Machine Learning - CS6301

Mini Project

Next Word Prediction

IMPLEMENTATION AND RESULTS

Done By:

Vignesh Kumar Murugavel Usha - 2018103079

Vivek Ramkumar – 2018103082

Contents

Next Word Prediction Using RNN Model

Introduction

Every day we usually need to type a lot of text. Some of us are faster writing long text but others will require more time and effort. In this project we will attempt to improve our typing experience by predicting the next word in a sentence fragment. This will help us type text faster and also avoid misspelling.

We use the Recurrent Neural Network for this purpose. This model was chosen because it provides a way to examine the previous input. LSTM, a special kind of RNN is also used for this purpose. The LSTM provides the mechanism to preserve the errors that can be backpropagated through time and layers which helps to reduce the **vanishing gradient** problem.

Objective:

To design a Recurrent Neural Network model to predict the next word of a sentence.

Method Used:

LSTMs (Long short term memory) :

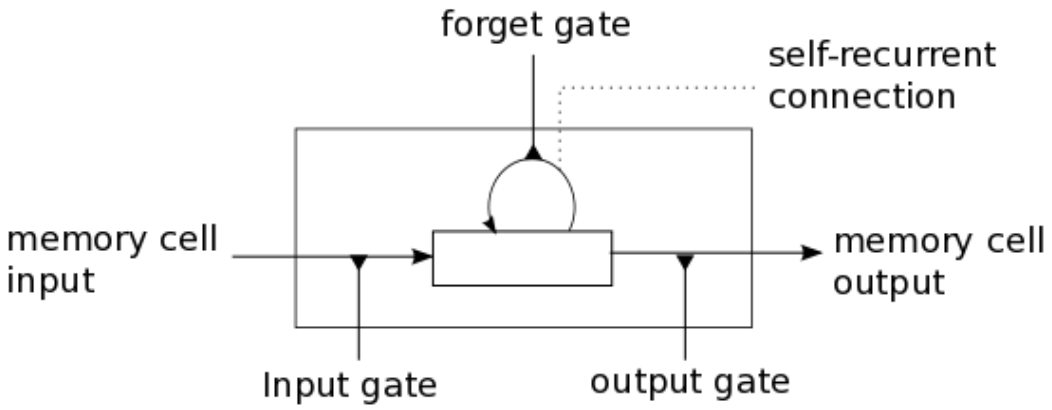
Two major problems torment the RNNs — vanishing and exploding gradients. In traditional RNNs the gradient signal can be multiplied a large number of times by the weight matrix. Thus, the magnitude of the weights of the transition matrix can play an important role. If the weights in the matrix are small, the gradient signal becomes smaller at every training step, thus making learning very slow or completely stops it. This is called vanishing gradient.

LSTM model is a special kind of RNN that learns long-term dependencies. It introduces new structure — the memory cell

that is composed of four elements: input, forget and output gates and a neuron that connects to itself.

The model it trained by giving a text file containing many English sentences as input.

LSTM Model:



Bi-LSTM:(Bi-directional long short term memory):

Bidirectional recurrent neural networks(RNN) are really just putting two independent RNNs together. This structure allows the networks to have both backward and forward information about the sequence at every time step

Using bidirectional will run your inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backward you preserve information from the future and using the two hidden states combined you are able in any point in time to preserve information from both past and future. What they are suited for is a very complicated question but BiLSTMs show very good results as they can understand the context

better. Let's say we try to predict the next word in a sentence, on a high level what a unidirectional LSTM will see is

“The boys went to ….”

