HOMEWORK 2
EXACT INFERENCE AND SUPERVISED LEARNING[1]

10-708 PROBABILISTIC GRAPHICAL MODELS (SPRING 2021)
http://708.mlcourse.org

OUT: Feb 24
DUE: March 10 at 11:59 PM
TAs: Alex, Helen, Xiang

START HERE: Instructions

Summary In this assignment, you will implement LSTM-CRF for labeling a constituency parse. Section A will help you develop a better understanding of inference and learning algorithms for graphical models. Then, in Section B, you will build on these intuitions to implement an LSTM-CRF model and compare its performance with a vanilla LSTM.

- **Collaboration policy:** The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes (including code) are shared, or are taken at that time, and provided learning is facilitated, not circumvented. The actual solution must be done by each student alone. The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved. See the Academic Integrity Section on the course site for more information: http://www.cs.cmu.edu/~mgormley/courses/10708/about.html#7-academic-integrity-policies
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10708/about.html#6-general-policies
- **Submitting your work to Gradescope:** We use Gradescope to collect PDF submissions of open-ended questions on the homework (e.g. mathematical derivations, plots, short answers). The course staff will manually grade your submission, and you'll receive feedback explaining your final marks. You will also submit your code for programming questions on the homework to Gradescope (https://www.gradescope.com/courses/228238). We will manually grade your code for completeness.
- For **multiple choice** or **select all that apply questions**, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For LaTeX users, replace `\choice` with `\CorrectChoice` to obtain a shaded box/circle, and don't change anything else.

[1]Compiled on Saturday 6th March, 2021 at 00:10

A Written Questions [75 pts]

Answer the following questions in the template provided. Then upload your solutions to Gradescope. You may use LaTeX or print the template and hand-write your answers then scan it in. Failure to use the template may result in a penalty. There are 37 points and 7 questions.

A.1 Markov Properties (Revisited)

1. (5 points) For a distribution P and an undirected graph G, show that (**P satisfies the global Markov property with respect to G**) $\Rightarrow$ (**P satisfies the local Markov property with respect to G**) $\Rightarrow$ (**P satisfies the pairwise Markov property with respect to G**). (hint: consider weak union property, $Y \perp X, W|Z \Rightarrow Y \perp X|W, Z$)

A.2 Variable Elimination

2. (4 points) In class, we looked at an example of variable elimination on an arbitrary graph. Let us now apply variable elimination to a familiar directed graphical model: a Hidden Markov Model. A Hidden Markov Model consists of two sets of variables: X_i (observations) and Y_i (states). States are unobserved latent variables which satisfy the Markov property that each state only depends on the state which immediately precedes it. Each state generates an observation. The complete structure of the model (for a sequence of length 5) looks as follows:

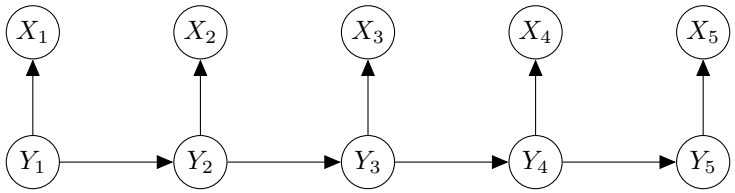


Figure A.1: Hidden Markov Model

Suppose we wish to compute the probability $P(Y_5 \mid X_1 ... X_5)$, which requires us to marginalize over $Y_1 ... Y_4$. Assume that we are eliminating variables in the order $Y_1 - Y_2 - Y_3 - Y_4$. Write down equations for the new factors that will be computed at each step of the elimination process.

Variable Eliminated	**Factor Computed**
Y_1	
Y_2	
Y_3	
Y_4	

3. Variable elimination is more efficient than naively computing the joint probability by brute force. In this problem, we will further study how the order in which variable elimination is carried out affects the efficiency of this method. Consider the following undirected graphical model:

