Irfan Ahmed
ICP 6
Due Date: 10/5/2020

The four hyper-parameters changed are(also few more for testing purposes):

- Epochs, Buffer size, Optimizer, Embedding dim, sequence length, batch size, temperature:
    - The number of **Epochs** is correlated with number of optimizations that are applied during training. Since having more rounds the training error will be reduced. But there is a high chance of overfitting the training data and it will start to lose performance. So, based on what I was testing the higher number of Epochs gave me the best result.
    - The **optimizer** I decided to use was 'Adamax' which is a variant of 'Adam. It is based on the infinity norm. In most cases it is found to be better and I wanted to test that out with my model. Based on my testing 'Adamax' did perform better.
    - Few things would need be considered when changing the **embedding dim** matrix. Since, not every word would get replaced by a vector, and instead it is replaced by index. This index can be used to look up the embedding matrix. Also, the embedded vectors are updated frequently during the training process it would make since for this case to increase the embedding dim to large number if the dataset is large. Which was the case when I decided to use Hamlet by Shakespeare.
    - **Batch size** can be one of the most important hyperparameters that will need to be tuned. One of the benefits of going with a smaller batch size is that it will not converge to the global optima. Also, a smaller batch size is known to have a faster convergence to good solutions. This means that the smaller batch sizes allow the model to start learning before looking at all the data. I decided to go with slightly higher batch size but not too high. I wanted to have all the variable looked at and not have errors that would be hard to account for.
    - The **sequence length** was changed as well to see if it would affect the output and based on my trial and error, I did not notice any change.
    - The **temperature** was also changed to higher number to see if the text generation would improve but it made it worse. Higher the number the worse the output.

a. What you learned in the ICP

One of the main takes away from this ICP 6 was understanding how the hyperparameters affect the data output. Changing the batch size can impact it negatively or positively. Based on my observation the higher the batch size performed better. This could be due to the large dataset that was being used. If I had used a smaller dataset the higher batch size would affect in negatively. Another, important take away from this ICP 6 was changing the optimizer and how it affects the training sample. Variant of 'Adam' was used ('Adamax'), which did perform better in my opinion. But hard to discern the two. The Epochs also played important role during the training and by going with the greater number it yielded better results. Also, few other like sequence lengths and embedding dim were hard to distinguish how they were affecting the data output. But changing the temperature did show improvement when using a low value number instead of high.

b. ICP description what was the task you were performing

1. Imported the required libraries/packages.
2. Used nltk to download ‘gutenberg which contained various data that could be used for text generation.
3. Used the raw data (nltk.corpus.gutenberg.raw(‘shakespear-hamlet.txt’)
4. Printed out the few character to see what the data looks like.
5. Created unique set of character in the file and sorted. Total of 67 unquiet character found in the data.
6. Mapped the character using enumerate by passing in the vocabular data set created(described in step 5).
7. Sequence of length initialized to 200. Then the text is divided into example sequence. Example_per_epoch = len(hamlet_text)//(sequence_length + 10).
8. Next the bath method applied to convert individual character to the desired sequence.
9. Then the sequence is duplicated and shifted to form the input and target text. For this the map method is used.
10. Looking at the input data and target values. Using print()
11. Created batch size, buffer size and the used dataset.shuffle on the buffer size .
12. The model defined and created. Also, the summary was looked at.
13. The output distribution was sampled and printed out the array list .
14. The model was trained and compiled. Also, the check points configured
15. The data was executed using 700 epochs.
16. Prediction loop was defined to generate the text.
17. Output the text.

c. Challenges that you faced

With practice I think I will get better at this. But had few issues trying to add more layers to the model. Also, trying to figure out how the hyperparameters affect the overall model performance. Feedback would be appreciated on what books to have when learning the neural network algorithms.

d. Screen shots that shows the successful execution of each required step of your code