And will try to predict the next word only by this context, with bidirectional LSTM you will be able to see information further down the road for example

Forward LSTM:

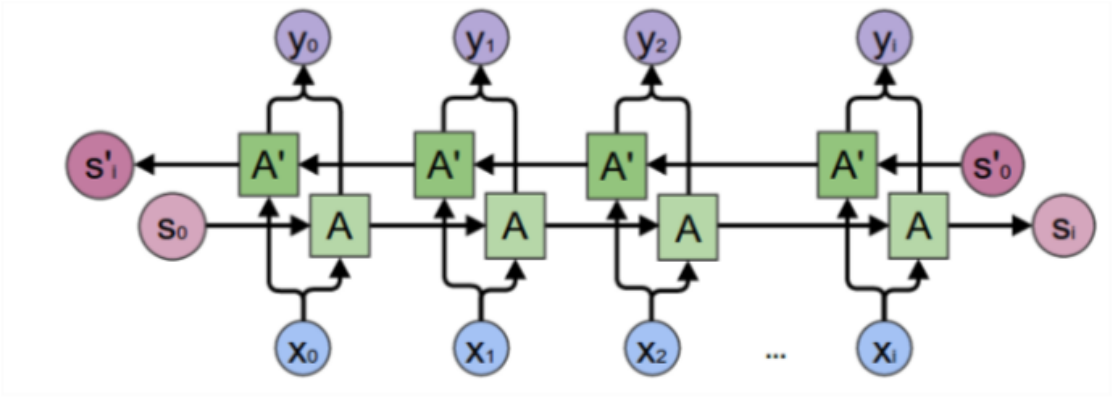
“The boys went to …”

Backward LSTM:

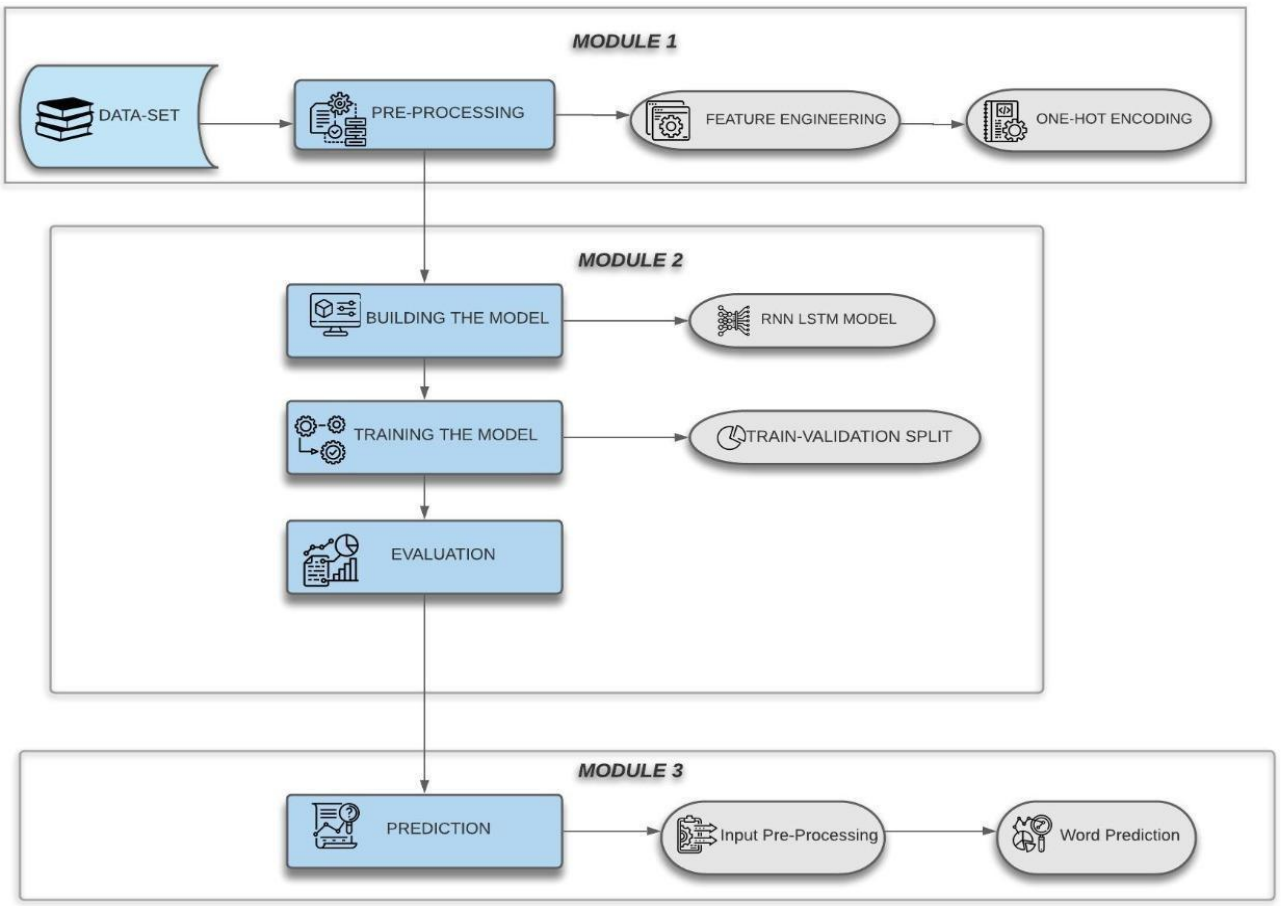
“… and then they got out of the pool”

