```
try:
  import google.colab
  IN_COLAB = True
except:
  IN_COLAB = False

# keras module for building LSTM
from keras.layers import Embedding, LSTM, Dense, Dropout, Flatten, Bidirectional
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
import keras.utils as ku
import keras
import tensorflow as tf
from tensorflow.keras.utils import Sequence

if IN_COLAB:
  !pip install Keras-Preprocessing
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import text_to_word_sequence

import pandas as pd
import numpy as np
import string, os
import matplotlib.pyplot as plt
import seaborn as sns
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting Keras-Preprocessing
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 5.4 MB/s eta 0:00:00
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.9/dist-packages (from Keras-Preprocessing) (1.16.0)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.9/dist-packages (from Keras-Preprocessing) (1.22.4)
Installing collected packages: Keras-Preprocessing
Successfully installed Keras-Preprocessing-1.1.2
```

```
# Helper to plot loss
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.legend()
  plt.grid(True)
  plt.show()

def plot_acc(history):
  plt.plot(history.history['acc'], label='accuracy')
  plt.legend()
  plt.grid(True)
  plt.show()
```

Text Generation with LSTM

As we have seen, LSTM models are excellent at dealing with sequential data. As luck would have it, text is also sequental data! We can train a model to predict the next word in a sentence, then use that smarts to generate new text. Basically ChatGPT, but far better.

Text for Training

We need some text from which to train our model to speak, I captured a small extract of text from Reddit posts, which vaguely resembles actual language. We'll first need to clean up our data a bit before we can assemble it for modelling. The inital cleaning bits are just like what we used in NLP, we just need to get rid of all the junk.

We can use pretty much anything that you can imagine as source, and assuming we can gather enough data and train our model, the generated speech will be styled after the source. I liken it to going on vacation in Indonesia and talking to Indonesians who spoke English like Australian surfer bros - their training data was a little weird, so the output was a little weird too. If you're looking to build your best ChatGPT competitor you will want a lot of data, specifically a lot of data that is representative of the full gamut of how you want your model to write. If you want slang in the new text, you can't really train on Shakespeare and Wikipedia.

```
# Reddit WSB data
train_text_file = keras.utils.get_file('train_text.txt', 'https://jrssbcrsefilesnait.blob.core.windows.net/3950data1/reddit_wsb.csv')
USE_COL = "body"

# Complaint data
#train_text_file = keras.utils.get_file('train_text3.txt', 'https://jrssbcrsefilesnait.blob.core.windows.net/3950data1/complaints_processed_clean.csv')
#USE_COL = "narrative"

train_text = pd.read_csv(train_text_file)
train_text.sample(10)
```

```
Downloading data from https://jrssbcrsefilesnait.blob.core.windows.net/3950data1/reddit_wsb.csv
43732027/43732027 [==============================] - 9s 0us/step
```

	title	score	id	url	comms_num	created	body	timestamp
15581	What to buy? To broke for GME	35	l75d2o	https://www.reddit.com/r/wallstreetbets/commen...	180	1.611889e+09	So I'm to broke for GME stocks, which other sh...	2021-01-29 05:01:10
11785	Foreshadowing?	1	l71ug5	https://i.redd.it/xfm4itasq3e61.jpg	0	1.611881e+09	NaN	2021-01-29 02:44:14
6139	So when the hedge funds lose a few of	29	l6zo32	https://i.redd.it/y6cvf6m4d3e61.jpg	8	1.611876e+09	NaN	2021-01-29

✓ 3s completed at 1:44 PM

46123	We've been compromised!	33930	n69ajb	https://v.redd.it/5zfzinkrlix61	1788	1.620342e+09	NaN	2021-05-07 02:01:47
431	GNUS still able to trade on platforms	0	l6x0kl	https://www.reddit.com/r/wallstreetbets/commen...	0	1.611871e+09	I hope someday this same kind of turned tables...	2021-01-28 23:48:33
9136	So is this the time to start boycotting tradin...	1	l70tgn	https://www.reddit.com/r/wallstreetbets/commen...	8	1.611879e+09	Like many I'm infuriated at the fact that I ca...	2021-01-29 02:09:13
53020	A Case for $ZIP	9	oxzeet	https://www.reddit.com/r/wallstreetbets/common...	11	1.628103e+09	ZipRecruiter ($ZIP) isn't the sexiest company...	2021-08-04 21:46:25

```
TOKENS = 500
OUTPUT_LENGTH = 45
OUT_DIM = 8
BATCH_SIZE = 128
SAMP = 7
SHUFFLE = 500
UNITS = 50

#Some larger values for Colab
TEST_EPOCHS = 5
TEST_BATCH = 512

if IN_COLAB:
    TEST_EPOCHS = 200
    TEST_BATCH = 1024
    !mkdir weights
```

Clean and Build Sequences

We can clean and process the text data with any strategy. Here we'll use a simple strategy of removing punctuation and converting all text to lowercase.

Next we'll use a Keras utility to convert our text into a sequence of words, which we'll then tokenize into integers. The sequence that is created is pretty simple, just a list of all the words in the sentence in the correct order. Basically an OCD version of a normal sentence.

```
raw_text = train_text[USE_COL]
## Remove punctuation
raw_text = raw_text.dropna()
raw_text = raw_text.apply(lambda x: x.replace('[{}]'.format(string.punctuation), '').lower())
vocab = set()
sentences = []
for sentence in raw_text:
  current_sentence = text_to_word_sequence(sentence)
  sentences.append(current_sentence)
  vocab.update(current_sentence)
#vocab
#sentences
max_length = max([len(sentence) for sentence in sentences])
sentences[SAMP]
```

```
['this',
 'is',
 'our',
 'time',
 'if',
 'anyone',
 'will',
 'listen',
 'to',
 'you',
 'please',
 'explain',
 'to',
 'them',
 'why',
 'gme',
 'is',
 'one',
 'of',
 'the',
 'very',
 'best',
 'investments',
 'they',
 'can',
 'make',
 'right',
 'now',
 'i',
 'have',
 'already',
 'gotten',
 '5',
 'people',
 'close',
 'to',
 'me',
 'to',
 'research',
 'what',
 'is',
 'happening',
 'and',
```

```
 'they',
 'decided',
 'to',
 'dump',
 'everything',
 'into',
 'gme',
 'not',
 'amc',
 'not',
 'nok',
 'not',
 'bb',
 'but',
 'gme',
```

Tokenize

We can take the text that we split above and encode it as a sequence of integers. We'll then use the tokenizer to convert our sequences into a sequence of integers.

```
tokenizer = Tokenizer(num_words=TOKENS)
tokenizer.fit_on_texts(sentences)

tokenized_sentences = tokenizer.texts_to_sequences(sentences)
tokenized_sentences[SAMP]
```

```
[9,
 6,
 131,
 78,
 28,
 380,
 25,
 2,
 14,
 305,
 2,
 76,
 128,
 43,
 6,
 73,
 5,
 1,
 165,
 138,
 17,
 40,
 99,
 145,
 57,
 8,
 20,
 229,
 112,
 69,
 317,
 2,
 89,
 2,
 452,
 44,
 6,
 3,
 17,
 2,
 421,
 96,
 43,
 23,
 175,
 23,
 23,
 374,
 24,
 43,
 136,
 126,
 8,
 135,
 23,
 228,
 308,
 22,
```

Construct Training Sequences

We are going to be a little slack in the construction of the datasets for training because we are limited in the amount of resources we can handle. All of the datasets that are fed to the model need to be the same length, so we'll set a cap and trucate it here for resource concerns. Our dataset will be constructed as:

- A sequence of (up to) 24 words as the X data.
- The next word as the Y data.

