Lab 6 - N-Grams and LSTM

Statistical Language Processing

- we are using stacks, using stacks we can do OCR, Spell check, speech rec, machine translation, pos tagging, parsing
- this uses stacks or corpora (database)
- disambiguate input
- depends on probability theory

Probability Theory

Conditional and Unconditional Probability

- Conditional - like having dependance on other factors, like sunny day depends on humidity, precipitation etc = P(A|B) where B is known event and A is unknown
- unconditional - does not depend on anything = P(A)
- Example P(put) is probability to see the word put in a text and P(on|put) is the probablity of seeing the word on after the word put

Bayes' Theorm

- used to calculate probability of A when B (P(A|B)) has occured given that we know the probability of P(B|A)
- calculate P(A|B) from given P(B|A)
- Formula: [P(B|A) * P(A)] / P(B)

Language Models

- models that assign probability to sequence of words are called language models (eg: LLM, N-Grams)
- n-grams is the simplest LM that assigns probabilities to sentences and sequences of words
- n-gram is a sequence of N words:
  1. 1-gram is uni-gram which is a single word sequence of words like "please" or "turn"
  2. 2-gram is bi-gram which is a two word sequence of words like "please turn" or "turn your" or "your homework"
  3. 3-gram is tri-gram which is a three word squence of words like "please turn your" or "turn your homework"

Probabilistic Language Models

- Used to assign probability to a sentence in many NLP tasks
- Machine Translation: P(high winds tonight) > P(large winds tonight)
- Spell Correction: Thek office is ten minutes from here: P(The office) > P(Then office)
- Speech Recognition: P(I saw a van) > P(eye awe of an)
- Summarization, question-answering
- goal is to compute the probability of a sentence or sequence of words W (=w1,w2,w3..)
- probability of upcoming word? P(w4|w1,w2,w3)

Chain Rule of Probability

- decompose
- P(the man from jupiter) = P(the)* P(man|the) * P(from|the man) * P(jupiter|the man from)

In [1]:
```
import nltk
import random
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict
```

In [11]:
```python
#Download the reuters corpus
#nltk.download('reuters')

#Tokenize the reuters corupus
tokens = reuters.words()

#Create bigrams and trigrams from tokenized text
b_tokens = list(bigrams(tokens))
t_tokens = list(trigrams(tokens))

#Create frequency distributions of bigrams and trigrams
b_freq = nltk.FreqDist(b_tokens)
t_freq = nltk.FreqDist(t_tokens)

#Create a dictionary to store trigram model
t_model = defaultdict(lambda: defaultdict(lambda:0))

#Build Trigram model
for w1, w2, w3 in t_tokens:
    t_model[ (w1, w2)][w3] += 1

# Function to generate text using the trigram model
def generate_text(seed_word,num_words):
    text = [seed_word [0], seed_word[1]]
    for i in range (num_words):
        next_word = random.choice(list(t_model[(text[-2], text[-1])].keys()))
        text.append (next_word)
    return ' '.join(text)

# Function to evaluate the model's ability to generate contextually relevant sentences
def evaluate_model(seed_word, num_sentences, num_words_per_sentence):
    print (f"Generating {num_sentences} sentences of {num_words_per_sentence} words each: ")
    for i in range(num_sentences):
        generated_sentence = generate_text(seed_word, num_words_per_sentence)
        print (f"Sentence{i+1}: {generated_sentence}")

# Example usage to generate contextually relevant sentences
seed_word = ("today", "on") # Initial seed word pair
num_sentences = 5
num_words_per_sentence = 10
evaluate_model (seed_word, num_sentences, num_words_per_sentence)
```

```
Generating 5 sentences of 10 words each:
Sentence1: today on its way out . Currently a farmer to forego antidumping
Sentence2: today on natural gas rate four pct increase to around 300 stg
Sentence3: today on natural rubber are among the 80 lodged " because a
Sentence4: today on Austria ' s Calona Wines Ltd , Algemene Bank Nederland
Sentence5: today on weather - related and the revival in short sterling interest
```

