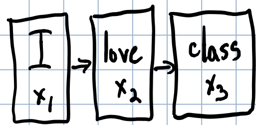
Traditional neural network architecture:



Each token is vectorized using one of the techniques we have learned about so far:

- one-hot encoding (0 or 1)
- word count (# of dimensions can be thousands or **tens of thousands**)
- TF-IDF (# of dimensions can be **thousands**, or **tens of thousands**)
- word2vec (# of dimensions usually **100, 300, or 500**)

Task: Predict the Sentiment of a Document

For the purposes of your group project, you can also think of this as "predict the tag" for your product.

Strategy 1: Traditional BOW Architecture

A **BOW (bag of words)** architecture does not take into account the sequence of the tokens in your document. Therefore, **I love class** is the same as **Love class I**, or **class I love**

$$\begin{array}{l} \text{I} \\ \text{love} \\ \text{class} \end{array} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \boxed{\begin{array}{c}\text{supervised} \\ \text{machine learning} \\ \text{model}\end{array}} \Rightarrow \begin{bmatrix} 0.5 \\ 0.9 \\ 0.3 \end{bmatrix} \Rightarrow \text{avg}(\ldots) \Rightarrow 0.8$$

0 is negative
1 is positive

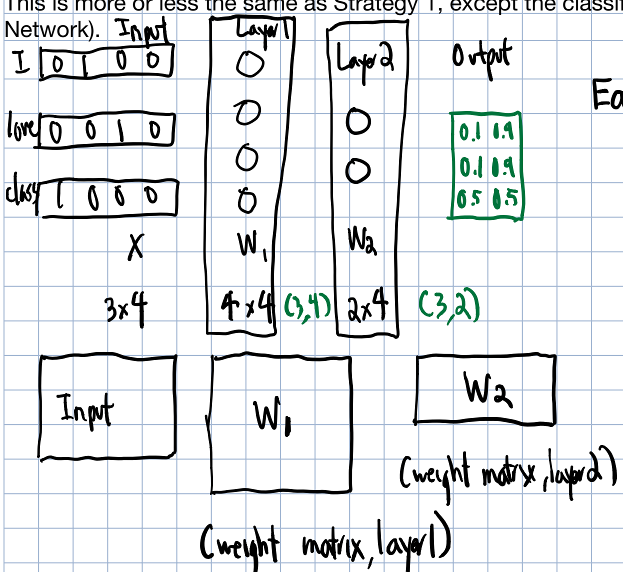
You can vectorize each word, run it through a supervised learning model classifier, and return (for each word), the likelihood that word is positive. Then you simply average the outputs to get a likelihood for the entire document of being positive.

Note: the supervised ML model can be anything:

- Deep neural network
- Logistic regression
- Decision tree
- Support Vector Machine (SVM)

Strategy 2: Deep Learning Neural Network

This is more or less the same as Strategy 1, except the classification model used is a feed-forward DNN (Deep Neural Network).



Each layer i's output is $a_i = g(a_{i-1} W_i^T + b_i)$

$g(x)$ is some non-linear activation function, usually

- tanh
- ReLU (or some variant)
- sigmoid

Issues:

- **model over-parameterization** - in our model, there's already **16 + 8** (from the weight matrices) + **4 + 2** (from the bias vectors) weight values to update. This is assuming a feature dimensionality of only **4.**
- **Still does not capture sequence information**

Strategy 3: Recurrent Neural Network

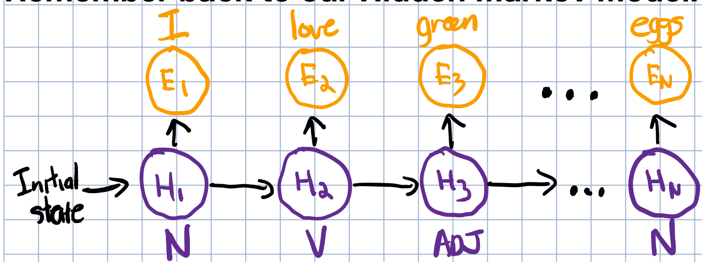
We want some way to capture sequential information. So far, we have had only two axis to work with for our data:

- N - the number of data points (ie. tokens, documents, etc.)
- M - the number of dimensions that represents a data point

We will now add in a **3rd dimension**:

- S - the sequence index: for instance, the third token in Document 1 might be represented as $\mathbf{x}_{1,3}$.

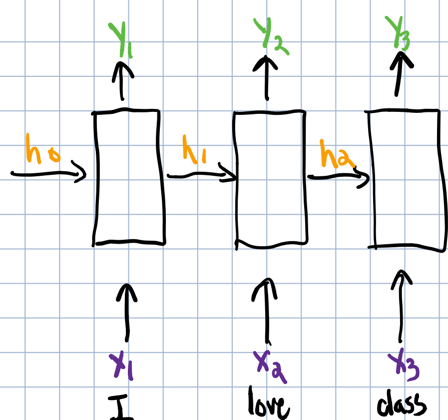
Remember back to our Hidden Markov Model:



E_N = the emitted (observed) state at index position N.

H_N = the hidden state at index position N that generates (emits) the observed state

Recurrent Neural Network Architecture



For each layer...

