MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.806/6.864 Advanced Natural Language Processing
Fall 2017

Assignment 2, Due: 9am on Thursday Oct.12. Upload pdf or scan to Stellar.

Hidden Markov Models

In many practical problems, we would like to model pairs of sequences. Consider, for instance, the task of part-of-speech (POS) tagging. Given a sentence, we would like to compute the corresponding tag sequence:

Input:"Faith is a fine invention"

Output:"Faith/N is/V a/D fine/A invention/N"

More generally, a *sequence labeling problem* involves mapping a sequence of observations $x_1, x_2, \ldots, x_n$ into a sequence of tags $y_1, y_2, \ldots, y_n$. In the example above, every word x is tagged by a single label y. One possible approach for solving this problem would be to label each word independently. For instance, a classifier could predict a part-of-speech tag based on the word, its suffix, its position in the sentence, etc. In other words, we could construct a feature vector on the basis of the observed "context" for the tag, and use the feature vector in a linear classifier. However, tags in a sequence are dependent on each other and this classifier would make each tagging decision independently of other tags. We would like our model to directly incorporate these dependencies. For instance, in our example sentence, the word *"fine"* can be either noun (N), verb (V) or adjective (A). The label V is not suitable since a tag sequence "D V" is very unlikely. Today, we will look at a model – a Hidden Markov Model – that allows us to capture some of these correlations.

Generative Tagging Model

Assume a finite set of words Σ and a finite set of tags $\mathcal{T}$. Define S to be the set of all sequence tag pairs $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, $x_i \in \Sigma$ and $y_i \in \mathcal{T}$ for $i = 1 \ldots n$. S here contains sequences of different lengths as well, i.e., n varies as well. A generative tagging model is a probability distribution p over pairs of sequences:

- $p(x_1, \ldots, x_n, y_1, \ldots, y_n) \geq 0, \ \forall (x_1, \ldots, x_n, y_1, \ldots, y_n) \in S$
- $\sum_{(x_1,\ldots,x_n,y_1,\ldots,y_n)\in S} p(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1$

If we have such a distribution, then we can use it to predict the most likely sequence of tags $y_1, \ldots, y_n$ for any observed sequence of words $x_1, \ldots, x_n$, as follows

$$f(x_1, \ldots, x_n) = \text{argmax}_{y_1,\ldots,y_n} \, p(x_1, \ldots, x_n, y_1, \ldots, y_n) \tag{1}$$

where we view f as a mapping from word sequences to tags.

Three key questions:

- How to specify $p(x_1, \ldots, x_n, y_1, \ldots, y_n)$ with a few number of parameters (degrees of freedom)
- How to estimate the parameters in this model based on observed sequences of words (and tags).
- How to predict, i.e., how to find the most likely sequence of tags for any observed sequence of words: evaluate $\text{argmax}_{y_1,\ldots,y_n} \, p(x_1, \ldots, x_n, y_1, \ldots, y_n)$

Model definition

Let $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ be sequences of random variables of length n. We wish to specify a joint probability distribution

$$P(X_1 = x_1, \ldots, X_n = x_n, Y_1 = y_1, \ldots, Y_n = y_n) \tag{2}$$

where $x_i \in \Sigma$, $y_i \in \mathcal{T}$. For brevity, we will write it as $p(x_1, \ldots, x_n, y_1, \ldots, y_n)$, i.e., treat it as a function of values of the random variables without explicating the variables themselves. We will define one additional random variable Y_{n+1}, which always takes the value STOP. Since our model is over variable length sequences, we will use the end symbol to model when to stop. In other words, if we observe $x_1, \ldots, x_n$, then clearly the symbol after $y_1, \ldots, y_n$, i.e., y_{n+1}, had to be STOP (otherwise we would have continued generating more symbols). We do not include STOP in $\mathcal{T}$.

Now, let's start by rewriting the distribution a bit according to general rules that apply to any distribution. The goal is to put the distribution in a form where we can easily explicate our assumptions. First,

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = p(y_1, \ldots, y_{n+1}) p(x_1, \ldots, x_n | y_1, \ldots, y_{n+1}) \qquad \text{(chain rule)}$$

Then we will use the chain rule repeatedly along the sequence of tags

$$p(y_1, \ldots, y_{n+1}) = p(y_1) p(y_2|y_1) p(y_3|y_1, y_2) \ldots p(y_{n+1}|y_1, \ldots, y_n) \qquad \text{(chain rule)}$$

So far, we have made no assumptions about the distribution at all. Since we don't expect tags to have very long dependences along the sequence, we will simply say that the next tag only depends on the current tag. In other words, we will "drop" the dependence on tags further back

$$\begin{aligned} p(y_1, \ldots, y_{n+1}) &\approx p(y_1) p(y_2|y_1) p(y_3|y_2) \ldots p(y_{n+1}|y_n) \qquad \text{(independence assumption)} \\ &= \prod_{i=1}^{n+1} p(y_i|y_{i-1}) \end{aligned}$$

Put another way, we assume that the tags form a Markov sequence (future tags are independent of the past tags given the current one). Let's now make additional assumptions about the observations as well

$$\begin{aligned} p(x_1, \ldots, x_n | y_1, \ldots, y_{n+1}) &= \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}, y_1, \ldots, y_{n+1}) \qquad \text{(chain rule)} \\ &\approx \prod_{i=1}^{n} p(x_i|y_i) \qquad \text{(independence assumption)} \end{aligned}$$

In other words, we say that the identity of each word only depends on the corresponding tags. This is a drastic assumption but still (often) leads to a reasonable tagging model, and simplifies our calculations. A more formal statement here is that the random variable X_i is conditionally independent of all the other variables in the model given Y_i (see more on conditional independence in the Bayesian networks lectures).

Now, we have a much simpler tagging model

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = p(y_1) \prod_{i=2}^{n+1} p(y_i|y_{i-1}) \prod_{i=1}^{n} p(x_i|y_i) \tag{3}$$

For notational convenience, we also assume a special fixed START symbol $y_0 = *$ so that $p(y_1)$ becomes $p(y_1|y_0)$. As a result, we can write

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} p(y_i|y_{i-1}) \prod_{i=1}^{n} p(x_i|y_i) \tag{4}$$

Let's understand this model a bit more carefully by looking at how the pairs of sequences could be generated from the model. Here's the recipe

1. Set $y_0 = *$ (we always start from the START symbol) and let $i = 1$.
2. Generate tag y_i from the conditional distribution $p(y_i|y_{i-1})$ where y_{i-1} already has a value (e.g., $y_0 = *$ when $i = 1$)
3. If $y_i = \text{STOP}$, we halt the process and return $y_1, \ldots, y_i, x_1, \ldots, x_{i-1}$. Otherwise we generate x_i from the output/emission distribution $p(x_i|y_i)$
4. Set $i = i + 1$, and return to step 2.

HMM formal definition

The model we have defined is a Hidden Markov Model or HMM for short. An HMM is defined by a tuple $\langle N, \Sigma, \theta \rangle$, where

- N is the number of states $1, \ldots, N$ (assume the last state N is the final state, i.e. what we called "STOP" earlier).
- Σ is the alphabet of output symbols. For example, $\Sigma = \{$"the", "dog"$\}$.
- $\theta = \langle a, b, \pi \rangle$ consists of three sets of parameters
  - Parameter $a_{i,j} = p(y_{next} = j|y = i)$ for $i = 1, \ldots, N-1$ and $j = 1, \ldots, N$ is the probability of transitioning from state i to state j: $\sum_{k=1}^{N} a_{i,k} = 1$
  - Parameter $b_j(o) = p(x = o|y = j)$ for $j = 1 \ldots N-1$ and $o \in \Sigma$ is the probability of emitting symbol o from state j: $\sum_{o \in \Sigma} b_j(o) = 1$.
  - Parameter $\pi_i = p(y_1 = i)$ for $i = 1 \ldots N$ specifies probability of starting at state i: $\sum_{i=1}^{N} \pi_i = 1$.

  Transition, emission, and starting probabilities can be zero for certain symbols/states.

(Question 1.1) [1 points] Write an expression for the dimensionality (i.e. number of scalar parameters) of θ as a function of N and Σ.

Example:

- N = 3. States are $\{1, 2, 3\}$
- Alphabet $\Sigma = \{the, dog\}$

- Distribution over initial states: $\pi_1 = 1, \pi_2 = \pi_3 = 0$.
- Parameters $a_{i,j}$ are

	$j=1$	$j=2$	$j=3$
$i=1$	0.5	0.5	0
$i=2$	0	0.8	0.2

- Parameters $b_j(o)$ are

	$o=$the	$o=$dog
$i=1$	0.9	0.1
$i=2$	0.1	0.9

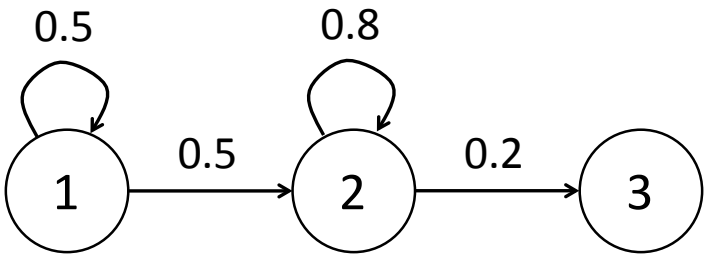

Figure 1: Transition graph for the example.

An HMM specifies a probability for each possible (x, y) pair, where $x = (x_1, \ldots, x_n)$ is a sequence of symbols drawn from Σ and $y = (y_1, \ldots, y_n)$ is a sequence of states drawn from the integers $1, \ldots, (N-1)$.

$$
\begin{aligned}
p(x, y|\theta) = & \pi_{y_1} \cdot && \text{(prob. of choosing } y_1 \text{ as an initial step)} \\
& a_{y_n, N} \cdot && \text{(prob. of transitioning to the final step)} \\
& \prod_{i=2}^{n} a_{y_{i-1}, y_i} \cdot && \text{(transition probability)} \\
& \prod_{i=1}^{n} b_{y_i}(x_i) && \text{(emission probability)}
\end{aligned}
$$

(Question 1.2) [1 points] Compute the probability of the following sequence: "the/1, dog/2, the/1", where "/1" and "/2" refer to the states in Figure 1.

Parameter estimation

We will first look at the fully observed case (complete data case), where our training data contains both xs and ys. We will do maximum likelihood estimation. Consider the examples: $\Sigma = \{e, f, g, h\}, N = 3$. Observation: $(e/1, g/2), (e/1, h/2), (f/1, h/2), (f/1, g/2)$. To find the MLE, we will simply look at the

counts of events, i.e., the number of transitions between tags, the number of times we saw an output symbol together with an specific tag (state). After normalizing the counts to yield valid probability estimates, we get

$$a_{i,j} = \frac{count(i,j)}{count(i)} \tag{5}$$

$$a_{1,2} = \frac{count(1,2)}{count(1)} = \frac{4}{4} = 1,\ a_{2,2} = \frac{count(2,2)}{count(2)} = \frac{0}{4} = 0, \ldots \tag{6}$$

where $count(i,j)$ is the number of times we have (i,j) as two successive states and $count(i)$ is the number of times state i appears in the sequence. Similarly,

$$b_i(o) = \frac{count(i \to o)}{count(i)} \tag{7}$$

$$b_1(e) = \frac{count(1 \to e)}{count(1)} = \frac{2}{4} = 0.5, \ldots \tag{8}$$

where $count(i \to o)$ is the number of times we see that state i emits symbol o. $count(i)$ was already defined above.

Decoding with HMM

Suppose now that we have the HMM parameters θ (see above) and the problem is to infer the underlying tags $y_1, \ldots, y_n$ corresponding to an observed sequence of words $x_1, \ldots, x_n$. In other words, we wish to evaluate

$$\text{argmax}_{y_1,\ldots,y_n}\, p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) \tag{9}$$

where

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} a_{y_{i-1},y_i} \prod_{i=1}^{n} b_{y_i}(x_i) \tag{10}$$

and $y_0 = *, y_{n+1} = \text{STOP}$. Note that by defining a fixed starting state, we have again folded the initial state distribution π into the transition probabilities $\pi_i = a_{*,i}$, $i \in \{1, \ldots, N\}$ (where N =STOP).

One possible solution for finding the most likely sequence of tags is to do brute force enumeration. Consider the example: $\Sigma = \{\text{the, dog}\}$, $x =$"the the the dog". The possible state sequences include:

$$1\ 1\ 1\ 2\ \text{STOP} \tag{11}$$

$$1\ 1\ 2\ 2\ \text{STOP} \tag{12}$$

$$1\ 2\ 2\ 2\ \text{STOP} \tag{13}$$

$$\vdots \tag{14}$$

(Question 2.1) [1 points] How many possible tag sequences can generate string of length n, assuming the size of tag alphabet to be $|\mathcal{T}|$?

Solving the tagging problem by enumerating the tag sequences will be prohibitively expensive.

Viterbi algorithm

The HMM has a simple dependence structure (recall, tags form a Markov sequence, observations only depend on the underlying tag). We can exploit this structure in a dynamic programming algorithm.

Input: $x = x_1, \ldots, x_n$ and model parameters θ.
Output: $\text{argmax}_{y_1,\ldots,y_{n+1}} p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1})$.

Now, let's look at a truncated version of the joint probability, focusing on the first k tags for any $k \in \{1, \ldots, n\}$. In other words, we define

$$r(y_1, \ldots, y_k) = \prod_{i=1}^{k} a_{y_{i-1},y_i} \prod_{i=1}^{k} b_{y_i}(x_i) \tag{15}$$

where y_k does not equal STOP. Note that our notation $r(y_1, \ldots, y_k)$ suppresses any dependence on the observation sequence. This is because we view $x_1, \ldots, x_n$ as given (fixed). Note that, according to our definition,

$$\begin{aligned} p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) =& r(y_1, \ldots, y_n) \cdot a_{y_n,y_{n+1}} \\ =& r(y_1, \ldots, y_n) \cdot a_{y_n,\text{STOP}} \end{aligned}$$

Let $S(k, v)$ be the set of tag sequences $y_1, \ldots, y_k$ such that $y_k = v$. In other words, $S(k, v)$ is a set of all sequences of length k whose last tag is v. The dynamic programming algorithm will calculate

$$\pi(k, v) = \max_{(y_1,\ldots,y_k) \in S(k,v)} r(y_1, \ldots, y_k) \tag{16}$$

recursively in the forward direction. In other words, $\pi(k, v)$ can be thought as solving the maximization problem partially, over all the tags $y_1, \ldots, y_{k-1}$ with the constraint that we use tag v for y_k. If we have $\pi(k, v)$, then $\max_v \pi(k, v)$ evaluates $\max_{y_1,\ldots,y_k} r(y_1, \ldots, y_k)$. We leave v in the definition of $\pi(k, v)$ so that we can extend the maximization one step further as we unravel the model in the forward direction. More formally,

- Base case reflects the assumption that $y_0 = *$.

  (Question 2.2) [1 points] Complete the following entries:

  $$\begin{aligned} \pi(0, *) =& \quad \text{(starting state, no observations)} \\ \pi(0, v) =& \quad \text{if } v \neq * \text{ (an actual state has observations)} \end{aligned}$$

- Moving forward recursively: for any $k \in \{1, \ldots, n\}$

  $$\pi(k, v) = \max_{u \in \mathcal{T}} \{\pi(k-1, u) \cdot a_{u,v} \cdot b_v(x_k)\} \tag{17}$$

  In other words, when extending $\pi(k-1, u)$, $u \in \mathcal{T}$, to $\pi(k, v)$, $v \in \mathcal{T}$, we must transition from $y_{k-1} = u$ to $y_k = v$ (part $a_{u,v}$) and generate the corresponding observation x_k (part $b_v(x_k)$). Then we maximize over the previous tag $y_{k-1} = u$ so that $\pi(k, v)$ only depends on the value of y_k.

(Question 2.3) [5 points] Provide detailed derivations for the Viterbi iteration, i.e., show why we have

$$\pi(k, v) = \max_{u \in \mathcal{T}} \{\pi(k-1, u) \cdot a_{u,v} \cdot b_v(x_k)\} \tag{18}$$

Finally, we must transition from y_n to STOP so that

$$\max_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_1, \ldots, y_n, y_{n+1} = \text{STOP}) = \max_{v \in \mathcal{T}} \{\pi(n, v) \cdot a_{v,\text{STOP}}\} \tag{19}$$

Now, having values $\pi(k, v)$, how do we reconstruct the most likely sequence of tags which we denote as $\hat{y}_1, \ldots, \hat{y}_n$? We can do this via *back-tracking*. In other words, at the last step, $\pi(n, v)$ represents maximizations of all $y_1, \ldots, y_n$ such that $y_n = v$. What is the best value for this last tag v, i.e., what is $\hat{y}_n$? It is

$$\hat{y}_n = \text{argmax}_v \left\{\pi(n, v) a_{v,\text{STOP}}\right\} \tag{20}$$

Now we can fix $\hat{y}_n$ and work backwards. What is the best value $\hat{y}_{n-1}$ such that we end up with tag $\hat{y}_n$ in position n? It is simply

$$\hat{y}_{n-1} = \text{argmax}_u \left\{\pi(n-1, u) a_{u,\hat{y}_n}\right\} \tag{21}$$

and so on.

(Question 2.4) [1 points] What is time complexity of Viterbi algorithm, as a function of tag set size $|\mathcal{T}|$ and sentence length n?

Hidden Variable Problem

When we no longer have the tags, we must resort to other ways of estimating the HMMs. It is not trivial to construct a model that agrees with the observations except in very simple scenarios. Here is one:

- We have an HMM with $N = 3, \Sigma = \{a, b, c\}$
- We see the following output sequences in training data: $(a, b), (a, c), (a, b)$.

How would you choose the parameter values for $\pi_i, a_{i,j}$ and $b_i(o)$? A reasonable choice is:

$$\pi_1 = 1.0, \pi_2 = \pi_3 = 0 \tag{22}$$

$$b_1(a) = 1.0, b_1(b) = b_1(c) = 0 \tag{23}$$

$$b_2(a) = 0, b_2(b) = b_2(c) = 0.5 \tag{24}$$

$$a_{1,2} = 1.0, a_{1,1} = a_{1,3} = 0 \tag{25}$$

$$a_{2,3} = 1.0, a_{2,1} = a_{2,2} = 0 \tag{26}$$

Expectation-Maximization (EM) for HMM

Suppose now that we have multiple observed sequences of outputs (no observed tags). We will denote these sequences with superscripts, i.e., $x^1, x^2, \ldots, x^m$. We will use subscripts to denote the symbols in a sequence i, i.e., $x_1^i, x_2^i, \ldots, x_n^i$. In the context of each sequence, we must evaluate a posterior probability over possible tag sequences. For estimation, we only need expected counts that are used in the re-estimation step (M-step). To this end, let count$(x^i, y, p \to q)$ be the numbers of times a transition from state p to state q occurs in a tag sequence y corresponding to observation x^i. We will only show here the derivations for transition probabilities; the equations for emission and initial state parameters are obtained analogously.

E-step: calculate expected counts, added across sequences

(Question 3.1) [4 points] Using count$(\cdot, \cdot, \cdot)$ and θ, complete the following expression:

$$\overline{count}(u \to v) = \tag{27}$$

M-step: re-estimate transition probabilities based on the expected counts

(Question 3.2) [4 points] Complete the following expression:

$$a_{u,v} = \tag{28}$$

The main problem in running the EM algorithm is calculating the sum over the possible tag sequences in the E-step.

The sum is over an exponential number of possible hidden state sequences y. Next we will discuss a dynamic programming algorithm – forward-backward algorithm. The algorithm is analogous to the Viterbi algorithm for maximizing over the hidden states.

The Forward-Backward Algorithm for HMMs[1]

Suppose we could efficiently calculate marginal posterior probabilities

$$p(y_j = p, y_{j+1} = q|x, \theta) = \sum_{y: y_j = p, y_{j+1} = q} p(y|x, \theta) \tag{29}$$

for any $p \in \{1, \ldots, (N-1)\}, q \in \{1, \ldots, N\}, j \in \{1, \ldots, n\}$. These are the posterior probabilities that the state in position j was p and we transitioned into q at the next step. The probability is conditioned on the observed sequence x and the current setting of the model parameters θ. Now, under this assumption, we could rewrite the difficult computation of fractional counts as:

$$\sum_y p(y|x^i, \theta^{t-1}) count(x^i, y, p \to q) = \sum_{j=1}^{n} p(y_j = p, y_{j+1} = q|x^i, \theta^{t-1}) \tag{30}$$

[1] Supplementary material: https://people.csail.mit.edu/rameshvs/content/hmms.pdf

The key remaining question is how to calculate these posterior marginals effectively. In other words, our goal is to evaluate $p(y_j = p, y_{j+1} = q|x^i, \theta^{t-1})$.

Now, consider a single observation sequence $x_1, \ldots, x_n$. We will make use of the following forward probabilities:

$$\alpha_p(j) = p(x_1, \ldots, x_{j-1}, y_j = p|\theta) \tag{31}$$

for all $j \in \{1, \ldots, n\}$, for all $p \in \{1, \ldots, N-1\}$. $\alpha_p(j)$ is the probability of emitting the symbols $x_1, \ldots, x_{j-1}$ and ending in state p in position j without (yet) emitting the corresponding output symbol. These are analogous to the $\pi(k, v)$ probabilities in the Viterbi algorithm with the exception that $\pi(k, v)$ included generating the corresponding observation in position k. Note that, unlike before, we are summing over all the possible sequences of states that could give rise to the observations $x_1, \ldots, x_{j-1}$. In the Viterbi algorithm, we maximized over the tag sequences.

Similarly to the forward probabilities, we can define the backward probabilities:

$$\beta_p(j) = p(x_j, \ldots, x_n|y_j = p, \theta) \tag{32}$$

for all $j \in \{1, \ldots, n\}$, for all $p \in \{1, \ldots, N-1\}$. $\beta_p(j)$ is the probability of emitting symbols $x_j, \ldots, x_n$ and transitioning into the final (STOP) state, given that we begun in state p in position j. Again, this definition involves summing over all the tag sequences that could have generated the observations from x_j onwards, provided that the tag at j is p.

Why are these two definitions useful? Suppose we had been able to evaluate α and β probabilities effectively. Then the marginal probability we were after could be calculated as:

$$p(y_j = p, y_{j+1} = q|x, \theta) = \frac{1}{Z}\alpha_p(j)a_{p,q}b_p(o_j)\beta_q(j+1)$$
$$Z = p(x_1, \ldots, x_n|\theta) = \sum_p \alpha_p(j)\beta_p(j) \text{ for any } j = 1 \ldots n$$

This is just the sum over all possible tag sequences that include the transition $y_j = p$ and $y_{j+1} = q$ and generates the observations, divided by the sum over all tag sequences that generate the observations. As a result, we obtain the relative probability of the transition, relative to all the alternatives given the observations, i.e., the posterior probability. Note that $\alpha_p(j)$ involves all the summations over tags $y_1, \ldots, y_{j-1}$, and $\beta_q(j+1)$ involves all the summations over the tags $y_{j+2}, \ldots, y_n$.

Let's finally discuss how we can calculate α and β.

As Fig. 2 shows, for every state sequence $y_1, y_2, \ldots, y_n$ there is

- a path through graph that has the sequence of states $s, \langle 1, y_1\rangle, \ldots, \langle n, y_n\rangle, f$.
- The path associated with state sequence $y_1, \ldots, y_n$ has weight equal to $p(x, y|\theta)$.
- $\alpha_p(j)$ is the sum of weights at all paths from s to the state (j, p).
- $\beta_p(j)$ is the sum of weights at all paths from state $\langle j, p\rangle$ to the final state f.

Given an input sequence $x_1, \ldots, x_n$, for any $p \in \{1, \ldots, N\}, j \in \{1, \ldots, n\}$, the forward and backward probability can be calculated recursively.

Forward probability:

$$\alpha_p(j) = p(x_1, \ldots, x_{j-1}, y_j = p|\theta) \tag{33}$$

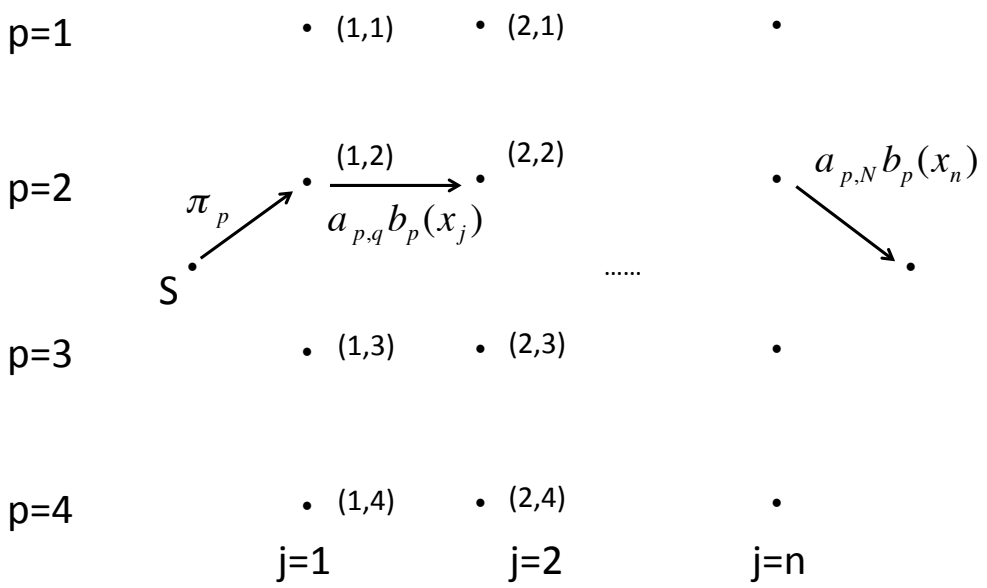

Figure 2: A path associated with state sequence.

- Base case:

$$\alpha_p(1) = \pi_p, \ \ \forall p \in \{1, \ldots, N-1\} \tag{34}$$

- Recursive case

$$\alpha_p(j+1) = \sum_q \alpha_q(j) a_{q,p} b_q(x_j) \ \ \forall p \in \{1, \ldots, N-1\}, j \in \{1, \ldots, n-1\} \tag{35}$$

Backward probability:

$$\beta_p(j) = p(x_j, \ldots, x_n | y_j = p, \theta) \tag{36}$$

- Base case:

$$\beta_p(n) = a_{p,N} b_p(x_n), \ \ \forall p \in \{1, \ldots, N-1\} \tag{37}$$

- Recursive case

$$\beta_p(j) = \sum_q a_{p,q} b_p(x_j) \beta_q(j+1), \ \ \forall p \in \{1, \ldots, N-1\}, j \in \{1, \ldots, n-1\} \tag{38}$$

(Question 3.3) [2 points] Show that $p(x_1, \ldots, x_n | \theta) = \sum_p \alpha_p(i)\beta_p(i)$ for any $i \in \{1, \ldots, n\}$.

(Question 3.4) [5 points] For $i \in \{1, \ldots, n-1\}$ and $p \in \{1, \ldots, N-1\}$, show that

$$p(y_i = p | x_1, \ldots, x_n, \theta) = \frac{\alpha_p(i)\beta_p(i)}{\sum_q \alpha_q(i)\beta_q(i)} \, . \tag{39}$$

Neural network classifier with word embeddings

The next problem is to be solved on the dataset provided in the previous assignment for the task of sentiment analysis.

You should develop the code in the following directory structure.

```
assignment2
└── sentiment
    ├── src
    ├── data
    └── word_vectors
```

You will need to submit only a zipped src directory as a part of the solutions along with the Answers.pdf report.

In this section, you will train a fully connected feed forward network classifier for sentiment analysis. As a part of this problem, you will understand the impact of various hyper-parameters and regularization techniques on a basic feed forward network. You will explore the use of word embedding features.

To implement a feed forward network based classifier, you will use PyTorch. You can refer to `http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html` and `http://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html` for inspiration.

Neural Network Architecture: The neural network will consist of three layers: input layer, hidden layer and output layer. The connection between the layers will be through tanh activation function. The input layer will contain nodes equal to the number of features. The output layer will contain 2 nodes and you will use LogSoftmax activation function for predicting the sentiment class label. You will use the NLLLoss function to train the model using the Adam optimizer. The architecture in PyTorch can be specified in the following steps

1. Input to Hidden Layer : Linear(input_dim, hidden_dim)
2. Tanh activation
3. Hidden to Output Layer : Linear(hidden_dim, 2)
4. LogSoftmax on top of the output layer

You are free to modify the activation functions, and even add activation functions before or after few layers for non-linearity. However, this framework and the modifications specified later should suffice.

Features: Word Embeddings provide a more meaningful representation than one-hot representations of words. As mentioned in class, the basic idea is that we represent a word with a vector ϕ defined by the context the word appears in. Further details on the training of word embeddings will be covered later in the course. Here, you will be using off-the-shelf (pretrained) Glove [1] word embeddings provided to you. These are 300 dimension word embeddings, and are stored in a per line format of *word vector* per line. Format of every line of glove.npz is *word+" "+embedding*. The embedding is also a space separated list of 300 floating point numbers. If there is no embedding provided for a word, you must ignore that particular word.

Training Details: You should train your network for a maximum of 50 iterations. It would be advised to use a large batch size of around 100. (173 might actually be a convenient number, so that you have 40 training

batches). While stating results, you will state (Dev and Test Accuracies % as in the previous homework.) Thus, the network will be optimized num_iterations * num_batches times. And you evaluate your model at the end of every epoch, where you have optimized the network over all the batches in the training set. **To control the randomness of initializations of the variables, set an initialization seed in torch. This can be done by specifying torch.manual_seed(1) in your code.**

Specific tasks are given below.

1. Let d be the size of the input layer (feature vector size = 300). Begin with an initial hidden layer size of d/2 nodes. Now, change the hidden layer size using values from the set $\{d/4, d/2, d, 2d\}$.
2. Learning rate - Pick a learning rate from the set $\{1e-5, 1e-3, 1e-1, 1e1\}$ for your experiments. This is a constant lr defined in your optimizer initialization.
3. A common regularization approach used is to include the norm (L2 norm) of the weights of the neural network in the total loss. The L2 norm portion of the loss is multiplied by a constant. This constant specifies the importance of the regularization loss with respect to the task loss. In PyTorch, including this is quite straight forward, by just specifying the weight_decay parameter value in the optimizer initialization. Pick the weight_decay constant from the set $\{1e-5, 1e-3, 1e1\}$.

(Question 4.1) [15 points] Compute the results on the Dev set for all combinations of the hyper-parameter values specified in the previous parts and pick the best Dev performing model. For each setting, you will train and evaluate the model from scratch for 50 iterations. *Specify the combination which works the best. In the answer set specify both Dev and Test accuracies for this combination.*

(Question 4.2) [10 points] For the best combination of hyper-parameters found, fix values of 2 parameters, and for every value of the third parameter, plot a graph of Dev Performance (Y-axis) and Values (X-axis). You will have 3 plots in all. *Submit the 3 graphs and state the performance trend corresponding to each parameter.* Note that you might find different variances for the 3 hyper-parameters.

Submit the code for this problem (only the best performing model architecture) in P4.py in the src directory. In the report explain your model architecture in detail.

Sentiment analysis to other domains

(Question 5.1) [15 points]

1. You have just seen several models for sentiment analysis of movie reviews. Will these models used work for other domains such as restaurant reviews, or hotel reviews (short reviews, similar to the ones you are given in the current dataset and expressing an overall sentiment towards the experience of the restaurant or hotel)? You don't need to run any experiments on any new dataset. *Justify your intuition.*
2. Why would the models you implemented not be a good idea for reviews with multiple aspects, in which you try to predict the sentiment of a specific aspect? For example, if it was for beer reviews and there was a sentiment for smell, a sentiment for appearance and a sentiment for taste in one single paragraph. How would you modify your model, to deal with this ?

Representations of text

This problem is for graduate students only.

For the sentiment analysis problems, to model text, you have looked at different representations of the text. These representations or encoders of text were kept fixed. They did not vary during the training phase; only the subsequent weights of the neural network or the coefficients of a logistic regression classifier were learned. For instance, the bag of words representation of a piece of text remained the same - before and after training. The word embeddings average feature representation of a piece of text remained the same - before and after training. In more complex models, the representations of text are also updated as a part of the training. As a part of this problem, you are given the following paper to read - `https://arxiv.org/pdf/1704.00051.pdf` which covers Machine Reading, the task of automating the process of answering questions from pieces of text. You will need a high level understanding of the paper to answer the following questions. Answers to the questions will be found in Sections 3.2 and 5.2 of the paper, and are pertaining to the Document Reader part of the system ONLY. The paper also describes methods to make the task Open Domain, but you don't need to understand those in detail.

(Question 6.1) [3 points] Consider the Document Reader part of the system, described in Section 3.2. There are two categories of text that the subtask (machine reading) takes as input (i.e. an input pair). What are they ? What is the output expected ?

(Question 6.2) [3 points] What are the neural network encodings used to represent these categories of text ? (Exact models will be found in section 5.2, don't need to get into alignment details)

(Question 6.3) [4 points] What is the output's maximum possible length? What is equation for the score of each output ?

(Question 6.4) [2 points] For a particular input pair, from all its possible candidate outputs, on what basis is an output selected ?

(Question 6.5) [3 points] During training, for each data point, say we have a correct output, and a set of incorrect outputs. What should the above asked encodings learn to do for the score of a correct output versus the scores of the incorrect outputs ? Thus through this training, a meaningful representation of text is learnt in context of the machine reading subtask.

Submission details

1. Create and submit report with answers to questions **as a PDF named Answers.pdf**.
2. Submit report and code to Stellar **independently**. Submit the code in a zipped file. Do not include the data directory in the zipped file, only the src directory.

References

[1] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.