ICP6_Irfan_Ahmed.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Use a different data and use the model provided in ICP6 to perform Text generation layers to model, changing hyperparameters etc) in the source code. Report your fin changes you made in the source code and why in your opinion these changes are lo

The four hyper-parameters changed are(also few more for testing purposes):

- Epochs, Buffer size, Optimizer, Embedding dim, sequence length, batch size, te

o The number of Epochs is correlated with number of optimizations that are applied will be reduced. But there is a high chance of overfitting the training data and it will the higher number of Epochs gave me the best result.

o The optimizer I decided to use was 'Adamax' which is a variant of 'Adam. It is base and I wanted to test that out with my model. Based on my testing 'Adamax' did perf

o Few things would need be considered when changing the embedding dim matrix. instead it is replaced by index. This index can be used to look up the embedding ma during the training process it would make since for this case to increase the embed the case when I decided to use Hamlet by Shakespeare.

o Batch size can be one of the most important hyperparameters that will need to be size is that it will not converge to the global optima. Also, a smaller batch size is kn means that the smaller batch sizes allow the model to start learning before looking but not too high. I wanted to have all the variable looked at and not have errors that

o The sequence length was changed as well to see if it would affect the output and

o The temperature was also changed to higher number to see if the text generation worse the output.

Sequence of length Epochs

Importing the required libraries and packages

```
[1]
import tensorflow as tf

import numpy as np
import os
import time
```

Downloading the data that will be used for this algorithm. using gutenberg to get a

```
[2] #importing nltk and using it to see which books are available.
import nltk
nltk.download('gutenberg')
from nltk.corpus import gutenberg as gut
print(gut.fileids(), end='')
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-
```

```
[3] #Extracting the hamlet by shakespeare
hamlet_text = nltk.corpus.gutenberg.raw('shakespeare-hamlet.txt')
```

Looking at the text that was uploaded.

```
print('Length of text: {} characters'.format(len(hamlet_text)))
```

```
Length of text: 162881 characters
```

Looking at the first few characters

```
[5] print(hamlet_text[:500])
```

```
[The Tragedie of Hamlet by William Shakespeare 1599]


Actus Primus. Scoena Prima.

Enter Barnardo and Francisco two Centinels.

  Barnardo. Who's there?
  Fran. Nay answer me: Stand & vnfold
your selfe

   Bar. Long liue the King

   Fran. Barnardo?
  Bar. He

   Fran. You come most carefully vpon your houre

   Bar. 'Tis now strook twelue, get thee to bed Francisco

   Fran. For this releefe much thankes: 'Tis bitter cold,
And I am sicke at heart

   Barn. Haue you had quiet Guard?
```

Creating a unique set of characters in the file. The set will be sorted and the unique charact

```
vocab = sorted(set(hamlet_text))
print('{} unique character'.format(len(vocab)))
```

```
67 unique character
```

Mapping the character

```
[7] char2idx = {u:i for i, u in enumerate(vocab)}
    idx2char = np.array(vocab)

    text_input = np.array([char2idx[c] for c in hamlet_text])
    print(char2idx)
    print(idx2char)
```

```
{'\n': 0, ' ': 1, '!': 2, '&': 3, "'": 4, '(': 5, ')': 6, ',': 7, '-': 8, '.':
['\n' ' ' '!' '&' "'" '(' ')' ',' '-' '.' '1' '5' '9' ':' ';' '?' 'A' 'B'
 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'V'
 'W' 'Y' 'Z' '[' ']' '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']
```

Dividing the text into example sequences

Sequence length doesn't really affect your model training but it's like having more training resetting it.

```
[8]  sequence_length = 200 #changed the sequence length to 200. Initialy it was a
     examples_per_epoch = len(hamlet_text) // (sequence_length + 10) #changed to
     print(examples_per_epoch)

     char_dataset = tf.data.Dataset.from_tensor_slices(text_input)

     for i in char_dataset.take(5):
       print(idx2char[i.numpy()])
```

```
775
[
T
h
e
```