Using LSTM

In [13]:
```
import numpy as np
import random
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

text = """The quick brown fox jumps over the lazy dog.
A quick brown dog outpaces a quick fox.
The dog and the fox like to run in the forest.
However, the lazy dog prefers to sleep all day."""

tokenizer = Tokenizer()
tokenizer. fit_on_texts ([text])
total_words = len(tokenizer.word_index) + 1

# Create input sequences
input_sequences = []
for line in text.split ('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i  in range(1,len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

#Pad Sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences,maxlen=max_sequence_len,padding = 'pre'))

#Create predictors and labels
x,y= input_sequences[:,:-1],input_sequences[:,-1]
y = tf.keras.utils.to_categorical(y,num_classes=total_words)

#Define the LSTM Model
model = Sequential()
model.add(Embedding(total_words,100))
model.add(LSTM(100))
model.add(Dense(total_words,activation='softmax'))

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

history= model.fit(x,y,epochs=100,verbose=1)

def generate_text(seed_text,next_words,model,max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list],maxlen=max_sequence_len-1,padding='pre')
        predicted = model.predict(token_list,verbose=0)
        predicted_index = np.argmax(predicted,axis = -1)
        output_word = ""
        for word,index in tokenizer.word_index.items():
            if index == predicted_index:
                output_word=word
                break
        seed_text += " "+output_word
    return seed_text

seed_text = 'quick brown fox'
next_words=5
generated_text = generate_text(seed_text,next_words,model,max_sequence_len)
print("Generated Text: ",generated_text)
```

```
Epoch 1/100
2/2 [==============================] - 2s 14ms/step - loss: 3.0939 - accuracy: 0.0606
Epoch 2/100
2/2 [==============================] - 0s 15ms/step - loss: 3.0813 - accuracy: 0.1515
Epoch 3/100
2/2 [==============================] - 0s 14ms/step - loss: 3.0730 - accuracy: 0.1515
Epoch 4/100
2/2 [==============================] - 0s 16ms/step - loss: 3.0650 - accuracy: 0.2121
Epoch 5/100
2/2 [==============================] - 0s 1ms/step - loss: 3.0571 - accuracy: 0.2121
Epoch 6/100
2/2 [==============================] - 0s 0s/step - loss: 3.0490 - accuracy: 0.1818
Epoch 7/100
2/2 [==============================] - 0s 0s/step - loss: 3.0401 - accuracy: 0.2424
Epoch 8/100
2/2 [==============================] - 0s 17ms/step - loss: 3.0299 - accuracy: 0.2424
Epoch 9/100
2/2 [==============================] - 0s 1ms/step - loss: 3.0189 - accuracy: 0.2424
Epoch 10/100
2/2 [==============================] - 0s 14ms/step - loss: 3.0057 - accuracy: 0.3030
Epoch 11/100
2/2 [==============================] - 0s 894us/step - loss: 2.9888 - accuracy: 0.2727
Epoch 12/100
2/2 [==============================] - 0s 2ms/step - loss: 2.9693 - accuracy: 0.1515
Epoch 13/100
2/2 [==============================] - 0s 16ms/step - loss: 2.9460 - accuracy: 0.1515
Epoch 14/100
2/2 [==============================] - 0s 16ms/step - loss: 2.9248 - accuracy: 0.0909
Epoch 15/100
2/2 [==============================] - 0s 17ms/step - loss: 2.9047 - accuracy: 0.0909
Epoch 16/100
2/2 [==============================] - 0s 15ms/step - loss: 2.8753 - accuracy: 0.0909
Epoch 17/100
2/2 [==============================] - 0s 13ms/step - loss: 2.8413 - accuracy: 0.0909
Epoch 18/100
2/2 [==============================] - 0s 17ms/step - loss: 2.8082 - accuracy: 0.0909
Epoch 19/100
2/2 [==============================] - 0s 15ms/step - loss: 2.7964 - accuracy: 0.0909
Epoch 20/100
2/2 [==============================] - 0s 1ms/step - loss: 2.7959 - accuracy: 0.0909
Epoch 21/100
2/2 [==============================] - 0s 4ms/step - loss: 2.8122 - accuracy: 0.0909
Epoch 22/100
