DL Prac 2

April 18, 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'
```

[21]:
```
from keras.datasets import imdb

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

[22]:
```
train_data[0]
```

[22]:
```
[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]
```

[23]:
```
train_labels[0]
```

[23]:
```
1
```

[24]:
```
max([max(sequence) for sequence in train_data])
```

[24]:
```
9999
```

[25]:
```
# 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]])
```

[26]:
```
decoded_review
```

[26]:
```
"? 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"
```

[27]:

```
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)
```

[28]:

```
x_train[0]
```

[28]:

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

[29]:

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

[30]:

```
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'))
```

[31]:

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

[13]:

```
from keras import optimizers

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

[14]:

```
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))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 1s - loss: 0.5103 - acc: 0.7911 -
val_loss: 0.4016 - val_acc: 0.8628
Epoch 2/20
15000/15000 [==============================] - 1s - loss: 0.3110 - acc: 0.9031 -
val_loss: 0.3085 - val_acc: 0.8870
Epoch 3/20
15000/15000 [==============================] - 1s - loss: 0.2309 - acc: 0.9235 -
val_loss: 0.2803 - val_acc: 0.8908
Epoch 4/20
15000/15000 [==============================] - 1s - loss: 0.1795 - acc: 0.9428 -
val_loss: 0.2735 - val_acc: 0.8893
Epoch 5/20
15000/15000 [==============================] - 1s - loss: 0.1475 - acc: 0.9526 -
val_loss: 0.2788 - val_acc: 0.8890
Epoch 6/20
15000/15000 [==============================] - 1s - loss: 0.1185 - acc: 0.9638 -
val_loss: 0.3330 - val_acc: 0.8764
Epoch 7/20
15000/15000 [==============================] - 1s - loss: 0.1005 - acc: 0.9703 -
val_loss: 0.3055 - val_acc: 0.8838
Epoch 8/20
15000/15000 [==============================] - 1s - loss: 0.0818 - acc: 0.9773 -
val_loss: 0.3344 - val_acc: 0.8769
Epoch 9/20
15000/15000 [==============================] - 1s - loss: 0.0696 - acc: 0.9814 -
val_loss: 0.3607 - val_acc: 0.8800
Epoch 10/20
15000/15000 [==============================] - 1s - loss: 0.0547 - acc: 0.9873 -
val_loss: 0.3776 - val_acc: 0.8785
Epoch 11/20
15000/15000 [==============================] - 1s - loss: 0.0453 - acc: 0.9895 -
val_loss: 0.4035 - val_acc: 0.8765
Epoch 12/20
15000/15000 [==============================] - 1s - loss: 0.0353 - acc: 0.9930 -
```

