Assignment 5.2

July 7, 2021

1 3.5 Classifying newswires: a multiclass classification example

1.1 3.5.1 The Reuters dataset

[1]:
```
from tensorflow.keras.datasets import reuters

# Importing the data to training and testing sets
(train_data, train_labels), (test_data, test_labels) = reuters.
 ↪load_data(num_words=10000)
```

```
C:\Users\hotal\AppData\Roaming\Python\Python38\site-
packages\tensorflow\python\keras\datasets\reuters.py:148:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
C:\Users\hotal\AppData\Roaming\Python\Python38\site-
packages\tensorflow\python\keras\datasets\reuters.py:149:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

[2]:
```
# Number of observations in the training data
len(train_data)
```

[2]: 8982

[3]:
```
# Number of observations in the testing data
len(test_data)
```

[3]: 2246

[4]:
```
# Printing an example from the dataset
print(train_data[10])
```

[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979, 3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]

[5]:
```python
# Getting the word dictionary from the dataset
word_index = reuters.get_word_index()

# Reversing the dictionary for easier indexing
reverse_word_index = dict([(value,key) for (key, value) in word_index.items()])

# Building a review using the training array and the reversed dictionary
decoded_review = ' '.join([reverse_word_index.get(i - 3 , '?') for i in␣
↪train_data[0]])

# Printing the review
print(decoded_review)
```

```
? ? ? said as a result of its december acquisition of space co it expects
earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986
the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs
in 1986 and rental operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it
said cash flow per share this year should be 2 50 to three dlrs reuter 3
```

[6]:
```python
# Checking the number of topics in the training set
len(set(train_labels))
```

[6]: 46

1.2 3.5.2 Preparing the data

[7]:
```python
import numpy as np

# Convert the training arrays into a 2D array with the columns representing the␣
↪words used.
def vectorize_sequence(sequences, dimensions = 10000):
    """
    Return a 2D array with the columns representing the word usage of each␣
↪entry.
    Parameters
    ----------
    sequences : array_like
        Array representing the word usage in each entry
    dimensions : data-type, optional
        Number of columns in the 2D array
    """

    results = np.zeros((len(sequences), dimensions))
    for i, sequences in enumerate(sequences):
        results[i, sequences] = 1
```

```
    return results
```

[8]:
```
# Vectorize the training and testing datasets
X_train = vectorize_sequence(train_data)
X_test = vectorize_sequence(test_data)
```

[9]:
```
def to_one_hot(labels):
    """
    Return a hot-encoded 2D array.
    Parameters
    ----------
    sequences : array_like
        Array representing the classifications of each article from Reuters
    """
    dimension = len(set(labels))
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1
    return results
```

[10]:
```
# Hot-encode the target attributes
y_train = to_one_hot(train_labels)
y_test = to_one_hot(test_labels)
```

1.3 3.5.3 Building your network

[11]:
```
from keras import models, layers
```

[12]:
```
model = models.Sequential()
model.add(layers.Dense(64, activation = 'relu', input_shape = (X_train.
 ↪shape[1],)))
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(len(set(train_labels)), activation = 'softmax'))
```

[13]:
```
model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics␣
 ↪= ['accuracy'])
```

1.4 3.5.4 Validating your approach

[14]:
```
X_val = X_train[:1000]
partial_X_train = X_train[1000:]

y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

[15]:
```
history = model.fit(partial_X_train,
                    partial_y_train,
```

```
                    epochs=20,
                    batch_size=512,
                    validation_data=(X_val, y_val))
```

