Lab 8 Neural Language Model

A language model predicts the next word in the sequence based on the specific words that have come before it in the sequence.

It is also possible to develop language models at the character level using neural networks. The benefit of character-based language models is their small vocabulary and flexibility in handling any words, punctuation, and other document structure. This comes at the cost of requiring larger models that are slower to train.

Nevertheless, in the field of neural language models, character-based models offer a lot of promise for a general, flexible and powerful approach to language modeling.

As a prerequisite for the lab, make sure to pip install:

- keras
- tensorflow
- h5py

Source Text Creation

To start out with, we'll be using a simple nursery rhyme. It's quite short so we can actually train something on your CPU and see relatively interesting results. Please copy and paste the following text in a text file and save it as "rhymes.txt". Place this in the same directory as this jupyter notebook:

```
!pip install tensorflow
!pip install keras
!pip install h5py
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes==0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.2)
Requirement already satisfied: tensorboard<2.15,>=2.14 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: keras<2.15,>=2.14.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.4
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensor
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorfl
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorb
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorb
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.15
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15,>=
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.15
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (3.9.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from h5py) (1.23.5)
```

```
s='Sing a song of sixpence,₩
A pocket full of rye.₩
```

```
Four and twenty blackbirds,₩
Baked in a pie.₩
When the pie was opened₩
The birds began to sing;₩
Wasn't that a dainty dish,₩
To set before the king.₩
The king was in his counting house,₩
Counting out his money;₩
The queen was in the parlour,₩
Eating bread and honey.₩
The maid was in the garden,₩
Hanging out the clothes,₩
When down came a blackbird₩
And pecked off her nose.'

with open('rhymes.txt','w') as f:
  f.write(s)
```

```
Sing a song of sixpence,
A pocket full of rye.
Four and twenty blackbirds,
Baked in a pie.

When the pie was opened
The birds began to sing;
Wasn't that a dainty dish,
To set before the king.

The king was in his counting house,
Counting out his money;
The queen was in the parlour,
Eating bread and honey.

The maid was in the garden,
Hanging out the clothes,
When down came a blackbird
And pecked off her nose.
```

Sequence Generation

A language model must be trained on the text, and in the case of a character-based language model, the input and output sequences must be characters.

The number of characters used as input will also define the number of characters that will need to be provided to the model in order to elicit the first predicted character.

After the first character has been generated, it can be appended to the input sequence and used as input for the model to generate the next character.

Longer sequences offer more context for the model to learn what character to output next but take longer to train and impose more burden on seeding the model when generating text.

We will use an arbitrary length of 10 characters for this model.

There is not a lot of text, and 10 characters is a few words.

We can now transform the raw text into a form that our model can learn; specifically, input and output sequences of characters.

```
#load doc into memory
def load_doc(filename):
   # open the file as read only
   file = open(filename, 'r')
   # read all text
   text = file.read()
   # close the file
   file.close()
   return text
# save tokens to file, one dialog per line
def save_doc(lines, filename):
   data = '₩n'.join(lines)
   file = open(filename, 'w')
   file.write(data)
   file.close()
```

```
#load text
raw_text = load_doc('rhymes.txt')
print(raw_text)

# clean
tokens = raw_text.split()
raw_text = ' '.join(tokens)

# organize into sequences of characters
length = 10
sequences = list()
for i in range(length, len(raw_text)):
   # select sequence of tokens
   seq = raw_text[i-length:i+1]
   # store
   sequences.append(seq)
print('Total Sequences: %d' % len(sequences))
```

```
Sing a song of sixpence,A pocket full of rye.Four and twenty blackbirds,Baked in a pie.When the pie was openedThe birds began to si
Total Sequences: 384
```

```
# save sequences to file
out_filename = 'char_sequences.txt'
save_doc(sequences, out_filename)
```

Train a Model

In this section, we will develop a neural language model for the prepared sequence data.

