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 "rhyme.txt". Place this in the same directory as this jupyter notebook:

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

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.9.2)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.27.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.12)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.19.6)
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.9.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.50.0)
Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.9.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.3.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.1.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (4.1.1)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: tensorboard<2.10,>=2.9 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.9.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow) (57.4.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (14.0.6)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.21.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tensorflow) (21.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tensorflow) (0.38.3)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tensorflow) (1.5.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (2.14.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (1.8.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (0.4.6)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (1.0.1)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (2.23.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.10,>=2.9->tensorflow) (0.6.1)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.10,>=2.9->tensorflow) (4.9)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.10,>=2.9->tensorflow) (0.2.8)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.10,>=2.9->tensorflow) (5.2.0)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.10,>=2.9->tensorflow) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorboard<2.10,>=2.9->tensorflow) (4.13.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<2.10,>=2.9->tensorflow) (3.10.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.10,>=2.9->tensorflow) (0.4.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.10,>=2.9->tensorflow) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.10,>=2.9->tensorflow) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.10,>=2.9->tensorflow) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.10,>=2.9->tensorflow) (2022.9.24)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.10,>=2.9->tensorflow) (3.2.2)
```

```
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->tensorflow) (3.0.9)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2.9.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.21.6)
```

```
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 sing;Wasn’t that a dainty dish,To set before the king.The king was in his counting house,Counting
 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

```
from keras.layers import Dropout


# define model
model = Sequential()
model.add(LSTM(200, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
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)
```

```
Epoch 72/100
12/12 [==============================] - 0s 27ms/step - loss: 0.0912 - accuracy: 0.9792
Epoch 73/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0883 - accuracy: 0.9818
Epoch 74/100
12/12 [==============================] - 0s 27ms/step - loss: 0.0870 - accuracy: 0.9922
Epoch 75/100
12/12 [==============================] - 0s 27ms/step - loss: 0.0883 - accuracy: 0.9896
Epoch 76/100
```

```
12/12 [==============================] - 0s 26ms/step - loss: 0.0820 - accuracy: 0.9870
Epoch 77/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0823 - accuracy: 0.9896
Epoch 78/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0679 - accuracy: 0.9922
Epoch 79/100
12/12 [==============================] - 0s 30ms/step - loss: 0.0718 - accuracy: 0.9896
Epoch 80/100
12/12 [==============================] - 0s 29ms/step - loss: 0.0629 - accuracy: 0.9922
Epoch 81/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0677 - accuracy: 0.9896
Epoch 82/100
12/12 [==============================] - 0s 29ms/step - loss: 0.0679 - accuracy: 0.9818
Epoch 83/100
12/12 [==============================] - 0s 28ms/step - loss: 0.1022 - accuracy: 0.9818
Epoch 84/100
12/12 [==============================] - 0s 25ms/step - loss: 0.1071 - accuracy: 0.9792
Epoch 85/100
12/12 [==============================] - 0s 26ms/step - loss: 0.0784 - accuracy: 0.9870
Epoch 86/100
12/12 [==============================] - 0s 26ms/step - loss: 0.0686 - accuracy: 0.9870
Epoch 87/100
12/12 [==============================] - 0s 25ms/step - loss: 0.0598 - accuracy: 0.9896
Epoch 88/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0808 - accuracy: 0.9792
Epoch 89/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0620 - accuracy: 0.9922
Epoch 90/100
12/12 [==============================] - 0s 29ms/step - loss: 0.0738 - accuracy: 0.9792
Epoch 91/100
12/12 [==============================] - 0s 31ms/step - loss: 0.0614 - accuracy: 0.9974
Epoch 92/100
12/12 [==============================] - 0s 28ms/step - loss: 0.0651 - accuracy: 0.9896
Epoch 93/100
12/12 [==============================] - 0s 27ms/step - loss: 0.0656 - accuracy: 0.9766
Epoch 94/100
12/12 [==============================] - 0s 27ms/step - loss: 0.0598 - accuracy: 0.9844
Epoch 95/100
12/12 [==============================] - 0s 26ms/step - loss: 0.0449 - accuracy: 0.9922
Epoch 96/100
12/12 [==============================] - 0s 40ms/step - loss: 0.0457 - accuracy: 0.9896
Epoch 97/100
12/12 [==============================] - 1s 53ms/step - loss: 0.0357 - accuracy: 0.9974
Epoch 98/100
12/12 [==============================] - 1s 60ms/step - loss: 0.0379 - accuracy: 0.9922
Epoch 99/100
12/12 [==============================] - 1s 58ms/step - loss: 0.0381 - accuracy: 0.9922
Epoch 100/100
12/12 [==============================] - 0s 32ms/step - loss: 0.0354 - accuracy: 0.9922
```

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

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 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
Sing a song of sixpence,A pock
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
king was in his counting house
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 23ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step

1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 25ms/step
```

```
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
hello worl,.Fn  ingg oue i wpe
```

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.

Q1: I did changes to the definition and fit part for the model and I tried using the regularization method of dropout. I also attempted different numbers for hidden layers and epochs.

