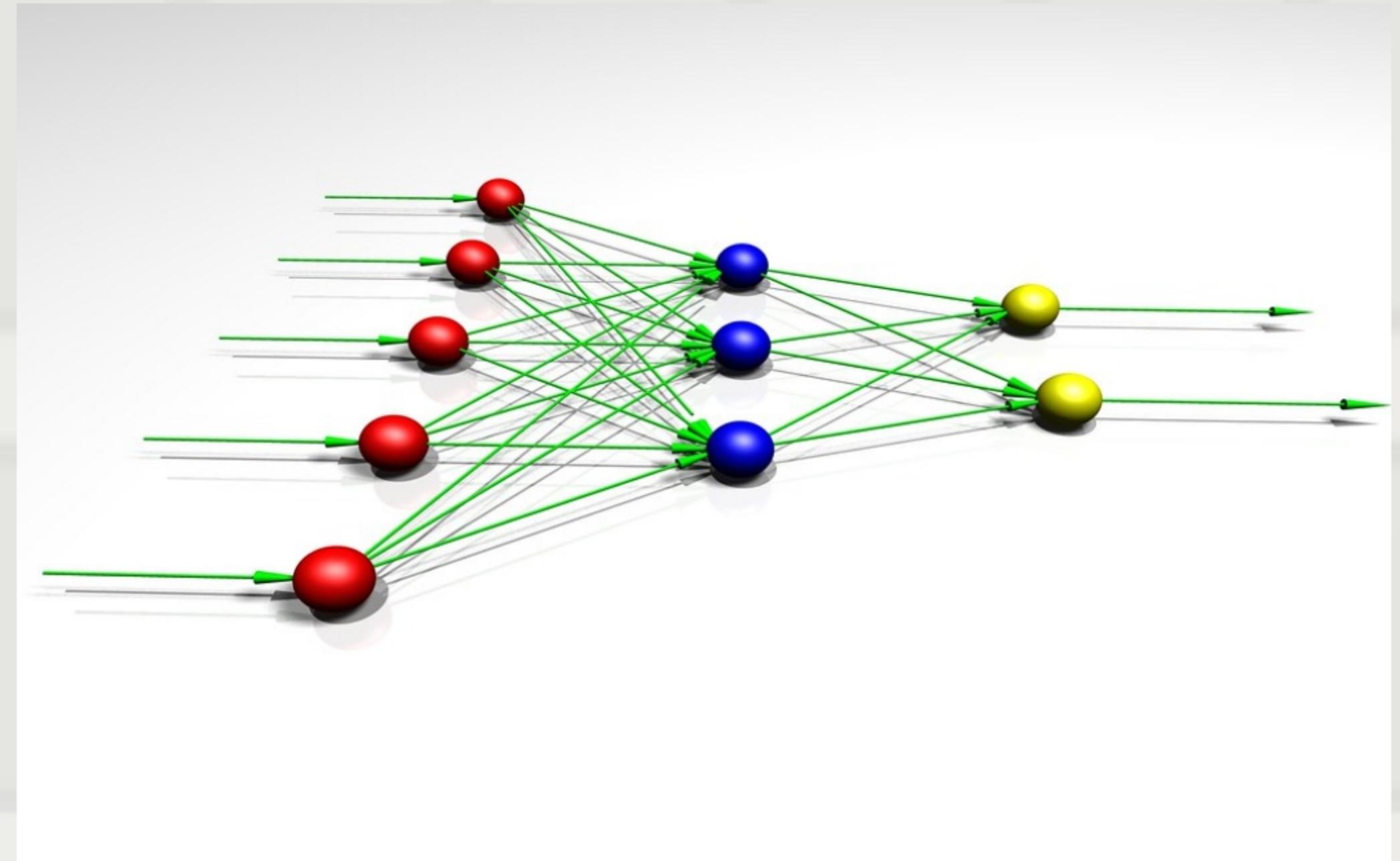


Sentiment Analyses Experience

Spencer Holley



THE DATASET

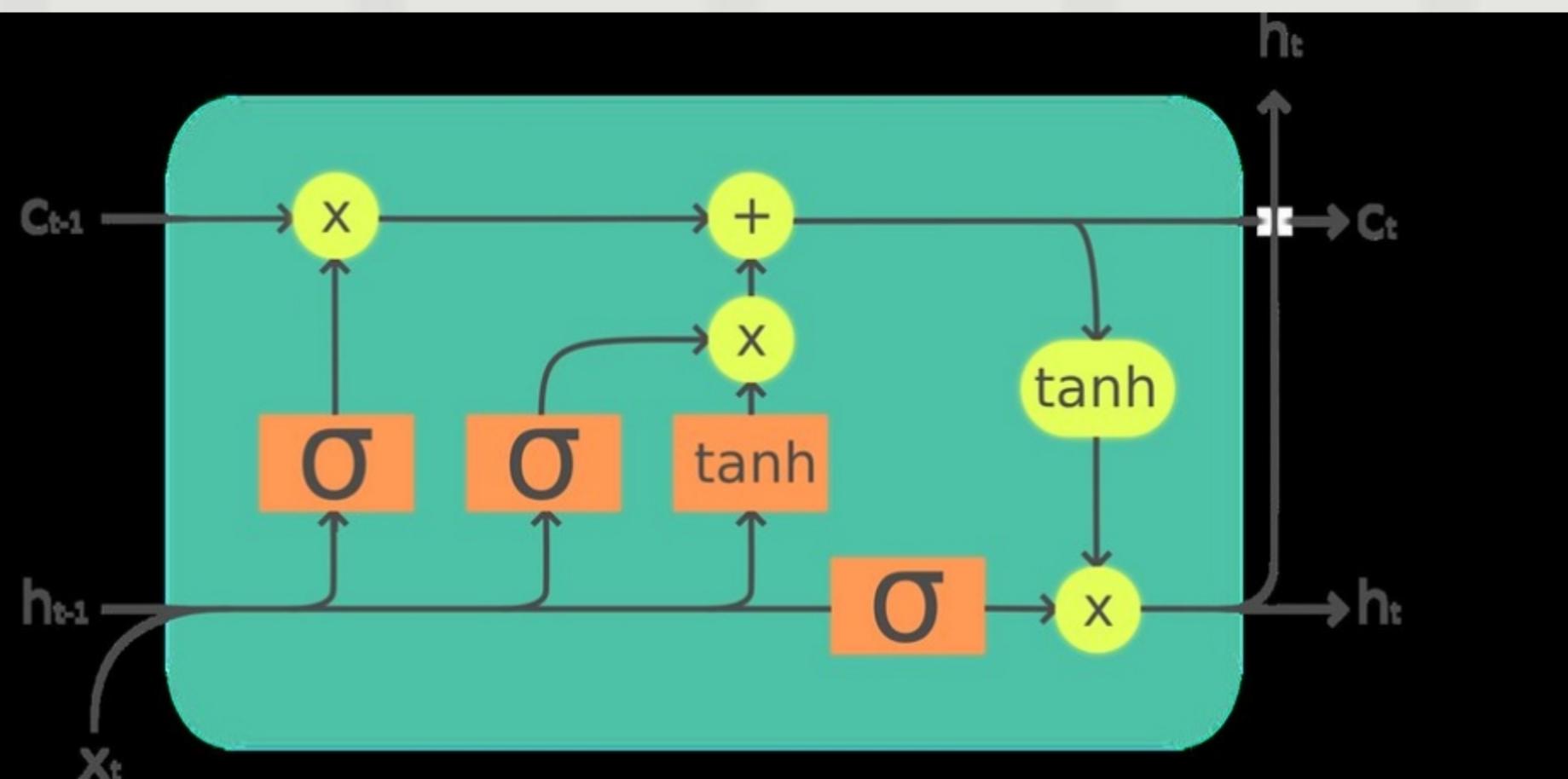
I used a dataset of 9,000 tweets provided by data.world provided by Crowdflower. The tweets are mainly public opinions of tech company's products such as Apple's iPad.