The model will read encoded characters and predict the next character in the sequence. A Long Short-Term Memory recurrent neural network hidden layer will be used to learn the context from the input sequence in order to make the predictions.

```
from numpy import array
from pickle import dump
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# load doc into memory
def load_doc(filename):
   # open the file as read only
   file = open(filename, 'r')
   # read all text
   text = file.read()
   # close the file
   file.close()
   return text


# load

in_filename = 'char_sequences.txt'
raw_text = load_doc(in_filename)
lines = raw_text.split('₩n')
```

The sequences of characters must be encoded as integers.This means that each unique character will be assigned a specific integer value and each sequence of characters will be encoded as a sequence of integers. We can create the mapping given a sorted set of unique characters in the raw input data. The mapping is a dictionary of character values to integer values.

Next, we can process each sequence of characters one at a time and use the dictionary mapping to look up the integer value for each character. The result is a list of integer lists.

We need to know the size of the vocabulary later. We can retrieve this as the size of the dictionary mapping.

```
# integer encode sequences of characters
chars = sorted(list(set(raw_text)))
mapping = dict((c, i) for i, c in enumerate(chars))
sequences = list()
for line in lines:
   # integer encode line
   encoded_seq = [mapping[char] for char in line]
   # store
   sequences.append(encoded_seq)

# vocabulary size
```

```
vocab_size = len(mapping)
print('Vocabulary Size: %d' % vocab_size)

# separate into input and output
sequences = array(sequences)
X, y = sequences[:,:-1], sequences[:,-1]
sequences = [to_categorical(x, num_classes=vocab_size) for x in X]
X = array(sequences)
y = to_categorical(y, num_classes=vocab_size)
```

```
Vocabulary Size: 38
```

The model is defined with an input layer that takes sequences that have 10 time steps and 38 features for the one hot encoded input sequences. Rather than specify these numbers, we use the second and third dimensions on the X input data. This is so that if we change the length of the sequences or size of the vocabulary, we do not need to change the model definition.

The model has a single LSTM hidden layer with 75 memory cells. The model has a fully connected output layer that outputs one vector with a probability distribution across all characters in the vocabulary. A softmax activation function is used on the output layer to ensure the output has the properties of a probability distribution.

The model is learning a multi-class classification problem, therefore we use the categorical log loss intended for this type of problem. The efficient Adam implementation of gradient descent is used to optimize the model and accuracy is reported at the end of each batch update. The model is fit for 50 training epochs.

To Do:

- Try different numbers of memory cells
- Try different types and amounts of recurrent and fully connected layers
- Try different lengths of training epochs
- Try different sequence lengths and pre-processing of data
- Try regularization techniques such as Dropout

```
# define model
model = Sequential()
model.add(LSTM(75, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(vocab_size, activation='softmax'))
print(model.summary())
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
history=model.fit(X, y, epochs=100)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 75)                34200

 dense (Dense)               (None, 38)                2888

=================================================================
Total params: 37088 (144.88 KB)
Trainable params: 37088 (144.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/100
12/12 [==============================] - 3s 14ms/step - loss: 3.6142 - accuracy: 0.0547
Epoch 2/100
12/12 [==============================] - 0s 12ms/step - loss: 3.5113 - accuracy: 0.1042
Epoch 3/100
12/12 [==============================] - 0s 14ms/step - loss: 3.2211 - accuracy: 0.1302
Epoch 4/100
12/12 [==============================] - 0s 11ms/step - loss: 3.1188 - accuracy: 0.1589
Epoch 5/100
12/12 [==============================] - 0s 12ms/step - loss: 3.0560 - accuracy: 0.1589
Epoch 6/100
12/12 [==============================] - 0s 11ms/step - loss: 3.0422 - accuracy: 0.1589
Epoch 7/100
12/12 [==============================] - 0s 12ms/step - loss: 3.0255 - accuracy: 0.1589
Epoch 8/100
12/12 [==============================] - 0s 21ms/step - loss: 3.0066 - accuracy: 0.1589
Epoch 9/100
12/12 [==============================] - 0s 13ms/step - loss: 2.9948 - accuracy: 0.1589
Epoch 10/100
12/12 [==============================] - 0s 13ms/step - loss: 2.9750 - accuracy: 0.1797
Epoch 11/100
12/12 [==============================] - 0s 19ms/step - loss: 2.9633 - accuracy: 0.1615
Epoch 12/100
12/12 [==============================] - 0s 11ms/step - loss: 2.9355 - accuracy: 0.1797
```

```
Epoch 13/100
12/12 [==============================] - 0s 13ms/step - loss: 2.9049 - accuracy: 0.1979
Epoch 14/100
12/12 [==============================] - 0s 11ms/step - loss: 2.8841 - accuracy: 0.1927
Epoch 15/100
12/12 [==============================] - 0s 12ms/step - loss: 2.8456 - accuracy: 0.2057
Epoch 16/100
12/12 [==============================] - 0s 12ms/step - loss: 2.8125 - accuracy: 0.2031
Epoch 17/100
12/12 [==============================] - 0s 11ms/step - loss: 2.7806 - accuracy: 0.2318
Epoch 18/100
12/12 [==============================] - 0s 10ms/step - loss: 2.7440 - accuracy: 0.2214
Epoch 19/100
12/12 [==============================] - 0s 12ms/step - loss: 2.7064 - accuracy: 0.2396
Epoch 20/100
12/12 [==============================] - 0s 17ms/step - loss: 2.6773 - accuracy: 0.2214
Epoch 21/100
12/12 [==============================] - 0s 13ms/step - loss: 2.6573 - accuracy: 0.2292
Epoch 22/100
12/12 [==============================] - 0s 11ms/step - loss: 2.6011 - accuracy: 0.2552
```

```
# save the model to file
model.save('model.h5')
# save the mapping
dump(mapping, open('mapping.pkl', 'wb'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file v
  saving_api.save_model(
```

Generating Text

We must provide sequences of 10 characters as input to the model in order to start the generation process. We will pick these manually. A given input sequence will need to be prepared in the same way as preparing the training data for the model.

```
from pickle import load
import numpy as np
from keras.models import load_model
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

# generate a sequence of characters with a language model
def generate_seq(model, mapping, seq_length, seed_text, n_chars):
    in_text = seed_text
    # generate a fixed number of characters
    for _ in range(n_chars):
        # encode the characters as integers
        encoded = [mapping[char] for char in in_text]
        # truncate sequences to a fixed length
        encoded = pad_sequences([encoded], maxlen=seq_length, truncating='pre')
        # one hot encode
        encoded = to_categorical(encoded, num_classes=len(mapping))
        # predict character
        yhat = np.argmax(model.predict(encoded), axis=-1)
        # reverse map integer to character
        out_char = ''
        for char, index in mapping.items():
            if index == yhat:
                out_char = char
                break
        # append to input
        in_text += char
    return in_text

# load the model
model = load_model('model.h5')
# load the mapping
mapping = load(open('mapping.pkl', 'rb'))
```

Running the example generates three sequences of text.

The first is a test to see how the model does at starting from the beginning of the rhyme. The second is a test to see how well it does at beginning in the middle of a line. The final example is a test to see how well it does with a sequence of characters never seen before.

```
# test start of rhyme
print(generate_seq(model, mapping, 10, 'Sing a son', 20))
# test mid-line
print(generate_seq(model, mapping, 10, 'king was i', 20))
```

```
# test not in original
print(generate_seq(model, mapping, 10, 'hello worl', 20))
```

```
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 32ms/step
Sing a song of sixpence,A pock
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
king was in his counting house
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
hello worls.Four nttth ctblba
```

If the results aren't satisfactory, try out the suggestions above or these below:

- Padding. Update the example to provides sequences line by line only and use padding to fill out each sequence to the maximum line length.
- Sequence Length. Experiment with different sequence lengths and see how they impact the behavior of the model.
- Tune Model. Experiment with different model configurations, such as the number of memory cells and epochs, and try to develop a better model for fewer resources.