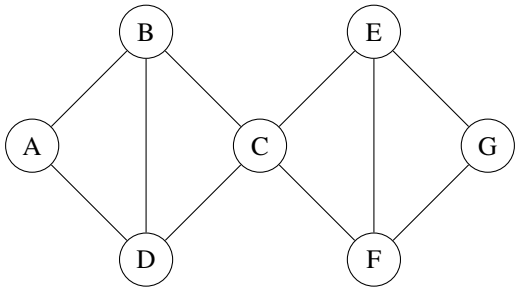


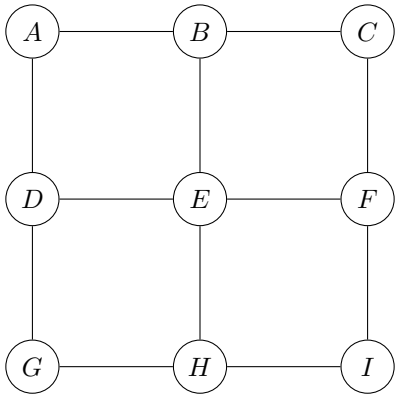
Figure A.2: Initial graph for variable elimination

(a) (2 points) Draw a factor graph for the model shown in Figure A.2, with each factor corresponding to a maximal clique in the graph

(b) (3 points) Consider the variable elimination order $A - G - B - D - E - F - C$ and $C - B - E - A - D - F - G$. Which of the above elimination orders is better and why?

(c) (1 point) Based on your observations, design a way to estimate which elimination order is better without going through the complete process. Describe your approach in a few sentences below.

4. Consider the following Markov network:



We are going to see how *tree-width*, a property of the graph, is related to the intrinsic complexity of variable elimination of a distribution.

(a) (2 points) Write down largest clique(s) for the elimination order $E, D, H, F, B, A, G, I, C$.

(b) (2 points) Write down largest clique(s) for the elimination order $A, G, I, C, D, H, F, B, E$.

(c) (1 point) Which of the above ordering is preferable? Explain briefly.

(d) (2 points) The *elimination width* of a given ordering is the size (i.e. number of variables) of the largest factor produced by variable elimination with that ordering. The *treewidth* is the minimum elimination width over all possible orderings, minus one. Give a reasonable ($\ll n^2$) upper bound on the tree-width of the $n \times n$ grid.

A.3 Message Passing

a	$\psi_A(a)$
0	1
1	2

b	$\psi_B(b)$
0	2
1	1

c	$\psi_C(c)$
0	1
1	1

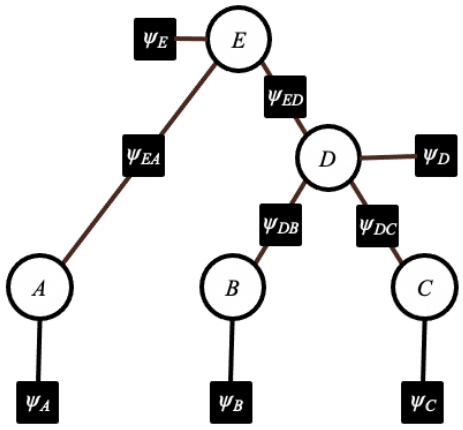


Figure A.3

d	$\psi_D(d)$
0	1
1	1

e	$\psi_E(e)$
0	1
1	2

a	e	$\psi_{EA}(a,e)$
0	0	1
0	1	1
1	0	1
1	1	1

d	e	$\psi_{ED}(d,e)$
0	0	1
0	1	1
1	0	2
1	1	1

b	d	$\psi_{DB}(b,d)$
0	0	1
0	1	2
1	0	1
1	1	1

c	d	$\psi_{DC}(c,d)$
0	0	1
0	1	1
1	0	1
1	1	3

5. Consider the factor graph in Figure A.3. On paper, carry out a run of belief propagation by sending messages first from the leaves ψ_A, ψ_B, ψ_C to the root ψ_E, and then from the root back to the leaves. Then answer the questions below. Assume all messages are un-normalized.

   (a) (2 points) **Numerical answer:** What is the message from A to ψ_{EA}?

   (b) (2 points) **Numerical answer:** What is the message from ψ_{DB} to B?

   (c) (2 points) **Numerical answer:** What is the belief at variable A?

(d) (2 points) **Numerical answer:** What is the belief at variable B?

(e) (2 points) **Numerical answer:** What is the belief at factor ψ_{DB}?

(f) (2 points) **Numerical answer:** What is the value of the partition function?