```
val_loss: 0.4437 - val_acc: 0.8766
Epoch 13/20
15000/15000 [==============================] - 1s - loss: 0.0269 - acc: 0.9956 -
val_loss: 0.4637 - val_acc: 0.8747
Epoch 14/20
15000/15000 [==============================] - 1s - loss: 0.0212 - acc: 0.9968 -
val_loss: 0.4877 - val_acc: 0.8714
Epoch 15/20
15000/15000 [==============================] - 1s - loss: 0.0162 - acc: 0.9977 -
val_loss: 0.6080 - val_acc: 0.8625
Epoch 16/20
15000/15000 [==============================] - 1s - loss: 0.0115 - acc: 0.9993 -
val_loss: 0.5778 - val_acc: 0.8698
Epoch 17/20
15000/15000 [==============================] - 1s - loss: 0.0116 - acc: 0.9979 -
val_loss: 0.5906 - val_acc: 0.8702
Epoch 18/20
15000/15000 [==============================] - 1s - loss: 0.0054 - acc: 0.9998 -
val_loss: 0.6204 - val_acc: 0.8639
Epoch 19/20
15000/15000 [==============================] - 1s - loss: 0.0083 - acc: 0.9984 -
val_loss: 0.6419 - val_acc: 0.8676
Epoch 20/20
15000/15000 [==============================] - 1s - loss: 0.0031 - acc: 0.9998 -
val_loss: 0.6796 - val_acc: 0.8683
```

[34]:

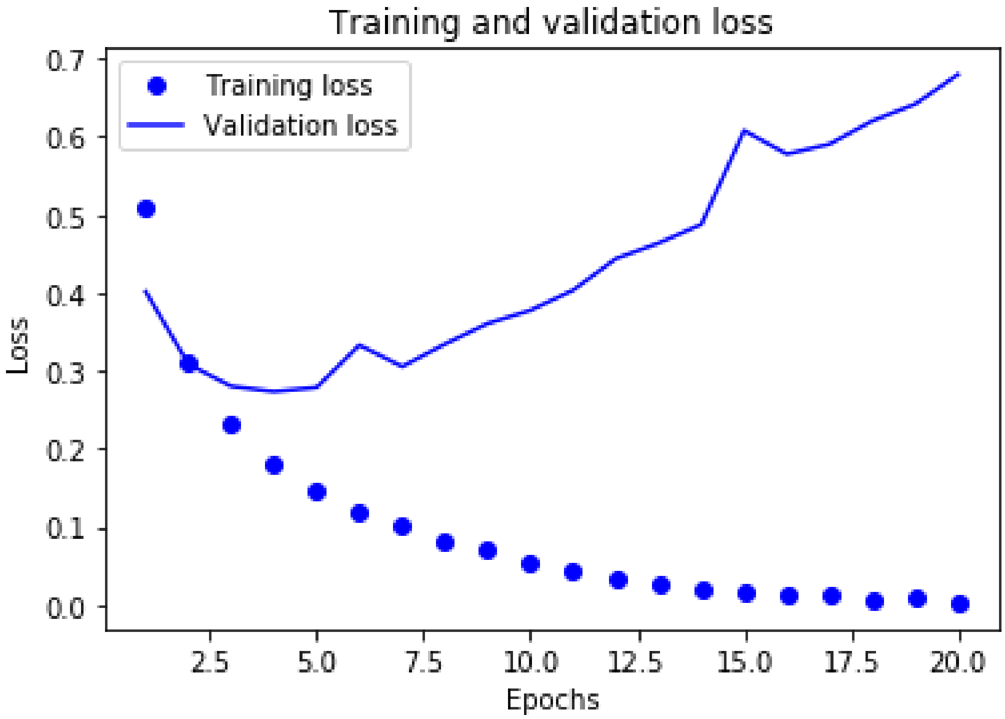
```python
history_dict = history.history
history_dict.keys()
```

[34]:

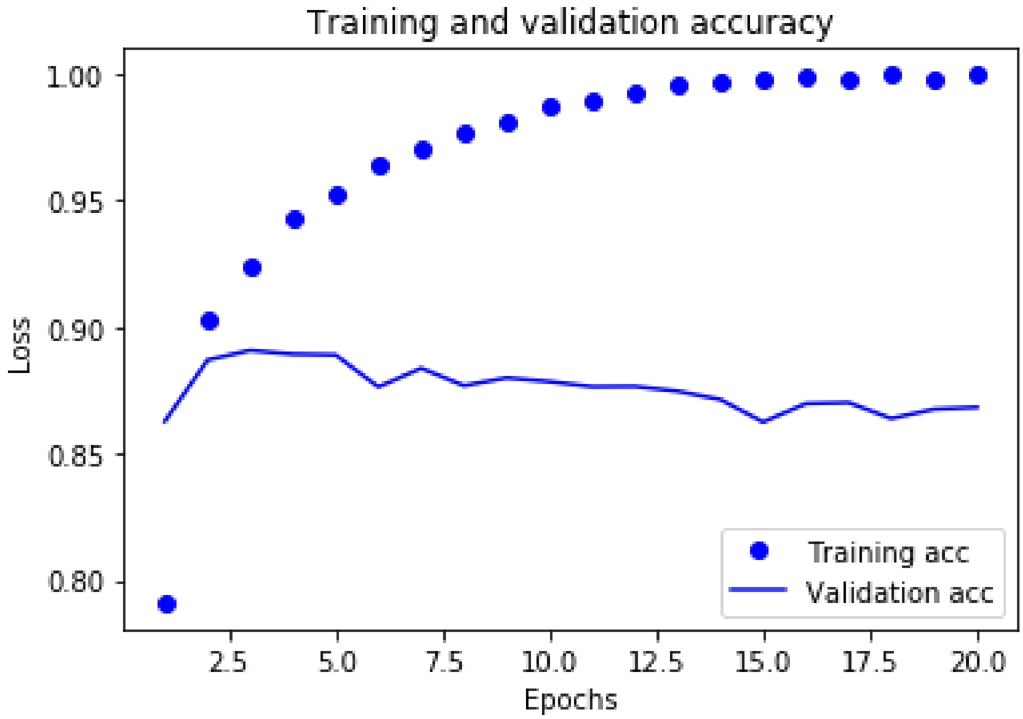
```
dict_keys(['val_acc', 'acc', 'val_loss', 'loss'])
```

[36]:

```python
import matplotlib.pyplot as plt

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()
```



[38]:
```
plt.clf()   # clear figure
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

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()
```



[40]:

```
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
25000/25000 [==============================] - 1s - loss: 0.4738 - acc: 0.8044
Epoch 2/4
25000/25000 [==============================] - 1s - loss: 0.2660 - acc: 0.9076
Epoch 3/4
25000/25000 [==============================] - 1s - loss: 0.2028 - acc: 0.9277
Epoch 4/4
25000/25000 [==============================] - 1s - loss: 0.1700 - acc: 0.9397
24544/25000 [============================>.] - ETA: 0s
```

[41]:

```
results
```

```
[41]: [0.29184698499679568, 0.88495999999999997]
```

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

```
[26]: array([[ 0.91966152],
              [ 0.86563045],
              [ 0.99936908],
              …,
              [ 0.45731062],
              [ 0.0038014 ],
              [ 0.79525089]], dtype=float32)
```