# 6.864 HW2

Name: Adarsh Jeewajee, ID: 910226809

---

## Problem: Hidden Markov Models

### Question 1.1

The dimensionality of $\theta$ as a function of $N$ and $\Sigma$ is:

$$N^2 - 2N + N|\Sigma| + N - 1$$

### Question 1.2

The probability can be expanded as:

$$\pi_1.b_1(the).a_{1,2}.b_2(dog).a_{2,1}.b_1(the)$$

It equals 0 as $a_{2,1} = 0$

### Question 2.1

Assuming 'Start' and 'Stop' symbols are not added to $T$, we have $|T|^n$ sequences of tags

### Question 2.2

The completed entries are:

$$\pi(0, *) = 1$$

$$\pi(0, v) = 0, v \neq *$$

### Question 2.3

We have $N$ words to guess the tags for where $N >= k$

$r(y_1, y_2, ..., y_k)$ stands for the joint probability of tags $y_1...y_k$ producing the words $w_1...w_k$

There are $|T|^k$ such sequences and their respective $r(y_1, y_2, ..., y_k)$'s

Let $S_{(k,v)}$ be the set of those sequences that end with $v$

$$\pi(k, v) = max_{S_(k-1,v')}(r(y_1, ..., y_{k-1}, v)) \tag{1}$$

$$\pi(0, v) = r(y_0) = \pi_v \tag{2}$$

$$\begin{aligned}\pi(1,v) &= max_{y_0}(r(y_0).a_{y_0,v}.b_v(w_1) \\ &= max_{v'}(\pi(0,v').a_{v',v}.b_v(w_1))\end{aligned} \tag{3}$$

$$\begin{aligned}\pi(2,v) &= max_{y_0,y_1}(r(y_0,y_1).a_{y_1,v}.b_v(w_2)) \\ &= max_{y_0,y_1}(r(y_0).a_{y_0,y_1}.b_{y_1}(w_1).a_{y_1,v}.b_v(w_2)) \\ &= max_{y_1}(max_{y_0}(r(y_0).a_{y_0,y_1}.b_{y_1}(w_1)).a_{y_1,v}.b_v(w_2)) \\ &= max_{v'}(\pi(1,v').a_{v',v}.b_v(w_2))\end{aligned} \tag{4}$$

$$\begin{aligned}\pi(k,v) &= max_{y_0,...,y_k}(r(y_0,...,y_{k-1}).a_{y_{k-1},v}.b_v(w_k)) \\ &= max_{y_0...y_{k-1}}(r(y_0,...,y_{k-2}).a_{y_{k-2},y_{k-1}}.b_{y_{k-1}}(w_{k-1}).a_{y_{k-1},v}.b_v(w_k)) \\ &= max_{y_{k-1}}(max_{y_0,...y_{k-2}}(r(y_0,...y_{k-2}).a_{y_{k-2},y_{k-1}}.b_{y_{k-1}}(w_{k-1})).a_{y_{k-1},v}.b_v(w_2)) \\ &= max_{v'}(\pi(k-1,v').a_{v',v}.b_{y_k}(w_k))\end{aligned} \tag{5}$$

**Question 2.4**

For each $n \in N$, for each $t \in T$, Need to do $max_u(\pi(n-1,u).a_{u,t}.b_t(x_n))$ for all $u \in T$

Then need to back-track for all $n \in N$, looking at values for all previous tags $t \in T$

Hence, complexity $= O(NT^2 + NT) = O(NT^2)$

**Question 3.1**

Count is given by:

$$\sum_{x^i} \sum_{y \in tagSequences} (count(x^i, y, u \to v).P(y/x^i, \theta))$$

**Question 3.2**

$a_{u,v}$ is given by:

$$\frac{\sum_{x^i} \sum_{y \in tagSequences} (count(x^i, y, u \to v).P(y/x^i, \theta))}{\sum_n \sum_{x^i} \sum_{y \in tagSequences} (count(x^i, y, u \to w_n).P(y/x^i, \theta))}$$