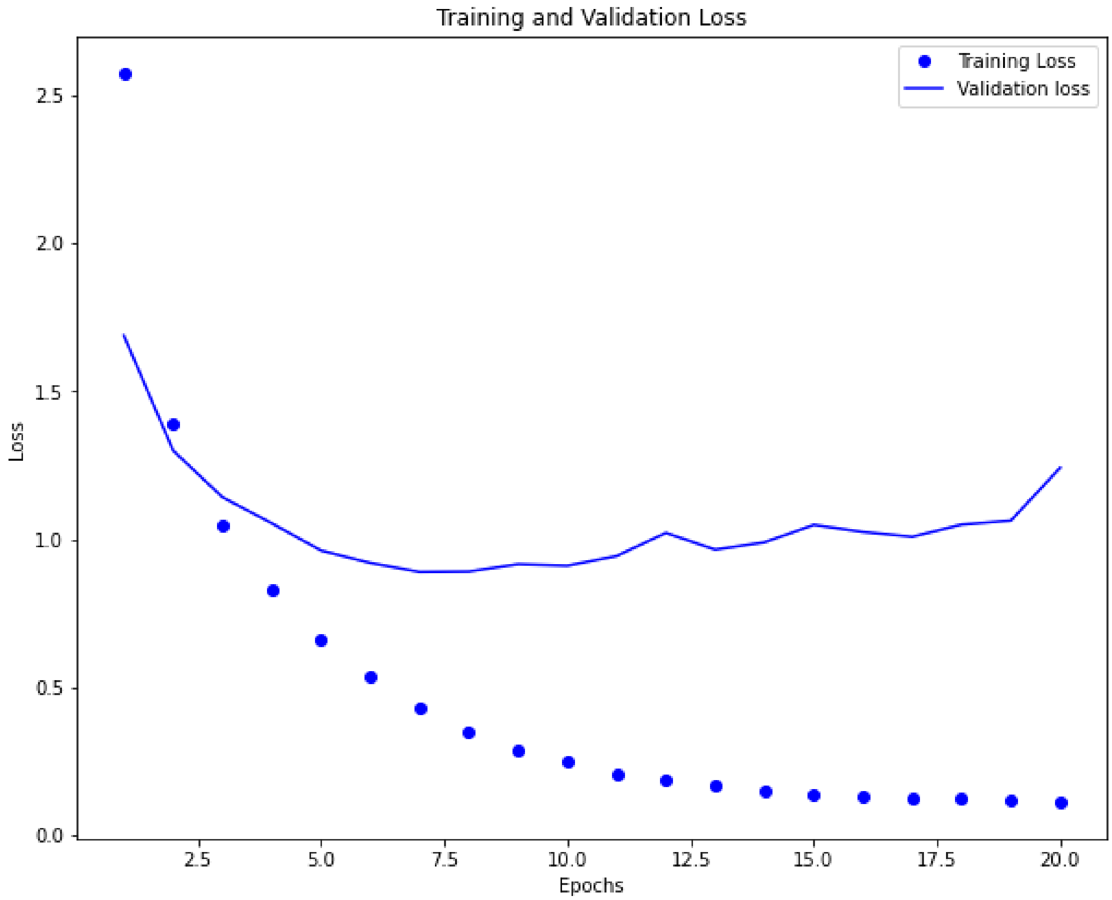
```
Epoch 1/20
16/16 [==============================] - 3s 113ms/step - loss: 3.0896 -
accuracy: 0.4043 - val_loss: 1.6882 - val_accuracy: 0.6580
Epoch 2/20
16/16 [==============================] - 1s 34ms/step - loss: 1.4818 - accuracy:
0.7047 - val_loss: 1.2996 - val_accuracy: 0.7190
Epoch 3/20
16/16 [==============================] - 1s 36ms/step - loss: 1.0819 - accuracy:
0.7673 - val_loss: 1.1421 - val_accuracy: 0.7360
Epoch 4/20
16/16 [==============================] - 1s 34ms/step - loss: 0.8595 - accuracy:
0.8146 - val_loss: 1.0540 - val_accuracy: 0.7690
Epoch 5/20
16/16 [==============================] - 1s 34ms/step - loss: 0.6781 - accuracy:
0.8579 - val_loss: 0.9619 - val_accuracy: 0.8050
Epoch 6/20
16/16 [==============================] - 1s 34ms/step - loss: 0.5383 - accuracy:
0.8908 - val_loss: 0.9199 - val_accuracy: 0.8150
Epoch 7/20
16/16 [==============================] - 1s 34ms/step - loss: 0.4368 - accuracy:
0.9124 - val_loss: 0.8893 - val_accuracy: 0.8160
Epoch 8/20
16/16 [==============================] - 1s 35ms/step - loss: 0.3442 - accuracy:
0.9282 - val_loss: 0.8908 - val_accuracy: 0.8210
Epoch 9/20
16/16 [==============================] - 1s 34ms/step - loss: 0.2863 - accuracy:
0.9374 - val_loss: 0.9156 - val_accuracy: 0.8040
Epoch 10/20
16/16 [==============================] - 1s 33ms/step - loss: 0.2359 - accuracy:
0.9472 - val_loss: 0.9104 - val_accuracy: 0.8100
Epoch 11/20
16/16 [==============================] - 1s 35ms/step - loss: 0.1934 - accuracy:
0.9527 - val_loss: 0.9435 - val_accuracy: 0.8040
Epoch 12/20
16/16 [==============================] - 1s 33ms/step - loss: 0.1751 - accuracy:
0.9576 - val_loss: 1.0214 - val_accuracy: 0.7970
Epoch 13/20
16/16 [==============================] - 1s 34ms/step - loss: 0.1588 - accuracy:
0.9581 - val_loss: 0.9648 - val_accuracy: 0.8100
Epoch 14/20
16/16 [==============================] - 1s 35ms/step - loss: 0.1388 - accuracy:
0.9620 - val_loss: 0.9896 - val_accuracy: 0.8140
Epoch 15/20
16/16 [==============================] - 1s 34ms/step - loss: 0.1330 - accuracy:
```

```
0.9603 - val_loss: 1.0483 - val_accuracy: 0.7970
Epoch 16/20
16/16 [==============================] - 1s 33ms/step - loss: 0.1214 - accuracy:
0.9601 - val_loss: 1.0247 - val_accuracy: 0.8020
Epoch 17/20
16/16 [==============================] - 1s 38ms/step - loss: 0.1161 - accuracy:
0.9608 - val_loss: 1.0085 - val_accuracy: 0.8110
Epoch 18/20
16/16 [==============================] - 1s 35ms/step - loss: 0.1116 - accuracy:
0.9606 - val_loss: 1.0494 - val_accuracy: 0.8100
Epoch 19/20
16/16 [==============================] - 1s 37ms/step - loss: 0.1084 - accuracy:
0.9567 - val_loss: 1.0629 - val_accuracy: 0.8010
Epoch 20/20
16/16 [==============================] - 0s 31ms/step - loss: 0.1036 - accuracy:
0.9607 - val_loss: 1.2412 - val_accuracy: 0.7760
```