2/2 [==============================] - 0s 15ms/step - loss: 2.8356 - accuracy: 0.0909
Epoch 23/100
2/2 [==============================] - 0s 16ms/step - loss: 2.8288 - accuracy: 0.0909
Epoch 24/100
2/2 [==============================] - 0s 14ms/step - loss: 2.8159 - accuracy: 0.0909
Epoch 25/100
2/2 [==============================] - 0s 1ms/step - loss: 2.8068 - accuracy: 0.0909
Epoch 26/100
2/2 [==============================] - 0s 17ms/step - loss: 2.7824 - accuracy: 0.0909
Epoch 27/100
2/2 [==============================] - 0s 2ms/step - loss: 2.7612 - accuracy: 0.0909
Epoch 28/100
2/2 [==============================] - 0s 16ms/step - loss: 2.7407 - accuracy: 0.1212
Epoch 29/100
2/2 [==============================] - 0s 5ms/step - loss: 2.7249 - accuracy: 0.1515
Epoch 30/100
2/2 [==============================] - 0s 14ms/step - loss: 2.7162 - accuracy: 0.1515
Epoch 31/100
2/2 [==============================] - 0s 692us/step - loss: 2.7117 - accuracy: 0.1818
Epoch 32/100
2/2 [==============================] - 0s 16ms/step - loss: 2.7030 - accuracy: 0.1818
Epoch 33/100
2/2 [==============================] - 0s 0s/step - loss: 2.6906 - accuracy: 0.1818
Epoch 34/100
2/2 [==============================] - 0s 3ms/step - loss: 2.6863 - accuracy: 0.2727
Epoch 35/100
2/2 [==============================] - 0s 7ms/step - loss: 2.6940 - accuracy: 0.3939
Epoch 36/100
2/2 [==============================] - 0s 16ms/step - loss: 2.6976 - accuracy: 0.3636
Epoch 37/100
2/2 [==============================] - 0s 1ms/step - loss: 2.6978 - accuracy: 0.3333
Epoch 38/100
2/2 [==============================] - 0s 14ms/step - loss: 2.6920 - accuracy: 0.3030
Epoch 39/100
2/2 [==============================] - 0s 19ms/step - loss: 2.6789 - accuracy: 0.3030
Epoch 40/100
2/2 [==============================] - 0s 2ms/step - loss: 2.6614 - accuracy: 0.3030
Epoch 41/100
2/2 [==============================] - 0s 14ms/step - loss: 2.6445 - accuracy: 0.3030
Epoch 42/100
2/2 [==============================] - 0s 6ms/step - loss: 2.6306 - accuracy: 0.3030
Epoch 43/100
2/2 [==============================] - 0s 13ms/step - loss: 2.6199 - accuracy: 0.3030
Epoch 44/100
2/2 [==============================] - 0s 5ms/step - loss: 2.6108 - accuracy: 0.2727
Epoch 45/100
2/2 [==============================] - 0s 14ms/step - loss: 2.6021 - accuracy: 0.2424
Epoch 46/100
2/2 [==============================] - 0s 4ms/step - loss: 2.5929 - accuracy: 0.2121
Epoch 47/100
2/2 [==============================] - 0s 16ms/step - loss: 2.5848 - accuracy: 0.2121
Epoch 48/100
2/2 [==============================] - 0s 911us/step - loss: 2.5753 - accuracy: 0.2424
Epoch 49/100
2/2 [==============================] - 0s 16ms/step - loss: 2.5647 - accuracy: 0.2424
Epoch 50/100
2/2 [==============================] - 0s 11ms/step - loss: 2.5546 - accuracy: 0.2727
Epoch 51/100
2/2 [==============================] - 0s 0s/step - loss: 2.5457 - accuracy: 0.3030
```

```
Epoch 52/100
2/2 [==============================] - 0s 17ms/step - loss: 2.5447 - accuracy: 0.3333
Epoch 53/100
2/2 [==============================] - 0s 6ms/step - loss: 2.5403 - accuracy: 0.2727
Epoch 54/100
2/2 [==============================] - 0s 8ms/step - loss: 2.5317 - accuracy: 0.2727
Epoch 55/100
2/2 [==============================] - 0s 15ms/step - loss: 2.5260 - accuracy: 0.2424
Epoch 56/100
2/2 [==============================] - 0s 10ms/step - loss: 2.5180 - accuracy: 0.2424
Epoch 57/100
2/2 [==============================] - 0s 16ms/step - loss: 2.5091 - accuracy: 0.1818
Epoch 58/100
2/2 [==============================] - 0s 17ms/step - loss: 2.5010 - accuracy: 0.1818
Epoch 59/100
2/2 [==============================] - 0s 16ms/step - loss: 2.4936 - accuracy: 0.1818
Epoch 60/100
2/2 [==============================] - 0s 0s/step - loss: 2.4818 - accuracy: 0.1818
Epoch 61/100
2/2 [==============================] - 0s 9ms/step - loss: 2.4637 - accuracy: 0.1818
