Project-9

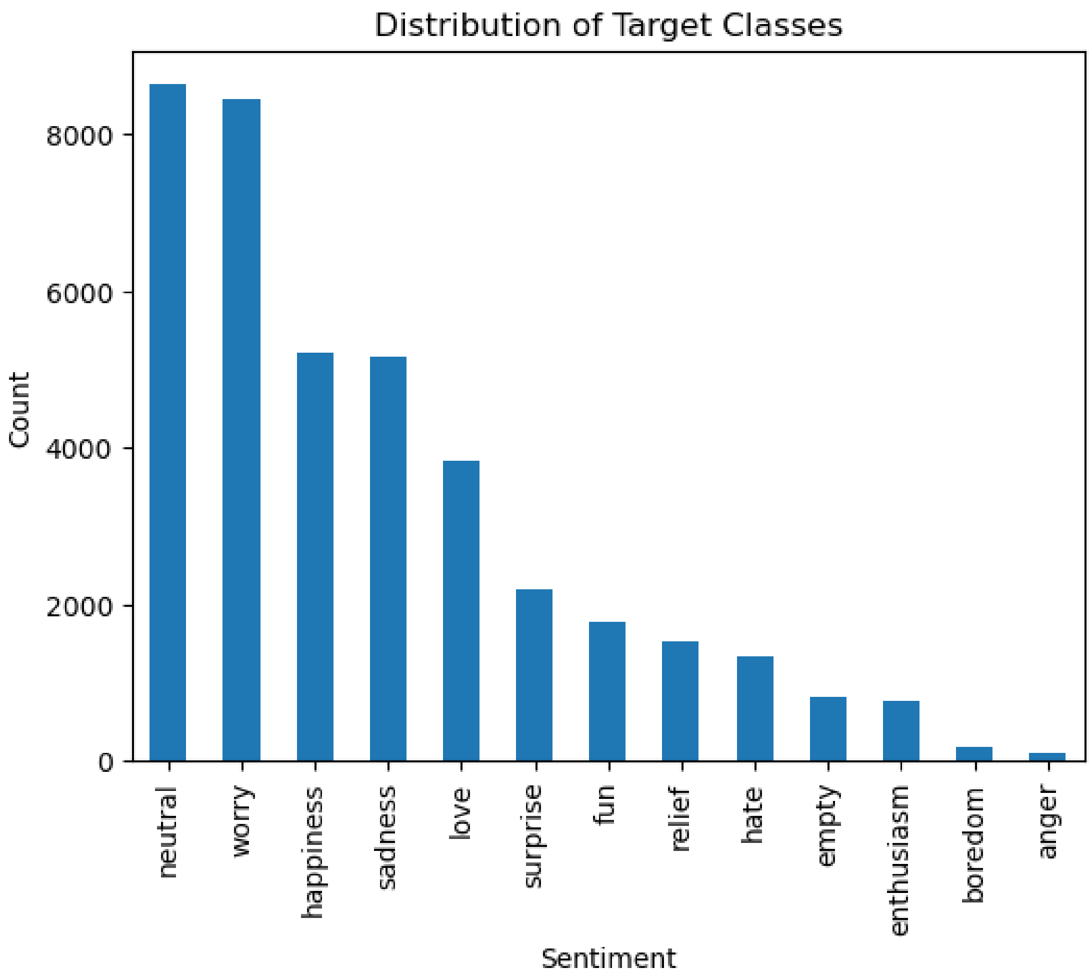
April 19, 2023

[1]:
```python
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from keras import models
from keras import layers
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
```

[2]:
```python
'''
Go to Kaggle.com. Find a text classification data set that interests you.␣
↪Divide into train/test.
Create a graph showing the distribution of the target classes. Describe the␣
↪data set and what the
model should be able to predict.
'''
# Load the df & remove redundant column
df = pd.read_csv('tweet_emotions.csv')
df = df.drop('tweet_id', axis=1)

# Test/Train Split
train_data, test_data, train_labels, test_labels =␣
↪train_test_split(df['content'], df['sentiment'], test_size=0.5,␣
↪random_state=1337)

print('Size of training and test data:', train_labels.shape, test_labels.shape)

# Graph the distribution of the target classes (Sentiment)
fig, ax = plt.subplots()
df['sentiment'].value_counts().plot(kind='bar', ax=ax)

ax.set_title('Distribution of Target Classes')
```

```
ax.set_xlabel('Sentiment')
ax.set_ylabel('Count')

plt.show()
```

```
Size of training and test data: (20000,) (20000,)
```