Converting the characters to desired size using batch method

```
[9]  sequence = char_dataset.batch(sequence_length + 1, drop_remainder=True)
     for j in sequence.take(5):
       print(repr(''.join(idx2char[j.numpy()])))
```

```
"[The Tragedie of Hamlet by William Shakespeare 1599]\n\n\nActus Primus. Sco
"e\n\n   Bar. Long liue the King\n\n   Fran. Barnardo?\n  Bar. He\n\n   Fran
"thankes: 'Tis bitter cold,\nAnd I am sicke at heart\n\n   Barn. Haue you ha
"ch, bid them make hast.\nEnter Horatio and Marcellus.\n\n  Fran. I thinke I
```

Each of the sequence is duplciated and then shifted from the input and the target text via

```
def split_input_target(chunk):
  input_text = chunk[:-1]
  target_text = chunk[1:]
  return input_text, target_text

dataset = sequence.map(split_input_target)
```

Printing the input and target values.

```
[11] for input_example, target_example in dataset.take(1):
       print("Input data: ", repr(''.join(idx2char[input_example.numpy()])), '\n')
       print("Target data: ", repr(''.join(idx2char[target_example.numpy()])), '\n
```

```
Input data:  "[The Tragedie of Hamlet by William Shakespeare 1599]\n\n\nActus

Target data:  "The Tragedie of Hamlet by William Shakespeare 1599]\n\n\nActus
```

Now, creating the training batches

```
[12] batch_size = 96 #Batch size changed from 64 to 96
     buffer_size = 500 # this is to shuffle the dataset and to prevent from shuffl
     dataset = dataset.shuffle(buffer_size).batch(batch_size, drop_remainder = Tru
     dataset
```

```
<BatchDataset shapes: ((96, 200), (96, 200)), types: (tf.int64, tf.int64)>
```

Building the model

```
vocab_size = len(vocab)
print(vocab_size)

embedding_dim = 512

rnn_units = 1024
```

```
67
```

```
[14] def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
       model = tf.keras.Sequential([
         tf.keras.layers.Embedding(vocab_size, embedding_dim,batch_input_shape=[b
         tf.keras.layers.GRU(rnn_units,return_sequences=True,stateful=True,recurr
         tf.keras.layers.Dense(vocab_size)
       ])
       return model
```

```
[15] model = build_model(
         vocab_size = len(vocab),
         embedding_dim=embedding_dim,
         rnn_units=rnn_units,
         batch_size=batch_size)
```

```
[16] for input_example_batch, target_example_batch in dataset.take(1):
       example_batch_predictions = model(input_example_batch)
       print(example_batch_predictions.shape, "# (batch_size, sequence_length, vo
```

```
(96, 200, 67) # (batch_size, sequence_length, vocab_size)
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (96, None, 512)           34304
_________________________________________________________________
gru (GRU)                    (96, None, 1024)          4724736
_________________________________________________________________
dense (Dense)                (96, None, 67)            68675
=================================================================
Total params: 4,827,715
Trainable params: 4,827,715
Non-trainable params: 0
_________________________________________________________________
```

