Madhukar_Ayachit_5.1

April 26, 2022

0.1 Assignment 5.1

[1]:
```
 # Import libraries
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
```

[2]:
```
import keras
keras.__version__
```

[2]:
```
'2.4.3'
```

[3]:
```
# Load the dataset
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.
 ↪load_data(num_words=10000)
```

[4]:
```
train_data[0]
```

[4]:
```
[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]
```

[6]:
```
train_labels[0]
```

[6]: 1

[7]:
```
#Since we restricted ourselves to the top 10,000 most frequent words, no word␣
↪index will exceed 10,000:
max([max(sequence) for sequence in train_data])
```

[7]: 9999

[9]:
```
# word_index is a dictionary mapping words to an integer index
word_index = imdb.get_word_index()
# We reverse it, mapping integer indices to words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
# We decode the review; note that our indices were offset by 3
# because 0, 1 and 2 are reserved indices for "padding", "start of␣
↪sequence",and "unknown".
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in␣
↪train_data[0]])
decoded_review
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
```

[9]:
```
"? 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 robert ? 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 brilliant 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 at 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 played 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 big profile for the whole film but these children are amazing and
should be praised 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"
```

[13]:
```python
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
# Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1  # set specific indices of results[i] to 1s
        return results
# Our vectorized training data
x_train = vectorize_sequences(train_data) # Our vectorized test data
x_test = vectorize_sequences(test_data)
```

[14]:
```python
# Print sample
x_train[0]
```

[14]:
```
array([0., 1., 1., …, 0., 0., 0.])
```

[15]:
```python
# vectorized labels

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

[17]:
```python
#The Keras implementation
from keras import models
from 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'))
```

[30]:
```python
model.
 ↪compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
```

[31]:
```python
from keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001),␣
 ↪loss='binary_crossentropy',metrics=['accuracy'])
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001), loss=losses.
 ↪binary_crossentropy, metrics=[metrics.binary_accuracy])
```

[32]:
```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[33]:
```
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 [==============================] - 1s 30ms/step - loss: 0.6932 -
binary_accuracy: 0.4983 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 2/20
30/30 [==============================] - 1s 26ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 3/20
30/30 [==============================] - 1s 27ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 4/20
30/30 [==============================] - 1s 30ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 5/20
30/30 [==============================] - 1s 29ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 6/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 7/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 8/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 9/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 10/20
30/30 [==============================] - 1s 27ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 11/20
30/30 [==============================] - 1s 27ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 12/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 13/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
```

```
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 14/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 15/20
30/30 [==============================] - 1s 30ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 16/20
30/30 [==============================] - 1s 27ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 17/20
30/30 [==============================] - 1s 31ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 18/20
30/30 [==============================] - 1s 28ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 19/20
30/30 [==============================] - 1s 27ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
Epoch 20/20
30/30 [==============================] - 1s 26ms/step - loss: 0.6931 -
binary_accuracy: 0.5035 - val_loss: 0.6932 - val_binary_accuracy: 0.4948
```

