final

December 10, 2023

[1]:
```
print("""
Riker Wachtler
11 December 2023
DL Final Project
https://github.com/RikerW/final-dl
""")
```

```
Riker Wachtler
11 December 2023
DL Final Project
https://github.com/RikerW/final-dl
```

[38]:
```
print("""
This is the completed version of my final project. I left short comments for␣
↪each thing I'm doing - but in short,
it's doing some pre-processing stuff as EDA and then getting into it.

This implements a mediocre language translation model (english to spanish)␣
↪using LTSMs, plus an encoder and decoder.

I'll be honest - I wrote most of this, then looked back and said oh whoops, I␣
↪need an AUTO-encoder for criteria 1, not just an encoder.
However, I think should more than count for "Deep Learning Models" - the␣
↪encoders are definitely a form of ANN?
They're just feed forward networks, unless I'm mistaken.

The callbacks I added are just logs, but hopefully that suffices. Anyway,␣
↪that's about it. This is a deceptively short project.
I reverse engineered a substantial portion from an example from Keras for␣
↪english->french, which is where I got the data from,
but I wrote all of this code myself.
""")
```

```
This is the completed version of my final project. I left short comments for
each thing I'm doing - but in short,
```

it's doing some pre-processing stuff as EDA and then getting into it.

This implements a mediocre language translation model (english to spanish) using LTSMs, plus an encoder and decoder.

I'll be honest - I wrote most of this, then looked back and said oh whoops, I need an AUTO-encoder for criteria 1, not just an encoder.
However, I think should more than count for "Deep Learning Models" - the encoders are definitely a form of ANN?
They're just feed forward networks, unless I'm mistaken.

[3]:
```
import pandas as pd, numpy as np, tensorflow as tf, seaborn as sns

from collections import Counter

from sklearn.model_selection import train_test_split
from keras.layers import LSTM, Input, Dense
from keras.callbacks import ModelCheckpoint
from keras.models import Model
from sklearn.preprocessing import OneHotEncoder

import warnings
warnings.filterwarnings("ignore")
```

```
C:\Users\riker\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A
NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy
(detected version 1.25.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[4]:
```
df = pd.read_csv("data/spa.txt", header=None, delimiter=r"\t+")
df = df.rename(columns={0: 'english', 1: 'spanish', 2: 'attribution'})
df.dropna(0)

# thanks for attributing it! but we don't need this information for our purposes
del df["attribution"]
```

[5]:
```
df.head(10)
```

[5]:

	english	spanish
0	Go.	Ve.
1	Go.	Vete.
2	Go.	Vaya.
3	Go.	Váyase.
4	Hi.	Hola.
5	Run!	¡Corre!
6	Run!	¡Corran!

```
7    Run!     ¡Huye!
8    Run!    ¡Corra!
9    Run!   ¡Corred!
```

[6]:
```
df.describe()
```

[6]:

	english	spanish
count	141370	141370
unique	119661	132831
top	You can put it there.	Estoy quebrado.
freq	68	13

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141370 entries, 0 to 141369
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   english  141370 non-null  object
 1   spanish  141370 non-null  object
dtypes: object(2)
memory usage: 2.2+ MB
```

[8]:
```
# this is at the top so I can find it again easily. this numbers are shorter␣
 ↪than they should be all things considered,
# because I don't want to spend years re-training models. num_samples should be␣
 ↪10k and epochs 100 for real use, probably.
batch_size = 64
epochs = 50
latent_dim = 256
num_samples = 1000
```

[9]:
```
# set up token counts n sequence lengths for encoder (eng), decoder (spa)
# all things considered this really just preps everything to OHE some stuff,␣
 ↪and gives us useful sizes for later

eng = df.filter(["english"], axis=1)
eng = eng[0:3000]

spa = df.filter(["spanish"], axis=1)
spa = spa[0:3000]

eng_chars = sorted(list(Counter(''.join(eng.unstack().values)).keys()))
spa_chars = sorted(list(Counter(''.join(spa.unstack().values)).keys()))

num_eng_chars = len(eng_chars)
num_spa_chars = len(spa_chars)
```

```
max_eng_len = eng.english.map(len).max()+1
max_spa_len = spa.spanish.map(len).max()+1

eng_index = dict([(char, i) for i, char in enumerate(eng_chars)])
spa_index = dict([(char, i) for i, char in enumerate(spa_chars)])

print(max_eng_len, eng_chars)
print(max_spa_len, spa_chars)
```

```
13 [' ', '!', '$', '"', ',', '.', '0', '1', '3', '5', '7', '8', '9', ':', '?',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'Y', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
'y', 'z']
39 [' ', '!', '"', ',', '.', '0', '1', '3', '5', '7', '8', ':', '?', 'A', 'B',
'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
'T', 'U', 'V', 'Y', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'x', 'y', 'z', '¡', '¿', 'Á',
'É', 'Ó', 'Ú', 'á', 'é', 'í', 'ñ', 'ó', 'ú', 'ü']
```