You can see that using the information from the future it could be easier for the network to understand what the next word is.

Bi-LSTM Model:



Block Diagram:



MODULES WITH INPUT AND OUTPUT:

MODULE 1: PREPROCESSING

- Input: The Adventures of Sherlock Holmes Novel
- Output: split the entire dataset into each word in order without the presence of special characters.

CODE:

```
path = 'ml_text_file.txt'
text = open(path).read().lower()
print(text)
print('corpus length:', len(text))
tokenizer = RegexpTokenizer(r'\w+')
words = tokenizer.tokenize(text)
```

FEATURE ENGINEERING

Input: Dataset split into each word in order without the presence of special characters.

Output: List of next words and previous words.

- For the feature engineering part, we need to have the unique sorted words list. We also need a dictionary(<key: value>) with each word form the unique_words list as key and its corresponding position as value.

- We define a SEQUENCE_LENGTH which means that the number of previous words that determines the next word. Also, we create an empty list called sentences to store a set of five previous words and its corresponding next word in the next_chars list. We fill these lists by looping over a range of 40 less than the length of words.

CODE:

```
chars = sorted(list(set(text)))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))

print(f'unique chars: {len(chars)}')

SEQUENCE_LENGTH = 40
step=3
sentences = []
next_chars = []
for i in range(len(text) - SEQUENCE_LENGTH):
    sentence.append(text[i:i + SEQUENCE_LENGTH])
    next_char.append(text[i + SEQUENCE_LENGTH])
```

FEATURE EXTRACTION –
ONE HOT ENCODING

Input: List of next words and previous words.

Output: One hot encoded list of next and previous words.

- Here, we create two numpy array X(for storing the features) and Y(for storing the corresponding label(here, next word)). We iterate X and Y if the word is present then the corresponding position is made 1.

CODE:

```
X = np.zeros((len(sentences), SEQUENCE_LENGTH, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        X[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
```

MODULE 2: CREATING CLASSIFIER MODELS:

In our project we used two LSTM models , a simple LSTM model and later a BI-LSTM model to improve accuracy

Input: One hot encoded list of next and previous words..

Output: Trained Model.

- We used a multi layer LSTM model with 128 neurons first and got lower accuracy
- Then , We used a Bi-directional Multi-layer LSTM model with 128 neurons, a fully connected layer, and a softmax function for activation.
- The softmax activation ensures that we receive a bunch of probabilities for the outputs equal to the vocab size.