[3]:
```
'''
Create a sequential model and evaluate on the test data
'''
# Vectorize/Encode the data
vectorizer = TfidfVectorizer(stop_words='english')
label_encoder = LabelEncoder()

x_train = vectorizer.fit_transform(train_data).toarray()
x_test = vectorizer.transform(test_data).toarray()
```

```python
y_train = label_encoder.fit_transform(train_labels)
y_test = label_encoder.transform(test_labels)

# Adjust the input shape to match the length of the vectorized features
input_shape = x_train.shape[1]

# Get the number of unique classes
num_classes = len(label_encoder.classes_)

# Build the model
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(input_shape,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

# Create a validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

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

# Train the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
20/20 [==============================] - 2s 64ms/step - loss: 2.5338 -
sparse_categorical_accuracy: 0.2014 - val_loss: 2.5009 -
val_sparse_categorical_accuracy: 0.2134
Epoch 2/20
20/20 [==============================] - 1s 26ms/step - loss: 2.4687 -
sparse_categorical_accuracy: 0.2090 - val_loss: 2.4331 -
val_sparse_categorical_accuracy: 0.2134
Epoch 3/20
20/20 [==============================] - 1s 29ms/step - loss: 2.3956 -
sparse_categorical_accuracy: 0.2090 - val_loss: 2.3608 -
val_sparse_categorical_accuracy: 0.2134
Epoch 4/20
20/20 [==============================] - 1s 30ms/step - loss: 2.3205 -
```

```
sparse_categorical_accuracy: 0.2090 - val_loss: 2.2921 -
val_sparse_categorical_accuracy: 0.2134
Epoch 5/20
20/20 [==============================] - 1s 26ms/step - loss: 2.2546 -
sparse_categorical_accuracy: 0.2090 - val_loss: 2.2387 -
val_sparse_categorical_accuracy: 0.2134
Epoch 6/20
20/20 [==============================] - 1s 28ms/step - loss: 2.2049 -
sparse_categorical_accuracy: 0.2090 - val_loss: 2.2025 -
val_sparse_categorical_accuracy: 0.2134
Epoch 7/20
20/20 [==============================] - 1s 33ms/step - loss: 2.1685 -
sparse_categorical_accuracy: 0.2090 - val_loss: 2.1772 -
val_sparse_categorical_accuracy: 0.2134
Epoch 8/20
20/20 [==============================] - 1s 27ms/step - loss: 2.1409 -
sparse_categorical_accuracy: 0.2119 - val_loss: 2.1601 -
val_sparse_categorical_accuracy: 0.2201
Epoch 9/20
20/20 [==============================] - 1s 29ms/step - loss: 2.1184 -
sparse_categorical_accuracy: 0.2426 - val_loss: 2.1476 -
val_sparse_categorical_accuracy: 0.2401
Epoch 10/20
20/20 [==============================] - 1s 32ms/step - loss: 2.0979 -
sparse_categorical_accuracy: 0.2903 - val_loss: 2.1370 -
val_sparse_categorical_accuracy: 0.2609
Epoch 11/20
20/20 [==============================] - 1s 27ms/step - loss: 2.0772 -
sparse_categorical_accuracy: 0.3142 - val_loss: 2.1267 -
val_sparse_categorical_accuracy: 0.2624
Epoch 12/20
20/20 [==============================] - 1s 33ms/step - loss: 2.0553 -
sparse_categorical_accuracy: 0.3248 - val_loss: 2.1167 -
val_sparse_categorical_accuracy: 0.2622
Epoch 13/20
20/20 [==============================] - 1s 31ms/step - loss: 2.0317 -
sparse_categorical_accuracy: 0.3199 - val_loss: 2.1066 -
val_sparse_categorical_accuracy: 0.2647
Epoch 14/20
20/20 [==============================] - 1s 27ms/step - loss: 2.0064 -
sparse_categorical_accuracy: 0.3273 - val_loss: 2.0965 -
val_sparse_categorical_accuracy: 0.2640
Epoch 15/20
20/20 [==============================] - 1s 31ms/step - loss: 1.9790 -
sparse_categorical_accuracy: 0.3312 - val_loss: 2.0864 -
val_sparse_categorical_accuracy: 0.2673
Epoch 16/20
20/20 [==============================] - 1s 33ms/step - loss: 1.9491 -
```

```
sparse_categorical_accuracy: 0.3334 - val_loss: 2.0764 -
val_sparse_categorical_accuracy: 0.2653
Epoch 17/20
20/20 [==============================] - 1s 28ms/step - loss: 1.9168 -
sparse_categorical_accuracy: 0.3424 - val_loss: 2.0661 -
val_sparse_categorical_accuracy: 0.2715
Epoch 18/20
20/20 [==============================] - 1s 30ms/step - loss: 1.8831 -
sparse_categorical_accuracy: 0.3606 - val_loss: 2.0572 -
val_sparse_categorical_accuracy: 0.2793
Epoch 19/20
20/20 [==============================] - 1s 32ms/step - loss: 1.8479 -
sparse_categorical_accuracy: 0.3805 - val_loss: 2.0496 -
val_sparse_categorical_accuracy: 0.2799
Epoch 20/20
20/20 [==============================] - 1s 26ms/step - loss: 1.8103 -
sparse_categorical_accuracy: 0.3959 - val_loss: 2.0416 -
val_sparse_categorical_accuracy: 0.2845
```