So each sequence is effectively one set of features, and its target is the next word. If we were doing this in reality, we'd want to prep more records from our sample:

- Suppose a sample sentence is "The quick brown fox jumps over the lazy dog". Our ideal data would have something like:
    - X = "the quick brown", Y = "fox"
    - X = "quick brown fox", Y = "jumps"
    - X = "brown fox jumps", Y = "over"

This is superior both because we are generating much more data to train the model and because we are training the model to predict words in all different positions in the sentence. We'd be predicting almost every word in the training dataset. The words that frequently end a sentence are not necessarily the same as the words that start a sentence or sit in the middle, so making predictions up and down the text will likely lead to a more useful model.

We'd end up with a better model if we generated more sequences from our data and/or added more data. The resource demands make that tough, so we have cut some corners that are easy to remedy in a real-world scenario.

```
# Find longest sequence
max_real_length = max([len(sentence) for sentence in tokenized_sentences])
print("Longest real sequence:", max_real_length)
if max_real_length > OUTPUT_LENGTH:
  extra_sequences = [s_[OUTPUT_LENGTH:] for s_ in tokenized_sentences]
  trunc_token_sequences = [t_[:OUTPUT_LENGTH] for t_ in tokenized_sentences]
trunc_token_sequences = trunc_token_sequences + extra_sequences
trunc_token_sequences[SAMP]
```

```
Longest real sequence: 4572
[9,
 6,
 131,
 78,
 28,
 380,
 25,
 2,
 14,
 305,
 2,
 76,
 128,
 43,
 6,
 73,
 5,
 1,
 165,
 138,
 17,
 40,
 99,
 145,
 57,
 8,
 20,
 229,
 112,
 69,
 317,
 2,
 89,
 2,
 452,
 44,
 6,
 3,
 17,
 2,
 421,
 96,
 43,
 23,
 175]
```

Pad Sequences

We need to pad our sequences so that they are all the same length, as our neural networks require that. The pad_sequences utility does just that, it will fill 0s at either the beginning or end of the sequence, depending on the "padding" option, and make everything the same length. We want to pad before the real data, because we are always planning on predicting the next token, so we want to work from the last value.

```
padded_trunc_token_sequences = pad_sequences(trunc_token_sequences, maxlen=OUTPUT_LENGTH, padding='pre')
padded_trunc_token_sequences[SAMP]
```

```
array([  9,   6, 131,  78,  28, 380,  25,   2,  14, 305,   2,  76, 128,
        43,   6,  73,   5,   1, 165, 138,  17,  40,  99, 145,  57,   8,
        20, 229, 112,  69, 317,   2,  89,   2, 452,  44,   6,   3,  17,
         2, 421,  96,  43,  23, 175], dtype=int32)
```

Write Tokenized Data to Disk

We can write the tokenized data to disk so that we can use it later. This will save us from having to redo that step that is slow. Since we chopped our data size down a bit, this isn't super needed. I tried to generate all of the sequences noted above and I both exceeded the Colab memory limits and it took a while to run. I wouldn't want to repeat that if I can avoid it, and hard drive space is cheap, so writing the interim results to disk is a good work around. In real scenarios when we had massive amounts of data we would need to load it incrementally from disk

anyway, so this is a free win.

```
#token_path = 'data/sample_tokenized_sentences.csv'
token_path = 'data/padded_sample_tokenized_sentences.csv'
if IN_COLAB:
  !mkdir data

if os.path.exists(token_path):
    df_prepped = pd.read_csv(token_path, header=None)
    prepped_sentences = np.array(df_prepped.values.tolist())
else:
    df_prepped = pd.DataFrame(padded_trunc_token_sequences)
    df_prepped.to_csv(token_path, header=False, index=False)
```

Dataset and Array

This data can be reasonably used either as a regular array or or a tensorflow dataset. We'll use both here, just to show that it can be done either way. If we we using the full set of sequences the data would be quite a bit larger with all the fully padded out sequences, we'd probably need to write the data to disk and load it with a dataset or generator.

```
print(padded_trunc_token_sequences.shape)
y_t = padded_trunc_token_sequences[:, -1].reshape(-1, 1)
y_t = ku.to_categorical(y_t, num_classes=TOKENS)
print("Y shape:", y_t.shape)
X_t = padded_trunc_token_sequences[:, :-1]
print("X shape:", X_t.shape)
```

```
    (49476, 45)
    Y shape: (49476, 500)
    X shape: (49476, 44)
```

Model

Now we model. The data that we made mirrors the construction of a sentence.

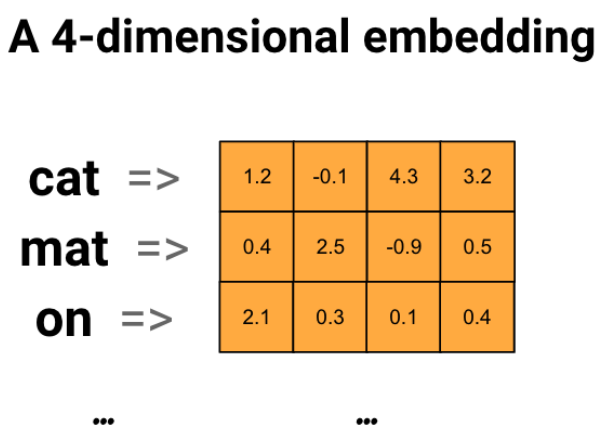
- X features - the sentence up to this point.
- Y target - the word(s) that should come next.

So, the model is effectively working to generate text just like a time series model works to predict the next value in a sequence of stock prices or hourly temperature. We train the model on, hopefully a large number of senteneces, where is sees many examples of "here are some words" (X values) and "here is the next word" (Y value). If we give it lots and lots of that training data, it should become better and better at determining what should come next, given the existing sentence.