**Question 3.3**

$$\begin{aligned} P(x_1, ..., x_n|\theta) &= \sum_p P(x_1, ..., x_n, y_i = p|\theta) \\ &= \sum_p P(x_i, ..., x_n|x_1, ..., x_{i-1}, y_i = p, \theta) P(x_1, ..., x_{i-1}, y_i = p|\theta) \\ &= \sum_p \alpha_p(i) P(x_i, ..., x_n|y_i = p, \theta) \\ &= \sum_p \alpha_p(i)\beta_p(i) \end{aligned} \tag{6}$$

**Question 3.4**

$$\begin{aligned} P(y_i = p|x_1, ..., x_n, \theta) &= \frac{P(x_1, ..., x_n, y_i = p|\theta)}{P(x_1, ..., x_n|\theta)} \\ &= \frac{\alpha_p(i)\beta_p(i)}{\sum_q \alpha_q(i)\beta_q(i)} \end{aligned} \tag{7}$$

Where for the last line we use our result from 3.3

## Problem: Neural Network Classifier with Word Embeddings

**Question 4.1**

For all the combinations of the Hyperparameters, below are the Dev Set Accuracies after training for 50 epochs on the training set:

| Hidden size/input size | Learning Rate | Weight Decay Constant | Dev Accuracies |
|---|---|---|---|
| 0.25 | 1e-05 | 1e-05 | 54.4797687861 |
| 0.25 | 1e-05 | 0.001 | 54.3930635838 |
| 0.25 | 1e-05 | 10 | 47.8323699422 |
| 0.25 | 0.001 | 1e-05 | 81.401734104 |
| 0.25 | 0.001 | 0.001 | 81.0260115607 |
| 0.25 | 0.001 | 10 | 52.1676300578 |
| 0.25 | 0.1 | 1e-05 | 84.8410404624 |
| 0.25 | 0.1 | 0.001 | 80.8092485549 |
| 0.25 | 0.1 | 10 | 52.1676300578 |
| 0.25 | 10 | 1e-05 | 47.8323699422 |
| 0.25 | 10 | 0.001 | 52.1676300578 |
| 0.25 | 10 | 10 | 47.8323699422 |
| 0.5 | 1e-05 | 1e-05 | 60.3034682081 |
| 0.5 | 1e-05 | 0.001 | 68.3092485549 |
| 0.5 | 1e-05 | 10 | 47.8323699422 |
| 0.5 | 0.001 | 1e-05 | 81.2716763006 |
| 0.5 | 0.001 | 0.001 | 81.3294797688 |
| 0.5 | 0.001 | 10 | 52.1676300578 |
| 0.5 | 0.1 | 1e-05 | 85.2312138728 |
| 0.5 | 0.1 | 0.001 | 80.2456647399 |
| 0.5 | 0.1 | 10 | 47.8323699422 |
| 0.5 | 10 | 1e-05 | 52.1676300578 |
| 0.5 | 10 | 0.001 | 47.8323699422 |
| 0.5 | 10 | 10 | 47.8323699422 |
| 1.0 | 1e-05 | 1e-05 | 66.5606936416 |
| 1.0 | 1e-05 | 0.001 | 61.2283236994 |
| 1.0 | 1e-05 | 10 | 52.1676300578 |
| 1.0 | 0.001 | 1e-05 | 81.6907514451 |
| 1.0 | 0.001 | 0.001 | 81.1271676301 |
| 1.0 | 0.001 | 10 | 52.1676300578 |
| 1.0 | 0.1 | 1e-05 | 83.6271676301 |

| Hidden size/input size | Learning Rate | Weight Decay Constant | Dev Accuracies |
|---|---|---|---|
| 1.0 | 0.1 | 0.001 | 74.161849711 |
| 1.0 | 0.1 | 10 | 40.9104046243 |
| 1.0 | 10 | 1e-05 | 47.8323699422 |
| 1.0 | 10 | 0.001 | 47.8323699422 |
| 1.0 | 10 | 10 | 52.1676300578 |
| 2.0 | 1e-05 | 1e-05 | 70.2312138728 |
| 2.0 | 1e-05 | 0.001 | 65.3323699422 |
| 2.0 | 1e-05 | 10 | 47.8323699422 |
| 2.0 | 0.001 | 1e-05 | 81.676300578 |
| 2.0 | 0.001 | 0.001 | 80.9826589595 |
| 2.0 | 0.001 | 10 | 52.1676300578 |
| 2.0 | 0.1 | 1e-05 | 87.2543352601 |
| 2.0 | 0.1 | 0.001 | 80.4768786127 |
| 2.0 | 0.1 | 10 | 47.8323699422 |
| 2.0 | 10 | 1e-05 | 52.1676300578 |
| 2.0 | 10 | 0.001 | 52.1820809249 |
| 2.0 | 10 | 10 | 58.887283237 |