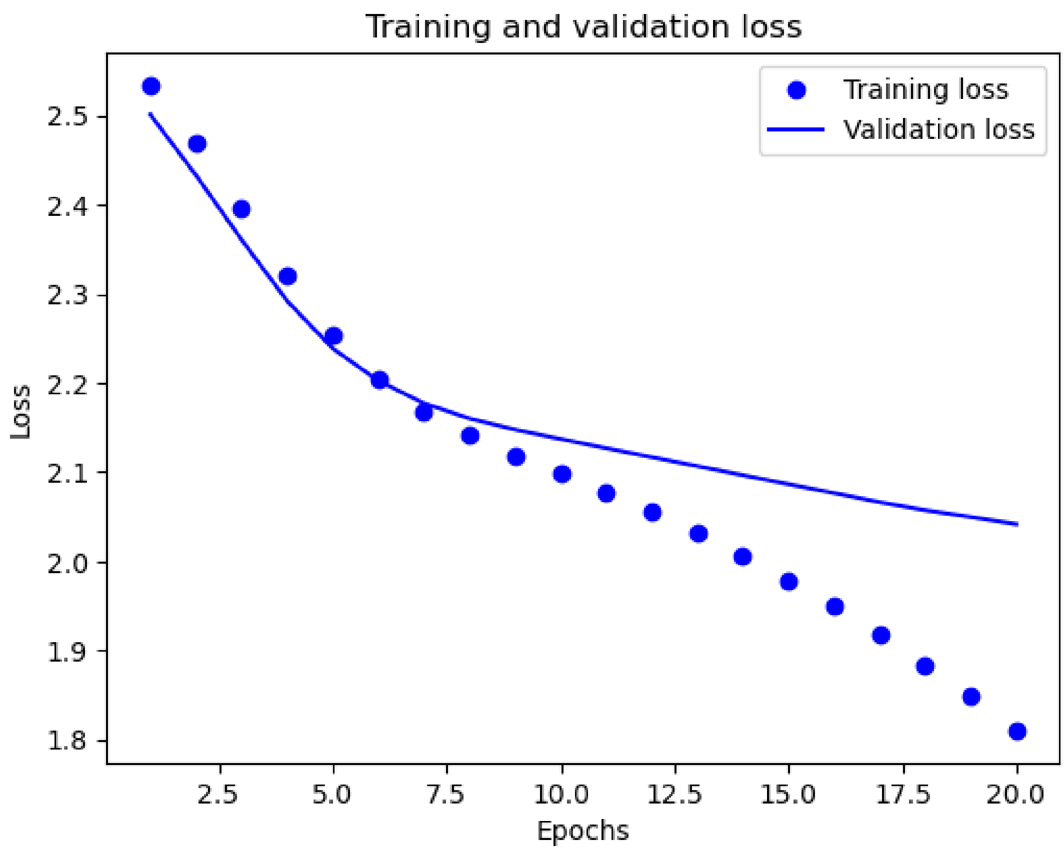
[4]:

```python
# Classifcation report (since it's multi-classification it will have more␣
 ↪values)
print("Classification Report")
prob = model.predict(x_test)
pred = np.argmax(prob, axis=1)
print(classification_report(y_test, pred, zero_division=1)) #Set non-predicted␣
 ↪to 1.00 instead of 0.00

# Tf Evaluation
print("TF Evaluation")
losses_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
print(losses_and_metrics)

# Plot the training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
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()
```

```
Classification Report
625/625 [==============================] - 1s 981us/step
```

	precision	recall	f1-score	support
0	1.00	0.00	0.00	58
1	1.00	0.00	0.00	84
2	1.00	0.00	0.00	423
3	1.00	0.00	0.00	365
4	1.00	0.00	0.00	873
5	0.42	0.00	0.00	2605
6	1.00	0.00	0.00	657
7	0.44	0.29	0.35	1895
8	0.26	0.63	0.37	4395
9	1.00	0.00	0.00	794
10	0.26	0.11	0.15	2555
11	1.00	0.00	0.00	1061
12	0.32	0.55	0.41	4235
accuracy			0.30	20000
macro avg	0.75	0.12	0.10	20000
weighted avg	0.47	0.30	0.22	20000

```
TF Evaluation
157/157 [==============================] - 0s 3ms/step - loss: 2.0219 -
sparse_categorical_accuracy: 0.2964
[2.0219314098358154, 0.2964000105857849]
```



[5]:
```
'''
Try a different architecture like RNN, CNN, etc and evaluate on the test data
'''
```

[5]:
```
'\nTry a different architecture like RNN, CNN, etc and evaluate on the test
data\n'
```

[6]:
```
#RNN
# Tokenize and pad the text sequences
max_words = 10000
max_sequence_length = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_data)

x_train = tokenizer.texts_to_sequences(train_data)
x_test = tokenizer.texts_to_sequences(test_data)

x_train = pad_sequences(x_train, maxlen=max_sequence_length)
x_test = pad_sequences(x_test, maxlen=max_sequence_length)
```

```python
# Encode the labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(train_labels)
y_test = label_encoder.transform(test_labels)

# Get the number of unique classes
num_classes = len(label_encoder.classes_)

# Build the model
model = models.Sequential()
model.add(layers.Embedding(max_words, 128, input_length=max_sequence_length))
model.add(layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

# Create a validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

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

# Train the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/10
79/79 [==============================] - 19s 218ms/step - loss: 2.1909 -
sparse_categorical_accuracy: 0.2263 - val_loss: 2.1311 -
val_sparse_categorical_accuracy: 0.2566
Epoch 2/10
79/79 [==============================] - 17s 220ms/step - loss: 2.0414 -
sparse_categorical_accuracy: 0.2987 - val_loss: 2.0141 -
val_sparse_categorical_accuracy: 0.3041
Epoch 3/10
79/79 [==============================] - 18s 225ms/step - loss: 1.7967 -
sparse_categorical_accuracy: 0.4041 - val_loss: 2.0326 -
val_sparse_categorical_accuracy: 0.3130
Epoch 4/10
79/79 [==============================] - 18s 226ms/step - loss: 1.5522 -
```