To do this well, we'd need a lot more data than we have, and much more time to train. We'd want to give the model enough data so that it can see lots and lots of examples of the same word in different contexts, and of similar contexts with different words. The patterns of language are really complex, so we need data that provides enough variation to demonstrate the patterns. **The actual performance of our model at creating sensible text will be bad, we should not get our hopes up, but the process is sound.**

Embedding Layer

We also use an embedding layer here, which accepts our integer inputs and converts them to a vector of a specified size. This is a way of representing the words in a way that is more useful for the model. We saw embeddings with word2vec during the NLP portions. Just as it did then, embedding will represent each of our tokens as a vector of a specified size, or as a value in N dimensional space.



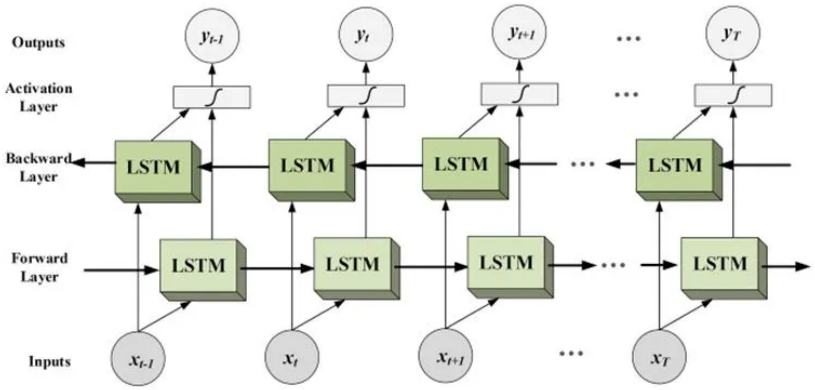
When we check the summary of the model we can see that the embedding layer has a lot of parameters, this is because it is learning the vector representation of each of the words in our vocabulary. When we used word2vec we used a pretrained model - the N dimensional space was already defined, and we placed our words into it. Here, we are letting the model learn the space and the vectors that represent the words, so the vector representation of each word is learned directly from the data. If we are dealing with a scenarion where we have a lot of data and a specialized vocabulary (such as in industry) this can be very useful. The model will learn which words are similar and which are not, based on the context of the text provided in training.

the context of the text provided in training.

If we consider embedding from the perspective of the actual prediction, it also makes a lot of sense that this is a prefered approach. We are not generally trying to predict the exact word, as we would with a classification that chooses the correct answer from a list of options. When dealing with language we are more likely trying to predict the next *type* of word - both if it is a noun/verb/etc and the meaning. Embedding the words allows us to use that multidimensional representation aim for words that are similar. When dealing with generative models with large vocabularies our model will follow "Do you want to go out sometime and grab a ____" with something like "beer" or "coffee" which are similar in this context, and close in their embedding representation. Using embeddings, the loss that is calculated each round is measuring the distance between the predicted word and the actual word in N dimensions, so grabbing words that are very close, or similar, is expected and desired.

Let's Go Bi

For this model we'll use a bidirectional LSTM layer. This is a layer that will process the input sequence in both directions, so it will see the sequence from the beginning and from the end. This is useful because it allows the model to consider what should come next, but also what should come before, which can be useful for determining the next word. The idea of a bidirectional layer is that if our data is in a sequence we can feed it through normally, but we can also reverse it. Doing this gives the model a differnet view on the data, and especially in cases like language creation, where the meaning of the output can depend on the entirety of the sequence.



Bidirectional layers are most prevalent in NLP, as adding the ability to look at the sequence of words in both directions can really help models build a better understanding of the context of the words. There is a speed penalty due to each layer doing roughly double the number of calculations, but for language tasks such as this, it is pretty likely to be worth it, particularly if we were to expand the dataset. Bidirectional layers can be added simply, as below; on the Keras documentation there are a few more details, mainly that the forward and backward layers can be configured separately then combined. When using a bidirectional layer, one option that we should commonly pay attention to is the "merge_mode" option, which determines how the forward and backward layers are combined to produce the final output. The default is "concat" which combines the outputs by concatenating them, but there are other options such as "sum" or "ave" which can be useful in some cases - hyperparameter tuning is the way to find the "best" option. We will use the default here, and can sidestep the other details of bidirectional layers for now.

```
model = Sequential()
model.add(Embedding(input_dim=TOKENS, output_dim=OUT_DIM, input_length=OUTPUT_LENGTH-1))
#model.add(LSTM(UNITS, return_sequences=True))
#model.add(LSTM(UNITS, return_sequences=True))
#model.add(LSTM(UNITS))
model.add(Bidirectional(LSTM(UNITS, return_sequences=True)))
model.add(Bidirectional(LSTM(UNITS, return_sequences=True)))
model.add(Bidirectional(LSTM(UNITS)))
model.add(Flatten())
model.add(Dense(TOKENS, activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['acc'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 44, 8)             4000

 bidirectional (Bidirectiona  (None, 44, 100)          23600
 l)

 bidirectional_1 (Bidirectio  (None, 44, 100)          60400
 nal)

 bidirectional_2 (Bidirectio  (None, 100)              60400
 nal)

 flatten (Flatten)           (None, 100)               0

 dense (Dense)               (None, 500)               50500

=================================================================
Total params: 198,900
Trainable params: 198,900
Non-trainable params: 0
_________________________________________________________________
None
```

In making this, lots of epochs definately gave better results, as did a swap to bidirectional layers. I saw this in both the raw accuracy scores and the relative sense of the generated text.

```
# Try with dataframes
early_stop = EarlyStopping(monitor='loss', patience=50)
save_weights = tf.keras.callbacks.ModelCheckpoint('weights/lstm_gen_weights.h5', save_best_only=True, monitor='loss', mode='min', save_weights_only=True)

history = model.fit(X_t, y_t, batch_size=TEST_BATCH, epochs=TEST_EPOCHS, verbose=1, callbacks=[early_stop, save_weights])
plot_acc(history)
plot_loss(history)
```