Deliverables to receive credit

1. (4 points) Optimize the cells above to tune the model so that it generates text that closely resembles the orginal line from the rhyme, or at least generates sensible words. It's okay if the third example using unseen text still looks somewhat strange though. Again, this is a toy problem, as language models require a lot of computation. This toy problem is great for rapid experimentation to explore different aspects of deep learning language models.
2. (3 points) Write a function to split the text corpus file into training and validation and pipe the validation data into the model.fit() function to be able to track validation error per epoch. Lookup Keras documentation to see how this is handled.
3. (3 points) Write a summary (methods and results) in the cells below of the different things you applied. You must include your intuitions behind what did work and what did not work well.

4. (Extra Credit 2.5 points) Do something even more interesting. Try a different source text. Train a word-level model. We'll leave it up to your creativity to explore and write a summary of your methods and results.

#1 Optimize the Cell

```
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import ModelCheckpoint


model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(vocab_size, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


checkpoint = ModelCheckpoint('best_model.h5', monitor='loss', verbose=1, save_best_only=True, mode='min')


model.fit(X, y, epochs=100, batch_size=128, callbacks=[checkpoint])
```

```
Epoch 1/100
3/3 [==============================] - ETA: 0s - loss: 3.6319 - accuracy: 0.0573
Epoch 1: loss improved from inf to 3.63193, saving model to best_model.h5
3/3 [==============================] - 6s 103ms/step - loss: 3.6319 - accuracy: 0.0573
Epoch 2/100
3/3 [==============================] - ETA: 0s - loss: 3.6075 - accuracy: 0.1380
Epoch 2: loss improved from 3.63193 to 3.60749, saving model to best_model.h5
3/3 [==============================] - 0s 92ms/step - loss: 3.6075 - accuracy: 0.1380
Epoch 3/100
3/3 [==============================] - ETA: 0s - loss: 3.5707 - accuracy: 0.1562
Epoch 3: loss improved from 3.60749 to 3.57071, saving model to best_model.h5
3/3 [==============================] - 0s 88ms/step - loss: 3.5707 - accuracy: 0.1562
Epoch 4/100
2/3 [===================>..........] - ETA: 0s - loss: 3.5209 - accuracy: 0.1484
Epoch 4: loss improved from 3.57071 to 3.49830, saving model to best_model.h5
3/3 [==============================] - 0s 74ms/step - loss: 3.4983 - accuracy: 0.1589
Epoch 5/100
2/3 [===================>..........] - ETA: 0s - loss: 3.4035 - accuracy: 0.1484
Epoch 5: loss improved from 3.49830 to 3.36269, saving model to best_model.h5
3/3 [==============================] - 0s 64ms/step - loss: 3.3627 - accuracy: 0.1562
Epoch 6/100
3/3 [==============================] - ETA: 0s - loss: 3.1951 - accuracy: 0.1562
Epoch 6: loss improved from 3.36269 to 3.19515, saving model to best_model.h5
3/3 [==============================] - 0s 62ms/step - loss: 3.1951 - accuracy: 0.1562
Epoch 7/100
3/3 [==============================] - ETA: 0s - loss: 3.1826 - accuracy: 0.1589
Epoch 7: loss improved from 3.19515 to 3.18263, saving model to best_model.h5
3/3 [==============================] - 0s 55ms/step - loss: 3.1826 - accuracy: 0.1589
Epoch 8/100
3/3 [==============================] - ETA: 0s - loss: 3.1615 - accuracy: 0.1224