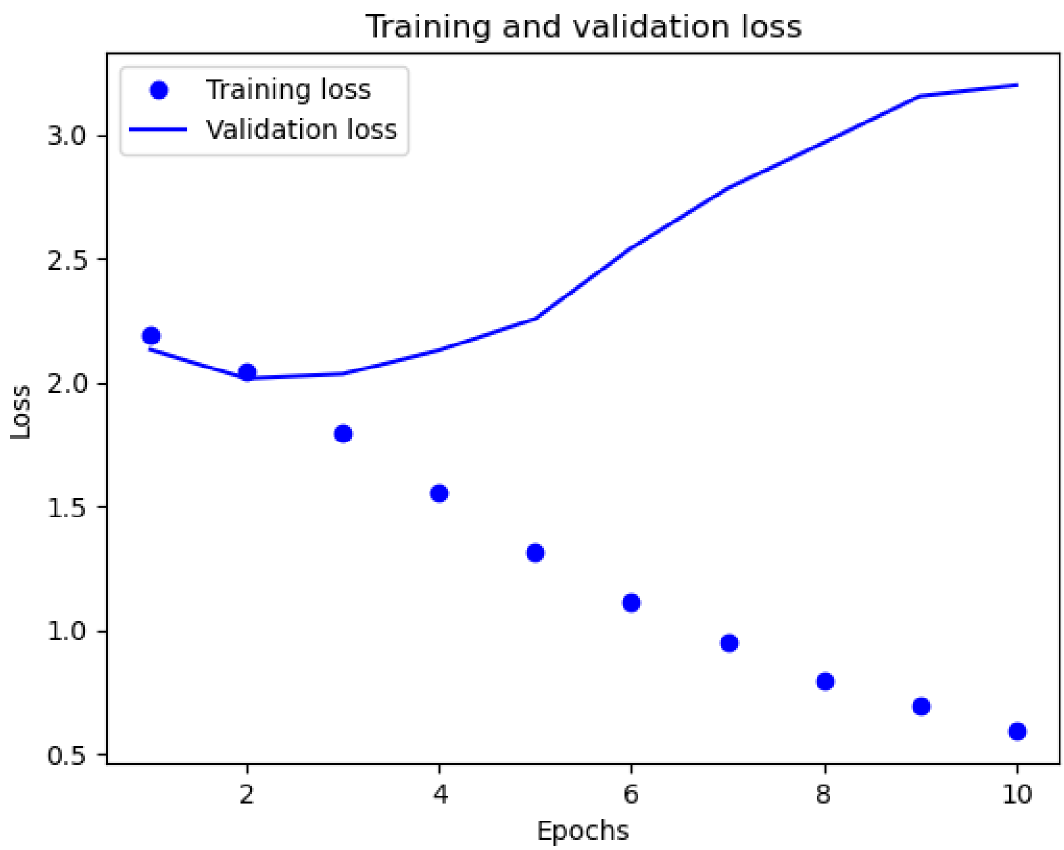
```
sparse_categorical_accuracy: 0.4874 - val_loss: 2.1282 -
val_sparse_categorical_accuracy: 0.2995
Epoch 5/10
79/79 [==============================] - 18s 228ms/step - loss: 1.3166 -
sparse_categorical_accuracy: 0.5733 - val_loss: 2.2554 -
val_sparse_categorical_accuracy: 0.2932
Epoch 6/10
79/79 [==============================] - 18s 230ms/step - loss: 1.1159 -
sparse_categorical_accuracy: 0.6457 - val_loss: 2.5416 -
val_sparse_categorical_accuracy: 0.2705
Epoch 7/10
79/79 [==============================] - 18s 233ms/step - loss: 0.9520 -
sparse_categorical_accuracy: 0.7039 - val_loss: 2.7822 -
val_sparse_categorical_accuracy: 0.2705
Epoch 8/10
79/79 [==============================] - 18s 229ms/step - loss: 0.7942 -
sparse_categorical_accuracy: 0.7532 - val_loss: 2.9667 -
val_sparse_categorical_accuracy: 0.2669
Epoch 9/10
79/79 [==============================] - 18s 233ms/step - loss: 0.6910 -
sparse_categorical_accuracy: 0.7886 - val_loss: 3.1549 -
val_sparse_categorical_accuracy: 0.2652
Epoch 10/10
79/79 [==============================] - 18s 232ms/step - loss: 0.5938 -
sparse_categorical_accuracy: 0.8140 - val_loss: 3.1992 -
val_sparse_categorical_accuracy: 0.2574
```