A.4 Belief Propagation for a Pairwise MRF

Consider an *acyclic* undirected graphical model defined by $G = (V, E)$, with vertices V and edges E. We assume that each element $i \in V$ of the vertex set is an index with a corresponding variable Y_i for that node. Assume $Y_i \in \mathcal{Y}$ for all i. We denote the neighbors of a node i as $\mathcal{N}(i)$. Assume that G is a tree. We can define a pairwise MRF for G as:

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{i \in V} \phi_i(y_i) \prod_{(i,j) \in E} \phi_{ij}(y_i, y_j)$$

where ϕ_i is the potential function for node i and ϕ_{ij} the potential for edge (i, j). Here we will design a belief propagation algorithm that is tailored to acyclic pairwise MRFs.

6. (4 points) Define a new belief propagation algorithm that *only* stores messages sent from one variable y_i to another variable y_j denoted as $m_{i \to j}(y_j)$. Define both how to compute each message and an appropriate message passing order. Here, you should describe only the message passing portion of the algorithm (i.e. do not define the beliefs yet).

7. (2 points) After your algorithm from above terminates, how can you use the messages $m_{j \to i}(y_i)$ to compute the variable marginals $p(y_i)$ (i.e. normalized beliefs)?

8. (3 points) After your algorithm from above terminates, how can you use the messages $m_{i \to j}(y_j)$ to compute the edge marginals $p(y_i, y_j)$ for all $(i, j) \in E$?

A.5 Learning Undirected Graphical Models

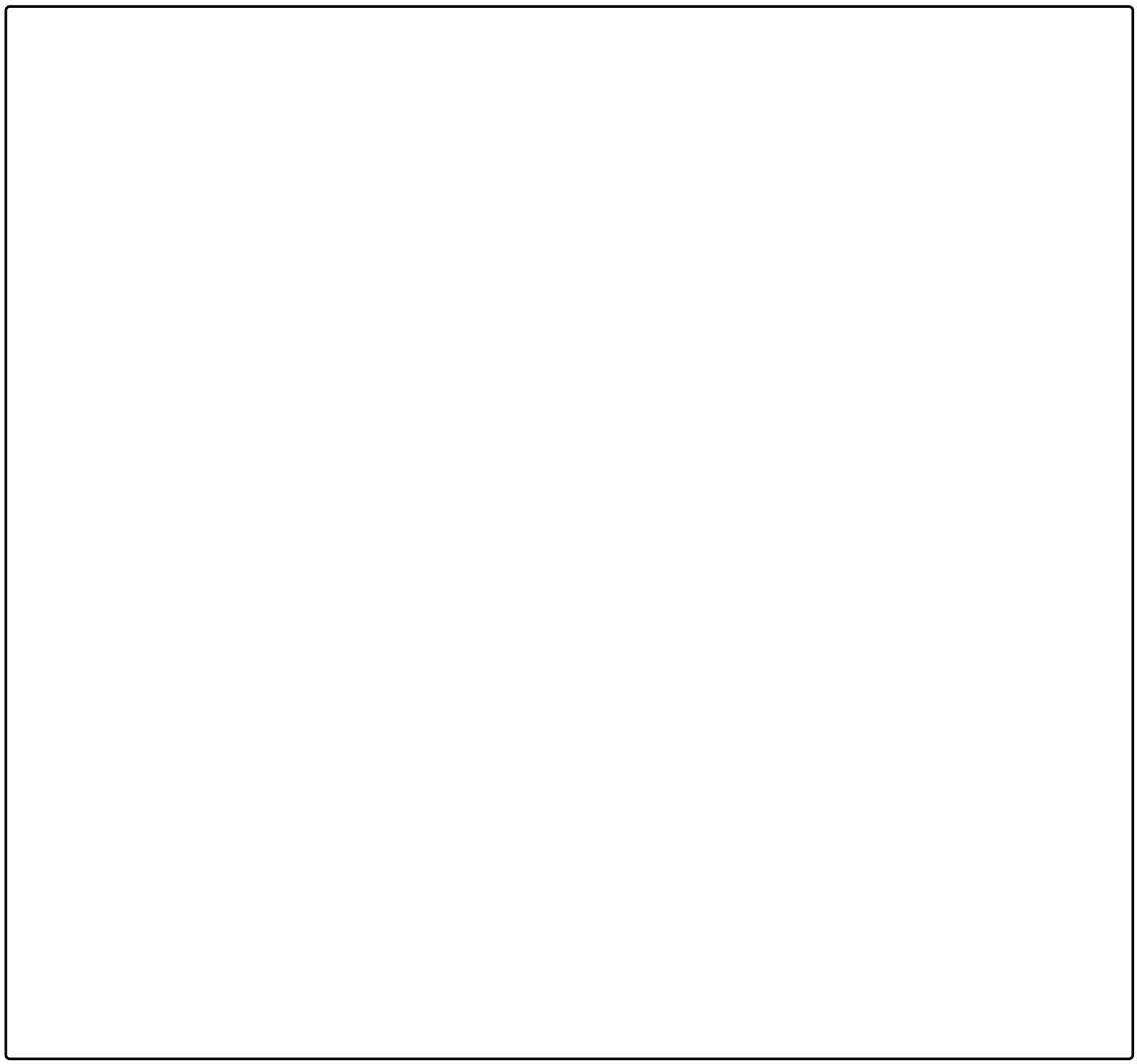
The log-probability of an undirected graphical model is given by:

$$\log p(\mathbf{y}) = \left[\sum_{\alpha} \log \psi_\alpha(\mathbf{y}_\alpha)\right] - \log \sum_{\mathbf{y}' \in \mathcal{Y}} \prod_{\alpha} \psi_\alpha(\mathbf{y}'_\alpha)$$

where y is the set of variables in the graphical model, ψ is a factor, and α is an index into a factor of interest.

9. (6 points) Let β be an index into a factor of interest. Explaining each step clearly with words, show that the derivative of the log-likelihood of an undirected graphical model with respect to a log factor $\log \psi_\beta(\mathbf{y}'_\beta)$ is:

$$\frac{\partial \log p(\mathbf{y})}{\partial \log \psi_\beta(\mathbf{y}'_\beta)} = \mathbb{1}(\mathbf{y}_\beta = \mathbf{y}'_\beta) - p(\mathbf{y}'_\beta)$$



10. (2 points) Show that the derivative of the log-likelihood of an undirected graphical model with respect to a factor $\psi_\beta(\mathbf{y}'_\beta)$ is:

$$\frac{\partial \log p(\mathbf{y})}{\partial \psi_\beta(\mathbf{y}'_\beta)} = \frac{\mathbb{1}(\mathbf{y}_\beta = \mathbf{y}'_\beta) - p(\mathbf{y}'_\beta)}{\psi_\beta(\mathbf{y}'_\beta)}$$

A.6 Empirical Questions

The following questions should be completed after you work through the programming portion of this assignment (Section B).

11. (1 point) **Select one:** If you feed the inputs shown in Figure A.3 into your belief propagation module implemented in PyTorch do you get the same answers that you worked out on paper? *(Hint: The correct answer is "Yes".)*