Deep Learning

I tried 4 different approaches, the last one preformed the best

1. FeedForward, a very simple fully connected neural network
2. LSTM
3. GRU
4. Bidirectional LSTM



Bidirectional LSTM

01

Uses an LSTM Architecture, which is great for modeling sequential data. The reason for this is that the model can control what it 'remembers' and forgets what is seen as less important. This makes it great for NLP tasks such as sentiment analyses.

02

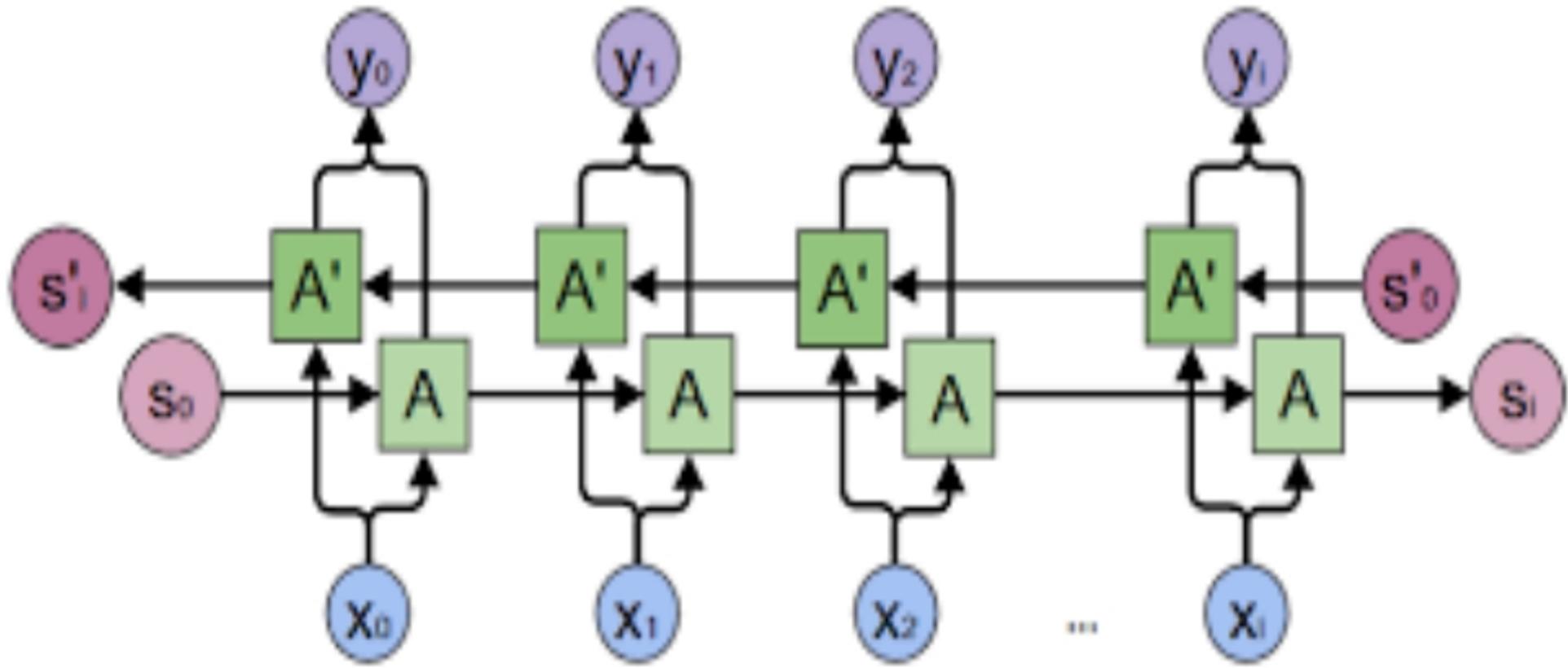
"Reads' text in both directions to gain a deeper intuition for how each word is used, as well as how it's used in relation to the words surrounding it