Epoch 8: loss improved from 3.18263 to 3.16150, saving model to best_model.h5
3/3 [==============================] - 0s 53ms/step - loss: 3.1615 - accuracy: 0.1224
Epoch 9/100
3/3 [==============================] - ETA: 0s - loss: 3.1073 - accuracy: 0.1484
Epoch 9: loss improved from 3.16150 to 3.10730, saving model to best_model.h5
3/3 [==============================] - 0s 57ms/step - loss: 3.1073 - accuracy: 0.1484
Epoch 10/100
3/3 [==============================] - ETA: 0s - loss: 3.1106 - accuracy: 0.1406
Epoch 10: loss did not improve from 3.10730
3/3 [==============================] - 0s 43ms/step - loss: 3.1106 - accuracy: 0.1406
Epoch 11/100
3/3 [==============================] - ETA: 0s - loss: 3.1117 - accuracy: 0.1510
Epoch 11: loss did not improve from 3.10730
3/3 [==============================] - 0s 47ms/step - loss: 3.1117 - accuracy: 0.1510
Epoch 12/100
3/3 [==============================] - ETA: 0s - loss: 3.1102 - accuracy: 0.1615
Epoch 12: loss did not improve from 3.10730
3/3 [==============================] - 0s 44ms/step - loss: 3.1102 - accuracy: 0.1615
Epoch 13/100
3/3 [==============================] - ETA: 0s - loss: 3.0756 - accuracy: 0.1589
Epoch 13: loss improved from 3.10730 to 3.07562, saving model to best_model.h5
3/3 [==============================] - 0s 60ms/step - loss: 3.0756 - accuracy: 0.1589
Epoch 14/100
3/3 [==============================] - ETA: 0s - loss: 3.0900 - accuracy: 0.1536
Epoch 14: loss did not improve from 3.07562
3/3 [==============================] - 0s 46ms/step - loss: 3.0900 - accuracy: 0.1536
Epoch 15/100
3/3 [==============================] - ETA: 0s - loss: 3.0777 - accuracy: 0.1589
```

```
#2 Split the Text

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_state=42)
history = model.fit(X_train, y_train, epochs=100, batch_size=128, validation_data=(X_val, y_val), callbacks=[checkpoint])
```

```
Epoch 1/100
3/3 [==============================] - ETA: 0s - loss: 1.1393 - accuracy: 0.6725
Epoch 1: loss improved from 1.16958 to 1.13933, saving model to best_model.h5
3/3 [==============================] - 4s 502ms/step - loss: 1.1393 - accuracy: 0.6725 - val_loss: 0.9573 - val_accuracy: 0.6923
Epoch 2/100
2/3 [===================>.....] - ETA: 0s - loss: 1.1696 - accuracy: 0.6602
Epoch 2: loss improved from 1.13933 to 1.13837, saving model to best_model.h5
3/3 [==============================] - 0s 75ms/step - loss: 1.1384 - accuracy: 0.6667 - val_loss: 0.9964 - val_accuracy: 0.6923
Epoch 3/100
3/3 [==============================] - ETA: 0s - loss: 1.0565 - accuracy: 0.6957
Epoch 3: loss improved from 1.13837 to 1.05647, saving model to best_model.h5
3/3 [==============================] - 0s 77ms/step - loss: 1.0565 - accuracy: 0.6957 - val_loss: 1.0581 - val_accuracy: 0.6923
Epoch 4/100
3/3 [==============================] - ETA: 0s - loss: 1.0396 - accuracy: 0.7101
Epoch 4: loss improved from 1.05647 to 1.03961, saving model to best_model.h5
3/3 [==============================] - 0s 70ms/step - loss: 1.0396 - accuracy: 0.7101 - val_loss: 1.0844 - val_accuracy: 0.6923
Epoch 5/100
3/3 [==============================] - ETA: 0s - loss: 1.0557 - accuracy: 0.7043
Epoch 5: loss did not improve from 1.03961
3/3 [==============================] - 0s 60ms/step - loss: 1.0557 - accuracy: 0.7043 - val_loss: 1.1580 - val_accuracy: 0.6154
Epoch 6/100