[9]:

```python
# Classifcation report (since it's multi-classification it will have more values)
print("Classification Report")
prob = model.predict(x_test)
pred = np.argmax(prob, axis=1)
print(classification_report(y_test, pred, zero_division=1)) #Set non-predicted to 1.00 instead of 0.00

# Tf Evaluation
print("TF Evaluation")
losses_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
print(losses_and_metrics)

# Plot the training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
```

```python
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Classification Report
625/625 [==============================] - 6s 10ms/step
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        58
           1       0.00      0.00      0.00        84
           2       0.04      0.04      0.04       423
           3       0.00      0.00      0.00       365
           4       0.08      0.06      0.07       873
           5       0.26      0.31      0.28      2605
           6       0.18      0.15      0.16       657
           7       0.33      0.35      0.34      1895
           8       0.32      0.39      0.35      4395
           9       0.07      0.08      0.08       794
          10       0.25      0.21      0.23      2555
          11       0.11      0.04      0.06      1061
          12       0.32      0.32      0.32      4235

    accuracy                           0.27     20000
   macro avg       0.23      0.15      0.15     20000
weighted avg       0.26      0.27      0.26     20000

TF Evaluation
157/157 [==============================] - 5s 33ms/step - loss: 3.0900 -
sparse_categorical_accuracy: 0.2693
[3.0900070667266846, 0.26930001378059387]
```



[10]:

```
# CNN
# Tokenize and pad
max_words = 10000
max_sequence_length = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_data)

x_train = tokenizer.texts_to_sequences(train_data)
x_test = tokenizer.texts_to_sequences(test_data)

x_train = pad_sequences(x_train, maxlen=max_sequence_length)
x_test = pad_sequences(x_test, maxlen=max_sequence_length)

# Encode the labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(train_labels)
y_test = label_encoder.transform(test_labels)

# Get the number of unique classes
```

```python
num_classes = len(label_encoder.classes_)

# Build the model
model = models.Sequential()
model.add(layers.Embedding(max_words, 128, input_length=max_sequence_length))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

# Create a validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

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

# Train the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(x_val, y_val))
```

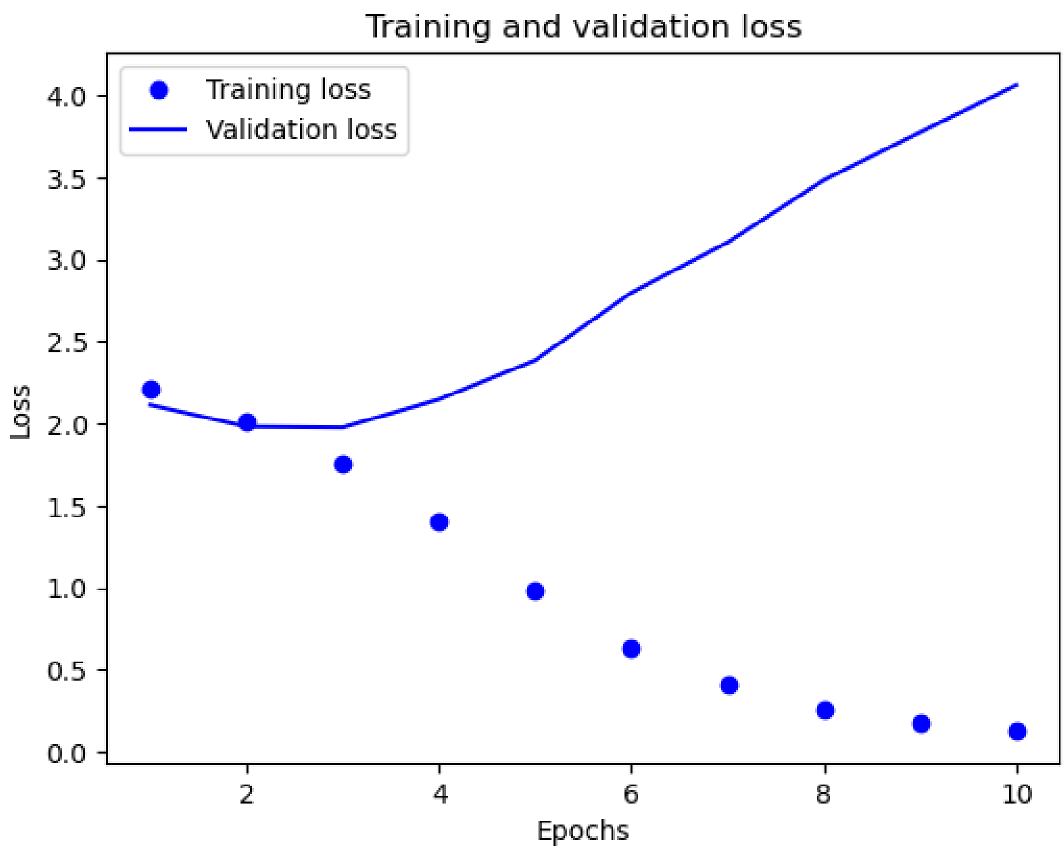
```
Epoch 1/10
79/79 [==============================] - 3s 32ms/step - loss: 2.2066 -
sparse_categorical_accuracy: 0.2150 - val_loss: 2.1135 -
val_sparse_categorical_accuracy: 0.2703
Epoch 2/10
79/79 [==============================] - 3s 32ms/step - loss: 2.0113 -
sparse_categorical_accuracy: 0.3162 - val_loss: 1.9814 -
val_sparse_categorical_accuracy: 0.3314
Epoch 3/10
79/79 [==============================] - 3s 32ms/step - loss: 1.7593 -
sparse_categorical_accuracy: 0.4290 - val_loss: 1.9764 -
val_sparse_categorical_accuracy: 0.3345
Epoch 4/10
79/79 [==============================] - 3s 32ms/step - loss: 1.4015 -
sparse_categorical_accuracy: 0.5638 - val_loss: 2.1469 -
val_sparse_categorical_accuracy: 0.3073
Epoch 5/10
79/79 [==============================] - 3s 32ms/step - loss: 0.9866 -
sparse_categorical_accuracy: 0.7003 - val_loss: 2.3846 -
```

```
val_sparse_categorical_accuracy: 0.3030
Epoch 6/10
79/79 [==============================] - 3s 35ms/step - loss: 0.6367 -
sparse_categorical_accuracy: 0.8202 - val_loss: 2.7964 -
val_sparse_categorical_accuracy: 0.2710
Epoch 7/10
79/79 [==============================] - 2s 31ms/step - loss: 0.4072 -
sparse_categorical_accuracy: 0.8974 - val_loss: 3.1020 -
val_sparse_categorical_accuracy: 0.2820
Epoch 8/10
79/79 [==============================] - 2s 31ms/step - loss: 0.2604 -
sparse_categorical_accuracy: 0.9414 - val_loss: 3.4819 -
val_sparse_categorical_accuracy: 0.2706
Epoch 9/10
79/79 [==============================] - 3s 33ms/step - loss: 0.1770 -
sparse_categorical_accuracy: 0.9616 - val_loss: 3.7725 -
val_sparse_categorical_accuracy: 0.2706
Epoch 10/10
79/79 [==============================] - 3s 33ms/step - loss: 0.1320 -
sparse_categorical_accuracy: 0.9718 - val_loss: 4.0595 -
val_sparse_categorical_accuracy: 0.2569
```