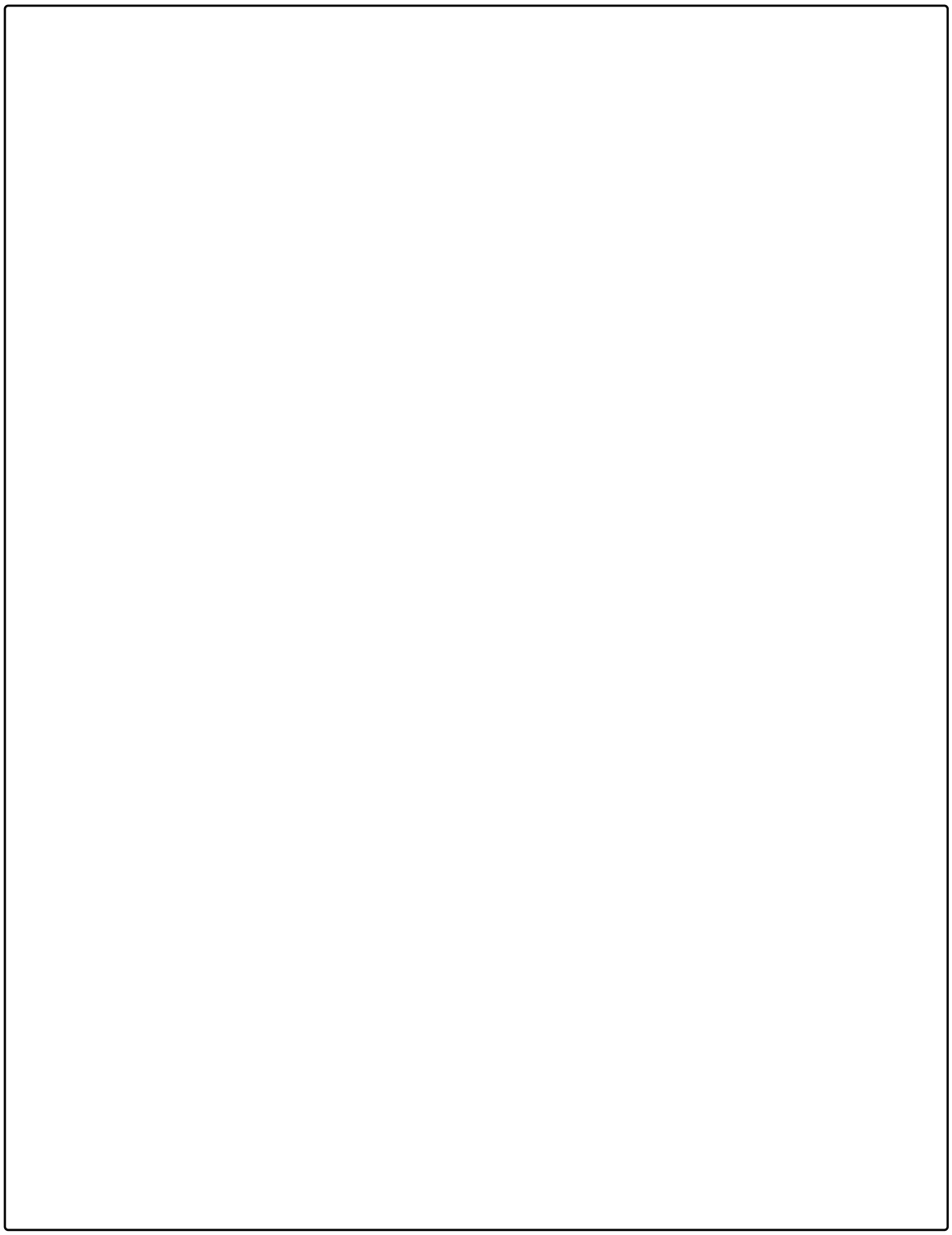
    ◯ Yes

    ◯ No

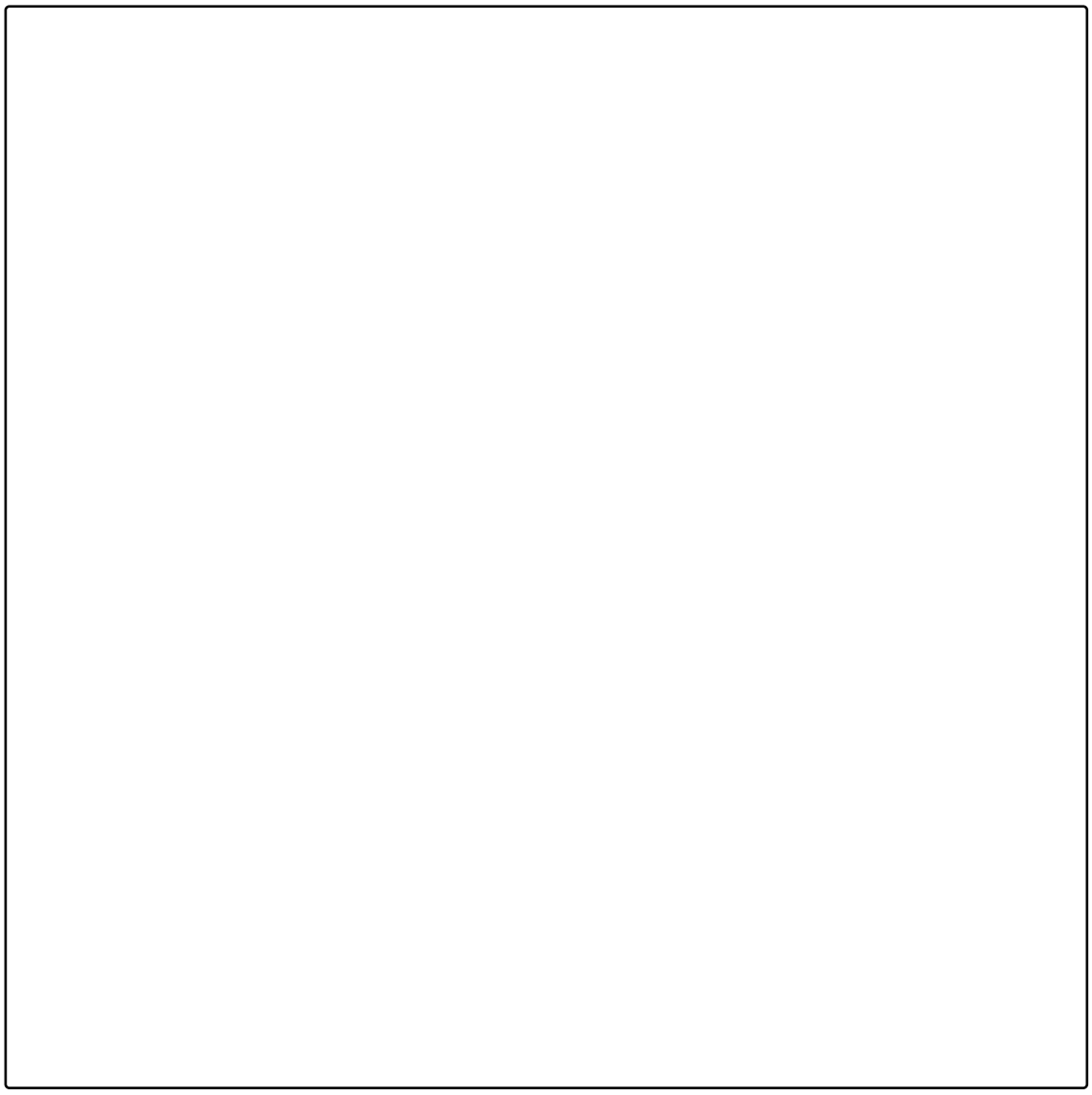
12. (10 points) Record your model's training, validation, and test set performance after one epoch in terms of the loss (cross entropy for the baseline, negative log-likelihood for the LSTM-CRF), accuracy (AC) and leaf accuracy (LAC). For the training metrics, report the average performance on the last 100 batches (where batch size is 1). *Note: Round each numerical value to two significant figures.*

Schedule	**Baseline**	**CRF**
Training Loss		
Training AC		
Training LAC		
Validation Loss		
Validation AC		
Validation LAC		
Test Loss		
Test AC		
Test LAC		

13. (6 points) Plot the training and validation accuracy curves for: *Baseline Model*, *LSTM-CRF Model*. Since the CRF can be slow to train and the training set is large, please plot the average training accuracy over the last 100 data points (i.e. the 100 most recent batches, since batch size should be 1). Plot the validation accuracy on the entire validation set once every 1,000 batches. The x-axis should be the number of batches, and the y-axis should be accuracy. Plot both the tree-level accuracy and the leaf-level accuracy. *Note: Your plots must be machine generated.*



14. (6 points) Plot the training and validation loss curves for: *Baseline Model*, *LSTM-CRF Model*. Since the CRF can be slow to train and the training set is large, please plot the average training loss over the last 100 data points (i.e. the 100 most recent batches, since batch size should be 1). Plot the validation accuracy on the entire validation set once every 1,000 batches. The x-axis should be the number of batches, and the y-axis should be loss. *Note: Your plots must be machine generated.*



A.7 Wrap-up Questions

15. (1 point) **Multiple Choice:** Did you correctly submit your code to Gradescope?

- ◯ Yes
- ◯ No

A.8 Collaboration Policy

After you have completed all other components of this assignment, report your answers to the collaboration policy questions detailed in the Academic Integrity Policies for this course.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details including names of people who helped you and the exact nature of help you received.

2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details including names of people you helped and the exact nature of help you offered.

3. Did you find or come across code that implements any part of this assignment? If so, include full details including the source of the code and how you used it in the assignment.

B Programming [25 pts]

Your goal in this assignment is to implement a CRF belief propagation algorithm for constituency parsing. Given the structure of the tree, you will implement a model to label the nodes with the appropriate tag. Your solution must be implemented in **PyTorch** using the data files we have provided. We have also provided template code for you to use.