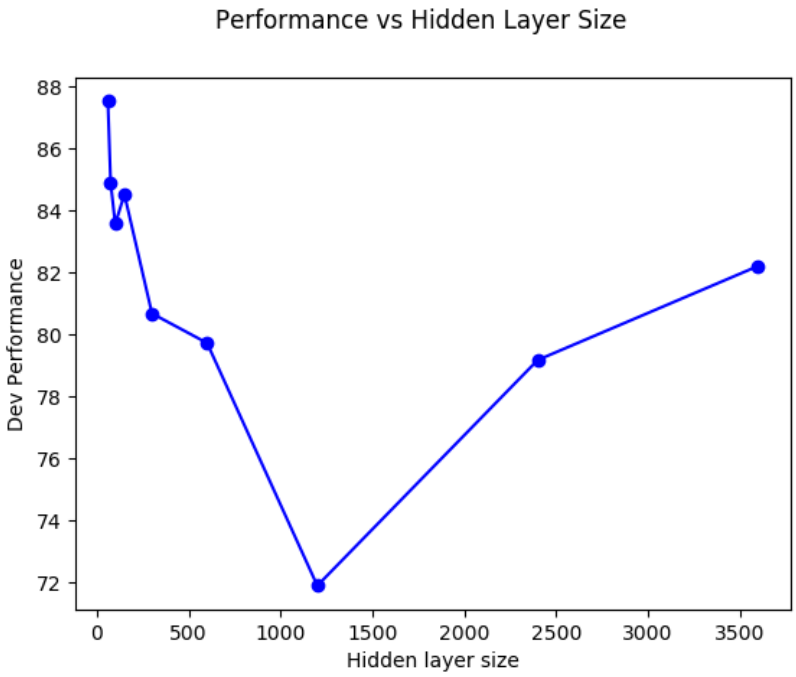
From there, I chose the model that performs the best on the Dev set which is the model that uses the parameters:
**Hidden layer size: 2.0**
**Learning rate: 0.1**
**Weight decay constant: 1e-05**

The **Model Architecture is explained in detail below**:

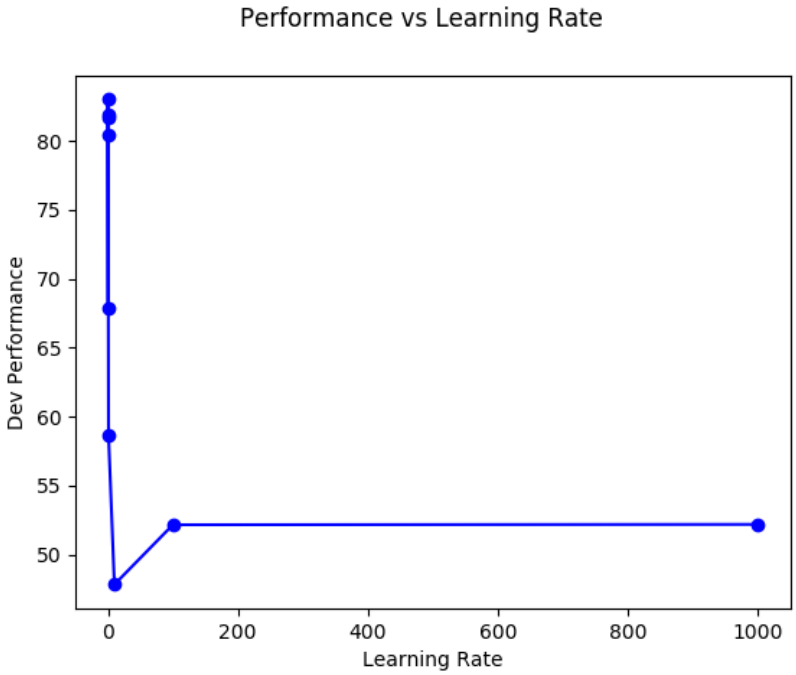
- The inputs are of dimension 300 and we use a hidden layer that is twice this size and is fully connected to the inputs
- We use *tanh* activation on each of those neurones before propagating the signal further to the output layer
- The output layer is a two-neurone layer that outputs a value pertaining to each of the two ratings a comment can get
- To convert those outputs to the loglikelihood of each rating, we use the LogSoftmax function on both values, and calculate the Negative Log Likelihood loss comparing these outputs to the actual ratings of each comment
- After a full cycle of feeding forward the signal, we backpropagate to get gradients of loss with respect to each parameter (weights between each neurone pair) and optimize those based on the loss using the Adam optimizer