[11]:
```
# Classifcation report (since it's multi-classification it will have more␣
 ↪values)
print("Classification Report")
prob = model.predict(x_test)
pred = np.argmax(prob, axis=1)
print(classification_report(y_test, pred, zero_division=1)) #Set non-predicted␣
 ↪to 1.00 instead of 0.00

# Tf Evaluation
print("TF Evaluation")
losses_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
print(losses_and_metrics)

# Plot the training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
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()
```

```
Classification Report
625/625 [==============================] - 2s 3ms/step
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        58
           1       0.00      0.00      0.00        84
           2       0.03      0.01      0.02       423
           3       0.02      0.02      0.02       365
           4       0.09      0.12      0.10       873
           5       0.28      0.32      0.30      2605
           6       0.20      0.19      0.19       657
           7       0.32      0.34      0.33      1895
           8       0.37      0.30      0.33      4395
           9       0.08      0.11      0.10       794
          10       0.25      0.28      0.26      2555
          11       0.11      0.10      0.11      1061
          12       0.31      0.30      0.30      4235

    accuracy                           0.26     20000
   macro avg       0.24      0.16      0.16     20000
weighted avg       0.27      0.26      0.26     20000

TF Evaluation
157/157 [==============================] - 1s 6ms/step - loss: 3.9375 -
sparse_categorical_accuracy: 0.2609
[3.937485456466675, 0.2609499990940094]
```



[17]:
```
'''
Try different embedding approaches and evaluate on the test data (Evaluation at␣
↪the end)
'''
# Tokenize the text data
max_words = 10000
max_sequence_length = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_data)

x_train_seq = tokenizer.texts_to_sequences(train_data)
x_test_seq = tokenizer.texts_to_sequences(test_data)

# Pad the sequences
x_train = pad_sequences(x_train_seq, maxlen=max_sequence_length)
x_test = pad_sequences(x_test_seq, maxlen=max_sequence_length)

y_train = label_encoder.fit_transform(train_labels)
y_test = label_encoder.transform(test_labels)
```

```python
# Get the number of unique classes
num_classes = len(label_encoder.classes_)

# Build the model
model = models.Sequential()
model.add(layers.Embedding(max_words, 100, input_length=max_sequence_length))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

# Create a validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

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

# Train the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
20/20 [==============================] - 1s 23ms/step - loss: 2.2679 -
sparse_categorical_accuracy: 0.1899 - val_loss: 2.2044 -
val_sparse_categorical_accuracy: 0.2134
Epoch 2/20
20/20 [==============================] - 0s 20ms/step - loss: 2.1948 -
sparse_categorical_accuracy: 0.2146 - val_loss: 2.1948 -
val_sparse_categorical_accuracy: 0.2111
Epoch 3/20
20/20 [==============================] - 0s 19ms/step - loss: 2.1814 -
sparse_categorical_accuracy: 0.2156 - val_loss: 2.1736 -
val_sparse_categorical_accuracy: 0.2111
Epoch 4/20
20/20 [==============================] - 0s 19ms/step - loss: 2.1690 -
sparse_categorical_accuracy: 0.2204 - val_loss: 2.1661 -
val_sparse_categorical_accuracy: 0.2111
Epoch 5/20
```