Sampling from the output distribution to obtain the character indices.

```
[18] sampled_indices = tf.random.categorical(example_batch_predictions[0], num_sam
     sampled_indices = tf.squeeze(sampled_indices,axis=-1).numpy()
     #This gives us, at each timestep, a prediction of the next character index:
     sampled_indices
```

```
array([38, 25, 20, 30, 65, 44, 53, 20, 48, 27,  3, 43, 29, 30, 30,  7,  0,
       12,  9, 40, 29, 38,  9, 65, 29, 46, 59,  2, 20, 50, 47, 18, 26, 12,
       61, 64, 10,  9, 13,  4, 66, 27, 37, 12, 54, 60, 51,  7, 29, 38,  7,
       59, 31, 33, 49, 52, 57, 14, 17, 25, 60, 18,  1, 56, 55, 64, 35, 41,
       58, 25, 48, 18, 58, 36, 31, 26, 35, 24, 58, 55, 58, 15, 28, 38, 31,
       10,  9, 47,  2, 55, 19,  2,  1, 14, 42, 11, 23, 24, 10, 60, 57, 11,
       15, 65, 61, 53,  4, 54, 59, 12, 36, 21, 52, 42, 49, 51, 10, 28, 34,
        8,  6, 46, 52, 56, 31, 26, 66, 36, 23, 36, 24, 14,  2, 60, 55, 12,
       42, 16, 32, 42, 59, 27, 57,  7, 55, 28, 18, 58, 56, 59, 43, 51, 51,
```

Training the model

```
def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, fro

example_batch_loss  = loss(target_example_batch, example_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size
print("scalar_loss:      ", example_batch_loss.numpy().mean())
```

```
Prediction shape:  (96, 200, 67)  # (batch_size, sequence_length, vocab_size)
scalar_loss:       4.2048426
```

using 'adam' optimizer to compile the model

```
[20] model.compile(optimizer='Adamax', loss=loss)
```

configuring the checkpoints

```
[21] checkpoint_dir = './training_checkpoints'
     checkpoint_prefix = os.path.join(checkpoint_dir,"ckpt_{epoch}")

     checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
         filepath = checkpoint_prefix,
         save_weights_only = True
     )
```

configuring the checkpoints

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir,"ckpt_{epoch}")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath = checkpoint_prefix,
    save_weights_only = True
)
```

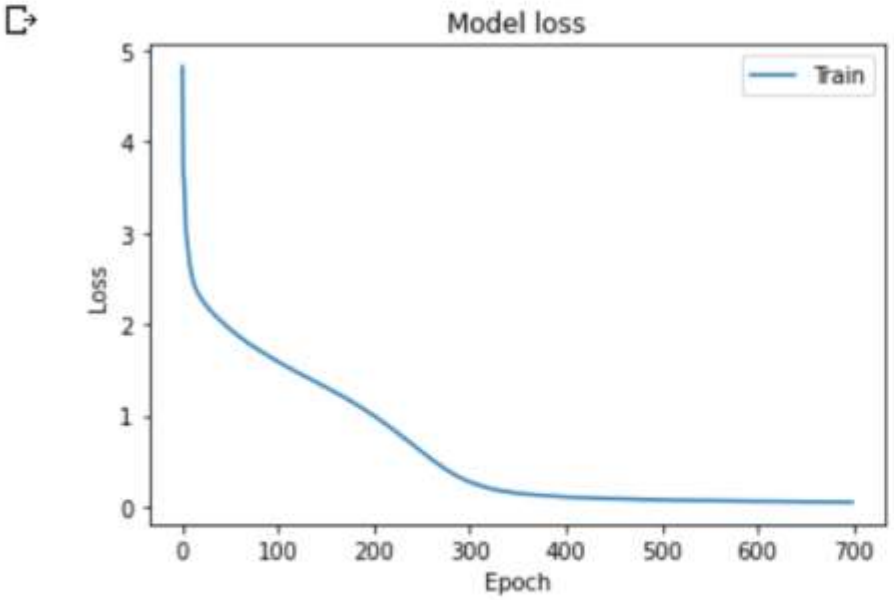
Executing the training data

```
[22] Epochs = 700 # Epoch is another hyper-paramter that is changed.
     history = model.fit(dataset, epochs=Epochs, callbacks=[checkpoint_callback])
```