03

Model: "sequential_100"

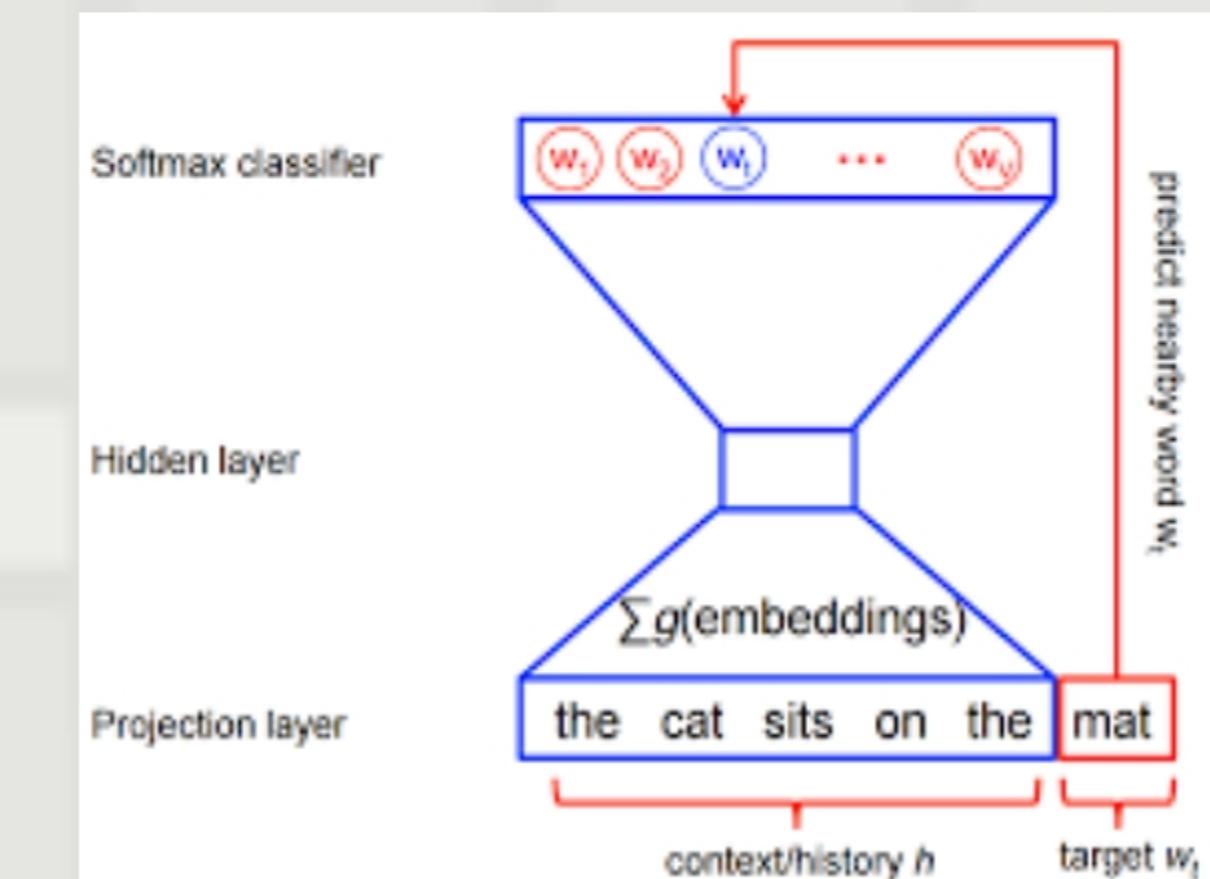
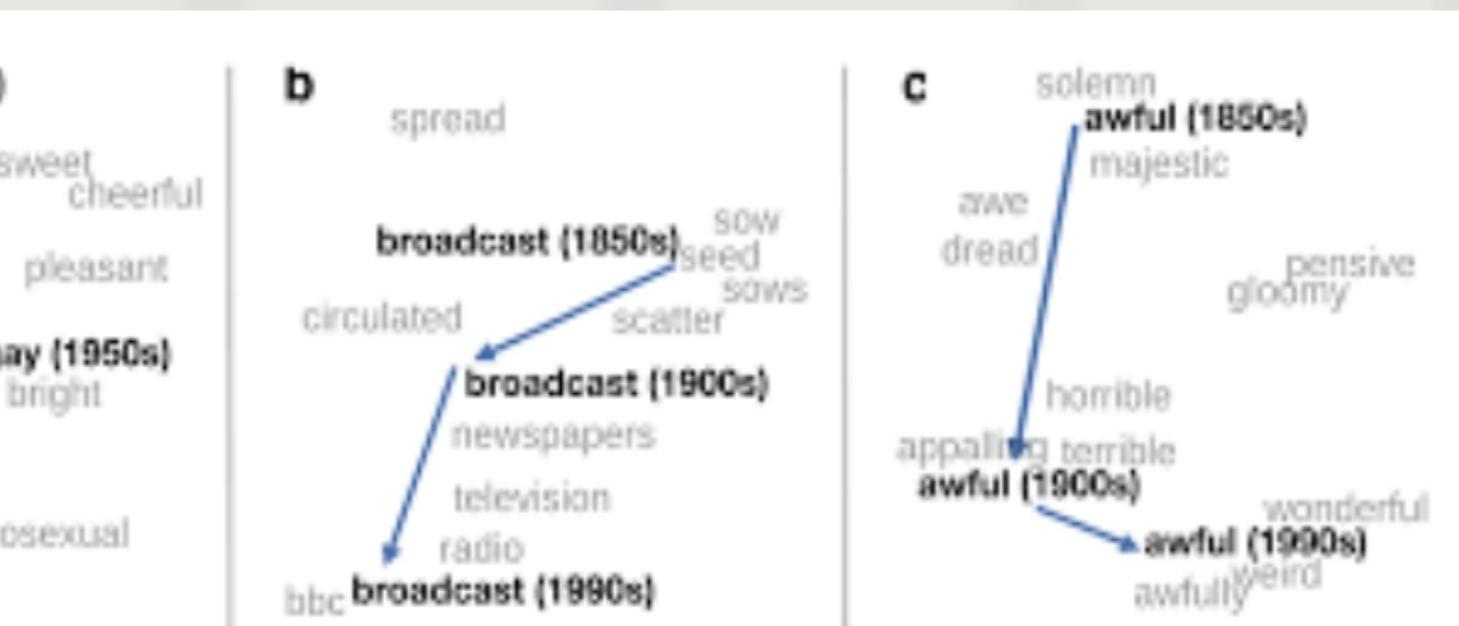
Layer (type)	Output Shape	Param #
<hr/>		
embedding_100 (Embedding)	(None, None, 128)	1600000
bidirectional_100 (Bidirecti	(None, 256)	263168
dense_300 (Dense)	(None, 64)	16448
dense_301 (Dense)	(None, 32)	2080
dense_302 (Dense)	(None, 3)	99
<hr/>		
Total params: 1,881,795		
Trainable params: 1,881,795		
Non-trainable params: 0		

