DL Prac 2

April 24, 2024

1 Practical - 2

1.0.1 Problem Statement

Classification using Deep neural network : Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset

[1]:
```
import keras
keras.__version__
```

[1]:
```
'3.2.1'
```

[2]:
```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.
 ↪load_data(num_words=10000)
```

[3]:
```
train_data[0]
```

[3]:
```
[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]
```

[4]:
```
train_labels[0]
```

[4]:
```
1
```

[5]:
```
max([max(sequence) for sequence in train_data])
```

[5]:
```
9999
```

[6]:
```
# 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]])
```

[7]:
```
decoded_review
```

[7]:
```
"? 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"
```

[8]:

```
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)
```

[9]:

```
x_train[0]
```

[9]:

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

[10]:

```
# Our vectorized labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

[11]:

```
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'))
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:86:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[12]:

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

[14]:

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[15]:

```
from keras import losses
from keras import metrics
```

```
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

[16]:

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

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

[17]:

```
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                 14s 151ms/step -
binary_accuracy: 0.6944 - loss: 0.6110 - val_binary_accuracy: 0.8473 - val_loss:
0.4153
Epoch 2/20
30/30                 1s 15ms/step -
binary_accuracy: 0.8871 - loss: 0.3473 - val_binary_accuracy: 0.8822 - val_loss:
0.3178
Epoch 3/20
30/30                 1s 15ms/step -
binary_accuracy: 0.9206 - loss: 0.2523 - val_binary_accuracy: 0.8894 - val_loss:
0.2835
Epoch 4/20
30/30                 1s 21ms/step -
binary_accuracy: 0.9381 - loss: 0.1950 - val_binary_accuracy: 0.8889 - val_loss:
0.2777
Epoch 5/20
30/30                 1s 19ms/step -
binary_accuracy: 0.9492 - loss: 0.1554 - val_binary_accuracy: 0.8838 - val_loss:
0.2844
Epoch 6/20
30/30                 1s 17ms/step -
binary_accuracy: 0.9608 - loss: 0.1322 - val_binary_accuracy: 0.8848 - val_loss:
0.2880
Epoch 7/20
30/30                 1s 16ms/step -
binary_accuracy: 0.9687 - loss: 0.1109 - val_binary_accuracy: 0.8855 - val_loss:
0.2999
Epoch 8/20
30/30                 1s 17ms/step -
binary_accuracy: 0.9750 - loss: 0.0936 - val_binary_accuracy: 0.8816 - val_loss:
```

```
0.3259
Epoch 9/20
30/30            1s 22ms/step -
binary_accuracy: 0.9773 - loss: 0.0835 - val_binary_accuracy: 0.8786 - val_loss:
0.3502
Epoch 10/20
30/30            1s 15ms/step -
binary_accuracy: 0.9781 - loss: 0.0755 - val_binary_accuracy: 0.8794 - val_loss:
0.3603
Epoch 11/20
30/30            1s 18ms/step -
binary_accuracy: 0.9854 - loss: 0.0598 - val_binary_accuracy: 0.8757 - val_loss:
0.3719
Epoch 12/20
30/30            1s 19ms/step -
binary_accuracy: 0.9900 - loss: 0.0488 - val_binary_accuracy: 0.8586 - val_loss:
0.4877
Epoch 13/20
30/30            1s 23ms/step -
binary_accuracy: 0.9874 - loss: 0.0496 - val_binary_accuracy: 0.8750 - val_loss:
0.4245
Epoch 14/20
30/30            1s 19ms/step -
binary_accuracy: 0.9913 - loss: 0.0406 - val_binary_accuracy: 0.8740 - val_loss:
0.4262
Epoch 15/20
30/30            1s 25ms/step -
binary_accuracy: 0.9947 - loss: 0.0321 - val_binary_accuracy: 0.8737 - val_loss:
0.4546
Epoch 16/20
30/30            1s 23ms/step -
binary_accuracy: 0.9961 - loss: 0.0260 - val_binary_accuracy: 0.8726 - val_loss:
0.4688
Epoch 17/20
30/30            1s 23ms/step -
binary_accuracy: 0.9974 - loss: 0.0205 - val_binary_accuracy: 0.8720 - val_loss:
0.4945
Epoch 18/20
30/30            1s 20ms/step -
binary_accuracy: 0.9972 - loss: 0.0196 - val_binary_accuracy: 0.8708 - val_loss:
0.5101
Epoch 19/20
30/30            1s 24ms/step -
binary_accuracy: 0.9986 - loss: 0.0154 - val_binary_accuracy: 0.8651 - val_loss:
0.5403
Epoch 20/20
30/30            1s 21ms/step -
binary_accuracy: 0.9989 - loss: 0.0131 - val_binary_accuracy: 0.8694 - val_loss:
```