Its **performance on the Test Set** is: **83.6849710983** if trained for 50 epochs although a higher accuracy of 85.2312138728 is obtained after 47 epochs.

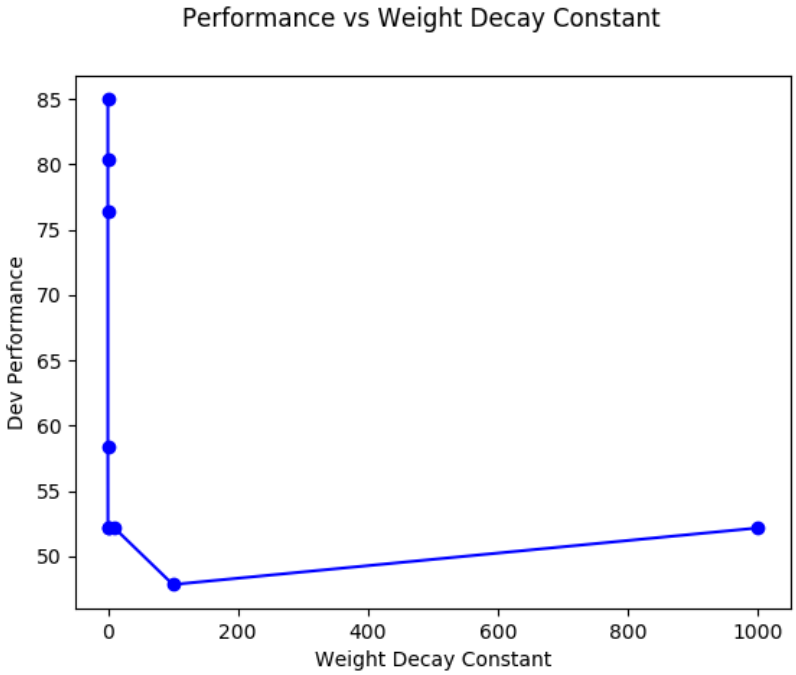
**Question 4.2**




For a small hidden layer state there is a small number of parameters and a smaller chance of overfitting so the performance is the best. As the hidden layer size increases from 200 and above, performance on dev set decreases as there are more parameters and more overfitting of the training set. Performance rises again after size 1200 as despite the risk of overfitting, we also have much more neurones that learn more detailed characteristics of the data.




When the learning rate is small enough, we are able to stay within the global loss minimum. As it increases above 1e-3, we are jumping too far ahead and skipping the global minimum for loss, hence performance keeps decreasing. After 1e2, we are so far off the global minimum that the performance is constant at around 52% since the loss curve has plateaued




At the smallest weight decay constants, we overfit the training data the least and generalize well over the dev set so performance is very high nut as we exceed 1e-3, we start overfitting more and more the training data and thus fail to generalize over the dev set so the performance keeps decreasing. After 1e2, we are almost at constant low performance as we are learning the training data too closely and slight difference in the unseen dev set causes errors from the model.