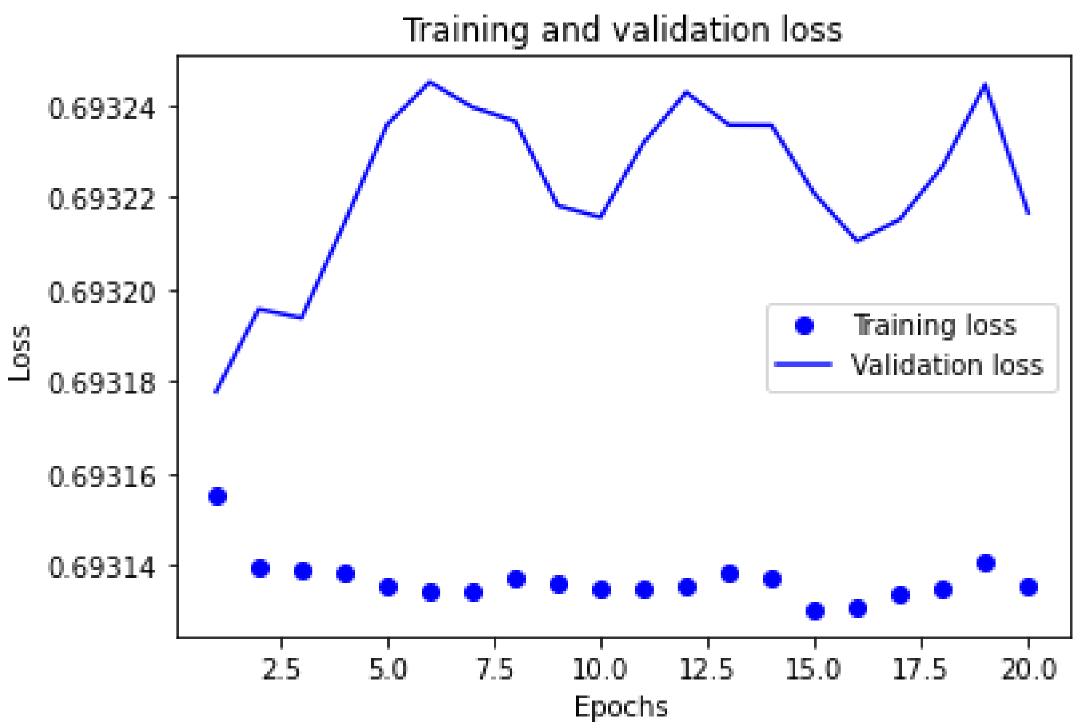
[34]:

```
history_dict = history.history
history_dict.keys()
#dict_keys(['loss', 'val_loss', 'binary_accuracy', 'val_binary_accuracy'])␣
 ↪#history_dict = history.history
print(history_dict)
```

```
{'loss': [0.69315505027771, 0.6931398510932922, 0.6931390762329102,
0.6931386590003967, 0.6931353807449341, 0.6931343674659729, 0.6931347846984863,
0.6931373476982117, 0.6931362748146057, 0.6931352019309998, 0.6931352615356445,
0.6931353807449341, 0.6931385397911072, 0.6931371688842773, 0.6931304931640625,
0.6931309700012207, 0.6931340098381042, 0.6931349039077759, 0.6931409239768982,
0.6931357979774475], 'binary_accuracy': [0.4983333349227905, 0.5035333037376404,
0.5035333037376404, 0.5035333037376404, 0.5035333037376404, 0.5035333037376404,
0.5035333037376404, 0.5035333037376404, 0.5035333037376404, 0.5035333037376404,
0.5035333037376404, 0.5035333037376404, 0.5035333037376404, 0.5035333037376404,
0.5035333037376404, 0.5035333037376404, 0.5035333037376404, 0.5035333037376404,
0.5035333037376404, 0.5035333037376404], 'val_loss': [0.6931777596473694,
0.6931957006454468, 0.693193793296814, 0.6932142972946167, 0.6932356953620911,
0.6932449340820312, 0.6932395100593567, 0.6932364702224731, 0.6932181119918823,
0.6932156085968018, 0.6932317614555359, 0.6932427287101746, 0.6932356357574463,
0.6932355761528015, 0.6932207345962524, 0.6932104229927063, 0.6932151317596436,
0.6932266354560852, 0.693244457244873, 0.6932166218757629],
'val_binary_accuracy': [0.49480000138282776, 0.49480000138282776,
0.49480000138282776, 0.49480000138282776, 0.49480000138282776,
0.49480000138282776, 0.49480000138282776, 0.49480000138282776,
```