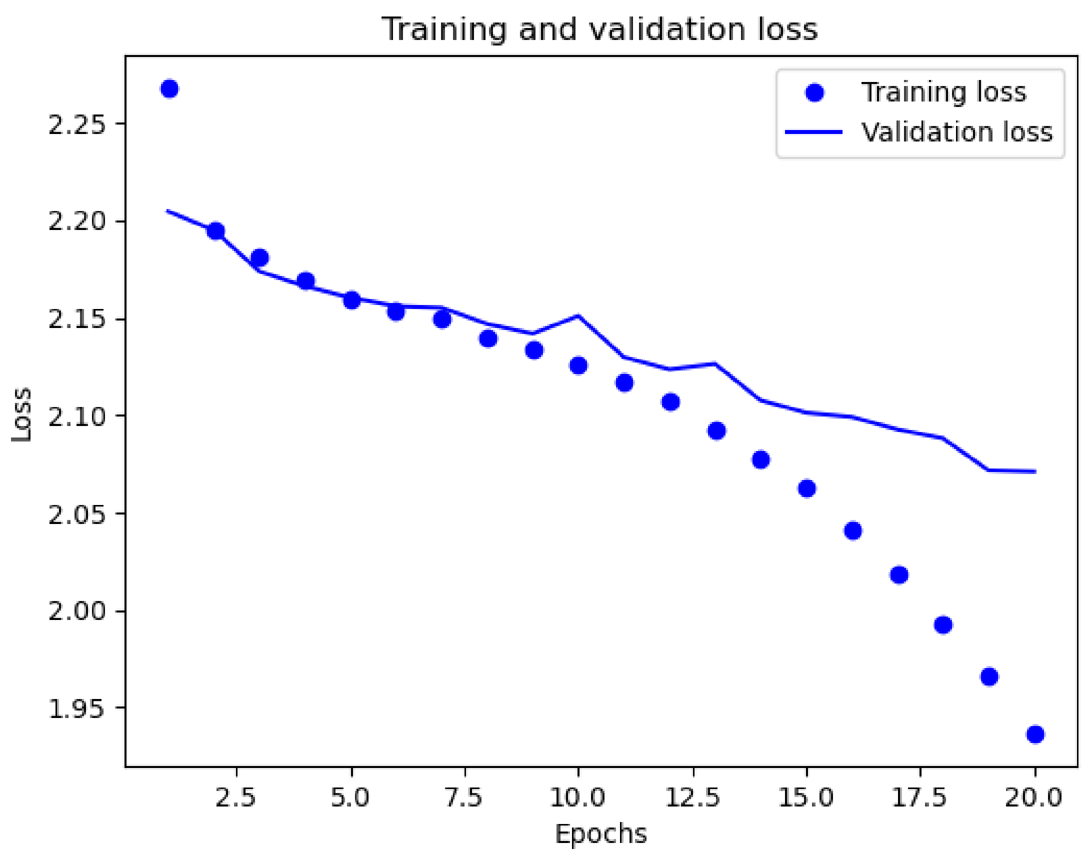
```
20/20 [==============================] - 0s 19ms/step - loss: 2.1599 -
sparse_categorical_accuracy: 0.2204 - val_loss: 2.1602 -
val_sparse_categorical_accuracy: 0.2499
Epoch 6/20
20/20 [==============================] - 0s 17ms/step - loss: 2.1538 -
sparse_categorical_accuracy: 0.2309 - val_loss: 2.1558 -
val_sparse_categorical_accuracy: 0.2122
Epoch 7/20
20/20 [==============================] - 0s 20ms/step - loss: 2.1496 -
sparse_categorical_accuracy: 0.2362 - val_loss: 2.1550 -
val_sparse_categorical_accuracy: 0.2546
Epoch 8/20
20/20 [==============================] - 0s 18ms/step - loss: 2.1398 -
sparse_categorical_accuracy: 0.2408 - val_loss: 2.1467 -
val_sparse_categorical_accuracy: 0.2542
Epoch 9/20
20/20 [==============================] - 0s 17ms/step - loss: 2.1339 -
sparse_categorical_accuracy: 0.2433 - val_loss: 2.1417 -
val_sparse_categorical_accuracy: 0.2537
Epoch 10/20
20/20 [==============================] - 0s 21ms/step - loss: 2.1259 -
sparse_categorical_accuracy: 0.2489 - val_loss: 2.1508 -
val_sparse_categorical_accuracy: 0.2567
Epoch 11/20
20/20 [==============================] - 0s 19ms/step - loss: 2.1171 -
sparse_categorical_accuracy: 0.2521 - val_loss: 2.1296 -
val_sparse_categorical_accuracy: 0.2571
Epoch 12/20
20/20 [==============================] - 0s 19ms/step - loss: 2.1073 -
sparse_categorical_accuracy: 0.2540 - val_loss: 2.1234 -
val_sparse_categorical_accuracy: 0.2595
Epoch 13/20
20/20 [==============================] - 0s 18ms/step - loss: 2.0922 -
sparse_categorical_accuracy: 0.2609 - val_loss: 2.1262 -
val_sparse_categorical_accuracy: 0.2582
Epoch 14/20
20/20 [==============================] - 0s 18ms/step - loss: 2.0775 -
sparse_categorical_accuracy: 0.2713 - val_loss: 2.1075 -
val_sparse_categorical_accuracy: 0.2645
Epoch 15/20
20/20 [==============================] - 0s 18ms/step - loss: 2.0630 -
sparse_categorical_accuracy: 0.2756 - val_loss: 2.1011 -
val_sparse_categorical_accuracy: 0.2748
Epoch 16/20
20/20 [==============================] - 0s 18ms/step - loss: 2.0407 -
sparse_categorical_accuracy: 0.2859 - val_loss: 2.0990 -
val_sparse_categorical_accuracy: 0.2696
Epoch 17/20
```

```
20/20 [==============================] - 0s 18ms/step - loss: 2.0184 -
sparse_categorical_accuracy: 0.2976 - val_loss: 2.0926 -
val_sparse_categorical_accuracy: 0.2851
Epoch 18/20
20/20 [==============================] - 0s 22ms/step - loss: 1.9928 -
sparse_categorical_accuracy: 0.3150 - val_loss: 2.0880 -
val_sparse_categorical_accuracy: 0.2793
Epoch 19/20
20/20 [==============================] - 0s 21ms/step - loss: 1.9666 -
sparse_categorical_accuracy: 0.3236 - val_loss: 2.0716 -
val_sparse_categorical_accuracy: 0.2903
Epoch 20/20
20/20 [==============================] - 0s 19ms/step - loss: 1.9366 -
sparse_categorical_accuracy: 0.3388 - val_loss: 2.0709 -
val_sparse_categorical_accuracy: 0.2897
```