CODE:

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
    from keras.layers import Dense, Activation, Dropout
    from keras.layers import LSTM, Input, Flatten, Bidirectional
    from keras.layers.normalization import BatchNormalization
    from keras.optimizers import Adam
    model = Sequential()
    model.add(Bidirectional(LSTM(128,
activation="relu",return_sequences=True),input_shape=(SEQU
ENCE_LENGTH, len(chars))))
    model.add(Bidirectional(LSTM(128)))
    model.add(Dropout(0.6))
    model.add(Dense(len(chars)))
    model.add(Activation('softmax'))


    optimizer = Adam(lr=0.001)
```

Callbacks:

- In order to improve the training process callbacks were used
- We will be importing the 3 required callbacks for training our model. The 3 important callbacks are ModelCheckpoint, ReduceLROnPlateau, and Tensorboard.
- **ModelCheckpoint** — This callback is used for storing the weights of our model after training. We save only the best weights of our model by specifying save_best_only=True. We will monitor our training by using the loss metric.
- **ReduceLROnPlateau** — This callback is used for reducing the learning rate of the optimizer after a specified number of epochs. Here, we have specified the patience as 3. If the accuracy does not improve after 3 epochs, then our learning rate is reduced accordingly by a factor of 0.2. The metric used for monitoring here is loss as well.
- **Tensorboard** — The tensorboard callback is used for plotting the visualization of the graphs, namely the graph plots for accuracy and the loss. Here, we will only be looking at the loss graph of the next word prediction.

CODE:

```
checkpoint = ModelCheckpoint("nextword1.h5", monitor='loss', verbose=1,
    save_best_only=True, mode='auto')
reduce = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=3, min_lr=0.0001, verbose = 1)
logdir='logsnextword1'
tensorboard_Visualization = TensorBoard(log_dir=logdir)
```

Fitting the Model:

- We are compiling and fitting our model in the final step. Here, we are training the model and saving the best weights to nextword1.h5 so that we don't have to re-train the model repeatedly and we can use our saved model when required. Here We have trained only on the training data and validation data with a split of 0.05.
- The loss we have used is categorical_crossentropy which computes the cross-entropy loss between the labels and predictions.
- The optimizer we will be using is Adam with a learning rate of 0.001 and we will compile our model on the metric loss.

CODE:

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

history = model.fit(X, y, validation_split=0.05, batch_size=128,

epochs=80, shuffle=True, callbacks=[checkpoint, reduce,

tensorboard_Visualization]).history
```

MODULE 3 : PREDICTING NEXT WORDS

Input: Sentences in English Language.

Output: Predicted list of next words.

- Now, we need to predict new words using this model. To do that we input the sample as a feature vector. we convert the input string to a single feature vector.

CODE:

```
def prepare_input(text):
    x = np.zeros((1, SEQUENCE_LENGTH, len(chars)))
    for t, char in enumerate(text):
        x[0, t, char_indices[char]] = 1.

    return x
```

- To choose the best possible n words after the prediction from the model is done by sample function.

CODE:

```
def sample(preds, top_n=3):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds)
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)

    return heapq.nlargest(top_n, range(len(preds)), preds.take)
```

- This function predicts next character until space is predicted .It does so by repeatedly preparing input, asking our model for predictions and sampling from them.

CODE:

```
def predict_completion(text):
    original_text = text
    generated = text
    completion = ''
    while True:
        x = prepare_input(text)
        preds = model.predict(x, verbose=0)[0]
        next_index = sample(preds, top_n=1)[0]
        next_char = indices_char[next_index]
        text = text[1:] + next_char
        completion += next_char
        if len(original_text + completion) + 2 > len(original_text) an
d next_char == ' ':
            return completion
```

- The final function that wraps everything and allow us to predict multiple completions:

CODE:

```
def predict_completions(text, n=3):
    x = prepare_input(text)
    preds = model.predict(x, verbose=0)[0]
    next_indices = sample(preds, n)
    return [indices_char[idx] + predict_completion(text[1:] + indic
es_char[idx]) for idx in next_indices]
```

