Report: Language Translation using Recurring Neural Networks

Introduction:

For my last project, I decided to work on a Language Translation model that would utilize Recurring Neural Networks to take the origin language, in this case English, and translate set sentences to the target language, which were German and Spanish.

The data used for the project was taken from the European Comission, in their Language Technologies Resources tab, and it's comprised of a piece of legislation named the 'Acquis Communautaire', which was translated to over 20 languages. The German and Spanish translations were used to train the model, and ultimately the German translation was used for the final model due to the large scale of each dataset. Both datasets were converted to csv files in the data wrangling stage and cleaned, which was minimal as the data was ultimately very clean.

After the data was cleaned, the dataset was preprocessed and split into testing and training sets to create the model. The model used was simple Neural Network that would take the English sentence and translate it to the target model. The model yielded around 91% accuracy, ultimately.

Stakeholders:

This project is to be used by the everyday person as tool to communicate in German, when the primary language is English.

Data Overview:

Data for the project was collected from the European Commission as text files and then converted to csv files. Each file, which for this project was the file for German and Spanish, had over 5 million sentences that had been translated to English.

Data source:
https://joint-research-centre.ec.europa.eu/language-technology-resources/dgt-translation-memory_en

Out[9]:

	EN	DE
0	Commission Regulation (EC) No 1788/2004	Verordnung (EG) Nr. 1788/2004 der Kommission
1	of 15 October 2004	vom 15. Oktober 2004
2	fixing the minimum selling prices for butter f...	zur Festsetzung der Mindestverkaufspreise für ...
3	THE COMMISSION OF THE EUROPEAN COMMUNITIES,	DIE KOMMISSION DER EUROPÄISCHEN GEMEINSCHAFTEN —
4	Having regard to the Treaty establishing the E...	gestützt auf den Vertrag zur Gründung der Euro...

In [10]:
```
EN_ES.head()
```

Out[10]:

	EN	ES
0	Commission Regulation (EC) No 1788/2004	Reglamento (CE) no 1788/2004 de la Comisión
1	of 15 October 2004	de 15 de octubre de 2004
2	fixing the minimum selling prices for butter f...	por el que se fijan los precios mínimos de ven...
3	THE COMMISSION OF THE EUROPEAN COMMUNITIES,	LA COMISIÓN DE LAS COMUNIDADES EUROPEAS,
4	Having regard to the Treaty establishing the E...	Visto el Tratado constitutivo de la Comunidad ...

In [11]:
```
EN_DE.shape
```

Out[11]: (5693624, 2)

In [12]:
```
EN_ES.shape
```

Out[12]: (5696850, 2)

The data cleaning for this project was very minimal, as for each set the amount of missing values was less than 1% and they were dropped from the datasets. The more significant part of wrangling the data was to create a subsample of each set to be able to created a pair of datasets that could be used for the model efficiently and with the available computer power. Ultimately, a subsample of 50,000 of the entries was created for each language.

The final features of the datasets were:

Original: Original language (‘EN’), target language (‘DE’, ‘ES’)

Created: ‘ENCounter’, ‘ESCounter’, ‘DECounter’, ‘CountDiff’

Preprocessing:

Before creating the model, a Tokenizer and padding functions were created to make the data compatible with the desired model.

The tokenizer function was created as the data was text data and numerical data was needed to run through a Neural Network:

```
# Tokenizer - to prep data for RNN
def tokenize(x):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(x)
    return tokenizer.texts_to_sequences(x), tokenizer
#Test
test = df.iloc[:5,0]
text_tokenized, text_tokenizer = tokenize(test)
print(text_tokenizer.word_index)
for sample_i, (sent, token_sent) in enumerate(zip(test, text_tokenized)):
    print('Sequence {} in x'.format(sample_i + 1))
    print('  Input:  {}'.format(sent))
    print('  Output: {}'.format(token_sent))
```

```
{"'the'": 1, "'": 2, "'for'": 3, "'to'": 4, "'commission'": 5, "'regulation'": 6, 'ec': 7, "'no'": 8, "'of'": 9, "'invitation'": 10, "'tender'": 11,
"'european'": 12, "'1788": 13, "2004'": 14, "'15'": 15, "'october'": 16, "'2004'": 17, "'fixing'": 18, "'minimum'": 19, "'selling'": 20, "'prices'": 2
1, "'butter'": 22, "'150th'": 23, "'individual'": 24, "'under'": 25, "'standing'": 26, "'provided'": 27, "'in'": 28, "'2571": 29, "97'": 30, "'communit
ies": 31, "'having'": 32, "'regard'": 33, "'treaty'": 34, "'establishing'": 35, "'community": 36}
Sequence 1 in x
  Input:  ['commission', 'regulation', '(ec)', 'no', '1788/2004']
  Output: [5, 6, 2, 7, 2, 8, 13, 14]
Sequence 2 in x
  Input:  ['of', '15', 'october', '2004']
  Output: [9, 15, 16, 17]
Sequence 3 in x
  Input:  ['fixing', 'the', 'minimum', 'selling', 'prices', 'for', 'butter', 'for', 'the', '150th', 'individual', 'invitation', 'to', 'tender', 'unde
r', 'the', 'standing', 'invitation', 'to', 'tender', 'provided', 'for', 'in', 'regulation', '(ec)', 'no', '2571/97']
  Output: [18, 1, 19, 20, 21, 3, 22, 3, 1, 23, 24, 10, 4, 11, 25, 1, 26, 10, 4, 11, 27, 3, 28, 6, 2, 7, 2, 8, 29, 30]
Sequence 4 in x
  Input:  ['the', 'commission', 'of', 'the', 'european', 'communities,']
  Output: [1, 5, 9, 1, 12, 31, 2]
Sequence 5 in x
  Input:  ['having', 'regard', 'to', 'the', 'treaty', 'establishing', 'the', 'european', 'community,']
  Output: [32, 33, 4, 1, 34, 35, 1, 12, 36, 2]
```