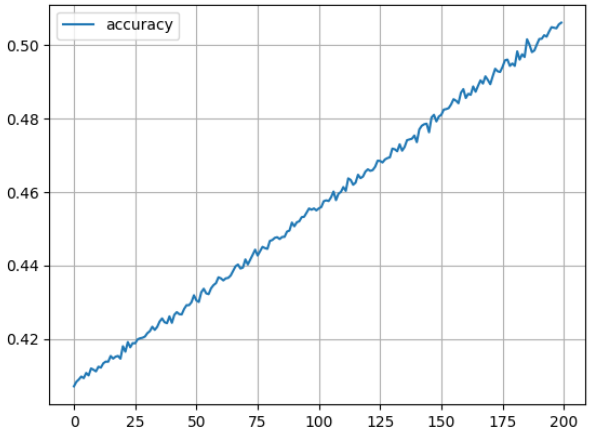
```
Epoch 1/200
49/49 [==============================] - 1s 22ms/step - loss: 2.8845 - acc: 0.4070
Epoch 2/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8788 - acc: 0.4083
Epoch 3/200
49/49 [==============================] - 1s 22ms/step - loss: 2.8761 - acc: 0.4089
Epoch 4/200
49/49 [==============================] - 1s 22ms/step - loss: 2.8685 - acc: 0.4097
Epoch 5/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8692 - acc: 0.4093
Epoch 6/200
49/49 [==============================] - 1s 22ms/step - loss: 2.8650 - acc: 0.4107
Epoch 7/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8590 - acc: 0.4100
Epoch 8/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8541 - acc: 0.4119
Epoch 9/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8575 - acc: 0.4116
Epoch 10/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8488 - acc: 0.4111
Epoch 11/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8452 - acc: 0.4124
Epoch 12/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8403 - acc: 0.4121
Epoch 13/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8386 - acc: 0.4133
Epoch 14/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8360 - acc: 0.4138
Epoch 15/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8277 - acc: 0.4137
Epoch 16/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8235 - acc: 0.4153
Epoch 17/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8263 - acc: 0.4146
Epoch 18/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8215 - acc: 0.4151
Epoch 19/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8156 - acc: 0.4153
Epoch 20/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8124 - acc: 0.4146
Epoch 21/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8071 - acc: 0.4180
Epoch 22/200
49/49 [==============================] - 1s 21ms/step - loss: 2.8084 - acc: 0.4165
Epoch 23/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7994 - acc: 0.4191
Epoch 24/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7973 - acc: 0.4177
Epoch 25/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7917 - acc: 0.4188
Epoch 26/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7913 - acc: 0.4187
Epoch 27/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7854 - acc: 0.4198
Epoch 28/200
49/49 [==============================] - 1s 20ms/step - loss: 2.7855 - acc: 0.4201
Epoch 29/200
49/49 [==============================] - 1s 22ms/step - loss: 2.7813 - acc: 0.4203
Epoch 30/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7787 - acc: 0.4206
Epoch 31/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7725 - acc: 0.4215
Epoch 32/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7690 - acc: 0.4221
Epoch 33/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7651 - acc: 0.4233
Epoch 34/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7604 - acc: 0.4224
Epoch 35/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7567 - acc: 0.4232
Epoch 36/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7533 - acc: 0.4247
Epoch 37/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7479 - acc: 0.4255
Epoch 38/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7470 - acc: 0.4245
Epoch 39/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7447 - acc: 0.4242
Epoch 40/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7423 - acc: 0.4261
Epoch 41/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7352 - acc: 0.4243
Epoch 42/200
```