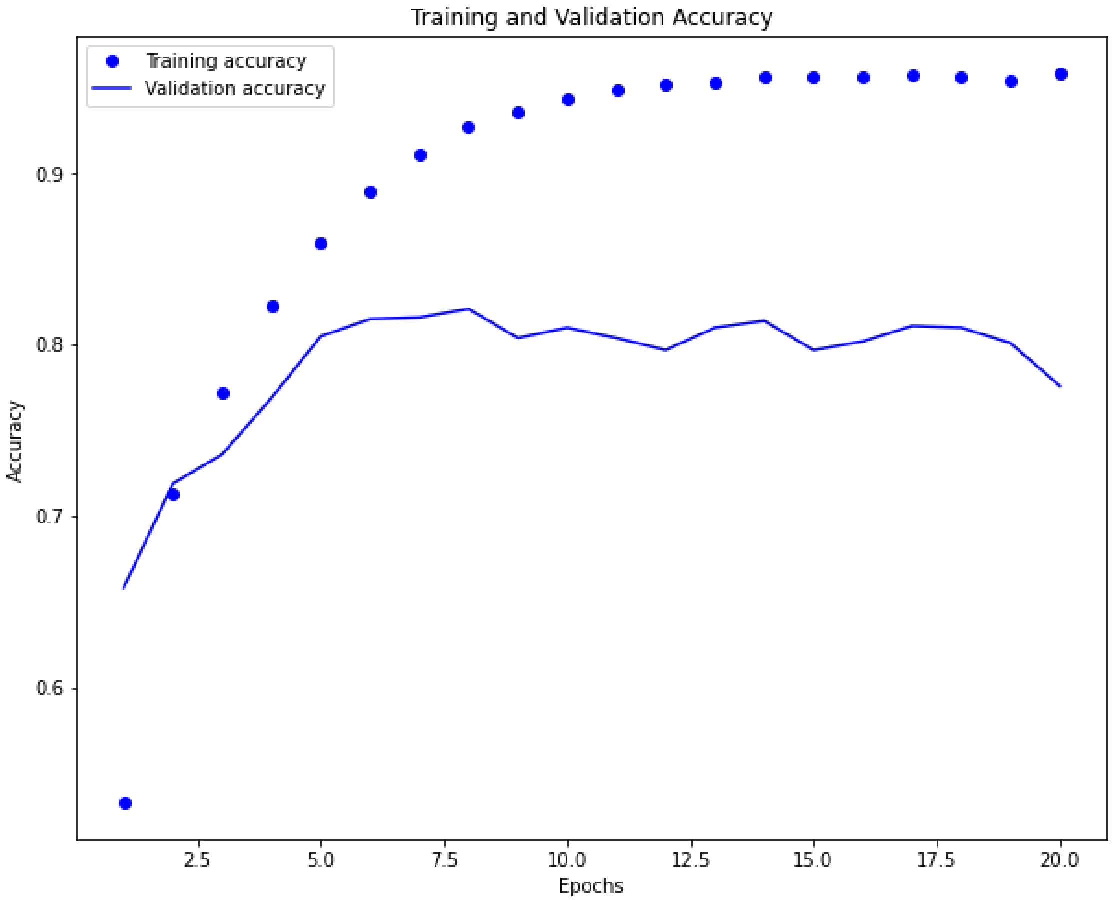
[16]:
```
import matplotlib.pyplot as plt
```