These are great for NLP tasks such as text classification or generation. They also work well with time series data, stock price prediction for example



Word embeddings

Word embeddings serve as a 'lookup table' for the Neural Network. Each unique word in the text corpus has vector of numbers representing how similar that word is to all the other words.



How to make word Embeddings

```
model2 = models.Sequential()
model2.add(layers.Embedding(12500, 128))
model2.add(layers.Bidirectional(layers.LSTM(128)))
model2.add(layers.Dense(64, activation='relu'))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(3, activation='softmax'))
```

Make it yourself with Keras

- This method worked the best for me
- it is very simple and can be done in Keras with one line of code
- the only downside it that it is very blackbox in the sense that you can't see the embeddings.

Use the Gensim Library

- This method didn't do too well in my experience, for this use case.
- However I still recommend it because you can see the results and get a feel for what words some words are similar to or the opposite of

```
[44]: from gensim.models import Word2Vec, keyedvectors
W = Word2Vec(tokens, size=33, window=3, min_count=1, workers=4)
W.train(sentences=tokens, total_examples=W.corpus_count, epochs=10)
wv = W.wv

[45]: e = W.wv.get_keras_embedding(train_embeddings=False)

[46]: from keras.metrics import Recall #the model did better with accuracy
from keras.metrics import Precision

model3 = models.Sequential()
model3.add(e)
```

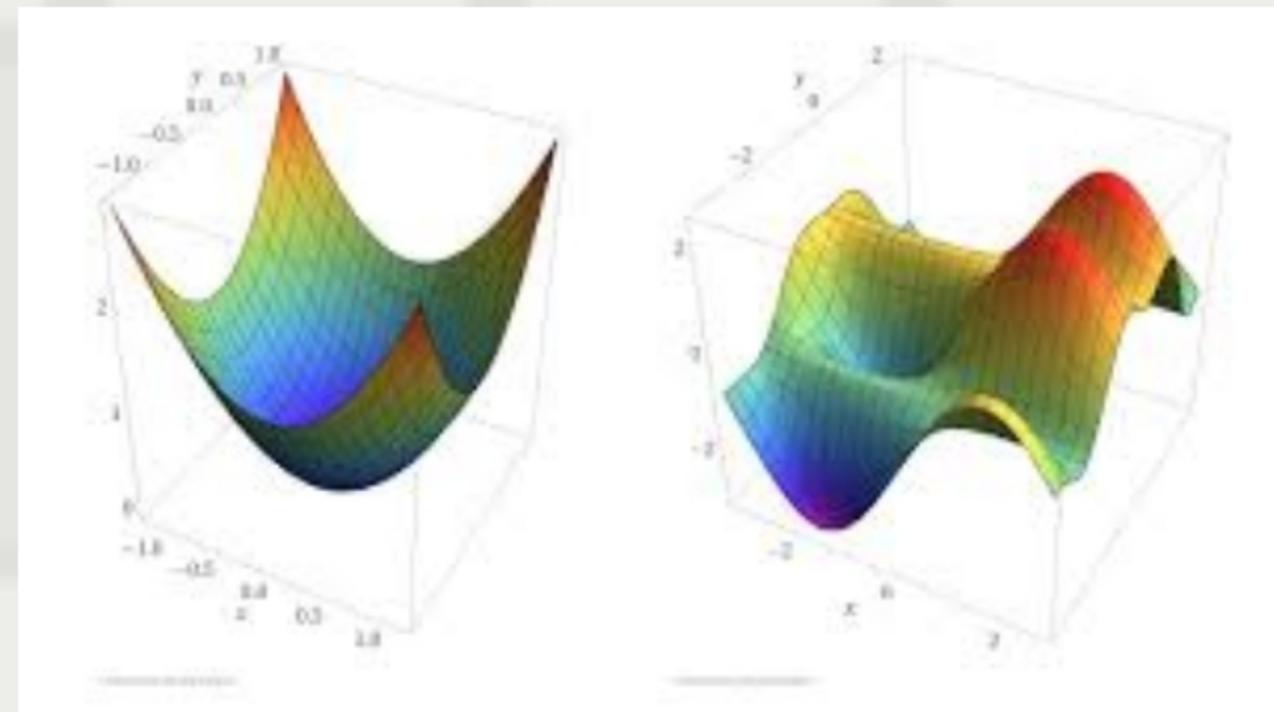
```
glove = {}
with open('glove.6B.50d.txt', 'rb') as f:
    for line in f:
        parts = line.split()
        word = parts[0].decode('utf-8')
        if word in total_vocabulary:
            vector = np.array(parts[1:], dtype=np.float32)
            glove[word] = vector
```

