Pros/Cons - Sentiment Analysis: Neural Network

By: Tyler McKay – 151087m

Cleaning the Data

- Remove extraneous info
  - string.replace()
  - regex.substitute()
- Convert the data into .csv files for easier access

```python
train_path = str("train_dirty.txt")
train_clean_path = str("test_dirty.csv")

dirty_lines = []
clean_lines = []

with open(train_path) as file:
    for line in file.readlines():
        line = re.sub(r"  +", '', line)
        line = re.sub(r"[\n\t]*", '', line)
        dirty_lines.append(line)
        line = line.replace("\"", "\'")
        line = line.replace("<Pros>", "\"")
        line = line.replace("<Cons>", "\"")
        line = line.replace("</Pros>", "\",1")
        line = line.replace("</Cons>", "\",0")
        clean_lines.append(line)

with open(train_clean_path, 'w') as file:
    for line in clean_lines:
        file.write(line + '\n')
```

✓ 0.7s

<Cons>not good for indoor pics, no zoom or lense adjustments</Cons>

=> *"not good for indoor pics, no zoom or lense adjustments",0*

Split into Train, Validation, and Test sets

- Read in data from .csv files
- Split up input data
  - 65% train (1300 instances)
  - 35% validation (700 instances)
- Prediction data
  - 43,000 instances

```python
validation_split = 0.65

predictData = pandas.read_csv(predict_clean_path, encoding='unicode_escape')
predictData.columns = ["Text", "Pro/Con"]

inputData = pandas.read_csv(train_clean_path, encoding='unicode_escape')
inputData.columns = ["Text", "Pro/Con"]

trainData = inputData.sample(frac=validation_split)
validationData = inputData.drop(trainData.index)

print("Train & Validation")
print(trainData.shape)
print(validationData.shape)
print("Predictions")
print(predictData.shape)
```

✓ 0.9s

```
Train & Validation
(1299, 2)
(700, 2)
Predictions
(43874, 2)
```

Tokenize Text

```
tokenizer = tfpp.text.Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_text)

train_sequences = tokenizer.texts_to_sequences(train_text)
train_padded = tfpp.sequence.pad_sequences(train_sequences, maxlen=max_length, truncating=trunc_type)

validation_sequences = tokenizer.texts_to_sequences(validation_text)
validation_padded = tfpp.sequence.pad_sequences(validation_sequences, maxlen=max_length)

predict_sequences = tokenizer.texts_to_sequences(predict_text)
predict_padded = tfpp.sequence.pad_sequences(predict_sequences, maxlen=max_length)
```

✓ 0.7s

- Use the *Tokenizer* class from TensorFlow
- Create an internal word vocabulary with *tokenizer*.**fit_on_texts()**
- Convert words into indexes if found in the vocabulary with *tokenizer*.**texts_to_sequences()**
- Pad out sequences so they are of all equal length with **pad_sequences()**

Model Architecture

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

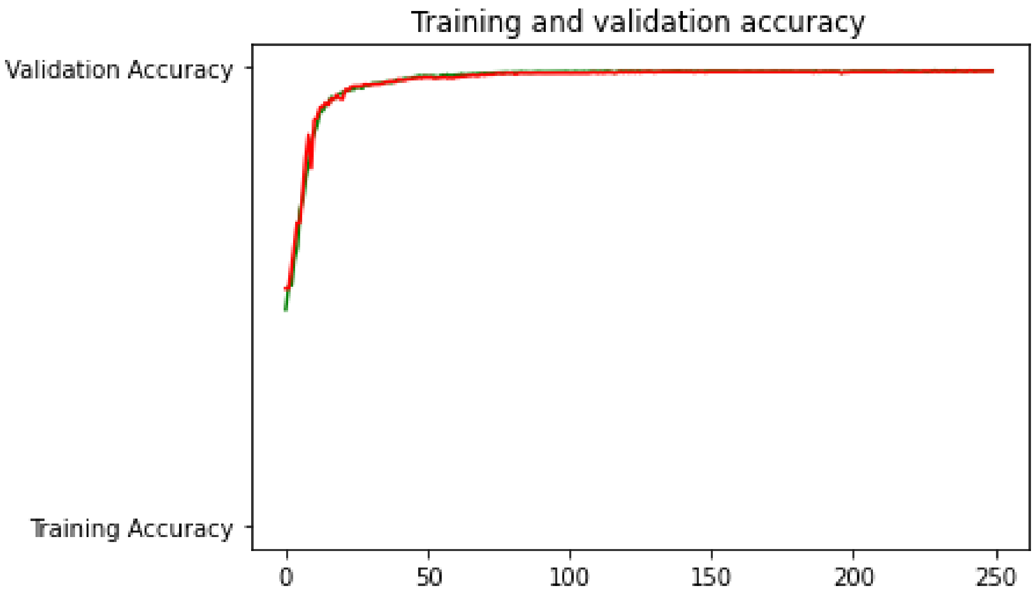
- Embedding layer is used to convert the word indexes into vectors usable in dense layers
- A Bidirectional Long Short-Term Memory (LSTM) layer which is used in language processing
    - Feeds the tokens into the layer in both directions
    - Allows for the model to better understand the tokens and their significance
- 2 Dense layers with 16 inputs each
- Final Dense layer with a single output to indicate either a ‘Pro’ or a ‘Con’