[17]:
```
history_dict = history.history
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1,len(acc)+ 1)
```

[18]:
```
plt.figure(figsize=(10,8))
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()
```



[19]:
```
plt.figure(figsize=(10,8))
plt.plot(epochs, acc,  'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[20]:
```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(partial_X_train,
          partial_y_train,
          epochs=9,
          batch_size=512,
          validation_data=(X_val, y_val))
```

```
Epoch 1/9
16/16 [==============================] - 2s 56ms/step - loss: 3.2293 - accuracy:
0.4208 - val_loss: 1.7683 - val_accuracy: 0.6430
Epoch 2/9
16/16 [==============================] - 1s 35ms/step - loss: 1.5527 - accuracy:
```

```
0.6963 - val_loss: 1.3136 - val_accuracy: 0.6970
Epoch 3/9
16/16 [==============================] - 1s 34ms/step - loss: 1.1197 - accuracy:
0.7580 - val_loss: 1.1324 - val_accuracy: 0.7570
Epoch 4/9
16/16 [==============================] - 1s 35ms/step - loss: 0.8585 - accuracy:
0.8135 - val_loss: 1.0375 - val_accuracy: 0.7820
Epoch 5/9
16/16 [==============================] - 1s 35ms/step - loss: 0.6747 - accuracy:
0.8603 - val_loss: 0.9665 - val_accuracy: 0.7960
Epoch 6/9
16/16 [==============================] - 1s 33ms/step - loss: 0.5241 - accuracy:
0.8939 - val_loss: 0.9301 - val_accuracy: 0.8090
Epoch 7/9
16/16 [==============================] - 1s 35ms/step - loss: 0.4275 - accuracy:
0.9152 - val_loss: 0.8728 - val_accuracy: 0.8210
Epoch 8/9
16/16 [==============================] - 1s 33ms/step - loss: 0.3425 - accuracy:
0.9298 - val_loss: 0.8895 - val_accuracy: 0.8170
Epoch 9/9
16/16 [==============================] - 1s 33ms/step - loss: 0.2848 - accuracy:
0.9384 - val_loss: 0.8993 - val_accuracy: 0.8140
```

```
[20]: <tensorflow.python.keras.callbacks.History at 0x2c308f44e20>
```

[21]:
```
results = model.evaluate(X_test, y_test)
results
```

```
71/71 [==============================] - 0s 2ms/step - loss: 1.0084 - accuracy:
0.7863
```