Sample Input:

```
quotes = [

    "where is the class happening today at what.",

    "That which does not kill us makes us always makes us stronger.",

    "Im not upset that you lied to me, Im only mildly dissapoined and upset that from now on I cant believe you.",

    "And those who were seen dancing were thought to be insane by those who could not hear the music.",

    "It is not a good day today for me as i was attacked by some people today and i got hurt badly" ,

    "in the match that was conducted today we had sucessfuly defated our opponents"


]
```

Predicting outputs for the following inputs:

```
for q in quotes:
    seq = q[:40].lower()
    print(seq)
    print(predict_completions(seq,10))
    print()
```

FINAL OUTPUT:

```
where is the class happening today at wh
['at ', 'om ', 'ich ', 'ere ', 'uch ', 'y ', 'ree ', 'le ', 'weer ', 'hat ']

that which does not kill us makes us alw
['ays ', 'ed ', 'ord ', 'y ', 'her ', ' it ', 'not ', 'moth, ', 'de ', 'so ']

im not upset that you lied to me, im onl
['y ', 'e ', 'a ', 'icent ', ' passer ', 'lar ', 'ves ', 'ressees ', 's ', 'der ']

and those who were seen dancing were tho
['ught ', 'rough ', 'oed ', 'e ', 'ne ', 'ight ', '-chain ', ' complie ', 'le ', 'fe ']

it is not a good day today for me as i w
['as ', 'ould ', 'ere ', 'ill ', 'rited ', 'hat ', 'ull ', 'lich ', 'ynorth ', 'wish ']

in the match that was conducted today we
['ll ', 'st ', 're ', 'dding, ', 'ary ', ' shall ', 'nt ', 'eks ', 'ys."\n\n"i ', 'ggeted.
']
```

SCREENSHOT OF EACH MODULES / INTERMEDIATE RESULTS

Code and the corresponding outputs :

MODULE 1: PRE PROCESSING

```
 text = open(path).read().lower()

 print(type(text))
 print('corpus length:', len(text))
 tokenizer = RegexpTokenizer(r'\w+')
words = tokenizer.tokenize(text)
#print(words)
text= ''
for i in words:
    text+=i
    text+=' '
print(type(text))
```

```
<class 'str'>
corpus length: 581886
<class 'str'>
```

In [5]:

```
chars = sorted(list(set(text)))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))

print(f'unique chars: {len(chars)}')
```

```
unique chars: 72
```

In [6]:

```
SEQUENCE_LENGTH = 40
step = 3
sentences = []
next_chars = []
for i in range(0, len(text) - SEQUENCE_LENGTH, step):
    sentences.append(text[i: i + SEQUENCE_LENGTH])
    next_chars.append(text[i + SEQUENCE_LENGTH])
print(f'num training examples: {len(sentences)}')
```

```
num training examples: 193949
```

```
X = np.zeros((len(sentences), SEQUENCE_LENGTH, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        X[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
```

```
sentences[100]
```

```
' at www gutenberg org title the adventur'
```

```
next_chars[100]
```

```
'e'
```

```
X[0][0]
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False])
```

In [22]:
```
y[0]
```

Out[22]:
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False])
```

MODULE 2: - TRAINING THE MODEL – LSTM

[12]:

```
model = Sequential()
model.add(LSTM(128,return_sequences=True, input_shape=(SEQUENCE_LENGTH, len(chars))))
model.add(LSTM(128))
model.add(Dense(len(chars),activation='relu'))
model.add(Activation('softmax'))
```

[13]:

```
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import TensorBoard

checkpoint = ModelCheckpoint("nextword1.h5", monitor='loss', verbose=1,
    save_best_only=True, mode='auto')

reduce = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=3, min_lr=0.0001, verbose = 1)

logdir='logsnextword1'
tensorboard_Visualization = TensorBoard(log_dir=logdir)
```