```
Epoch 1/700
8/8 [==============================] - 1s 185ms/step - loss: 4.8233
Epoch 2/700
8/8 [==============================] - 1s 181ms/step - loss: 3.6793
Epoch 3/700
8/8 [==============================] - 1s 179ms/step - loss: 3.5125
Epoch 4/700
8/8 [==============================] - 1s 181ms/step - loss: 3.2373
Epoch 5/700
8/8 [==============================] - 1s 182ms/step - loss: 2.9867
Epoch 6/700
8/8 [==============================] - 1s 181ms/step - loss: 2.8965
Epoch 7/700
8/8 [==============================] - 1s 186ms/step - loss: 2.7895
Epoch 8/700
8/8 [==============================] - 2s 196ms/step - loss: 2.7092
Epoch 9/700
8/8 [==============================] - 2s 189ms/step - loss: 2.6344
```

```
[23] import matplotlib.pyplot as plt
     plt.plot(history.history['loss'])

     plt.title('Model loss')
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.legend(['Train', 'Val'], loc='upper right')
     plt.show()
```



generateing the text

```
[24] tf.train.latest_checkpoint(checkpoint_dir)
     model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)

     model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))

     model.build(tf.TensorShape([1, None]))
     model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (1, None, 512)            34304
_________________________________________________________________
gru_1 (GRU)                  (1, None, 1024)           4724736
_________________________________________________________________
dense_1 (Dense)              (1, None, 67)             68675
=================================================================
Total params: 4,827,715
Trainable params: 4,827,715
Non-trainable params: 0
_________________________________________________________________
```

Prediction loop will generate the text

```
def generate_text(model, start_string):
  # Evaluation step (generating text using the learned model)

  # Number of characters to generate
  num_generate = 1000

  # Converting our start string to numbers (vectorizing)
  input_eval = [char2idx[s] for s in start_string]
  input_eval = tf.expand_dims(input_eval, 0)

  # Empty string to store our results
  text_generated = []

  # Low temperatures results in more predictable text.
  # Higher temperatures results in more surprising text.
  # Experiment to find the best setting.
  temperature = 1.0

  # Here batch size == 1
  model.reset_states()
  for i in range(num_generate):
    predictions = model(input_eval)
    # remove the batch dimension
    predictions = tf.squeeze(predictions, 0)

    # using a categorical distribution to predict the character returned by the model
    predictions = predictions / temperature
    predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()

    # We pass the predicted character as the next input to the model
    # along with the previous hidden state
    input_eval = tf.expand_dims([predicted_id], 0)

    text_generated.append(idx2char[predicted_id])

  return (start_string + ''.join(text_generated))
```

```
[26] print(generate_text(model, start_string=u"Hamlet:"))
```

```
Hamlet: Crie
New lighted on a heauen-kissing hill:
A Combination, and a forme indeed,
Where euery God did seeme to set his Seale,
To giue the world assurance of a man.
This was your Husband. Looke you now what forgiue me my Father

   Qu. Calmely good Laertes

   Laer. That drop of blood, that calmes
Proclaimes me Bastard:
Cries Cuckold to my Father, brands the Harlot
Euen heere betweene the chaste vnsmirched by himselfe:
Exchange forgiuenes to the King: They are they keepe:
What company, at what expence: and finding
By this encompassement and drift of question,
That they doe know my sonne: Come you more neerer
Then your particular demands will to Hecuba,
That he should weepe for her? What would he doe,
Had he the Motiue and the Cue for passion
That I haue? He would drowne the Stope of pracis

   Bap, vndertake it Players

   Other. What second in eares.

  King. Now must your conscience my acquittance seal,
And you must put me in your heart for Friend,
Sith you haue heard, and with a knowing
```

e. Output file link if applicable

https://github.com/UMKC-APL-BigDataAnalytics/icp6-irfancheemaa

f. Video link (YouTube or any other publicly available video platform)

https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=269042d7-f5f0-4b83-8314-ac4b0189875e

g. Any inside about the data or the ICP in general

Overall, very interesting topic to learn about. I would like to have more background in the code itself. Try to fully understand how it all interacts. But enjoy it so far.