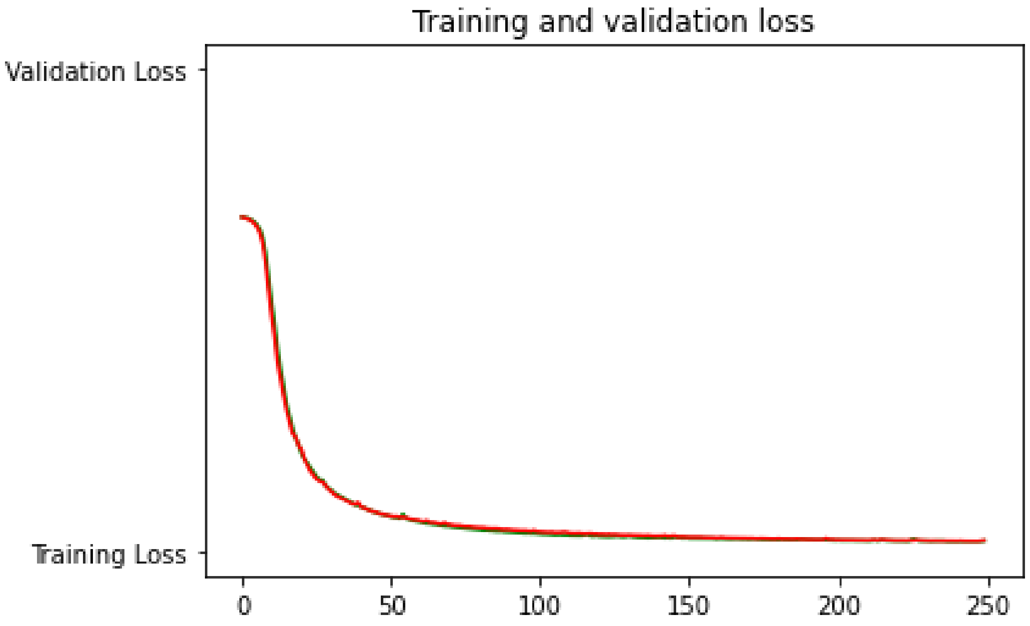
Training

```python
num_epochs = 250

model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate),
    metrics=[tf.metrics.BinaryAccuracy(name="accuracy")],
)

history: list = model.fit(
    train_padded,
    train_label,
    epochs=num_epochs,
    validation_data=(validation_padded, validation_label),
).history

model.save(str(f".\models\model_{num_epochs}e.h5"))
```

✓ 24m 7.9s

- Train on the training data with validation data to avoid overfitting
- 250 epochs





Predictions

- Use the trained model to make predictions
- Loop over each of the predictions and classify into either ‘Pro’ or ‘Con’

```
predictions: numpy.array = model.predict(predict_padded, batch_size=64, verbose=1)
```

✓ 17.4s

```
686/686 [==============================] - 17s 24ms/step
```

Classify the predictions into 'Pro or 'Con

```
classifications = []
for value in predictions:
    if (value > 0.5): classifications.append("Pro")
    else: classifications.append("Con")
classifications = numpy.asarray(classifications)

print(predictions[:6].T)
print(classifications[:6].T)
```

✓ 0.1s

```
[[9.9998820e-01 9.9998420e-01 9.9999058e-01 9.9998295e-01 3.3100920e-07
  1.1033661e-06]]
['Pro' 'Pro' 'Pro' 'Pro' 'Con' 'Con']
```

Comparing Predictions Against the Text

```
for index in random.sample(range(len(predict_text)), 8):
    print(predict_text[index])
    print(f" => {classifications[index]}")
```

✓ 0.4s

```
speed, simplicity, quality output
 => Pro
No color screen, screen size small, a little bit heavy and bulky.
 => Con
All in one device, 2 batteries, color screen, good browser, Palm software.
 => Pro
Excellent optics, SLR in digital is here now
 => Pro
none
 => Con
Durable build, tons of options, responsive menus, great reception
 => Pro
Uses batteries quickly, recommend an AC adapter.
 => Con
inexpensive
 => Pro
```

- Loop over the classifications and prediction text
- Compare the text with the classification
- Lots of correct predictions on first glance
- However certain mistakes are clear
- ‘none’ produces a ‘Con’ classification but it is not evident if that is true or not
- ‘inexpensive/cheap’ could be a ‘Pro’ or ‘Con’ depending on your perspective

Let’s confuse it now…

"Not great, but if you are working on a budget, you can't do worse“
=> **Pro**

"Absolutely the worst“
=> **Pro**

"Broken upon arrival“
=> **Pro**

"I want to love this, but can't really recommend“
=> **Con**

"Pickup when on sale, not worth full price“
=> **Pro**

"Great for amateurs“
=> **Pro**

"Recommend only to professionals“
=> **Pro**

"fantastic for dumpsters“
=> **Pro**

"recommend if you have no idea what you are doing“
=> **Con**

"goes right in the dumpster“
=> **Pro**