Use a pretrained model such as GloVe

- Goals & Requirements
- Application flow
- Wireframes & interactions
- Creative concept
- Visual design & Identity
- UI Specification

Setting Parameters

I strongly advise using a grid search to hypertune your model, I especially recommend experimenting with the number of nodes in each layer and activation functions relu and tanh are certainly both worth trying! I would also advise testing between rmsprop and adam as far as optimizers go but would void sgd as it's very likley to get stuck in global minima.



```
def TestDL(self, params, func, task, X, y, X_val=None, y_val=None, batch_size=64, epochs=50):
    early_stopping = [EarlyStopping(patience=10), ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5')]
    if task == 'classification':
        k = KerasClassifier(func)
    if task == 'regression':
        k = KerasRegressor(func)
    grid = GridSearchCV(k, params, cv=3)
    if type(X_val) != np.ndarray:
        grid.fit(X, y, batch_size=batch_size, epochs=epochs, validation_split=0.2)
    else:
        grid.fit(X, y, batch_size=batch_size, epochs=epochs, validation_data=(X_val, y_val), callbacks=early_stopping)
    return grid
```

Simple Machine Learning

Basic Machine learning strategies can work just as well as Deep Learning!

However, I tried a very strange approach that didn't work very well!

If you want to use basic ML, I would advise using a Gradient Boosted Random Forest, add it the word embeddings, technically this part is deep learning.

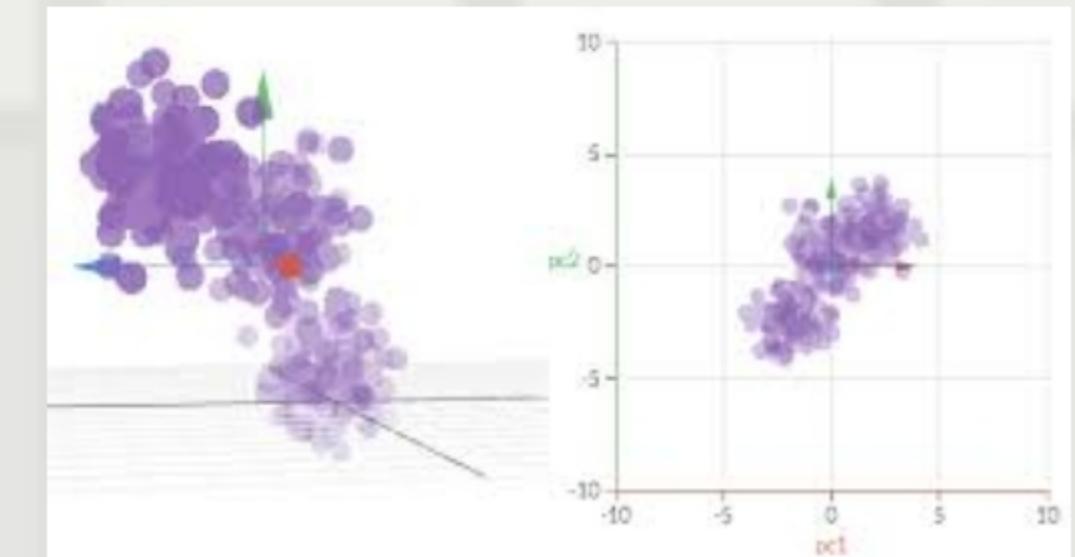
What I Tried

than using word embeddings
to One Hot Encode each tweet!

This means that each tweet was represented
as a vector of 1's and 0's, a 1 for each word
in theocab that was used and a 0 for each
word that wasn't



	Red	Yellow	Green
1	1	0	0
1	0	0	0
0	1	0	0
0	0	0	1



2. Next, I compressed the data to make it less sparse, each vector had 12,500 numbers of which, about 12,470 were 0's. I used PCA to compress each vector to a length of 300

3. Lastly, I fit the data to a Gradient Boosted Random Fo



Future works

With more time to work on this project I would use my model to label tweets by their sentiment and use that feature alongside other features to create a polarity predictor!