[18]:
```
# Classifcation report (since it's multi-classification it will have more␣
 ↪values)
print("Classification Report")
prob = model.predict(x_test)
pred = np.argmax(prob, axis=1)
print(classification_report(y_test, pred, zero_division=1)) #Set non-predicted␣
 ↪to 1.00 instead of 0.00

# Tf Evaluation
print("TF Evaluation")
losses_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
print(losses_and_metrics)

# Plot the training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
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()
```

```
Classification Report
625/625 [==============================] - 1s 953us/step
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        58
```

```
           1       1.00      0.00      0.00        84
           2       1.00      0.00      0.00       423
           3       1.00      0.00      0.00       365
           4       1.00      0.00      0.00       873
           5       0.27      0.17      0.21      2605
           6       1.00      0.00      0.00       657
           7       0.54      0.17      0.25      1895
           8       0.29      0.54      0.38      4395
           9       1.00      0.00      0.00       794
          10       1.00      0.00      0.00      2555
          11       1.00      0.00      0.00      1061
          12       0.29      0.67      0.41      4235

    accuracy                           0.30     20000
   macro avg       0.80      0.12      0.10     20000
weighted avg       0.56      0.30      0.22     20000

TF Evaluation
157/157 [==============================] - 0s 2ms/step - loss: 2.0579 -
sparse_categorical_accuracy: 0.2986
[2.0579071044921875, 0.2985999882221222]
```



[8]:
```python
'''
Write up your analysis of the performance of various approaches
'''

'''
For my program I ran multi-classification rather than a binary classification.
This meant some changes in how the functionality was implemented but the␣
 ↪biggest change was a lower base accuracy.
This was probably due to more choices and more complexity leading to more␣
 ↪possibilities for mistakes.

For the sequential model the data was pretty inaccurate, the highest sparse␣
 ↪categorical accuracy at 20 was 0.3959.
It did have a pretty steady climb in accuracy and was pretty fast to compute␣
 ↪each epoch though.

For RNN the data started out relatively similar in accuracy but went much␣
 ↪higher with each epoch.
The highest sparse categorical accuracy was 0.8140 which was a significant␣
 ↪improvement.
The time to compute each epoch was significantly higher than the sequential␣
 ↪model.

CNN was overall the best, it went up quickly in accuracy for each epoch and␣
 ↪reached the highest sparse categorical accuracy.
It ended up at 0.9718 for the last epoch, and computed fairly quickly.

For the TF evaluation the sequential model was actually the highest in accuracy␣
 ↪but they all ranked fairly similarly.

For training & loss validation RNN & CNN performed similarly, with training␣
 ↪loss going up and validation loss going down.
For sequential both the training and validation loss went down a bit.

When adding embedding to the sequential model it resulted in a lower amount of␣
 ↪training loss with more epochs.
It also resulted in slightly higher sparse categorical accuracy. Overall it was␣
 ↪an improvement.
'''
```

[8]:
```
'\nFor my program I ran multi-classification rather than a binary
classification. \nThis meant some changes in how the functionality was
implemented but the biggest change was an overall lower accuracy.\nThis was
probably due to more choices and more complexity leading to more possibilities
for mistakes.\n\n\n'
```