3/3 [==============================] - ETA: 0s - loss: 1.0018 - accuracy: 0.7362
Epoch 6: loss improved from 1.03961 to 1.00182, saving model to best_model.h5
3/3 [==============================] - 0s 83ms/step - loss: 1.0018 - accuracy: 0.7362 - val_loss: 1.1680 - val_accuracy: 0.6154
Epoch 7/100
3/3 [==============================] - ETA: 0s - loss: 0.9920 - accuracy: 0.7333
Epoch 7: loss improved from 1.00182 to 0.99204, saving model to best_model.h5
3/3 [==============================] - 0s 84ms/step - loss: 0.9920 - accuracy: 0.7333 - val_loss: 1.1770 - val_accuracy: 0.5897
Epoch 8/100
3/3 [==============================] - ETA: 0s - loss: 1.0118 - accuracy: 0.7246
Epoch 8: loss did not improve from 0.99204
3/3 [==============================] - 0s 129ms/step - loss: 1.0118 - accuracy: 0.7246 - val_loss: 1.1950 - val_accuracy: 0.6154
Epoch 9/100
3/3 [==============================] - ETA: 0s - loss: 0.9667 - accuracy: 0.7478
Epoch 9: loss improved from 0.99204 to 0.96673, saving model to best_model.h5
3/3 [==============================] - 0s 151ms/step - loss: 0.9667 - accuracy: 0.7478 - val_loss: 1.1877 - val_accuracy: 0.6667
Epoch 10/100
3/3 [==============================] - ETA: 0s - loss: 0.9674 - accuracy: 0.7362
Epoch 10: loss did not improve from 0.96673
3/3 [==============================] - 0s 112ms/step - loss: 0.9674 - accuracy: 0.7362 - val_loss: 1.3311 - val_accuracy: 0.5897
Epoch 11/100
3/3 [==============================] - ETA: 0s - loss: 0.9392 - accuracy: 0.7246
Epoch 11: loss improved from 0.96673 to 0.93922, saving model to best_model.h5
3/3 [==============================] - 0s 130ms/step - loss: 0.9392 - accuracy: 0.7246 - val_loss: 1.3394 - val_accuracy: 0.5641
Epoch 12/100
3/3 [==============================] - ETA: 0s - loss: 0.9339 - accuracy: 0.7362
Epoch 12: loss improved from 0.93922 to 0.93391, saving model to best_model.h5
3/3 [==============================] - 0s 105ms/step - loss: 0.9339 - accuracy: 0.7362 - val_loss: 1.3392 - val_accuracy: 0.5385
Epoch 13/100
3/3 [==============================] - ETA: 0s - loss: 0.8664 - accuracy: 0.7797
Epoch 13: loss improved from 0.93391 to 0.86640, saving model to best_model.h5
3/3 [==============================] - 0s 120ms/step - loss: 0.8664 - accuracy: 0.7797 - val_loss: 1.4036 - val_accuracy: 0.5897
Epoch 14/100
3/3 [==============================] - ETA: 0s - loss: 0.8927 - accuracy: 0.7797
Epoch 14: loss did not improve from 0.86640
3/3 [==============================] - 0s 116ms/step - loss: 0.8927 - accuracy: 0.7797 - val_loss: 1.3681 - val_accuracy: 0.5128
Epoch 15/100
3/3 [==============================] - ETA: 0s - loss: 0.8752 - accuracy: 0.7652
```

```
#3 Summary
#In this assignment, the focus was on optimizing a model (specify type, like LSTM, CNN, etc.) for a specific task. The data was carefully prep

#Reflecting on the process, certain strategies like specific architectural choices and data preprocessing techniques worked well, contributi


#4 Extra Credit
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from keras.utils import to_categorical
import numpy as np

tokenizer = Tokenizer()
tokenizer.fit_on_texts([raw_text])
encoded = tokenizer.texts_to_sequences([raw_text])[0]

sequence_length = 9
sequences = []
for i in range(sequence_length, len(encoded)):
```

```
    sequence = encoded[i - sequence_length:i + 1]
    sequences.append(sequence)

sequences = np.array(sequences)