```
[21]: [1.0083519220352173, 0.7862867116928101]
```

[22]:
```
# Establishing a random classification baseline
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
hits_array = np.array(test_labels) == np.array(test_labels_copy)
float(np.sum(hits_array)) / len(test_labels)
```

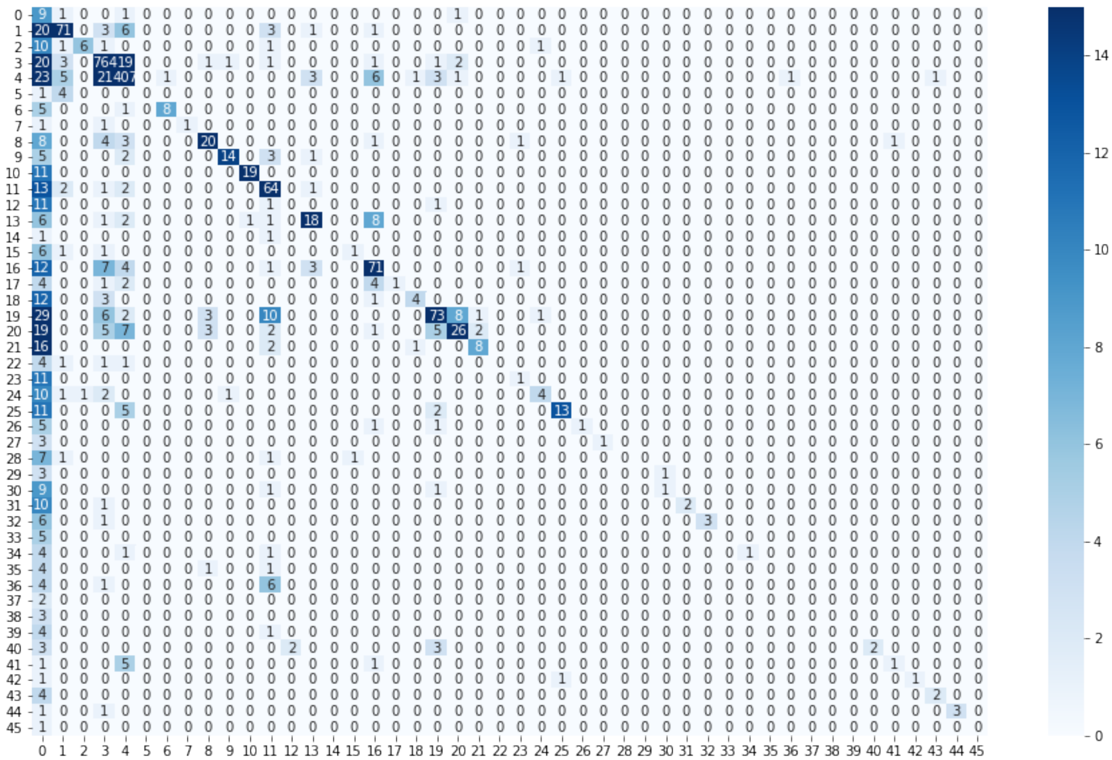
```
[22]: 0.1856634016028495
```

1.5 3.5.5 Generating predictions on new data

[23]:
```
predictions = model.predict(X_test)
```

[24]:
```
import sklearn.metrics as metrics
from seaborn import heatmap
```

```
confusion_matrix = metrics.confusion_matrix(y_true=y_test.argmax(axis=1),␣
↪y_pred=np.round(predictions).argmax(axis=1))
fig, ax = plt.subplots(figsize=(16,10))
heatmap(confusion_matrix, annot = True, cmap='Blues', fmt='g', ax = ax,␣
↪vmax=15);
```



[25]:
```
predictions[0].shape
```

[25]: (46,)

[26]:
```
np.sum(predictions[0])
```

[26]: 1.0

[27]:
```
np.argmax(predictions[0])
```

[27]: 3

1.6 3.5.6 A different way to handle the labels and the loss

[28]:
```
y_train2 = np.array(train_labels)
y_test2 = np.array(test_labels)
```

[29]:
```
y_val2 = y_train2[:1000]
partial_y_train2 = y_train2[1000:]
```

[30]:
```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

history = model.fit(partial_X_train,
                    partial_y_train2,
                    epochs=9,
                    batch_size=512,
                    validation_data=(X_val, y_val2), verbose = False)
results = model.evaluate(X_test, y_test2)
results
```