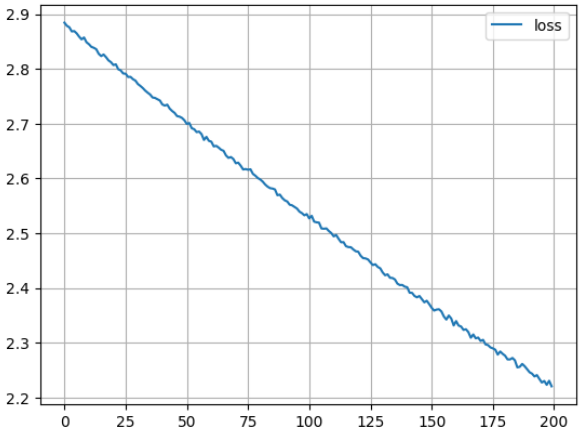
```
49/49 [==============================] - 1s 22ms/step - loss: 2.7331 - acc: 0.4265
Epoch 43/200
49/49 [==============================] - 1s 20ms/step - loss: 2.7348 - acc: 0.4272
Epoch 44/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7275 - acc: 0.4267
Epoch 45/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7231 - acc: 0.4266
Epoch 46/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7197 - acc: 0.4281
Epoch 47/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7139 - acc: 0.4291
Epoch 48/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7129 - acc: 0.4291
Epoch 49/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7106 - acc: 0.4299
Epoch 50/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7063 - acc: 0.4318
Epoch 51/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6995 - acc: 0.4304
Epoch 52/200
49/49 [==============================] - 1s 21ms/step - loss: 2.7014 - acc: 0.4300
Epoch 53/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6917 - acc: 0.4327
Epoch 54/200
49/49 [==============================] - 1s 22ms/step - loss: 2.6900 - acc: 0.4336
Epoch 55/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6847 - acc: 0.4324
Epoch 56/200
49/49 [==============================] - 1s 20ms/step - loss: 2.6856 - acc: 0.4321
Epoch 57/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6810 - acc: 0.4337
Epoch 58/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6704 - acc: 0.4346
Epoch 59/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6759 - acc: 0.4351
Epoch 60/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6687 - acc: 0.4367
Epoch 61/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6672 - acc: 0.4364
Epoch 62/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6586 - acc: 0.4359
Epoch 63/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6596 - acc: 0.4364
Epoch 64/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6564 - acc: 0.4366
Epoch 65/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6524 - acc: 0.4372
Epoch 66/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6505 - acc: 0.4385
Epoch 67/200
49/49 [==============================] - 1s 22ms/step - loss: 2.6420 - acc: 0.4397
Epoch 68/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6376 - acc: 0.4402
Epoch 69/200
49/49 [==============================] - 1s 20ms/step - loss: 2.6391 - acc: 0.4391
Epoch 70/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6355 - acc: 0.4394
Epoch 71/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6277 - acc: 0.4416
Epoch 72/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6289 - acc: 0.4402
Epoch 73/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6233 - acc: 0.4415
Epoch 74/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6164 - acc: 0.4429
Epoch 75/200
49/49 [==============================] - 1s 20ms/step - loss: 2.6172 - acc: 0.4442
Epoch 76/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6159 - acc: 0.4426
Epoch 77/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6167 - acc: 0.4438
Epoch 78/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6083 - acc: 0.4450
Epoch 79/200
49/49 [==============================] - 1s 21ms/step - loss: 2.6050 - acc: 0.4447
Epoch 80/200
49/49 [==============================] - 1s 22ms/step - loss: 2.6007 - acc: 0.4445
Epoch 81/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5981 - acc: 0.4466
Epoch 82/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5942 - acc: 0.4469
Epoch 83/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5890 - acc: 0.4474
Epoch 84/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5850 - acc: 0.4476
Epoch 85/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5823 - acc: 0.4471
Epoch 86/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5816 - acc: 0.4477
Epoch 87/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5800 - acc: 0.4477
Epoch 88/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5694 - acc: 0.4492
Epoch 89/200
49/49 [==============================] - 1s 20ms/step - loss: 2.5705 - acc: 0.4494
Epoch 90/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5642 - acc: 0.4516
```

```
Epoch 91/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5601 - acc: 0.4506
Epoch 92/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5580 - acc: 0.4517
Epoch 93/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5521 - acc: 0.4520
Epoch 94/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5509 - acc: 0.4531
Epoch 95/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5480 - acc: 0.4532
Epoch 96/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5450 - acc: 0.4543
Epoch 97/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5394 - acc: 0.4555
Epoch 98/200
49/49 [==============================] - 1s 22ms/step - loss: 2.5367 - acc: 0.4551
Epoch 99/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5327 - acc: 0.4555
Epoch 100/200
49/49 [==============================] - 1s 20ms/step - loss: 2.5349 - acc: 0.4549
Epoch 101/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5270 - acc: 0.4555
Epoch 102/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5317 - acc: 0.4558
Epoch 103/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5211 - acc: 0.4574
Epoch 104/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5199 - acc: 0.4577
Epoch 105/200
49/49 [==============================] - 1s 22ms/step - loss: 2.5193 - acc: 0.4575
Epoch 106/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5086 - acc: 0.4585
Epoch 107/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5082 - acc: 0.4601
Epoch 108/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5088 - acc: 0.4577
Epoch 109/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5041 - acc: 0.4595
Epoch 110/200
49/49 [==============================] - 1s 21ms/step - loss: 2.5003 - acc: 0.4599
Epoch 111/200
49/49 [==============================] - 1s 22ms/step - loss: 2.4943 - acc: 0.4613
Epoch 112/200
49/49 [==============================] - 1s 20ms/step - loss: 2.4968 - acc: 0.4602
Epoch 113/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4897 - acc: 0.4636
Epoch 114/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4835 - acc: 0.4632
Epoch 115/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4837 - acc: 0.4619
Epoch 116/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4763 - acc: 0.4625
Epoch 117/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4747 - acc: 0.4646
Epoch 118/200
49/49 [==============================] - 1s 22ms/step - loss: 2.4745 - acc: 0.4637
Epoch 119/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4703 - acc: 0.4641
Epoch 120/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4670 - acc: 0.4655
Epoch 121/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4663 - acc: 0.4661
Epoch 122/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4589 - acc: 0.4657
Epoch 123/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4545 - acc: 0.4659
Epoch 124/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4540 - acc: 0.4668
Epoch 125/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4521 - acc: 0.4684
Epoch 126/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4467 - acc: 0.4684
Epoch 127/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4419 - acc: 0.4680
Epoch 128/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4433 - acc: 0.4688
Epoch 129/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4384 - acc: 0.4692
Epoch 130/200
49/49 [==============================] - 1s 22ms/step - loss: 2.4359 - acc: 0.4694
Epoch 131/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4283 - acc: 0.4717
Epoch 132/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4233 - acc: 0.4716
Epoch 133/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4249 - acc: 0.4710
Epoch 134/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4186 - acc: 0.4729
Epoch 135/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4182 - acc: 0.4712
Epoch 136/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4155 - acc: 0.4722
Epoch 137/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4080 - acc: 0.4740
Epoch 138/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4053 - acc: 0.4743
Epoch 139/200
```

```
49/49 [==============================] - 1s 20ms/step - loss: 2.4056 - acc: 0.4744
Epoch 140/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4024 - acc: 0.4753
Epoch 141/200
49/49 [==============================] - 1s 21ms/step - loss: 2.4014 - acc: 0.4735
Epoch 142/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3914 - acc: 0.4769
Epoch 143/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3911 - acc: 0.4779
Epoch 144/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3854 - acc: 0.4784
Epoch 145/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3829 - acc: 0.4785
Epoch 146/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3854 - acc: 0.4762
Epoch 147/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3796 - acc: 0.4802
Epoch 148/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3738 - acc: 0.4809
Epoch 149/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3766 - acc: 0.4791
Epoch 150/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3706 - acc: 0.4805
Epoch 151/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3639 - acc: 0.4809
Epoch 152/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3587 - acc: 0.4824
Epoch 153/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3606 - acc: 0.4825
Epoch 154/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3611 - acc: 0.4828
Epoch 155/200
49/49 [==============================] - 1s 24ms/step - loss: 2.3571 - acc: 0.4838
Epoch 156/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3479 - acc: 0.4852
Epoch 157/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3423 - acc: 0.4848
Epoch 158/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3497 - acc: 0.4841
Epoch 159/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3443 - acc: 0.4869
Epoch 160/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3316 - acc: 0.4879
Epoch 161/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3395 - acc: 0.4855
Epoch 162/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3314 - acc: 0.4867
Epoch 163/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3297 - acc: 0.4864
Epoch 164/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3235 - acc: 0.4887
Epoch 165/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3246 - acc: 0.4872
Epoch 166/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3194 - acc: 0.4888
Epoch 167/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3091 - acc: 0.4903
Epoch 168/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3151 - acc: 0.4894
Epoch 169/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3080 - acc: 0.4914
Epoch 170/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3095 - acc: 0.4904
Epoch 171/200
49/49 [==============================] - 1s 21ms/step - loss: 2.3032 - acc: 0.4893
Epoch 172/200
49/49 [==============================] - 1s 20ms/step - loss: 2.3053 - acc: 0.4915
Epoch 173/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2970 - acc: 0.4935
Epoch 174/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2958 - acc: 0.4928
Epoch 175/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2915 - acc: 0.4926
Epoch 176/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2897 - acc: 0.4940
Epoch 177/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2873 - acc: 0.4958
Epoch 178/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2784 - acc: 0.4960
Epoch 179/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2840 - acc: 0.4943
Epoch 180/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2797 - acc: 0.4949
Epoch 181/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2763 - acc: 0.4943
Epoch 182/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2696 - acc: 0.4982
Epoch 183/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2695 - acc: 0.4960
Epoch 184/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2722 - acc: 0.4974
Epoch 185/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2679 - acc: 0.4967
Epoch 186/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2550 - acc: 0.5015
Epoch 187/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2561 - acc: 0.5000
```

```
Epoch 188/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2613 - acc: 0.4980
Epoch 189/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2571 - acc: 0.4985
Epoch 190/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2515 - acc: 0.5001
Epoch 191/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2461 - acc: 0.5016
Epoch 192/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2436 - acc: 0.5016
Epoch 193/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2384 - acc: 0.5026
Epoch 194/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2409 - acc: 0.5022
Epoch 195/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2339 - acc: 0.5036
Epoch 196/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2274 - acc: 0.5048
Epoch 197/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2302 - acc: 0.5046
Epoch 198/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2232 - acc: 0.5045
Epoch 199/200
49/49 [==============================] - 1s 20ms/step - loss: 2.2305 - acc: 0.5056
Epoch 200/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2204 - acc: 0.5060
```