Then the padding function was created to make the sentence lengths equal, as they varied in length:

```
# Pad function to standardize the length of the sentences
def pad(x, length=None):
    return pad_sequences(x, maxlen=length, padding='post')

#Test
test_pad = pad(text_tokenized)
for sample_i, (token_sent, pad_sent) in enumerate(zip(text_tokenized, test_pad)):
    print('Sequence {} in x'.format(sample_i + 1))
    print('  Input:  {}'.format(np.array(token_sent)))
    print('  Output: {}'.format(pad_sent))
```

```
Sequence 1 in x
  Input:  [ 5  6  2  7  2  8 13 14]
  Output: [ 5  6  2  7  2  8 13 14  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0]
Sequence 2 in x
  Input:  [ 9 15 16 17]
  Output: [ 9 15 16 17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0]
Sequence 3 in x
  Input:  [18  1 19 20 21  3 22  3  1 23 24 10  4 11 25  1 26 10  4 11 27  3 28  6
  2  7  2  8 29 30]
  Output: [18  1 19 20 21  3 22  3  1 23 24 10  4 11 25  1 26 10  4 11 27  3 28  6
  2  7  2  8 29 30]
Sequence 4 in x
  Input:  [ 1  5  9  1 12 31  2]
  Output: [ 1  5  9  1 12 31  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0]
Sequence 5 in x
  Input:  [32 33  4  1 34 35  1 12 36  2]
  Output: [32 33  4  1 34 35  1 12 36  2  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0]
```

Once this function was completed, the data was split into testing and training sets, and then another function was applied that applied the tokenizer and padding function. Finally, before the model, a final function was created that would convert the data back to text once the model had been applied.

Logits to text function:

```
# Function to give the German translation
def logits_to_text(logits, tokenizer):
    """
    Turn logits from a neural network into text using the tokenizer
    :param logits: Logits from a neural network
    :param tokenizer: Keras Tokenizer fit on the labels
    :return: String that represents the text of the logits
    """
    index_to_words = {id: word for word, id in tokenizer.word_index.items()}
    index_to_words[0] = ''

    return ' '.join([index_to_words[prediction] for prediction in np.argmax(logits, 1)])
```

Modeling:

Once the preprocessing was complete, the model was created. I utilized a simple Neural Network with the following parameters:

- Model: Sequential - appropriate as we input one sentence and output a single translation.
- Learning rate: 0.005
- Batch size: 512
- Epochs: 10

```
# Model builder
def simple_model(input_shape, output_sequence_length, english_vocab_size, german_vocab_size):
    learning_rate = 0.005
    model = Sequential()
    model.add(GRU(256, input_shape=input_shape[1:], return_sequences=True))
    model.add(TimeDistributed(Dense(1024, activation='relu')))
    model.add(Dropout(0.5))
    model.add(TimeDistributed(Dense(german_vocab_size, activation='softmax')))

    # Compile model
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])
    return model

# Reshaping the input to work with a basic RNN
tmp_x = pad(pre_EN, max_DE_length)
tmp_x = tmp_x.reshape((-1, pre_DE.shape[-2], 1))

# Train the neural network
simple_rnn_model = simple_model(
    tmp_x.shape,
    max_DE_length,
    EN_vocab_size,
    DE_vocab_size)

print(simple_rnn_model.summary())

simple_rnn_model.fit(tmp_x, pre_DE, batch_size=512, epochs=10, validation_split=0.2)

# Print prediction(s)
print(logits_to_text(simple_rnn_model.predict(tmp_x[:1])[0], DE_token))
```

The final project yielded a validation accuracy of approximately 91%, which were ideal results for the project.

Key Takeaways:

After completing this project, the main takeaways from it were:

- Although the accuracy of the model was ideal, the results could have been optimized if the sample sizes were closer to the original size of the datasets. Even after creating a subsample of the data, it had to be cut down further for the model to run efficiently without crashing.
- Possible next steps for this project could to create a mode that is bidirection, which could translate the sentences back to English.