```
71/71 [==============================] - 0s 2ms/step - loss: 0.9888 - acc:
0.7916
```

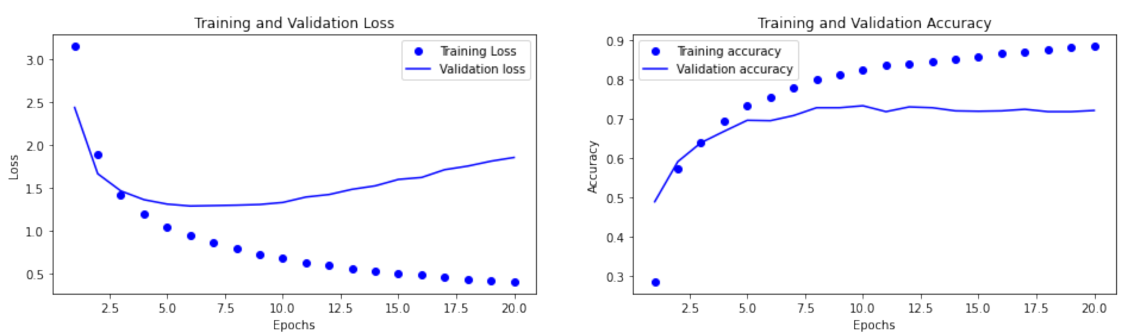
[30]:
```
[0.9887732267379761, 0.7916295528411865]
```

1.7 3.5.7 The importance of having sufficiently large intermediate layers

[31]:
```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(X_val, y_val),
                    verbose = False)
```

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

history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
fig, [ax1, ax2] = plt.subplots(1,2, figsize=(16,4))

ax1.plot(epochs, loss_values,  'bo', label = 'Training Loss')
ax1.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.legend()

ax2.plot(epochs, acc,  'bo', label = 'Training accuracy')
ax2.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.legend()
plt.show()
```

```
71/71 [==============================] - 0s 2ms/step - loss: 2.2046 - acc:
0.6915
[2.20456600189209, 0.6914514899253845]
```



1.8 3.5.8 Further experiments

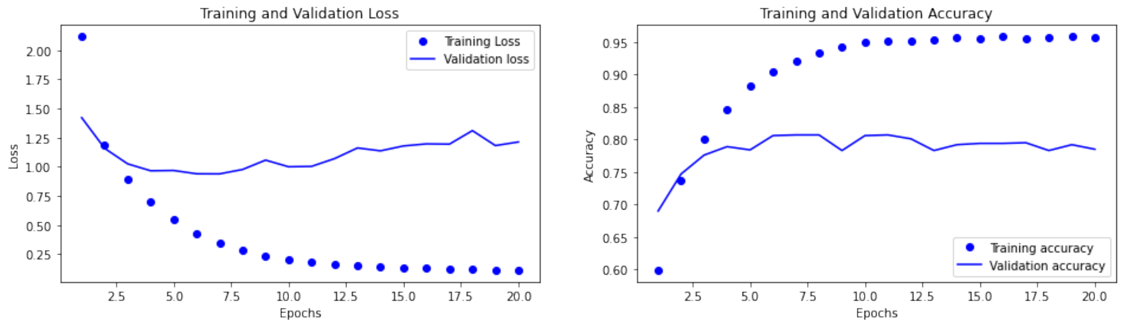
Try using larger or smaller layers: 32 units, 128 units, and so on.

1.8.1 32 Units

[32]:
```
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(X_val, y_val),
                    verbose = False)

results = model.evaluate(X_test, y_test)
print(results)

history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
fig, [ax1, ax2] = plt.subplots(1,2, figsize=(16,4))

ax1.plot(epochs, loss_values,  'bo', label = 'Training Loss')
ax1.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.legend()

ax2.plot(epochs, acc,  'bo', label = 'Training accuracy')
ax2.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
```

```
ax2.legend()
plt.show()
```

```
71/71 [==============================] - 0s 2ms/step - loss: 1.4523 - acc:
0.7752
[1.4522982835769653, 0.7751558423042297]
```