If you haven't changed anything, this will pick up where the training above left off, we really only need one or the other, but we're getting wild and crazy here. If you're running this locally, one or the other will probably do the job.

```
train_ds = tf.data.Dataset.from_tensor_slices((X_t, y_t)).shuffle(SHUFFLE).batch(TEST_BATCH).prefetch(tf.data.experimental.AUTOTUNE)
```

```
# Or with datasets...
history = model.fit(train_ds, epochs=TEST_EPOCHS, verbose=1, callbacks=[early_stop, save_weights])
plot_acc(history)
plot_loss(history)
```

```
Epoch 1/200
49/49 [==============================] - 2s 48ms/step - loss: 2.2493 - acc: 0.4989
Epoch 2/200
49/49 [==============================] - 2s 45ms/step - loss: 2.2458 - acc: 0.5005
Epoch 3/200
49/49 [==============================] - 2s 45ms/step - loss: 2.2473 - acc: 0.5003
Epoch 4/200
49/49 [==============================] - 2s 33ms/step - loss: 2.2384 - acc: 0.5018
Epoch 5/200
49/49 [==============================] - 2s 39ms/step - loss: 2.2302 - acc: 0.5043
Epoch 6/200
```

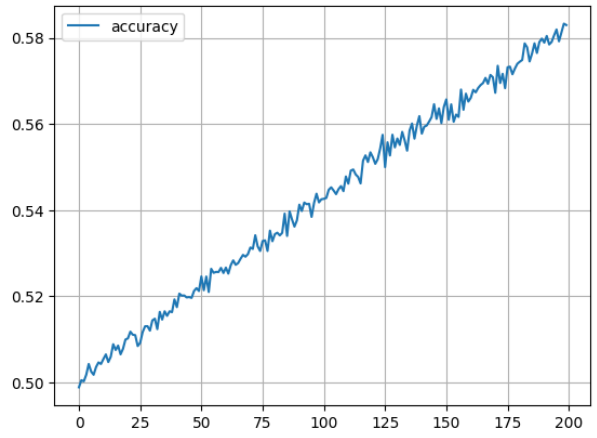
```
49/49 [==============================] - 2s 36ms/step - loss: 2.2299 - acc: 0.5025
Epoch 7/200
49/49 [==============================] - 2s 36ms/step - loss: 2.2395 - acc: 0.5018
Epoch 8/200
49/49 [==============================] - 1s 27ms/step - loss: 2.2275 - acc: 0.5036
Epoch 9/200
49/49 [==============================] - 1s 24ms/step - loss: 2.2186 - acc: 0.5047
Epoch 10/200
49/49 [==============================] - 1s 23ms/step - loss: 2.2226 - acc: 0.5043
Epoch 11/200
49/49 [==============================] - 1s 30ms/step - loss: 2.2179 - acc: 0.5055
Epoch 12/200
49/49 [==============================] - 1s 30ms/step - loss: 2.2181 - acc: 0.5065
Epoch 13/200
49/49 [==============================] - 2s 37ms/step - loss: 2.2166 - acc: 0.5047
Epoch 14/200
49/49 [==============================] - 1s 30ms/step - loss: 2.2101 - acc: 0.5060
Epoch 15/200
49/49 [==============================] - 2s 34ms/step - loss: 2.2041 - acc: 0.5089
Epoch 16/200
49/49 [==============================] - 1s 30ms/step - loss: 2.2079 - acc: 0.5076
Epoch 17/200
49/49 [==============================] - 1s 27ms/step - loss: 2.1990 - acc: 0.5086
Epoch 18/200
49/49 [==============================] - 1s 24ms/step - loss: 2.2113 - acc: 0.5065
Epoch 19/200
49/49 [==============================] - 1s 21ms/step - loss: 2.2037 - acc: 0.5078
Epoch 20/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1914 - acc: 0.5100
Epoch 21/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1850 - acc: 0.5102
Epoch 22/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1859 - acc: 0.5118
Epoch 23/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1925 - acc: 0.5111
Epoch 24/200
49/49 [==============================] - 1s 27ms/step - loss: 2.1849 - acc: 0.5110
Epoch 25/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1938 - acc: 0.5085
Epoch 26/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1872 - acc: 0.5092
Epoch 27/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1784 - acc: 0.5117
Epoch 28/200
49/49 [==============================] - 1s 27ms/step - loss: 2.1770 - acc: 0.5131
Epoch 29/200
49/49 [==============================] - 1s 28ms/step - loss: 2.1749 - acc: 0.5131
Epoch 30/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1796 - acc: 0.5120
Epoch 31/200
49/49 [==============================] - 1s 27ms/step - loss: 2.1638 - acc: 0.5144
Epoch 32/200
49/49 [==============================] - 1s 27ms/step - loss: 2.1686 - acc: 0.5148
Epoch 33/200
49/49 [==============================] - 1s 20ms/step - loss: 2.1708 - acc: 0.5124
Epoch 34/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1599 - acc: 0.5164
Epoch 35/200
49/49 [==============================] - 1s 20ms/step - loss: 2.1654 - acc: 0.5146
Epoch 36/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1524 - acc: 0.5165
Epoch 37/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1587 - acc: 0.5155
Epoch 38/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1505 - acc: 0.5165
Epoch 39/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1512 - acc: 0.5163
Epoch 40/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1447 - acc: 0.5193
Epoch 41/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1495 - acc: 0.5175
Epoch 42/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1427 - acc: 0.5206
Epoch 43/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1358 - acc: 0.5202
Epoch 44/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1343 - acc: 0.5202
Epoch 45/200
49/49 [==============================] - 1s 20ms/step - loss: 2.1423 - acc: 0.5197
Epoch 46/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1356 - acc: 0.5198
Epoch 47/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1408 - acc: 0.5196
Epoch 48/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1347 - acc: 0.5213
Epoch 49/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1230 - acc: 0.5219
Epoch 50/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1301 - acc: 0.5212
Epoch 51/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1193 - acc: 0.5246
Epoch 52/200
49/49 [==============================] - 1s 24ms/step - loss: 2.1257 - acc: 0.5214
Epoch 53/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1129 - acc: 0.5246
Epoch 54/200
49/49 [==============================] - 1s 20ms/step - loss: 2.1249 - acc: 0.5210
```