[10]:
```
# oh yeah, we truncated this to 3000 instead of ~14000 because I don't like␣
↪generating obscenely large one-hot vectors (jupyter said 18.7 GB?)
total_length = len(eng)
assert total_length == len(spa)
print(total_length)
```

3000

[11]:
```
# Set up encoder inputs - we're one-hot encoding them, with the code below :)
encoder_input  = np.zeros((total_length, max_eng_len, num_eng_chars),␣
↪dtype='float32')
decoder_input  = np.zeros((total_length, max_spa_len, num_spa_chars),␣
↪dtype='float32')
decoder_target = np.zeros((total_length, max_spa_len, num_spa_chars),␣
↪dtype='float32')
```

[12]:
```
# OHE the suckers - stagger the decoder target by one character, since we're␣
↪working on predicting the next one lol

for i in range(total_length):
    cur_eng = eng.english[i]
    cur_spa = spa.spanish[i]

    for j in range(len(cur_eng)):
        encoder_input[i, j, eng_index[cur_eng[j]]] = 1
    encoder_input[i, j+1, eng_index[' ']] = 1
```

```
    for j in range(len(cur_spa)):
        decoder_input[i, j, spa_index[cur_spa[j]]] = 1
        if j > 0:
            decoder_target[i, j-1, spa_index[cur_spa[j]]] = 1
    decoder_input[i, j+1, spa_index[' ']] = 1
    decoder_target[i, j, spa_index[' ']] = 1
    decoder_target[i, j+1, spa_index[' ']] = 1
```

[42]:
```
# Create encoder model - input, LTSM, into a model from the states. not very␣
 ↪complicated
eng_inp = Input(shape=(None, num_eng_chars))
eng_enc = LSTM(latent_dim, return_sequences=True, return_state=True)
eng_out, estate_h, estate_c = eng_enc(eng_inp)

eng_states = [estate_h, estate_c]

eng_model = Model(eng_inp, eng_states)
```

[14]:
```
eng_states, eng_model
```

[14]:
```
([<KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'lstm')>,
  <KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'lstm')>],
 <keras.src.engine.functional.Functional at 0x152db8429d0>)
```

[15]:
```
# Create a decoder model - more complicated. you need to an LTSM with states␣
 ↪like normal,
# but we also need to make it fancier since this one needs to actually set it␣
 ↪up to go once to set up our starting state
spa_inp = Input(shape=(None, num_spa_chars))
spa_dec = LSTM(latent_dim, return_sequences=True, return_state=True)
spa_out, sstate_h, sstate_c = spa_dec(spa_inp, initial_state=eng_states)

spa_h_inp = Input(shape=(latent_dim,))
spa_c_inp = Input(shape=(latent_dim,))
spa_inps = [spa_h_inp, spa_c_inp]

spa_out, sstate_h, sstate_c = spa_dec(spa_inp, initial_state=spa_inps)
spa_states = [sstate_h, sstate_c]

spa_dense = Dense(num_spa_chars, activation='softmax')
spa_out = spa_dense(spa_out)
spa_model = Model([spa_inp] + spa_inps, [spa_out] + spa_states)
```

[16]:
```
spa_states, spa_model
```

[16]:
```
([<KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'lstm_1')>,
  <KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'lstm_1')>],
```

```
<keras.src.engine.functional.Functional at 0x152dc358130>)
```