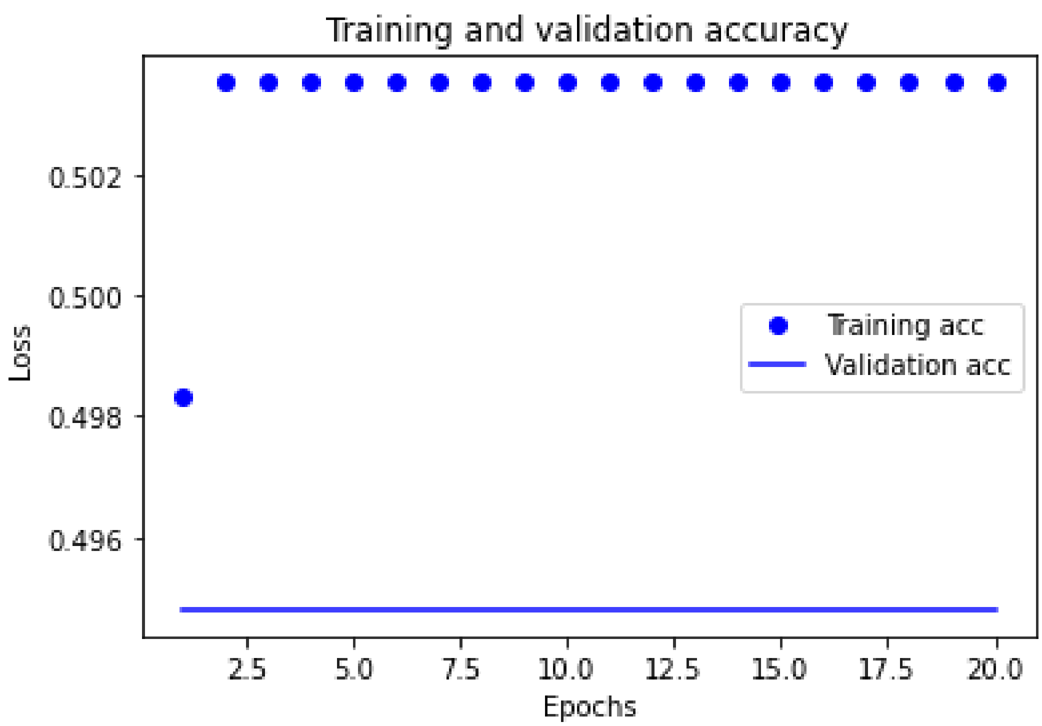
0.49480000138282776, 0.49480000138282776, 0.49480000138282776,
0.49480000138282776, 0.49480000138282776, 0.49480000138282776,
0.49480000138282776, 0.49480000138282776, 0.49480000138282776,
0.49480000138282776, 0.49480000138282776, 0.49480000138282776]}

[35]:
```python
import matplotlib.pyplot as plt
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
#acc = history.history['acc']
#val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[36]:

```
plt.clf() # clear figure
#acc_values = history_dict['acc']
#val_acc_values = history_dict['val_acc']
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_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('Loss')
plt.legend()
plt.show()
```



[37]:

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

```
Epoch 1/4
```

```
49/49 [==============================] - 0s 9ms/step - loss: 0.6932 - accuracy:
0.5000
Epoch 2/4
49/49 [==============================] - 0s 9ms/step - loss: 0.6931 - accuracy:
0.4989
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.6931 - accuracy:
0.4987
Epoch 4/4
49/49 [==============================] - 0s 7ms/step - loss: 0.6931 - accuracy:
0.4983
782/782 [==============================] - 2s 2ms/step - loss: 0.6932 -
accuracy: 0.5000
```

[38]:
```
results
```

[38]:
```
[0.6931527256965637, 0.5]
```

[39]:
```
model.predict(x_test)
```

[39]:
```
array([[0.56660044],
       [0.50014085],
       [0.50014085],
       …,
       [0.50014085],
       [0.50014085],
       [0.50014085]], dtype=float32)
```

[ ]:
```

```