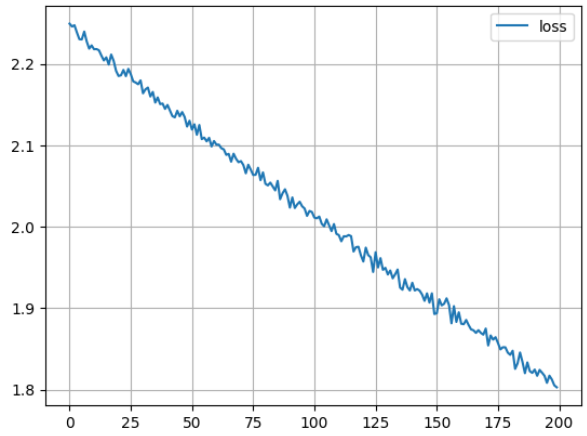
```
Epoch 55/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1076 - acc: 0.5264
Epoch 56/200
49/49 [==============================] - 1s 20ms/step - loss: 2.1094 - acc: 0.5255
Epoch 57/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1048 - acc: 0.5257
Epoch 58/200
49/49 [==============================] - 1s 20ms/step - loss: 2.1092 - acc: 0.5256
Epoch 59/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0985 - acc: 0.5266
Epoch 60/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1052 - acc: 0.5254
Epoch 61/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1007 - acc: 0.5267
Epoch 62/200
49/49 [==============================] - 1s 21ms/step - loss: 2.1007 - acc: 0.5253
Epoch 63/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0964 - acc: 0.5272
Epoch 64/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0950 - acc: 0.5284
Epoch 65/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0882 - acc: 0.5273
Epoch 66/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0894 - acc: 0.5277
Epoch 67/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0799 - acc: 0.5287
Epoch 68/200
49/49 [==============================] - 1s 20ms/step - loss: 2.0897 - acc: 0.5296
Epoch 69/200
49/49 [==============================] - 1s 24ms/step - loss: 2.0834 - acc: 0.5292
Epoch 70/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0792 - acc: 0.5298
Epoch 71/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0806 - acc: 0.5313
Epoch 72/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0757 - acc: 0.5310
Epoch 73/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0656 - acc: 0.5342
Epoch 74/200
49/49 [==============================] - 1s 20ms/step - loss: 2.0761 - acc: 0.5316
Epoch 75/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0701 - acc: 0.5305
Epoch 76/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0636 - acc: 0.5328
Epoch 77/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0638 - acc: 0.5330
Epoch 78/200
49/49 [==============================] - 1s 24ms/step - loss: 2.0724 - acc: 0.5305
Epoch 79/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0572 - acc: 0.5353
Epoch 80/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0667 - acc: 0.5328
Epoch 81/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0527 - acc: 0.5344
Epoch 82/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0508 - acc: 0.5348
Epoch 83/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0542 - acc: 0.5342
Epoch 84/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0495 - acc: 0.5347
Epoch 85/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0446 - acc: 0.5392
Epoch 86/200
49/49 [==============================] - 1s 20ms/step - loss: 2.0563 - acc: 0.5340
Epoch 87/200
49/49 [==============================] - 1s 24ms/step - loss: 2.0338 - acc: 0.5397
Epoch 88/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0411 - acc: 0.5379
Epoch 89/200
49/49 [==============================] - 1s 20ms/step - loss: 2.0460 - acc: 0.5362
Epoch 90/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0380 - acc: 0.5376
Epoch 91/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0235 - acc: 0.5413
Epoch 92/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0360 - acc: 0.5398
Epoch 93/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0230 - acc: 0.5418
Epoch 94/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0275 - acc: 0.5414
Epoch 95/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0307 - acc: 0.5415
Epoch 96/200
49/49 [==============================] - 1s 20ms/step - loss: 2.0251 - acc: 0.5384
Epoch 97/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0224 - acc: 0.5417
Epoch 98/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0133 - acc: 0.5438
Epoch 99/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0194 - acc: 0.5418
Epoch 100/200
49/49 [==============================] - 1s 24ms/step - loss: 2.0181 - acc: 0.5425
Epoch 101/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0112 - acc: 0.5426
Epoch 102/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0103 - acc: 0.5428
Epoch 103/200
```

```
49/49 [==============================] - 1s 20ms/step - loss: 2.0125 - acc: 0.5447
Epoch 104/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0037 - acc: 0.5453
Epoch 105/200
49/49 [==============================] - 1s 25ms/step - loss: 2.0005 - acc: 0.5446
Epoch 106/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0091 - acc: 0.5437
Epoch 107/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0018 - acc: 0.5449
Epoch 108/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9947 - acc: 0.5456
Epoch 109/200
49/49 [==============================] - 1s 21ms/step - loss: 2.0033 - acc: 0.5444
Epoch 110/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9912 - acc: 0.5478
Epoch 111/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9897 - acc: 0.5462
Epoch 112/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9822 - acc: 0.5492
Epoch 113/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9883 - acc: 0.5495
Epoch 114/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9881 - acc: 0.5483
Epoch 115/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9898 - acc: 0.5477
Epoch 116/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9884 - acc: 0.5462
Epoch 117/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9696 - acc: 0.5515
Epoch 118/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9749 - acc: 0.5527
Epoch 119/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9754 - acc: 0.5512
Epoch 120/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9649 - acc: 0.5534
Epoch 121/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9573 - acc: 0.5523
Epoch 122/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9744 - acc: 0.5508
Epoch 123/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9652 - acc: 0.5518
Epoch 124/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9625 - acc: 0.5543
Epoch 125/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9445 - acc: 0.5575
Epoch 126/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9687 - acc: 0.5500
Epoch 127/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9500 - acc: 0.5558
Epoch 128/200
49/49 [==============================] - 1s 24ms/step - loss: 1.9615 - acc: 0.5527
Epoch 129/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9471 - acc: 0.5575
Epoch 130/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9495 - acc: 0.5546
Epoch 131/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9414 - acc: 0.5567
Epoch 132/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9463 - acc: 0.5552
Epoch 133/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9367 - acc: 0.5582
Epoch 134/200
49/49 [==============================] - 1s 20ms/step - loss: 1.9416 - acc: 0.5562
Epoch 135/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9475 - acc: 0.5538
Epoch 136/200
49/49 [==============================] - 1s 24ms/step - loss: 1.9253 - acc: 0.5585
Epoch 137/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9227 - acc: 0.5601
Epoch 138/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9357 - acc: 0.5566
Epoch 139/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9259 - acc: 0.5596
Epoch 140/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9218 - acc: 0.5618
Epoch 141/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9312 - acc: 0.5578
Epoch 142/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9217 - acc: 0.5594
Epoch 143/200
49/49 [==============================] - 1s 24ms/step - loss: 1.9235 - acc: 0.5597
Epoch 144/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9215 - acc: 0.5606
Epoch 145/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9168 - acc: 0.5616
Epoch 146/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9090 - acc: 0.5646
Epoch 147/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9182 - acc: 0.5612
Epoch 148/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9069 - acc: 0.5636
Epoch 149/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9183 - acc: 0.5602
Epoch 150/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8929 - acc: 0.5640
Epoch 151/200
49/49 [==============================] - 1s 24ms/step - loss: 1.8938 - acc: 0.5657
```

```
Epoch 152/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9110 - acc: 0.5610
Epoch 153/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9035 - acc: 0.5646
Epoch 154/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9055 - acc: 0.5605
Epoch 155/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9121 - acc: 0.5622
Epoch 156/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9039 - acc: 0.5617
Epoch 157/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8815 - acc: 0.5680
Epoch 158/200
49/49 [==============================] - 1s 21ms/step - loss: 1.9025 - acc: 0.5633
Epoch 159/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8831 - acc: 0.5671
Epoch 160/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8950 - acc: 0.5652
Epoch 161/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8809 - acc: 0.5661
Epoch 162/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8804 - acc: 0.5680
Epoch 163/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8856 - acc: 0.5674
Epoch 164/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8794 - acc: 0.5684
Epoch 165/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8740 - acc: 0.5691
Epoch 166/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8729 - acc: 0.5695
Epoch 167/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8697 - acc: 0.5707
Epoch 168/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8731 - acc: 0.5693
Epoch 169/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8697 - acc: 0.5714
Epoch 170/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8675 - acc: 0.5709
Epoch 171/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8750 - acc: 0.5673
Epoch 172/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8542 - acc: 0.5735
Epoch 173/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8666 - acc: 0.5695
Epoch 174/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8614 - acc: 0.5717
Epoch 175/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8644 - acc: 0.5683
Epoch 176/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8572 - acc: 0.5732
Epoch 177/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8495 - acc: 0.5733
Epoch 178/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8521 - acc: 0.5716
Epoch 179/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8520 - acc: 0.5729
Epoch 180/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8455 - acc: 0.5740
Epoch 181/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8428 - acc: 0.5745
Epoch 182/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8478 - acc: 0.5749
Epoch 183/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8256 - acc: 0.5787
Epoch 184/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8323 - acc: 0.5778
Epoch 185/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8456 - acc: 0.5745
Epoch 186/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8341 - acc: 0.5764
Epoch 187/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8202 - acc: 0.5787
Epoch 188/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8335 - acc: 0.5765
Epoch 189/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8227 - acc: 0.5790
Epoch 190/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8206 - acc: 0.5798
Epoch 191/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8247 - acc: 0.5789
Epoch 192/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8171 - acc: 0.5805
Epoch 193/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8243 - acc: 0.5785
Epoch 194/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8208 - acc: 0.5791
Epoch 195/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8171 - acc: 0.5806
Epoch 196/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8084 - acc: 0.5820
Epoch 197/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8172 - acc: 0.5792
Epoch 198/200
49/49 [==============================] - 1s 20ms/step - loss: 1.8126 - acc: 0.5812
Epoch 199/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8054 - acc: 0.5833
Epoch 200/200
```

```
Epoch 200/200
49/49 [==============================] - 1s 21ms/step - loss: 1.8029 - acc: 0.5830
```