```
Epoch 00018: loss did not improve from 1.73542
Epoch 19/25
1378/1378 [==============================] - 15s 11ms/step - loss: 2.2887 - accuracy: 0.4704 - val_loss: 2.8678 - val_accuracy: 0.3474

Epoch 00019: loss did not improve from 1.73542
Epoch 20/25
1378/1378 [==============================] - 14s 10ms/step - loss: 2.2894 - accuracy: 0.4707 - val_loss: 2.8709 - val_accuracy: 0.3464

Epoch 00020: loss did not improve from 1.73542
Epoch 21/25
1378/1378 [==============================] - 15s 11ms/step - loss: 2.2828 - accuracy: 0.4733 - val_loss: 2.8724 - val_accuracy: 0.3464

Epoch 00021: loss did not improve from 1.73542
Epoch 22/25
1378/1378 [==============================] - 14s 10ms/step - loss: 2.2873 - accuracy: 0.4720 - val_loss: 2.8722 - val_accuracy: 0.3470

Epoch 00022: loss did not improve from 1.73542
Epoch 23/25
1378/1378 [==============================] - 15s 11ms/step - loss: 2.2811 - accuracy: 0.4727 - val_loss: 2.8741 - val_accuracy: 0.3468

Epoch 00023: loss did not improve from 1.73542
Epoch 24/25
1378/1378 [==============================] - 14s 10ms/step - loss: 2.2778 - accuracy: 0.4741 - val_loss: 2.8746 - val_accuracy: 0.3476

Epoch 00024: loss did not improve from 1.73542
Epoch 25/25
1378/1378 [==============================] - 14s 10ms/step - loss: 2.2764 - accuracy: 0.4749 - val_loss: 2.8791 - val_accuracy: 0.3458

Epoch 00025: loss did not improve from 1.73542
```

+ Code + Markdown

MODULE 2: -TRAINING THE MODEL- BI-LSTM

]:

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.layers import Dense, Activation, Dropout
from keras.layers import LSTM, Input, Flatten, Bidirectional
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam
model = Sequential()
model.add(Bidirectional(LSTM(128, activation="relu",return_sequences=True),input_shape=(SEQUENCE_LENGTH, len(chars))))
model.add(Bidirectional(LSTM(128)))
model.add(Dropout(0.6))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
```

n [8]:

```
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import TensorBoard

checkpoint = ModelCheckpoint("nextword1.h5", monitor='loss', verbose=1,
    save_best_only=True, mode='auto')

reduce = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=3, min_lr=0.0001, verbose = 1)

logdir='logsnextword1'
tensorboard_Visualization = TensorBoard(log_dir=logdir)
```

n [9]:

```
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
history = model.fit(X, y, validation_split=0.05, batch_size=128, epochs=80, shuffle=True, callb
acks=[checkpoint, reduce, tensorboard_Visualization]).history
```

```
Epoch 1/80
1440/1440 [==============================] - 191s 129ms/step - loss: 2.8192 - accuracy: 0.22
54 - val_loss: 2.4447 - val_accuracy: 0.3127

Epoch 00001: loss improved from inf to 2.51976, saving model to nextword1.h5
```

```
Epoch 00077: loss did not improve from 0.88108
Epoch 78/80
1440/1440 [==============================] - 184s 127ms/step - loss: 0.8787 - accuracy: 0.72
38 - val_loss: 2.7784 - val_accuracy: 0.4611

Epoch 00078: loss did not improve from 0.88108
Epoch 79/80
1440/1440 [==============================] - 184s 128ms/step - loss: 0.8762 - accuracy: 0.72
29 - val_loss: 2.7556 - val_accuracy: 0.4643

Epoch 00079: loss did not improve from 0.88108
Epoch 80/80
1440/1440 [==============================] - 186s 129ms/step - loss: 0.8813 - accuracy: 0.72
05 - val_loss: 2.7564 - val_accuracy: 0.4575

Epoch 00080: loss did not improve from 0.88108
```