(1,3)

$h_1 = g^a(x_1 W_x^h + h_0 W_h^h + b^a)$ ← Step 1

$y_1 = g^y(h_1 W_h^y + b^y)$ ← Step 2

$h_i = 1\times3$

$x_i = 1\times300$ (assume we use word2vec)

$y_i = 1\times K$ (assume K classes we are trying to predict) ✓ ✓

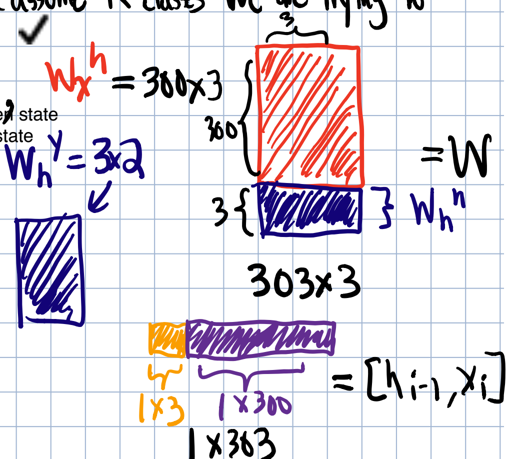
$W_h^h = 3\times3$, $W_x^h = 300\times3$

square matrix: represent hidden state transition into next hidden state

original hidden state: (1,3)

Assume that our hidden layer has 3 dimensions

$W_h^y = 3\times2$



Advantages:

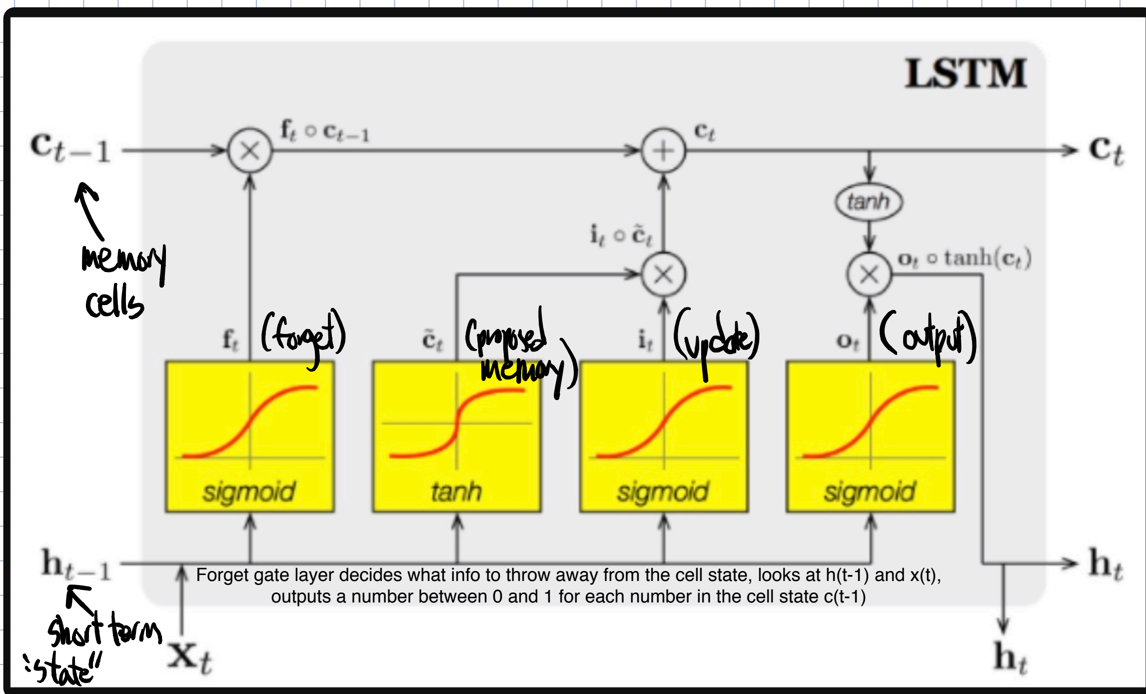
- incorporates the information from prior words/tokens
- significantly **fewer parameters** to update than a traditional deep learning neural network
- Can provide a prediction at **each step of the sequence**
- Better at modeling sequences of data

Disadvantages:

- **Explainability is lost** compared to count vectorization models
- Although it incorporates sequence, most of information comes from close neighbors (**struggles to handle long-term dependencies**)
- **Vanishing/exploding gradients**

After living in Madrid for three years, seeing the quiet tranquility of the people, their effusive personalities, and the amazingly deep conversations they engaged in over cups of coffee at the local cafe, I decided it was worth it to learn __________.

Strategy 4: LSTM (Long Short Term Memory) Units



① $\tilde{C}_t = \tanh(W_h^c h_{t-1} + W_x^c x_t + b_c)$ proposed "new" memory for time t

② $G_u = \sigma(W_h^u h_{t-1} + W_x^u x_t + b_u)$ update gate for time t

$G_F \quad \sigma(W_h^f h_{t-1} + W_x^f x_t + b_f)$ forget gate for time t

$G_o = \sigma(W_h^o h_{t-1} + W_x^o x_t + b_o)$ output gate for time t

get gate values

③ $C_t = G_u \otimes \tilde{C}_t + G_f \otimes C_{t-1}$

$h_t = G_o \otimes \tanh(C_t)$