**Question 5.1.1**

There are two ways of looking at this question:-

One if the **model architecture is kept the same but the model is trained on hotel or restaurant training data directly then tested on them**

- We did not vary the layers that we have in our model and the activation functions between them. Neither did we vary the loss function or optimizer used.
- Taking this model, when run on the training data with data similar to what will be seen on the test set, we optimized the hidden layer size to have enough neurones to classify this data accurately, the learning rate so that we do not minimize loss locally only or do not miss a global minimum for loss and also the weight decay constant that is suitable for this kind of data so we do not overfit

- The same process will be repeated from a fresh model with the same architecture to tune the parameters on either restaurant or hotel reviews
- Because once tuned the architecture worked well on the common task of classifying sentiment from reviews, it will perform well on the new type of data too, but with different hyperparameter values more tailored to the kind of words and patterns it will extract on that new domain

One if the **model is trained on our data set then used on hotel or restaurant reviews**

- Following the same explanation above given for the model being trained afresh on the new domain, we now train on our dataset then move to a different domain of sentiment analysis
- Even though the model does classify reviews well, it does not mean that different domains of reviews are going to require the same hyperparameters
- For example, maybe because of the difference in jargon and style of writing for restaurant reviews, that the minimum loss would occur at a different combination of weights, maybe there is a lesser or higher degree of overfitting, maybe the local maxima are so wide that we need a bigger learning rate etc
- Not training on the restaurant/hotel dataset first, does not help in preventing these problems and we will not get the best classification possible

**Question 5.1.2**

The models we implemented attempt to classify review only for positivity or negativity i.e. the overall sentiment. They are not trained to discern between aspects. For example, a review that talks about bad taste even though it is very positive in terms if smell might be predicted as having a positive overall rating but this would not be a desired result if we are only interested in 'taste' sentiment prediction which should be predicted as negative.
To remedy to this, we could:

- First, if we are only interested in taste, we should use a more specialized data set where reviews are scored for taste only and not other aspects, hence instead of a 0 or 1 for overall positivity or negativity, a 0 would indicate bad taste sentiment and a 1 would indicate good taste sentiment
- Training on this data set would allow the model to pick up the sentiment we are interested in
- Another option is to use a different set of embeddings where taste-related words are given priority over other sentiments' words and priority could be higher magnitude vectors for taste words versus smell words etc

- Another option is to connect the hidden layer, through tanh, to an output layer that has 2n neurones, each pair for a probability distribution output over the positivity and negativity of a certain sentiment, of which there are n in total. This way, the net not only specializes in its sentiment being looked for but it also is able to differentiate between sentiments.

## Problem: Representations of text

### Question 6.1

The two categories of text are:

- The question, $q$, that was asked (made of L tokens: $q_1, ..., q_L$) and is processed into a vector $q$ as described in next question
- A set of $n$ paragraphs filtered by the document retriever as potential answer-holders (each paragraph is made of m tokens: $p_1, ..., p_m$ where the raw tokens are manipulated through feature vectors then an RNN as seen in the next question.

We want the output to be a span of tokens which is a subset of $p_1 top_m$, that is mostly likely to have the answer to our question

### Question 6.2

After embedding the inputs into feature vectors (one for each token in a paragraph and one vector for a question by leveraging embeddings e.g. Glove, part-of-speech tags or entity tags), the neural network encodings are achieved by feeding either the paragraph token vectors or the question into an 3-layer bidirectional LSTM (special RNN) with dropout of 0.3. The final representation is a concatenation of all the hidden layer values.

### Question 6.3

The maximum length possible is 16 and the score is $P_{start}(i) x P_{end}(i')$ where $i$ is the index of the starting token for the output and $i'$ is the end token.

### Question 6.4

We choose the output that maximizes $P_{start}(i) x P_{end}(i')$. The outputs are all spans $i$ to $i'$ with length less than 16.

### Question 6.5

The encodings need to be modified i.e the weights of the RNN need to be modified such that the returned encoding for a vector that produces a correct output persists for that vector but the encoding for the vectors giving incorrect outputs should be altered. This can be done by measuring the loss and backpropagating

through the network. The new produced embeddings after adjjusting weights is a better one and thus we achieve a dynamic kind of embedding.