```
0.5558
```

[18]:

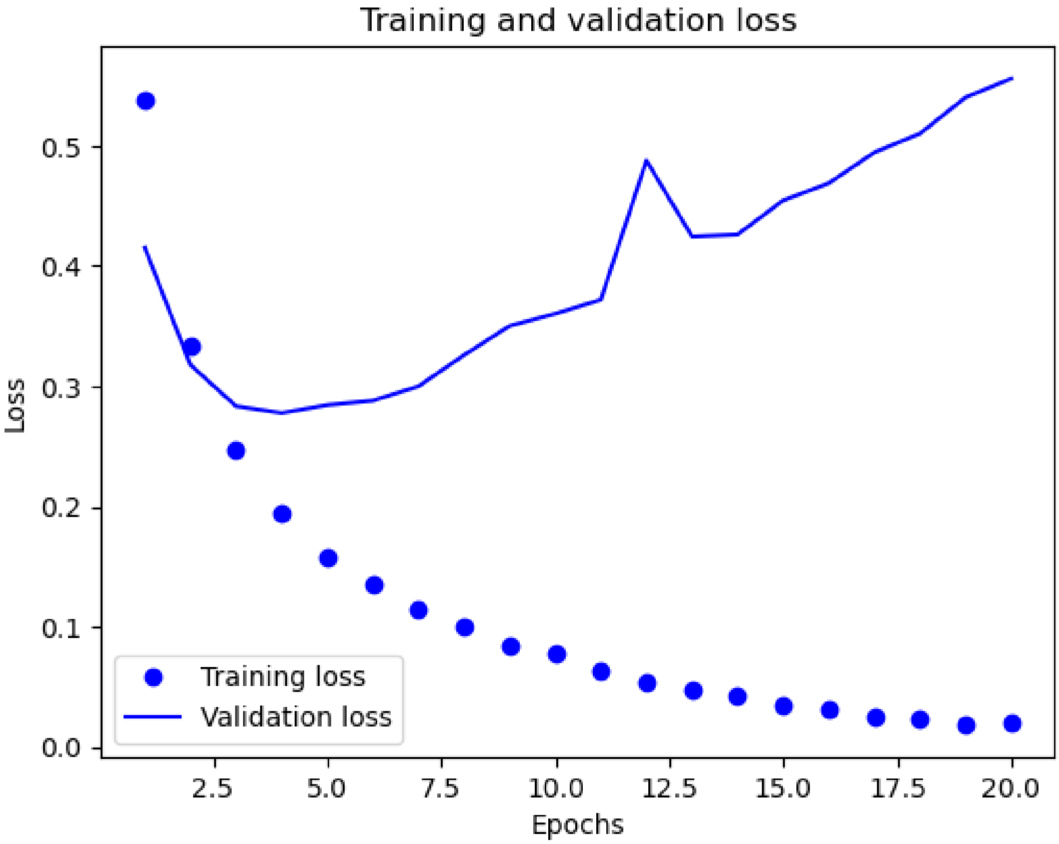
```
history_dict = history.history
history_dict.keys()
```

[18]:

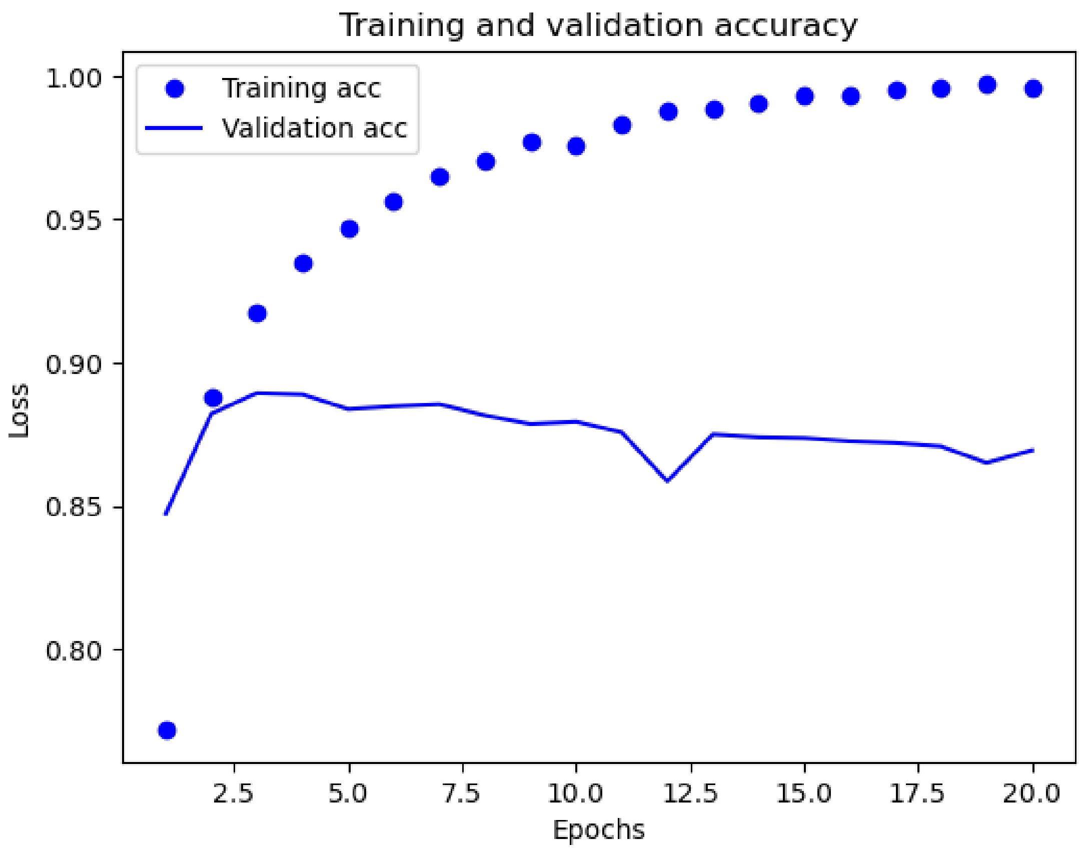
```
dict_keys(['binary_accuracy', 'loss', 'val_binary_accuracy', 'val_loss'])
```

[20]:

```
import matplotlib.pyplot as plt

acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
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()
```



[21]:
```
plt.clf()   # clear figure
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['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()
```



[22]:

```
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)
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:86:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/4
49/49                 4s 11ms/step -
accuracy: 0.7243 - loss: 0.5640
```

```
Epoch 2/4
49/49                 1s 11ms/step -
accuracy: 0.8996 - loss: 0.2904
Epoch 3/4
49/49                 1s 11ms/step -
accuracy: 0.9268 - loss: 0.2143
Epoch 4/4
49/49                 1s 12ms/step -
accuracy: 0.9402 - loss: 0.1760
782/782               3s 2ms/step -
accuracy: 0.8859 - loss: 0.2855
```

[23]:
```
results
```

[23]:
```
[0.283999502658844, 0.8870000243186951]
```

[24]:
```
model.predict(x_test)
```

```
782/782               2s 3ms/step
```

[24]:
```
array([[0.21269305],
       [0.9994701 ],
       [0.7967413 ],
       …,
       [0.10996707],
       [0.06536405],
       [0.5542662 ]], dtype=float32)
```

[ ]:
```

```