MODULE 3:PREDICTION

In [45]:
```
def prepare_input(text):
    x = np.zeros((1, SEQUENCE_LENGTH, len(chars)))
    for t, char in enumerate(text):
        x[0, t, char_indices[char]] = 1.

    return x
prepare_input("This is an example of input for our LSTM".lower())
```

Out[45]:
```
array([[[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]])
```

In [46]:
```
def sample(preds, top_n=3):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds)
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)

    return heapq.nlargest(top_n, range(len(preds)), preds.take)
```

In [54]:
```
def predict_completion(text):
    original_text = text
    generated = text
    completion = ''
    while True:
        x = prepare_input(text)
        preds = model.predict(x, verbose=0)[0]
        next_index = sample(preds, top_n=1)[0]
        next_char = indices_char[next_index]
        text = text[1:] + next_char
        completion += next_char

        if len(original_text + completion) + 2 > len(original_text) and next_char == ' ':
            return completion
```

[18]:
```
def predict_completions(text, n=3):
    x = prepare_input(text)
    preds = model.predict(x, verbose=0)[0]
    next_indices = sample(preds, n)
    return [indices_char[idx] + predict_completion(text[1:] + indices_char[idx]) for idx in next_indices]
```

[19]:
```
quotes = [
    "where is the class happening today at what.",
    "That which does not kill us makes us always makes us stronger.",
    "Im not upset that you lied to me, Im only mildly dissapoined and upset that from now on I cant believe you.",
    "And those who were seen dancing were thought to be insane by those who could not hear the music.",
    "It is not a good day today for me as i was attacked by some people today and i got hurt badly"

]
```

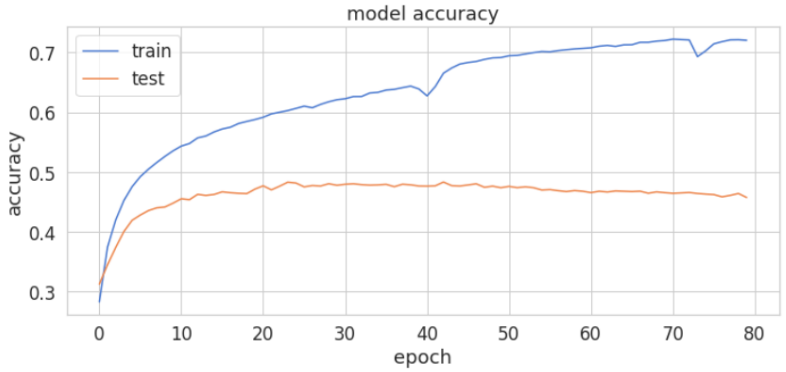
```
for q in quotes:
    seq = q[:40].lower()
    print(seq)
    print(predict_completions(seq,10))
    print()
```

FINAL RESULT : PREDICTION

```
where is the class happening today at wh
['at ', 'om ', 'ich ', 'ere ', 'uch ', 'y ', 'ree ', 'le ', 'weer ', 'hat ']

that which does not kill us makes us alw
['ays ', 'ed ', 'ord ', 'y ', 'her ', ' it ', 'not ', 'moth, ', 'de ', 'so ']

im not upset that you lied to me, im onl
['y ', 'e ', 'a ', 'icent ', ' passer ', 'lar ', 'ves ', 'ressees ', 's ', 'der ']

and those who were seen dancing were tho
['ught ', 'rough ', 'oed ', 'e ', 'ne ', 'ight ', '-chain ', ' complie ', 'le ', 'fe ']

it is not a good day today for me as i w
['as ', 'ould ', 'ere ', 'ill ', 'rited ', 'hat ', 'ull ', 'lich ', 'ynorth ', 'wish ']

in the match that was conducted today we
['ll ', 'st ', 're ', 'dding, ', 'ary ', ' shall ', 'nt ', 'eks ', 'ys."\n\n"i ', 'ggeted.
']
```

