
```

[ ]:
```
from keras import models
from keras import layers
from keras import regularizers

# Tried different configuration of L1= 0.005, 0.01 and L2= 0.01, 0.02
model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.
 ↪02), activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.
 ↪02), activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

[11]:
```
### Including Dropout
```

[12]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Tried different combinations of dropout values (0.1, 0.5, 0.8)
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(10000,)))
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
```

```python
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

Compiling the model

[13]:
```python
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

[ ]:
```python
# Using mean_square_error as loss function
#model.compile(optimizer="rmsprop",
#               loss="mean_squared_error",
#               metrics=["accuracy"])
```

1.1.4 Validating your approach

Setting aside a validation set

[14]:
```python
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training your model

[15]:
```python
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 40ms/step - loss: 0.6275 - accuracy:
0.6387 - val_loss: 0.5196 - val_accuracy: 0.8353
Epoch 2/20
30/30 [==============================] - 1s 21ms/step - loss: 0.5108 - accuracy:
0.7792 - val_loss: 0.4247 - val_accuracy: 0.8700
Epoch 3/20
30/30 [==============================] - 1s 27ms/step - loss: 0.4304 - accuracy:
0.8312 - val_loss: 0.3685 - val_accuracy: 0.8692
Epoch 4/20
30/30 [==============================] - 1s 21ms/step - loss: 0.3674 - accuracy:
0.8627 - val_loss: 0.3076 - val_accuracy: 0.8849
Epoch 5/20
30/30 [==============================] - 1s 22ms/step - loss: 0.3200 - accuracy:
0.8910 - val_loss: 0.2895 - val_accuracy: 0.8892
Epoch 6/20
30/30 [==============================] - 1s 24ms/step - loss: 0.2795 - accuracy:
```

```
0.9067 - val_loss: 0.2770 - val_accuracy: 0.8900
Epoch 7/20
30/30 [==============================] - 1s 24ms/step - loss: 0.2444 - accuracy:
0.9237 - val_loss: 0.3006 - val_accuracy: 0.8856
Epoch 8/20
30/30 [==============================] - 1s 24ms/step - loss: 0.2156 - accuracy:
0.9302 - val_loss: 0.2833 - val_accuracy: 0.8870
Epoch 9/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1872 - accuracy:
0.9386 - val_loss: 0.2915 - val_accuracy: 0.8891
Epoch 10/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1667 - accuracy:
0.9483 - val_loss: 0.3165 - val_accuracy: 0.8876
Epoch 11/20
30/30 [==============================] - 1s 24ms/step - loss: 0.1486 - accuracy:
0.9549 - val_loss: 0.3496 - val_accuracy: 0.8866
Epoch 12/20
30/30 [==============================] - 1s 24ms/step - loss: 0.1293 - accuracy:
0.9611 - val_loss: 0.3437 - val_accuracy: 0.8884
Epoch 13/20
30/30 [==============================] - 1s 24ms/step - loss: 0.1207 - accuracy:
0.9634 - val_loss: 0.3917 - val_accuracy: 0.8877
Epoch 14/20
30/30 [==============================] - 1s 24ms/step - loss: 0.1156 - accuracy:
0.9655 - val_loss: 0.3912 - val_accuracy: 0.8864
Epoch 15/20
30/30 [==============================] - 1s 24ms/step - loss: 0.1016 - accuracy:
0.9691 - val_loss: 0.4151 - val_accuracy: 0.8845
Epoch 16/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0914 - accuracy:
0.9725 - val_loss: 0.4630 - val_accuracy: 0.8853
Epoch 17/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0862 - accuracy:
0.9743 - val_loss: 0.4805 - val_accuracy: 0.8867
Epoch 18/20
30/30 [==============================] - 1s 20ms/step - loss: 0.0801 - accuracy:
0.9760 - val_loss: 0.5169 - val_accuracy: 0.8831
Epoch 19/20
30/30 [==============================] - 1s 20ms/step - loss: 0.0753 - accuracy:
0.9758 - val_loss: 0.5220 - val_accuracy: 0.8785
Epoch 20/20
30/30 [==============================] - 1s 18ms/step - loss: 0.0743 - accuracy:
0.9781 - val_loss: 0.5739 - val_accuracy: 0.8848
```