Generate Some Text

We can generate text once the model is trained. We'll start with a seed sentence, and then we'll use the model to predict the next word. We'll then append that word to the sentence and use the model to predict the next word, and so on. There is a little helper function to do this for us with limited repetition.

Another new thing is that we create an inverse dictionary to map the encoded words back to the original words.

Temperature

One weirdly named factor that is important in text generation is the temperature. The temperature is a factor that we can use to control the randomness of the output. To generate text, we are essentially using a probability distribution to determine what the next word should be - the softmax output of the model will tell us the most likely next word. The issue is that certain words are way more likely than others - "the", "it", "a", "and", etc. are all very common words so we can expect the model to predict them as "most likely" a lot, probably too often.

The most direct way to combat this is to add some degree of randomness to which word we select - we'll still pick the most likely word more often than any other, but we'll also pick other words that have some degree of likelihood at random. The higher the temperature the more randomness is introduced. A correct value requires tuning with human feedback, and it'll vary depending on the base quality of the model - large models that are trained on huge volumes of text and are deep enough to pick up on the "what type of word should be here" patterns will be able to generate better text with lower temperatures. Our model here is small and kind of sucks, so the temperature needs to be higher to get anything remotely usable. The implementation here is stolen shamelessly from the internet, the details don't really matter all that much, we just need to vary our predictions away from always simply picking the most likely word.

```
inverse_dict = {v: k for k, v in tokenizer.word_index.items()}
```

```
from random import *
```

```
def sampleWord(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds.flatten(), 1)
    return np.argmax(probas)

def generate_text(seed_text, next_words, model, length_max, tok, inverse, temperature=1.0):
    out = seed_text
    for _ in range(next_words):
        token_list = tok.texts_to_sequences([out])[0]
        token_list = pad_sequences([token_list], maxlen=length_max-1, padding='pre')
        #print(token_list)
        predicted = model.predict(token_list, verbose=0)
        #print(tok.sequences_to_texts(predicted))
        word = inverse[sampleWord(predicted, temperature=temperature)]
        out += " "+word
        #print(out)
    return out
```

Fake text time!

```
print (generate_text("Last night I went on a date with", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
print (generate_text("We are going to the ", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
#print (generate_text("Where are all of the", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
#print (generate_text("I am scared", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
print (generate_text("School is out for", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
```

```
Last night I went on a date with bb of here in dip hold retards all 🚀 above gamestop life one shit apes strong 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀

We are going to the  moon selling before all they out or a end this week price in same trade its at least down 12 short short at 1 market open no just

School is out for the last 12 days shares stocks hold 12 19 3 end calls trading 6 25 shares shares 7 shares 2 5 0 shares a stock please short a on my
```

A few long ones... as with a time series, we shouldn't be surprised if the text becomes worse the farther we go from

```
print (generate_text("Last night I went on a date with", next_words=100, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
#print (generate_text("When is work happening", next_words=100, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict), "\n")
print (generate_text("The Toronto Raptors won and Rob Ford's body rose from the dead", next_words=100, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer
```

```
Last night I went on a date with it soon selling for gme with robinhood two reddit wallstreetbets have the trade the to in this market short at html t

The Toronto Raptors won and Rob Ford's body rose from the dead way got it doing this big this point app of people them to this market trading account
```

All in all, this isn't really that much worse than actual wallstreetbets posts. Also, in one run while making this my model generated a rocket ship emoji, because we are going to the moon. Emojis are just weird character sequences, so that makes sense, but I had never really considered that as a possibility before.

And if we make the temperature low, we likely get worse results.

```
print (generate_text("Last night I went on a date with", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict, tempera
print (generate_text("We are going to the ", next_words=30, model=model, length_max=OUTPUT_LENGTH, tok=tokenizer, inverse=inverse_dict, temperature=.1), "\
```

```
Last night I went on a date with it i to my fucking wsb gme gme 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 x200b 🚀 buy buy 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀

We are going to the  moon 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀 🚀
```

Homemade Text Generation Results

The LSTM model works well for text generation, if we had more data, more time, and spent a bit more effort cleaning up the edge cases in our code here we could likely get something pretty legible.

The biggest difference between this and the impressive larger models is that they are higher capacity, trained with more data, and sometimes tuned with human feedback. These things combine to make the model more likely to pick up on the patterns of language and generate text that is more likely to be grammatically correct and follow the expected flow of language. Our model is really struggling to just find a reasonable word to use next, we haven't allowed it to learn enough to really construct logical sentences.

For our next trick, we can borrow the power of gpt-2...

Colab paid products - Cancel contracts here