```
#Q2
def model_fit_with_train_val_split(X, y, split = 0):
    return model.fit(X, y, epochs = 100, validation_split = split)

model_fit_with_train_val_split(X, y, split = 0.25)
```

```
9/9 [==============================] - 0s 36ms/step - loss: 0.0808 - accuracy: 0.9861 - val_loss: 0.5884 - val_accuracy: 0.7917
Epoch 73/100
9/9 [==============================] - 0s 36ms/step - loss: 0.0829 - accuracy: 0.9826 - val_loss: 0.4448 - val_accuracy: 0.8750
Epoch 74/100
9/9 [==============================] - 0s 42ms/step - loss: 0.1020 - accuracy: 0.9896 - val_loss: 0.4170 - val_accuracy: 0.8750
Epoch 75/100
9/9 [==============================] - 0s 41ms/step - loss: 0.0311 - accuracy: 1.0000 - val_loss: 0.4333 - val_accuracy: 0.8958
Epoch 76/100
9/9 [==============================] - 0s 37ms/step - loss: 0.0339 - accuracy: 1.0000 - val_loss: 0.3857 - val_accuracy: 0.9062
Epoch 77/100
9/9 [==============================] - 0s 40ms/step - loss: 0.0195 - accuracy: 1.0000 - val_loss: 0.3883 - val_accuracy: 0.8854
Epoch 78/100
9/9 [==============================] - 0s 36ms/step - loss: 0.0151 - accuracy: 1.0000 - val_loss: 0.3934 - val_accuracy: 0.8750
Epoch 79/100
9/9 [==============================] - 0s 35ms/step - loss: 0.0124 - accuracy: 1.0000 - val_loss: 0.3694 - val_accuracy: 0.8958
Epoch 80/100
9/9 [==============================] - 0s 40ms/step - loss: 0.0110 - accuracy: 1.0000 - val_loss: 0.3588 - val_accuracy: 0.9271
Epoch 81/100
9/9 [==============================] - 0s 40ms/step - loss: 0.0102 - accuracy: 1.0000 - val_loss: 0.3485 - val_accuracy: 0.9271
Epoch 82/100
9/9 [==============================] - 1s 61ms/step - loss: 0.0112 - accuracy: 1.0000 - val_loss: 0.3485 - val_accuracy: 0.9062
Epoch 83/100
9/9 [==============================] - 1s 72ms/step - loss: 0.0078 - accuracy: 1.0000 - val_loss: 0.3616 - val_accuracy: 0.8958
Epoch 84/100
9/9 [==============================] - 1s 86ms/step - loss: 0.0078 - accuracy: 1.0000 - val_loss: 0.3595 - val_accuracy: 0.9062
Epoch 85/100
9/9 [==============================] - 1s 62ms/step - loss: 0.0084 - accuracy: 1.0000 - val_loss: 0.3508 - val_accuracy: 0.9167
Epoch 86/100
9/9 [==============================] - 0s 54ms/step - loss: 0.0068 - accuracy: 1.0000 - val_loss: 0.3507 - val_accuracy: 0.9167
Epoch 87/100
9/9 [==============================] - 1s 61ms/step - loss: 0.0068 - accuracy: 1.0000 - val_loss: 0.3564 - val_accuracy: 0.9167
Epoch 88/100
9/9 [==============================] - 0s 36ms/step - loss: 0.0061 - accuracy: 1.0000 - val_loss: 0.3627 - val_accuracy: 0.9167
Epoch 89/100
9/9 [==============================] - 0s 40ms/step - loss: 0.0057 - accuracy: 1.0000 - val_loss: 0.3725 - val_accuracy: 0.9062
Epoch 90/100
```

```
Epoch 90/100
9/9 [==============================] - 0s 36ms/step - loss: 0.0058 - accuracy: 1.0000 - val_loss: 0.3783 - val_accuracy: 0.9062
Epoch 91/100
9/9 [==============================] - 0s 38ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.3780 - val_accuracy: 0.9062
Epoch 92/100
9/9 [==============================] - 0s 36ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 0.3764 - val_accuracy: 0.9062
Epoch 93/100
9/9 [==============================] - 0s 33ms/step - loss: 0.0057 - accuracy: 1.0000 - val_loss: 0.3767 - val_accuracy: 0.9062
Epoch 94/100
9/9 [==============================] - 0s 33ms/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 0.3766 - val_accuracy: 0.9062
Epoch 95/100
9/9 [==============================] - 0s 32ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 0.3732 - val_accuracy: 0.9062
Epoch 96/100
9/9 [==============================] - 0s 39ms/step - loss: 0.0050 - accuracy: 1.0000 - val_loss: 0.3757 - val_accuracy: 0.9062
Epoch 97/100
9/9 [==============================] - 0s 32ms/step - loss: 0.0064 - accuracy: 1.0000 - val_loss: 0.3871 - val_accuracy: 0.9062
Epoch 98/100
9/9 [==============================] - 0s 35ms/step - loss: 0.0046 - accuracy: 1.0000 - val_loss: 0.3925 - val_accuracy: 0.9062
Epoch 99/100
9/9 [==============================] - 0s 34ms/step - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.3938 - val_accuracy: 0.9062
Epoch 100/100
9/9 [==============================] - 0s 31ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.3923 - val_accuracy: 0.9062
<keras.callbacks.History at 0x7f902fffbc50>
```

Q3. I first changed the number of memory cells for the model to 200, which increased the validation test accuracy. This makes sense since increasing the number of memory cells makes the model more complex, meaning it can fit data better. I also tried to use Dropout to decrease overtraining.