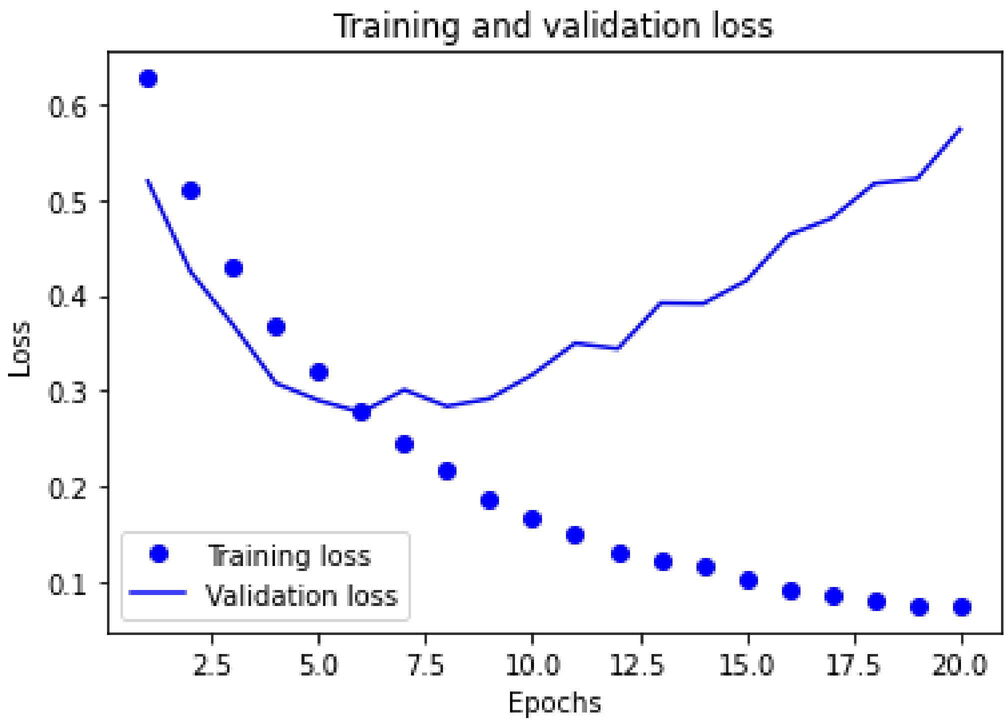
[16]:

```
history_dict = history.history
history_dict.keys()
```

[16]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

Plotting the training and validation loss

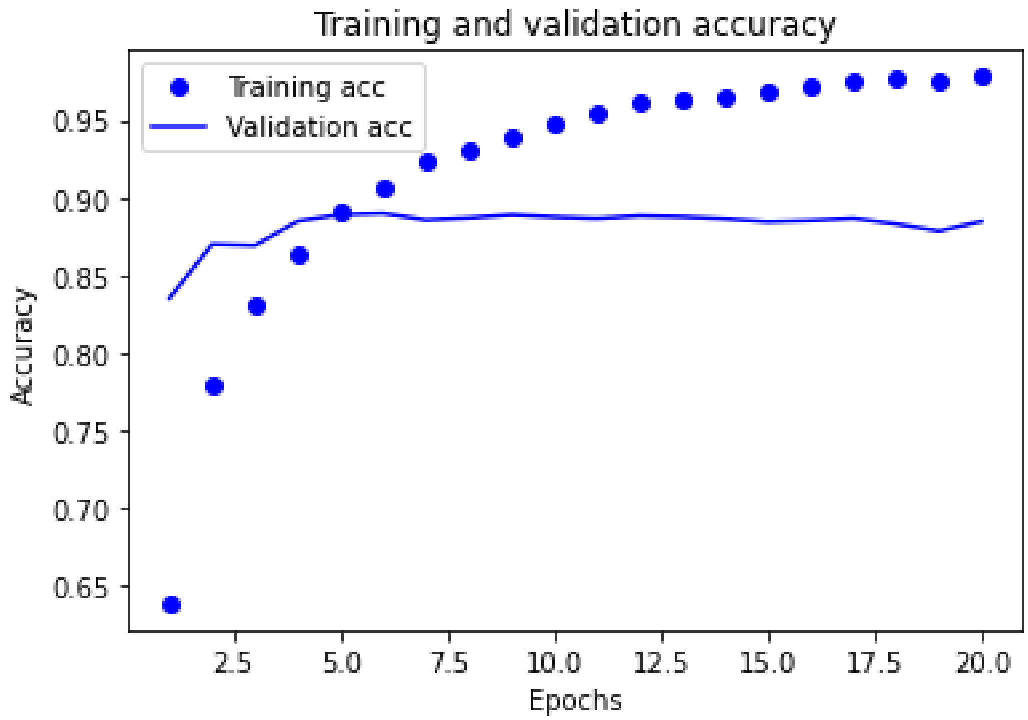
[17]:
```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the training and validation accuracy

[18]:
```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
```

```
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining a model from scratch

[19]:
```
#Tried different combinations of layers (3,2,1) and dense units (16,32,64)
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
# using tanh as activation function
    #layers.Dense(16, activation="tanh"),
    #layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# Using mean_square_error as loss function
#model.compile(optimizer="rmsprop",
#               loss="mean_squared_error",
#               metrics=["accuracy"])
```

```python
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 19ms/step - loss: 0.4644 - accuracy:
0.8238
Epoch 2/4
49/49 [==============================] - 1s 17ms/step - loss: 0.2671 - accuracy:
0.9063
Epoch 3/4
49/49 [==============================] - 1s 14ms/step - loss: 0.2035 - accuracy:
0.9273
Epoch 4/4
49/49 [==============================] - 1s 13ms/step - loss: 0.1703 - accuracy:
0.9387
782/782 [==============================] - 1s 594us/step - loss: 0.2921 -
accuracy: 0.8837
```

[20]:
```python
results
```

[20]:
```
[0.2920909821987152, 0.883679986000061]
```

1.1.5 Using a trained model to generate predictions on new data

[21]:
```python
model.predict(x_test)
```

[21]:
```
array([[0.16282263],
       [0.99993783],
       [0.852559  ],
       …,
       [0.1051057 ],
       [0.05724409],
       [0.66419065]], dtype=float32)
```