Epoch 62/100
2/2 [==============================] - 0s 0s/step - loss: 2.4475 - accuracy: 0.1515
Epoch 63/100
2/2 [==============================] - 0s 21ms/step - loss: 2.4265 - accuracy: 0.2424
Epoch 64/100
2/2 [==============================] - 0s 6ms/step - loss: 2.4014 - accuracy: 0.3636
Epoch 65/100
2/2 [==============================] - 0s 13ms/step - loss: 2.3820 - accuracy: 0.2727
Epoch 66/100
2/2 [==============================] - 0s 18ms/step - loss: 2.3708 - accuracy: 0.3030
Epoch 67/100
2/2 [==============================] - 0s 15ms/step - loss: 2.3643 - accuracy: 0.3333
Epoch 68/100
2/2 [==============================] - 0s 0s/step - loss: 2.3586 - accuracy: 0.3333
Epoch 69/100
2/2 [==============================] - 0s 0s/step - loss: 2.3554 - accuracy: 0.3030
Epoch 70/100
2/2 [==============================] - 0s 0s/step - loss: 2.3493 - accuracy: 0.2424
Epoch 71/100
2/2 [==============================] - 0s 16ms/step - loss: 2.3393 - accuracy: 0.2424
Epoch 72/100
2/2 [==============================] - 0s 16ms/step - loss: 2.3240 - accuracy: 0.2424
Epoch 73/100
2/2 [==============================] - 0s 0s/step - loss: 2.3017 - accuracy: 0.2121
Epoch 74/100
2/2 [==============================] - 0s 15ms/step - loss: 2.2770 - accuracy: 0.2121
Epoch 75/100
2/2 [==============================] - 0s 17ms/step - loss: 2.2563 - accuracy: 0.2121
Epoch 76/100
2/2 [==============================] - 0s 16ms/step - loss: 2.2356 - accuracy: 0.2121
Epoch 77/100
2/2 [==============================] - 0s 0s/step - loss: 2.2161 - accuracy: 0.2121
Epoch 78/100
2/2 [==============================] - 0s 0s/step - loss: 2.2034 - accuracy: 0.2121
Epoch 79/100
2/2 [==============================] - 0s 12ms/step - loss: 2.1893 - accuracy: 0.2424
Epoch 80/100
2/2 [==============================] - 0s 0s/step - loss: 2.1710 - accuracy: 0.2424
Epoch 81/100
2/2 [==============================] - 0s 17ms/step - loss: 2.1557 - accuracy: 0.2727
Epoch 82/100
2/2 [==============================] - 0s 17ms/step - loss: 2.1333 - accuracy: 0.2727
Epoch 83/100
2/2 [==============================] - 0s 1ms/step - loss: 2.1096 - accuracy: 0.2727
Epoch 84/100
2/2 [==============================] - 0s 17ms/step - loss: 2.0878 - accuracy: 0.2424
Epoch 85/100
2/2 [==============================] - 0s 0s/step - loss: 2.0617 - accuracy: 0.2424
Epoch 86/100
2/2 [==============================] - 0s 6ms/step - loss: 2.0292 - accuracy: 0.2727
Epoch 87/100
2/2 [==============================] - 0s 17ms/step - loss: 2.0053 - accuracy: 0.2727
Epoch 88/100
2/2 [==============================] - 0s 0s/step - loss: 1.9804 - accuracy: 0.2727
Epoch 89/100
2/2 [==============================] - 0s 17ms/step - loss: 1.9619 - accuracy: 0.2727
Epoch 90/100
2/2 [==============================] - 0s 13ms/step - loss: 1.9479 - accuracy: 0.2727
Epoch 91/100
2/2 [==============================] - 0s 0s/step - loss: 1.9409 - accuracy: 0.3333
Epoch 92/100
2/2 [==============================] - 0s 0s/step - loss: 1.9375 - accuracy: 0.3333
Epoch 93/100
2/2 [==============================] - 0s 0s/step - loss: 1.9221 - accuracy: 0.3030
Epoch 94/100
2/2 [==============================] - 0s 2ms/step - loss: 1.9032 - accuracy: 0.3333
Epoch 95/100
2/2 [==============================] - 0s 17ms/step - loss: 1.8805 - accuracy: 0.3333
Epoch 96/100
2/2 [==============================] - 0s 16ms/step - loss: 1.8631 - accuracy: 0.3636
Epoch 97/100
2/2 [==============================] - 0s 16ms/step - loss: 1.8422 - accuracy: 0.3939
Epoch 98/100
2/2 [==============================] - 0s 16ms/step - loss: 1.8187 - accuracy: 0.4242
Epoch 99/100
2/2 [==============================] - 0s 0s/step - loss: 1.7923 - accuracy: 0.3939
Epoch 100/100
2/2 [==============================] - 0s 14ms/step - loss: 1.7620 - accuracy: 0.3939
Generated Text:  quick brown fox the dog dog to to
```