B.1 Background: The Constituency Tree Labeling Task

Constituency parsing aims to extract a parse tree from a sentence that represents its syntactic structure according to a phrase structure grammar. Terminals are the words in the sentence, non-terminals in the tree are types of phrases, and the edges are unlabeled. The pre-terminals (i.e. nodes just above the leaves, aka words) are called part-of-speech tags. The other non-terminals are clause or phrase level tags. For more information on tags, look here. Throughout this assignment, we use *nodes* to refer to the set of all non-terminals.

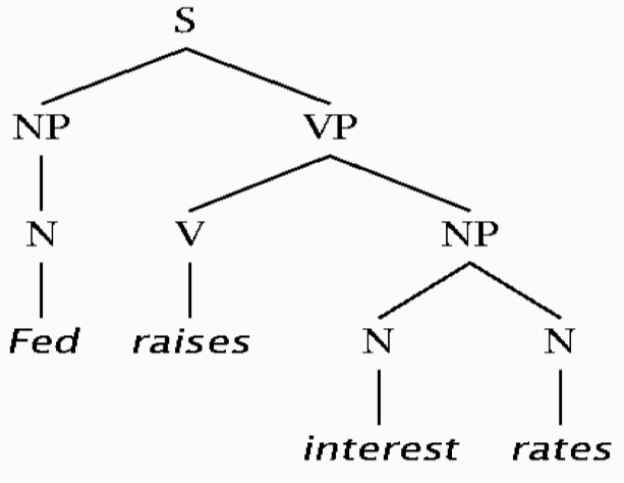


Figure B.1: A parse tree with eight non-terminals: four part-of-speech tags (N, V, N, N) and four phrase-level tags (NP, VP, NP, S).

In this assignment, we will assume that for each example, the *branching structure* of the tree is *known*, but the tags are not. Given the structure, our goal is to successfully predict the appropriate tag for each node in the tree. We define the accuracy of the model as the average accuracy over all examples where each example consists of a tree structure with T nodes and L leaf nodes. Accuracy for a single tree is defined as:

$$\text{Acc} = \frac{\text{number of correctly predicted nodes}}{T}$$

Note that this accuracy is computed across all nodes in the graph. In practice, however, we may particularly care about the POS tags corresponding to each word in the sentence. Thus, we define leaf accuracy as:

$$\text{Leaf Acc} = \frac{\text{number of correctly predicted leaf nodes}}{L}$$

B.2 Background: The Data

We have provided a pre-processed version of Penn Tree Bank with 13,000 examples, divided into 10,000 training examples `ptb-train-10000.txt`, 1,000 development examples `ptb-dev-1000.txt`, and 2,000 test examples `ptb-test-2000.txt`. Each line consists of one tree. For example, the tree shown above would be encoded as:

```
(S (NP (N Fed)) (VP (V raises) (NP (N interest) (N rates))))
```

The trees have already been binarized such that each node has at most two children. We have provided **starter code** to read each line into an NLTK tree data structure, and a custom tree structure, which you can modify.

The task has the following input/output:

- *Given Input:* An input sentence and the associated skeleton of its constituency parse tree
- *Predicted Output:* The labels of the non terminals in the parse tree

B.3 Baseline Model: LSTM with independent tag predictions (5 points)

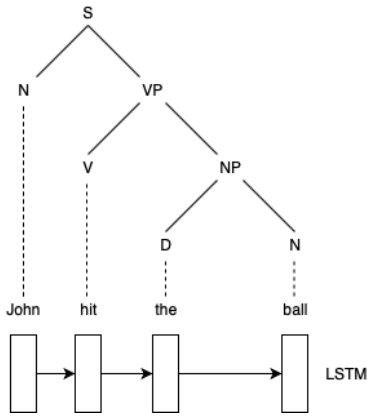


Figure B.2: Baseline model using a unidirectional LSTM.

In this section, you will implement a working baseline LSTM model. Starter code for the model can be found in `baseline_template.py`.

First, use a torch.nn.Embedding layer to convert your sentence to a tensor representation. Then, use an LSTM layer (torch.nn.LSTM) to output a "hidden state" for each node. For the leaf nodes (i.e. nodes corresponding to each token in the sentence), this hidden state is simply the corresponding output of the LSTM. For intermediate nodes, use a linear layer on the concatenation of the hidden states of the left and the right child. If there is only one child, concatenate its hidden state with itself. Use the direct child nodes, *not* the leftmost descendent and rightmost descendent. The output should be a distributions over tags. Train the model using cross entropy loss.

Your implementation must have a single-layer unidirectional LSTM which outputs a hidden state of dimension 128. The embedding size should be 256. Set your optimizer to be Adam with a learning rate of 0.0001. Due to the complexity associated with building the tree and computing its potentials, you can use a batch size of 1.

B.4 Main Model: LSTM + CRF

In this section, you will implement a CRF layer on top of an LSTM representation (Figure B.2). This will involve computing the unary potentials (a.k.a. factors) corresponding to each node and binary potentials corresponding to each edge in the tree. For an example of unary and binary potentials, see problem A.3. Starter code can be found in `lstm_crf_template.py`.

1. (3 points) **Representation of potentials.** Instead of keeping an explicit lookup table for the unary and binary potentials, these potentials will be computed by applying a linear layer to the LSTM hidden representation computed for each node (or in the case of binary potentials, concatenation of hidden representations). You need to compute a unary potential for every node in the graph and an edge potential for every edge. Note that the dotted lines in Figure B.2 do not count as edges.

   To ensure that the potentials are positive and to provide better numerical stability, we assume that the output of the linear layer is the *logarithm* of the potentials. From there, **all further computation should be carried out in logspace**.

2. (10 points) **Belief propagation.** Now that you've set up the unary and binary potentials, it's time to implement belief propagation. The sum-product belief propagation algorithm proceeds as follows:

   (a) *Send messages from the leaves to the root.* Starting at the leaves, compute the factor and variable messages to be sent upward. Recall from Lecture 5 that factor messages are computed as:

   $$\mu_{\alpha \to i}(x_i) = \sum_{x_\alpha : x_\alpha[i] = x_i} \psi_\alpha(x_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(x_\alpha[j])$$

   where the message is being sent from factor α to node i, and a matrix-vector product is being computed between the binary factor $\psi_\alpha(x_\alpha)$ and the product of incoming messages to α. A variable message is computed as:

   $$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

   where the message is being sent from variable i to factor α.

   (b) *Send messages from the root to the leaves*: After having computed all messages to send upwards to the root, we can re-use some of these messages when sending messages downward from the root. (See the slides from Lecture 5, starting at slide 67, for a visualization of this process.)

   (c) *Compute beliefs*: Given these messages, we are able to compute the beliefs and hence the marginals at each node. A variable belief is computed as:

   $$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)$$

   and a factor belief is computed as:

   $$b_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(x_\alpha[i])$$

   (Note: you will not technically need to compute all beliefs, but comparing the resulting partition functions with each other could be a good sanity check.)

3. **Useful trick: logsumexp** When numbers are too small or too large, precision can be lost to floating point errors and overflow errors can occur. To circumvent this numerical stability issue, we can compute all messages and potentials in the log space. Multiplication operations on messages would then become addition of log messages, since $\log(ab) = \log(a) + \log(b)$. Addition of messages can be done using the logsumexp operation. Available in pytorch as torch.logsumexp, this function is equivalent to exponentiating the elements of a tensor, summing them along a specified dimension, and then taking the log to put everything back in logspace.
4. (5 points) **Loss function: negative log-likelihood.** The loss function should be the negative log-likelihood (NLL), computed from the CRF potentials. This can be done for each sample by looking up and adding up the potentials associated with each node's true label, and subtracting out the partition function. (Hint: How do you compute the partition function from your log belief? Can you think of any sanity checks that can come from the computing the partition function?)
5. (2 points) **Computing accuracy.** To help evaluate your model, compute the accuracy for each tree as well as the accuracy among the leaves. To do so, predict the tag with highest marginal probability for each node, and compare against the true labels.

Additional details: Again, use a hidden dimension of 128 and embedding dimension of 256. Use the Adam optimizer with learning rate 0.0001.

B.5 Code Submission [25 pts]

You must submit all of your code to the appropriate slot on Gradescope. If you have multiple `.py` files, you must upload all of them. Your code will **not** be autograded on Gradescope. Instead, we will grade your code by hand; that is, we will read through and attempt to run your code in order to grade it. As such, please carefully identify major sections of the code via comments.