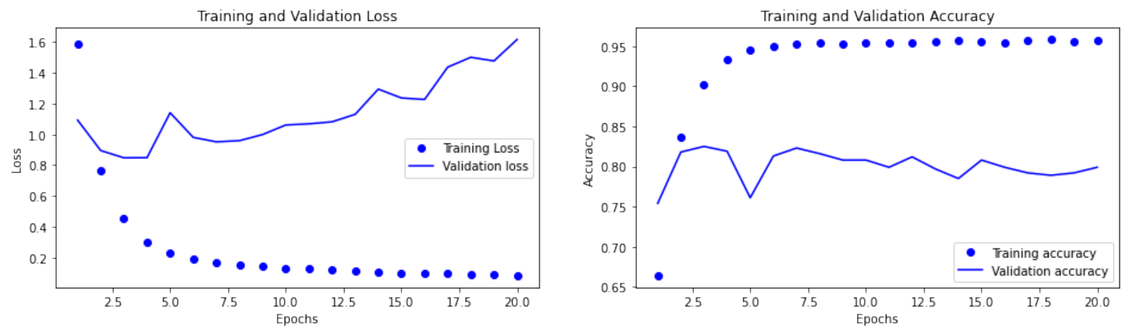


1.8.2 128 Units

[33]:
```
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(X_val, y_val),
                    verbose = False)

results = model.evaluate(X_test, y_test)
print(results)

history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)
```

```
# Plotting metrics
fig, [ax1, ax2] = plt.subplots(1,2, figsize=(16,4))

ax1.plot(epochs, loss_values,  'bo', label = 'Training Loss')
ax1.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.legend()

ax2.plot(epochs, acc,  'bo', label = 'Training accuracy')
ax2.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.legend()
plt.show()
```

```
71/71 [==============================] - 0s 3ms/step - loss: 2.0258 - acc:
0.7823
[2.025777578353882, 0.7822796106338501]
```



You used two hidden layers. Now try using a single hidden layer, or three hidden layers

1.8.3 1 Hidden Layer

[34]:
```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])
```

```python
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(X_val, y_val),
                    verbose = False)

results = model.evaluate(X_test, y_test)
print(results)

history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
fig, [ax1, ax2] = plt.subplots(1,2, figsize=(16,4))

ax1.plot(epochs, loss_values,  'bo', label = 'Training Loss')
ax1.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.legend()

ax2.plot(epochs, acc,  'bo', label = 'Training accuracy')
ax2.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.legend()
plt.show()
```

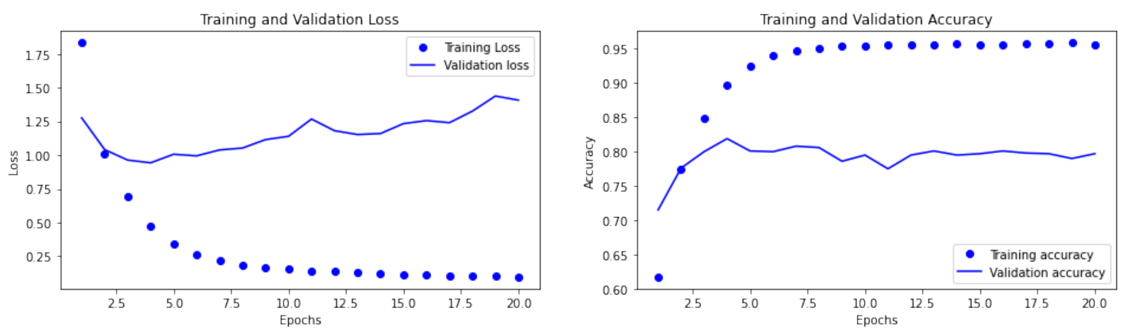
```
71/71 [==============================] - 0s 2ms/step - loss: 1.3920 - acc:
0.7850
[1.392011284828186, 0.7849510312080383]
```

1.8.4 3 Hidden Layers

[35]:
```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(X_val, y_val),
                    verbose = False)

results = model.evaluate(X_test, y_test)
print(results)

history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
fig, [ax1, ax2] = plt.subplots(1,2, figsize=(16,4))

ax1.plot(epochs, loss_values,  'bo', label = 'Training Loss')
ax1.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
```

```
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.legend()

ax2.plot(epochs, acc,  'bo', label = 'Training accuracy')
ax2.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.legend()
plt.show()
```

```
71/71 [==============================] - 0s 2ms/step - loss: 1.7357 - acc:
0.7778
[1.7356630563735962, 0.777827262878418]
```



[ ]: