CSE 517: Homework 2 - HMM

Lilly Kumari

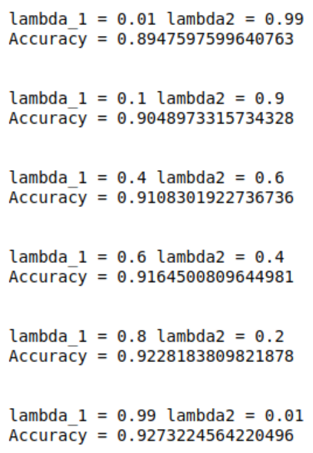
Problem 1. Bigram HMMs

$$\begin{aligned} p(\mathbf{s}, \mathbf{y}) &= p\left(x_1 x_2 \ldots x_n, y_1 y_2 \ldots y_{n+1}\right) \\ &= q\left(y_{n+1} = \mathrm{STOP} \mid y_n\right) \prod_{i=1}^{n} q\left(y_i \mid y_{i-1}\right) e\left(x_i \mid y_i\right) \end{aligned}$$

Solution Design Choices:

- Smoothing - used linear interpolation, bigram model weight ($\lambda_1 = 0.99$) and unigram model weight ($\lambda_2 = 0.01$). Unigram probability of START token is same as the unigram probability as STOP token.
- OOV handling - Used regex for replacing hashtag, mention, RT (retweet), emoticon, smiley, numbers and URL with their globally defined token such as <*hashtag*>. The words (text) which were not in the vocabulary constructed from the training data nor matching any of the above regular expressions were replaced by <*unk*> *token*. In the training data, after replacing words matching above mentioned regex, the words which appeared less than 2 times were replaced with an <*unk*> *token*.

Figure 1 shows the accuracy (on dev set) of a bigram (linear interpolation) HMM model trained on the train set corresponding to different values of lambdas.



```
lambda_1 = 0.01 lambda2 = 0.99
Accuracy = 0.8947597599640763

lambda_1 = 0.1 lambda2 = 0.9
Accuracy = 0.9048973315734328

lambda_1 = 0.4 lambda2 = 0.6
Accuracy = 0.9108301922736736

lambda_1 = 0.6 lambda2 = 0.4
Accuracy = 0.9164500809644981

lambda_1 = 0.8 lambda2 = 0.2
Accuracy = 0.9228183809821878

lambda_1 = 0.99 lambda2 = 0.01
Accuracy = 0.9273224564220496
```

Figure 1:

Best Model - bigram model weight ($\lambda_1 = 0.99$) and unigram model weight ($\lambda_2 = 0.01$) Accuracy on test set = 0.925

Error Analysis

- ([’STEPH’, ’ˆ´],) [’!’] — since this is a single word sentence which starts with a name, hence it’s really confusing to tag it either with a Noun tag or an exclamation tag.
- ([’Darling’, ’N’], [’Taylin’, ’N’], [’https://t.co/UeY9mzvxUf’, ’U’]) [’ ˆ´, ’ ˆ´, ’U’] — noun tags mixed up by Name tags. Since the words begin with a capital letter, the HMM model lacks in making distinction between Noun & Name based words and tags them with Name tags.
- ([’BEEF’, ’N’], [’KEEMA’, ’N’], [’STUFFED’, ’V’], [’CHAPPATI’, ’N’], [’https://t.co/Eu7rgPd5Oq’, ’U’]) [’ ˆ´, ’ ˆ´, ’V’, ’ ˆ´, ’U’] — the model correctly tags STUFFED with Verb tag but wrongly identifies KEEMA, CHAPPATI as Names, rather than Nouns.

Figure 2 shows the confusion matrix computed from the predicted POS tags on dev set.

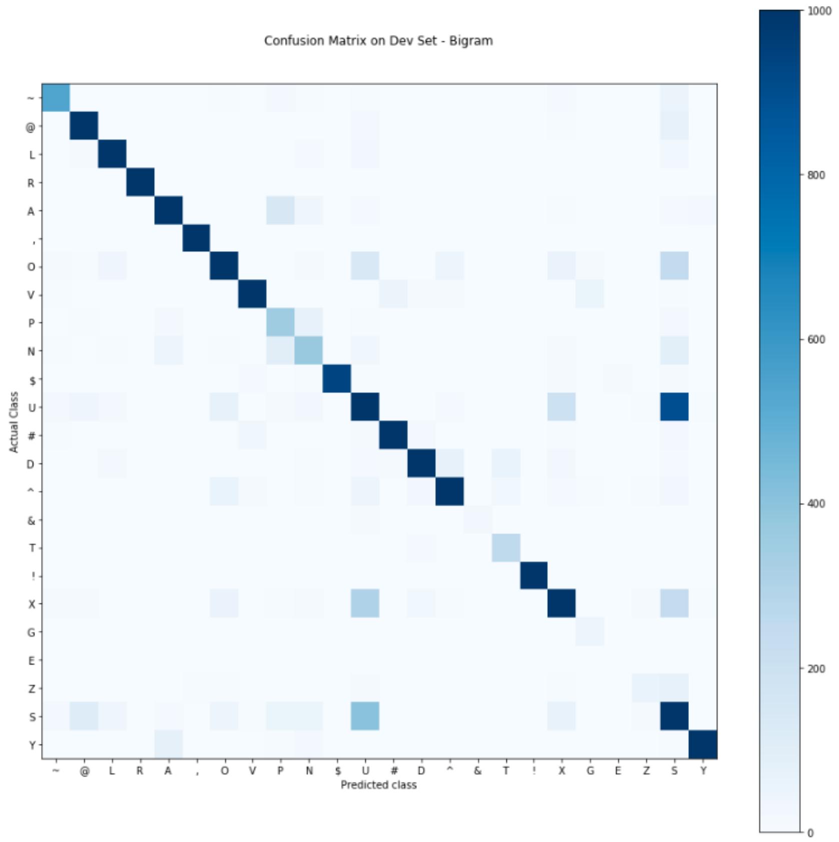


Figure 2:

Figure 3 shows the confusion matrix computed from the predicted POS tags on test set.

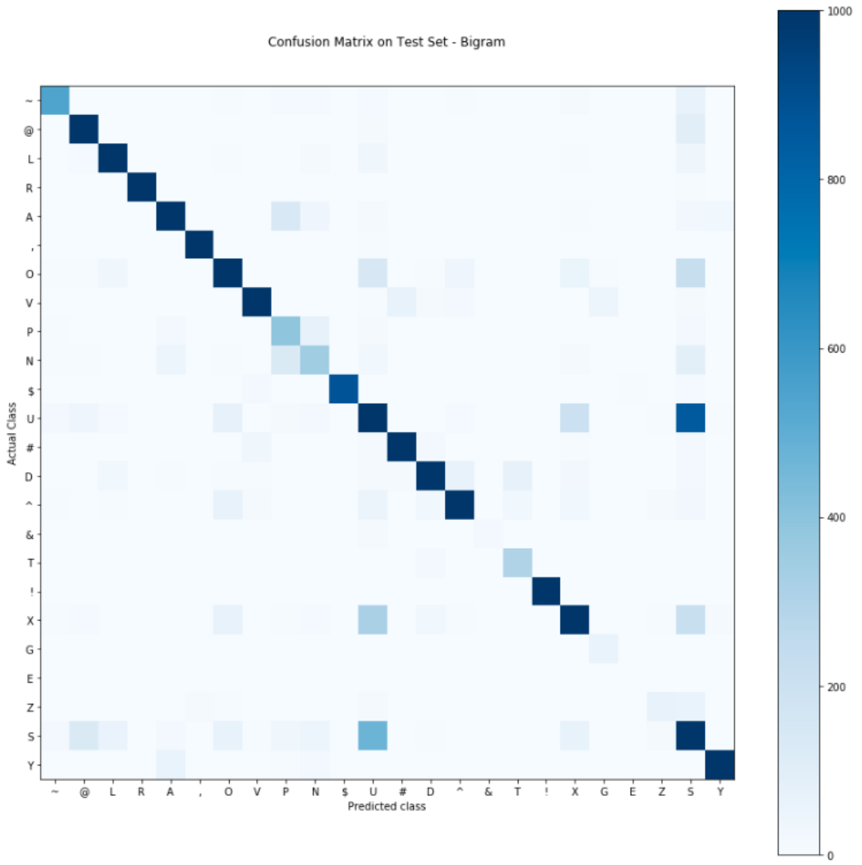


Figure 3:

Problem 2. Trigram HMMs

$$p(\mathbf{s}, \mathbf{y}) = p\left(x_1 x_2 \ldots x_n, y_1 y_2 \ldots y_{n+1}\right)$$

$$= q\left(y_{n+1} = \text{STOP} \mid y_n, y_{n-1}\right) \prod_{i=1}^{n} q\left(y_i | y_{i-1}, y_{i-2}\right) e\left(x_i | y_i\right)$$

where $y_0 = y_{-1} =< START >$

$$q(y_i|y_{i-1}, y_{i-2}) = \frac{c(y_i, y_{i-1}, y_{i-2})}{c(y_{i-1}, y_{i-2})}$$

$$e(x_i|y_i) = \frac{c(y_i \rightarrow x_i)}{c(y_i)}$$

Viterbi decoding algorithm for trigram HMM

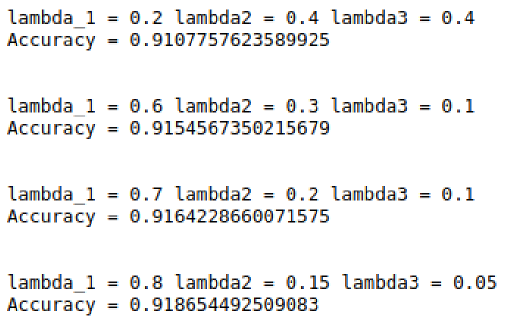
- Given a sentence $x_1 \ldots x_n$, and MLE estimates $q(s|u, v)$ and $e(x|s)$
- $\mathcal{K}_{-1} = \mathcal{K}_0 = \{START\}$ and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \ldots n$ where $\mathcal{K}$ is the set of possible POS tags
- Base case: Set $\pi(0, START, START) = 1$

```
• For $k = 1 \dots n$,
  – For $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$,
        $\pi(k,u,v) = \max_{w \in \mathcal{K}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e\left(x_k|v\right)\right)$
        $bp(k,u,v) = \arg\max_{w \in \mathcal{K}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e\left(x_k|v\right)\right)$
• Set $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n,u,v) \times q(\text{STOP}|u,v))$
• For $k = (n-2) \dots 1$,
  $y_k = bp\left(k+2, y_{k+1}, y_{k+2}\right)$
• Return $y_1 \dots y_n$
```

Design Choices:

- Smoothing - used linear interpolation,trigram model weight ($\lambda_1 = 0.8$), bigram model weight ($\lambda_2 = 0.15$) and unigram model weight ($\lambda_3 = 0.05$). If a bigram doesn't exist in the bigrams list, then its probability is taken as 0. Since I constructed a trigram smoothed dictionary which consists of all possible combinations of tags including START and STOP tokens, each combination has some probability score between 0 and 1. And that removed the task of handling of unknown bi-grams while testing on unseen data.
- OOV handling - Used regex for replacing hashtag, mention, RT (retweet), emoticon, smiley, numbers and URL with their globally defined token such as <*hashtag*>. The words (text) which were not in the vocabulary constructed from the training data nor matching any of the above regular expressions were replaced by <*unk*> *token*. In the training data, after replacing words matching above mentioned regex, the words which appeared less than 2 times were replaced with an <*unk*> *token*.

Figure 4 shows the accuracy (on dev set) of a trigram (linear interpolation) HMM model trained on the train set corresponding to different values of lambdas.



```
lambda_1 = 0.2 lambda2 = 0.4 lambda3 = 0.4
Accuracy = 0.9107757623589925


lambda_1 = 0.6 lambda2 = 0.3 lambda3 = 0.1
Accuracy = 0.9154567350215679


lambda_1 = 0.7 lambda2 = 0.2 lambda3 = 0.1
Accuracy = 0.9164228660071575


lambda_1 = 0.8 lambda2 = 0.15 lambda3 = 0.05
Accuracy = 0.918654492509083
```

Figure 4:

Best Model - trigram model weight ($\lambda_1 = 0.8$), bigram model weight ($\lambda_2 = 0.15$) and unigram model weight ($\lambda_3 = 0.05$)

Accuracy on test set = 0.916

Comparison of bigram and trigram HMM models Both the models perform almost similarily. For example, the given tweet (['Forex', 'N'], ['Triple', 'N'], ['Force', 'N'], ['Robot', 'N'], ['https://t.co/3hadhmCORw', 'U']) is labeled the same by both as [' ^´, ' ^´, ' ^´ , ' ^´ , 'U']. Both the models incorrectly tag Nouns with Name tags.

For the following tweet, (['Darling', 'N'], ['Taylin', 'N'], ['https://t.co/UeY9mzvxUf', 'U']), the bigram model identifies first two words as Names while the trigram identifies them as S (apostrophe) and G. So, neither of them always correctly identify a Noun word.

For the tweet, (['STEPH', '^´],), the trigram model correctly labels it with [' ^´] Name tag while the bigram model labels it with ['!'] exclamation tag.

The Bigram HMM accuracy on dev set and test set are 0.927 and 0.925 respectively. While the trigram HMM accuracy on dev set and test set are 0.919 and 0.916 respectively.

Figure 5 shows the confusion matrix computed from the predicted POS tags on dev set.

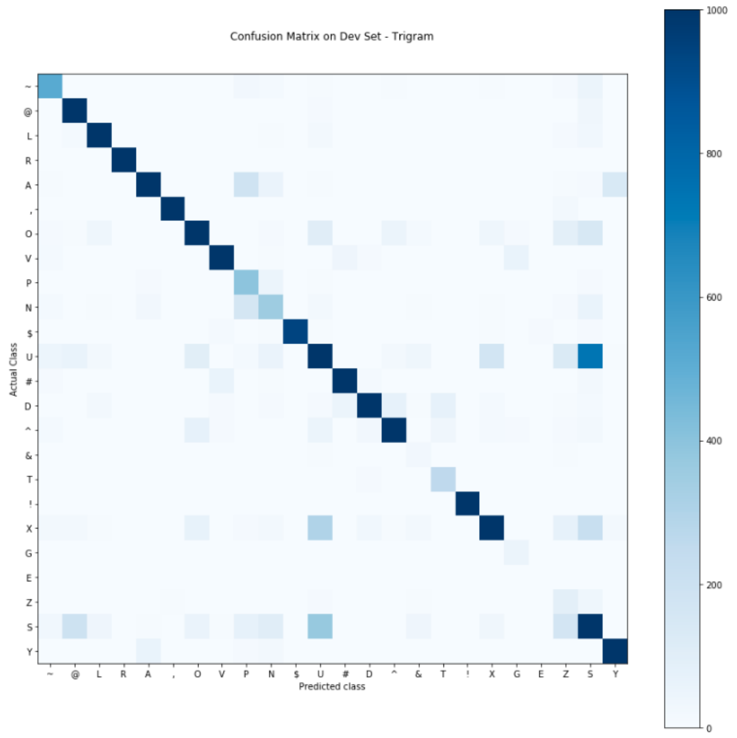


Figure 5:

Figure 6 shows the confusion matrix computed from the predicted POS tags on test set.

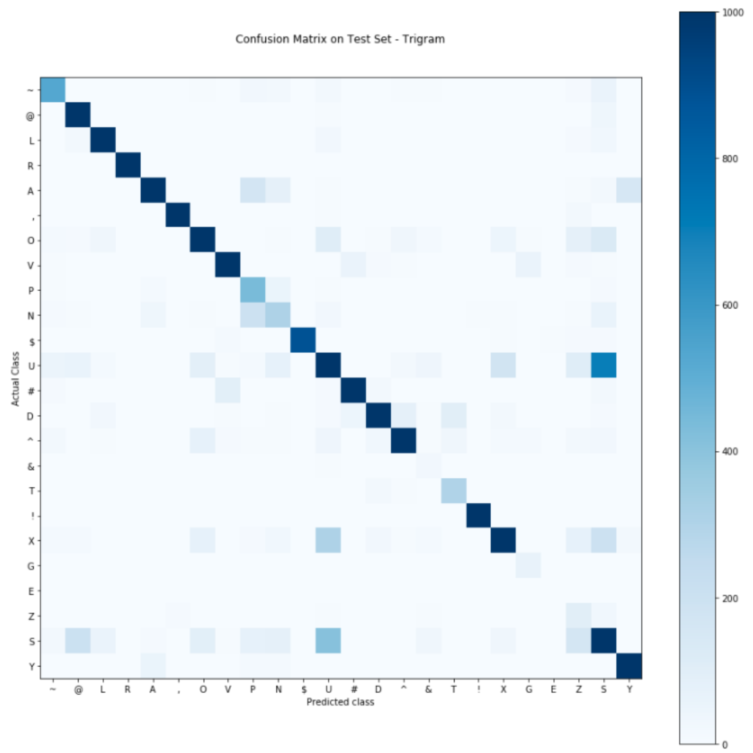


Figure 6:

Problem 4. PCFG Language Models

Solution PCFG language models benefit from the syntactic structure of a sentence, which tends to produce results which are more syntactically right at local and moreover global space. While N-gram language models work by modeling the probabilistic estimation of N, (N-1), ..., 1-gram models (using linear interpolation). This technique preserves the semantic structure at a local space, hence producing results which are more semantically correct but here the search is more constrained. *For example*, the sentence ”Show me the flights Ive Boston to San Francisco on Monday” is more or less semantically correct at local scale.

Based on the number of rules used in the PCFG grammar, the time complexity for a vanilla PCFG is $\mathcal{O}(|rules| * n^3)$ where *n* is the length of the sentence. This could be computationally expensive as compared to simple n-gram language model where the complexity of determining n-grams is linear with respect to the size of the corpus. But n-gram language models are memory-wise inefficient because of the sparsity problem which could be handled to a certain extent using smoothing techniques. In general, a vocabulary of size V will have V n-grams (e.g., 20,000 words will have 400 million bigrams, and 8 trillion trigrams!)

So, overall, both PCFG (P_{tree}) and N-grams ((P_{seq})) based language models have their own merits and limitations. Saying which one is best depends on the the problem and resources at hand.

Moreover, one can also combine both models to get semantically and syntactically correct sentences, which again depends on the resources available at disposal.

Problem 5. Playing with Off-the-shelf Parsers

Solution • PCFG parsers (parser used - http://tomato.banatao.berkeley.edu:8080/parser/parser.html)

– how to train your dragon to spit ice?

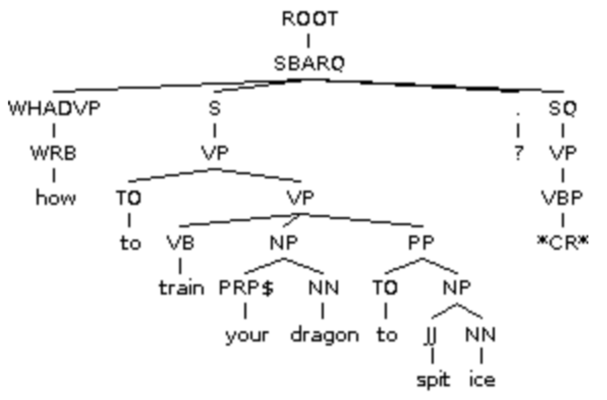


Figure 7:

Figure 7: (spit, ice) is tagged as NP, whereas it should have been VP. Also, (spit) is tagged as JJ (adjective) but it should have been VB.

– People sometimes get lost in their very own train of forgotten thoughts while thinking about their wants and needs.

Figure 8: (forgotten) tagged as VBN, it should be JJ (adjective), (wants and needs) tagged as VP, it should be linked to 'their'. (wants) and (needs) should be NN instead of VBZ.

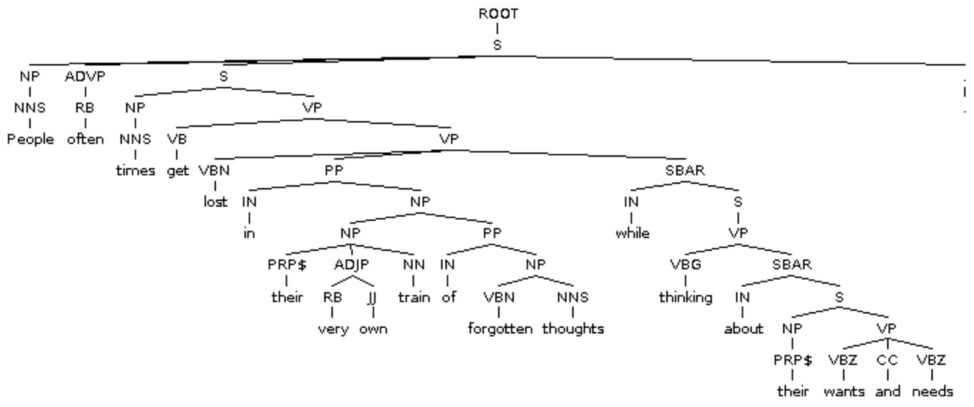


Figure 8:

• Dependency parsers (parser used - https://spacy.io/demos/displacy)

– The car houses undergraduates and graduates.

Figure 9: *nsubj* rule marked from (undergraduates) to (houses) is incorrect, it should have been from (houses) to (car). The *compound* rule is tagged incorrectly too.

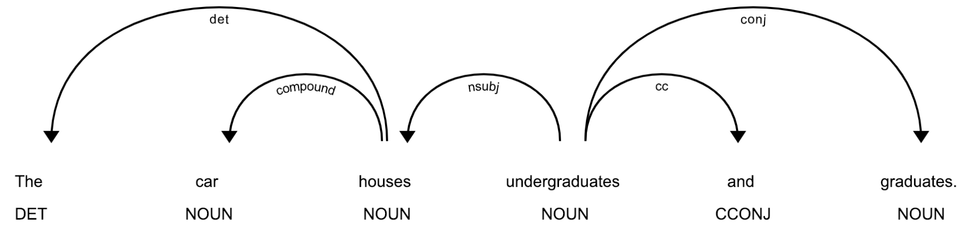


Figure 9:

- The left the right the center its always upto you to decide.
  Figure 10: *dobj* between (left, right), (left, center) are incorrectly associated. Also, (left, the) are incorrectly associated with *nsubj*.

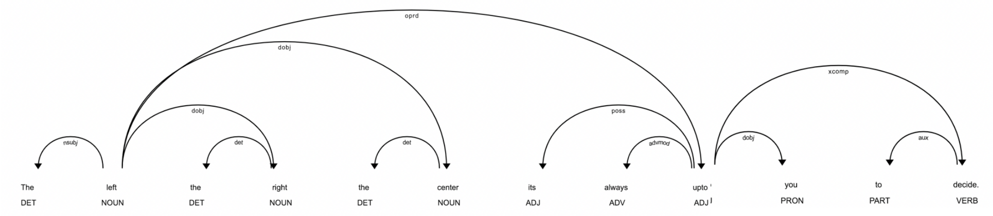
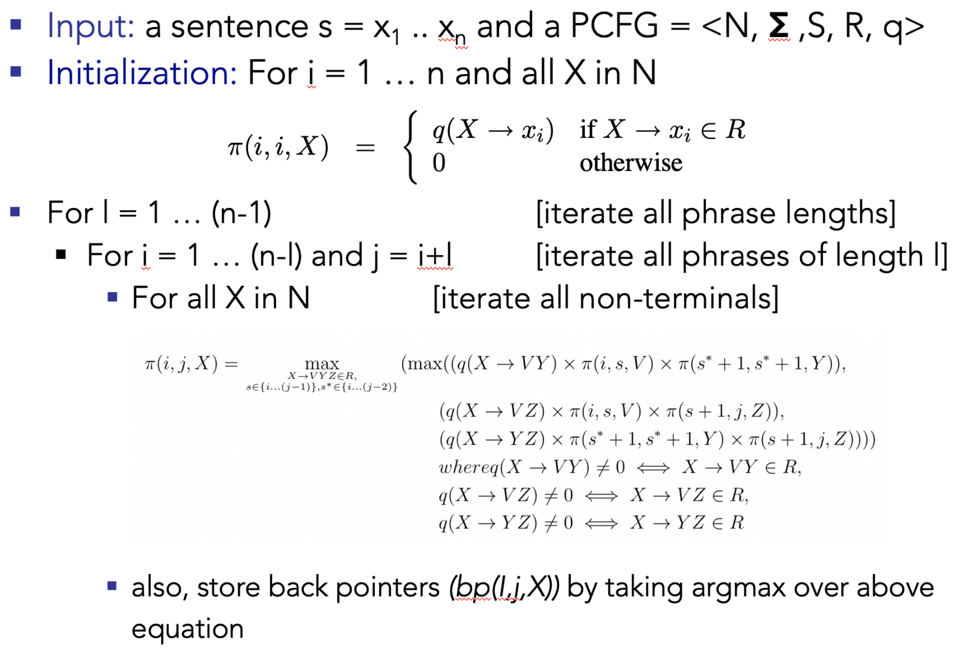


Figure 10:

Problem 6. Variations to CKY Algorithm

Solution Figure 11



```
▪ Input: a sentence s = $x_1$ .. $x_n$ and a PCFG = <N, Σ ,S, R, q>
▪ Initialization: For i = 1 … n and all X in N
      $\pi(i,i,X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$
▪ For l = 1 … (n-1)                          [iterate all phrase lengths]
    ▪ For i = 1 … (n-l) and j = i+l          [iterate all phrases of length l]
        ▪ For all X in N                     [iterate all non-terminals]
              $\pi(i,j,X) = \max_{X \rightarrow VYZ \in R,\ s \in \{i \ldots (j-1)\}, s^* \in \{i \ldots (j-2)\}} (\max((q(X \rightarrow VY) \times \pi(i,s,V) \times \pi(s^*+1,s^*+1,Y)),$
                  $(q(X \rightarrow VZ) \times \pi(i,s,V) \times \pi(s+1,j,Z)),$
                  $(q(X \rightarrow YZ) \times \pi(s^*+1,s^*+1,Y) \times \pi(s+1,j,Z))))$
                  $where q(X \rightarrow VY) \neq 0 \iff X \rightarrow VY \in R,$
                  $q(X \rightarrow VZ) \neq 0 \iff X \rightarrow VZ \in R,$
                  $q(X \rightarrow YZ) \neq 0 \iff X \rightarrow YZ \in R$
        ▪ also, store back pointers (bp(l,j,X)) by taking argmax over above equation
```

Figure 11: Pseudo code for modified CKY Algorithm