X, y = sequences[:, :-1], sequences[:, -1]
y = to_categorical(y, num_classes=len(tokenizer.word_index) + 1)

model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=50, input_length=9))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100))
model.add(Dense(100, activation='relu'))
model.add(Dense(len(tokenizer.word_index) + 1, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
model.fit(X, y, epochs=100, batch_size=128)
```

```
9/9 [==============================] - 0s 39ms/step - loss: 4.1940 - accuracy: 0.1471
Epoch 16/100
9/9 [==============================] - 0s 46ms/step - loss: 4.0989 - accuracy: 0.1471
Epoch 17/100
9/9 [==============================] - 0s 45ms/step - loss: 4.0021 - accuracy: 0.1614
Epoch 18/100
9/9 [==============================] - 0s 44ms/step - loss: 3.9241 - accuracy: 0.1671
Epoch 19/100
9/9 [==============================] - 0s 40ms/step - loss: 3.8505 - accuracy: 0.1757
Epoch 20/100
9/9 [==============================] - 0s 41ms/step - loss: 3.7946 - accuracy: 0.1738
Epoch 21/100
9/9 [==============================] - 0s 44ms/step - loss: 3.7215 - accuracy: 0.1843
Epoch 22/100
9/9 [==============================] - 0s 41ms/step - loss: 3.6588 - accuracy: 0.1815
Epoch 23/100
9/9 [==============================] - 0s 42ms/step - loss: 3.5662 - accuracy: 0.2063
Epoch 24/100
9/9 [==============================] - 0s 42ms/step - loss: 3.4871 - accuracy: 0.2101
Epoch 25/100
9/9 [==============================] - 0s 39ms/step - loss: 3.4336 - accuracy: 0.2178
Epoch 26/100
9/9 [==============================] - 0s 44ms/step - loss: 3.3733 - accuracy: 0.2053
Epoch 27/100
9/9 [==============================] - 0s 39ms/step - loss: 3.3092 - accuracy: 0.2321
Epoch 28/100
9/9 [==============================] - 0s 43ms/step - loss: 3.2270 - accuracy: 0.2455
Epoch 29/100
9/9 [==============================] - 0s 42ms/step - loss: 3.1550 - accuracy: 0.2550
Epoch 30/100
9/9 [==============================] - 0s 41ms/step - loss: 3.1113 - accuracy: 0.2550
Epoch 31/100
9/9 [==============================] - 0s 43ms/step - loss: 3.0298 - accuracy: 0.2741
Epoch 32/100
9/9 [==============================] - 0s 47ms/step - loss: 2.9812 - accuracy: 0.2779
Epoch 33/100
9/9 [==============================] - 0s 42ms/step - loss: 2.9200 - accuracy: 0.2846
Epoch 34/100
9/9 [==============================] - 0s 42ms/step - loss: 2.8622 - accuracy: 0.3009
Epoch 35/100
9/9 [==============================] - 0s 39ms/step - loss: 2.8126 - accuracy: 0.3142
Epoch 36/100
9/9 [==============================] - 0s 43ms/step - loss: 2.7777 - accuracy: 0.3056
Epoch 37/100
9/9 [==============================] - 0s 46ms/step - loss: 2.7701 - accuracy: 0.2980
Epoch 38/100
9/9 [==============================] - 1s 69ms/step - loss: 2.6884 - accuracy: 0.3161
Epoch 39/100
9/9 [==============================] - 1s 77ms/step - loss: 2.6376 - accuracy: 0.3305
Epoch 40/100
9/9 [==============================] - 1s 73ms/step - loss: 2.6013 - accuracy: 0.3391
Epoch 41/100
9/9 [==============================] - 1s 77ms/step - loss: 2.5596 - accuracy: 0.3410
Epoch 42/100
9/9 [==============================] - 1s 73ms/step - loss: 2.5105 - accuracy: 0.3515
Epoch 43/100
9/9 [==============================] - 1s 76ms/step - loss: 2.4791 - accuracy: 0.3563
Epoch 44/100
9/9 [==============================] - 0s 51ms/step - loss: 2.4322 - accuracy: 0.3639
```