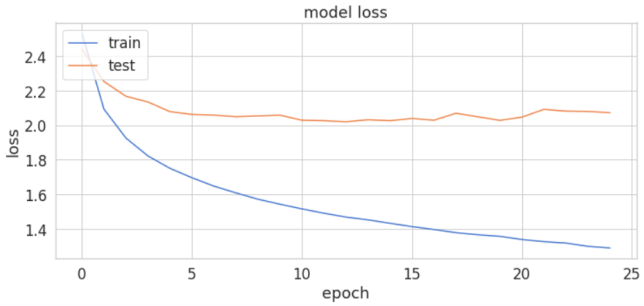
EVALUATION - Accuracy & Classification (BI-LSTM):

In [21]:
```
plt.plot(history['accuracy'])
plt.plot(history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left');
```



As the number of epoch's increases, the accuracy of the model also increases.

```
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left');
```



As number of epoch's increases, the loss of the model decreases.

COMPARISON OF MODELS :

For 80 Epochs BI-LSTM - gives an accuracy of 75%.

```
results = model.evaluate(X, y, batch_size=128,)
```

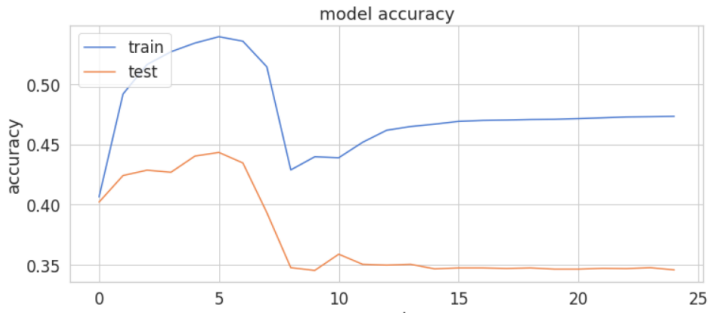
```
1516/1516 [==============================] - 22s 14ms/step - loss: 0.843
7 - accuracy: 0.7499
```

Earlier for single layer LSTM gave an accuracy of only 40% after just 25 epochs the model was overfitting and the loss increased.

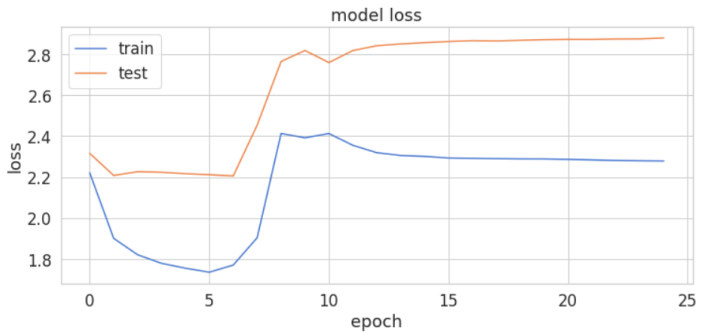
```
results = model.evaluate(X, y, batch_size=128,)
```

```
1451/1451 [==============================] - 7s 5ms/step - loss: 2.0095 - accuracy: 0.4320
```

```
plt.plot(history['accuracy'])
plt.plot(history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left');
```



```
plt.legend(['train', 'test'], loc='upper left');
```



Due to overfitting , the loss increased In subsequent epochs

Conclusion:

A special type of RNN - LSTM models can be used over traditional approach of Next Word Prediction. The model breaks down raw chunks of words into input sequences, of which it is trained to understand over successive iterations. It will then display the output, the next word automatically from the neural network.

Bi-LSTM Model was trained and used , we got a higher accuracy with a low loss value and the predicted words were much better compared to simple LSTM’s