[54]:
```
# Here's a demo that shows i know how to use logs

nspa_inp = Input(shape=(None, num_spa_chars))
neng_inp = Input(shape=(None, num_eng_chars))
neng_enc = LSTM(latent_dim, return_sequences=True, return_state=True)
neng_out, nestate_h, nestate_c = neng_enc(neng_inp)

neng_states = [nestate_h, nestate_c]

nspa_dec = LSTM(latent_dim, return_sequences=True, return_state=True)
nspa_out, _, _ = nspa_dec(nspa_inp, initial_state=neng_states)
nspa_out = Dense(num_spa_chars, activation='softmax')(nspa_out)

nmodel = Model([neng_inp, nspa_inp], nspa_out)

# Generic callbacks to use
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{accuracy:.
 ↪2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]

# Run training
nmodel.compile(optimizer='rmsprop', loss='categorical_crossentropy', ␣
 ↪metrics=['accuracy'])
nmodel.fit([encoder_input, decoder_input], decoder_target,␣
 ↪batch_size=batch_size, epochs=epochs, validation_split=0.2,␣
 ↪callbacks=my_callbacks)
```

```
Epoch 1/50
38/38 [==============================] - 8s 132ms/step - loss: 1.2092 -
accuracy: 0.7157 - val_loss: 1.1955 - val_accuracy: 0.7089
Epoch 2/50
38/38 [==============================] - 4s 115ms/step - loss: 1.0698 -
accuracy: 0.7398 - val_loss: 1.1877 - val_accuracy: 0.7088
Epoch 3/50
38/38 [==============================] - 4s 116ms/step - loss: 1.0601 -
accuracy: 0.7405 - val_loss: 1.1912 - val_accuracy: 0.7088
Epoch 4/50
38/38 [==============================] - 4s 112ms/step - loss: 1.0593 -
accuracy: 0.7402 - val_loss: 1.1937 - val_accuracy: 0.7088
```

[54]:
```
<keras.src.callbacks.History at 0x15304e73f10>
```

[20]:

```
# actually try to predict something :)

rev_spa_index = {v: k for k, v in spa_index.items()}

def decode_sequence(input_seq):
    states = eng_model.predict(input_seq)

    target = np.zeros((1, 1, num_spa_chars))
    target[0, 0, spa_index['\t']] = 1.

    out = ''
    while True:
        # predict some tokens baby
        out_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # actually add a token to our output :)
        out_token = np.argmax(out_tokens[0, -1, :])
        out_char = rev_spa_index[out_token]
        out += out_cha

        # if our decoder thinks we're done, or we hit the len cap, end it
        if (out_cha == '\n' or len(out) > max_spa_len):
            break;

        # otherwise, keep us going and update our target + state for the next␣
 ↪loop
        target = np.zeros((1, 1, num_decoder_tokens))
        target[0, 0, sampled_token_index] = 1.
        states = [h, c]

    return out
```

[37]:

```
import random

pid = random.randint(0, 100)
input_seq = encoder_input[pid]
decoded_sentence = decode_sequence(input_seq)
print('-')
print('Input sentence:', eng.english[pid])
print('Decoded sentence:', decoded_sentence)
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
-
Input sentence: Exhale.
Decoded sentence: Espírad..
```

[55]:
```
print("""

Overall takeaways - it's actually remarkably involved to do what I thought␣
↪wouldn't take very long at first.
Not super hard, but certainly complex. We worked with similar LTSM stuff - but␣
↪pre-loaded models. Combining them with writing my own encoders and decoders␣
↪is VERY finicky as well.
```

```
I'm pretty sure despite re-running this repeatedly there's at least one odd␣
↪variable name that breaks things because of how many times I rewrote this.

I definitely have more of an appreciation for machine translation. Also, mine␣
↪sucked a little - I re-ran it a couple times and this is more or less what␣
↪it outputs 70% of the time, I think i'm off by one for my OHE unfortunately.
Nonetheless, definitely an interesting project.

-- Riker
""")
```

Overall takeaways - it's actually remarkably involved to do what I thought wouldn't take very long at first.
Not super hard, but certainly complex. We worked with similar LTSM stuff - but pre-loaded models. Combining them with writing my own encoders and decoders is VERY finicky as well.
I'm pretty sure despite re-running this repeatedly there's at least one odd variable name that breaks things because of how many times I rewrote this.

I definitely have more of an appreciation for machine translation. Also, mine sucked a little - I re-ran it a couple times and this is more or less what it outputs 70% of the time, I think i'm off by one for my OHE unfortunately.
Nonetheless, definitely an interesting project.

-- Riker

[ ]:

```

```