HOMEWORK 2 GRAPHICAL MODELS[1]

10-418 / 10-618 MACHINE LEARNING FOR STRUCTURED DATA (FALL 2019)
https://piazza.com/cmu/fall2019/1041810618

OUT: Sep. 26, 2019
DUE: Oct. 09, 2019 11:59 PM
TAs: Karthika, Aakanksha, Austin

START HERE: Instructions

Summary In this assignment, you will implement a baseline LSTM model for constituency parsing followed by a general CRF (Conditional Random Field) and train both jointly. Section 1 will help you develop a better understanding of directed and undirected graphical models through some warm-up problems. Then, in Section 2, you will build on these intuitions to implement an LSTM-CRF model and compare its performance with a vanilla LSTM.

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: http://www.cs.cmu.edu/~mgormley/courses/10418/about.html#7-academic-integrity-policies
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10418/about.html#6-general-policies
- **Autolab:** You will submit your code for programming questions on the homework to Autolab (https://autolab.andrew.cmu.edu/). After uploading your code, we will manually grade your code by hand. We will not use Autolab to autograde your code.
- **Submitting your work to Gradescope:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (https://gradescope.com/). Please use the provided template. Submissions can be handwritten, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. For short answer questions you **should not** include your work in your solution. If you include your work in your solutions, your assignment may not be graded correctly by our AI assisted grader.

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 12th October, 2019 at 20:38

1 Written Questions [66 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 66 points and 16 questions.

1.1 Conditional Independencies

1. Consider the Bayesian Network described in Figure 1.1

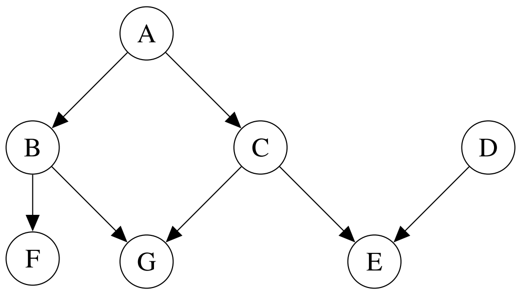


Figure 1.1: Bayesian Network Structure

Based on this network structure, answer the following questions:

(a) (1 point) Write down the equation for the joint probability distribution $P(A, B, C, D, E, F, G)$

$P(A)P(B|A)P(C|A)P(F|B)P(G|B,C)P(D)P(E|C,D)$

(b) (1 point) Is $C \perp D \mid E$?

○ True

[✓ False]

(c) (1 point) Is $A \perp F \mid B$?

[✓ True]

○ False

(d) (1 point) Is $A \perp G \mid B$?

○ True

[✓ False]

(e) (1 point) Which nodes are present in the Markov blanket of B?

A, F, G, C

(f) (1 point) Which nodes are present in the Markov blanket of D?

C, E

2. Now consider an undirected graphical model with the same set of nodes and edges as the bayesian network from figure 1.2. This model structure looks as follows:

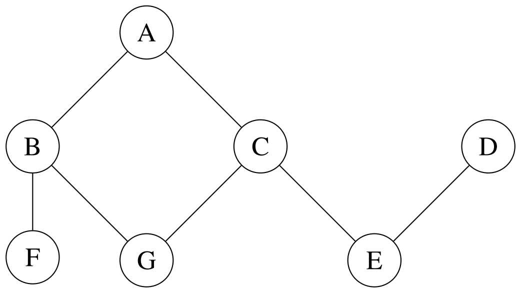


Figure 1.2: Undirected Graphical Model

For this model structure, answer the following questions:

(a) (1 point) Is $C \perp D \mid E$?

[✓ True]

False

(b) (1 point) Is $A \perp F \mid B$?

[✓ True]

False

(c) (1 point) Is $A \perp G \mid B$?

True

[✓ False]

(d) (1 point) Which nodes are present in the Markov blanket of B?

A, F, G

(e) (1 point) Which nodes are present in the Markov blanket of D?

E

3. Let us now compare both models (1.1 and 1.2).

(a) (1 point) Do both models (1.1 and 1.2) have the same set of conditional independencies?

Yes

[✓ No]

(b) (2 points) If you answered yes to the above question, list out all the conditional independencies. If you answered no, provide an example of a graph which does have the same set of conditional independencies for both directed and undirected variants.

In both, we have that $A \perp C \mid B$.

$$\begin{tikzcd} A \arrow[r] & B \arrow[r] & C \end{tikzcd}$$

$$\begin{tikzcd} A \arrow[r, dash] & B \arrow[r, dash] & C \end{tikzcd}$$

(c) (2 points) For the directed bayesian network, we decomposed the joint probability distribution into a product of conditional probability distributions associated with each node. However, we did not do so for the undirected model. Is it possible to write joint probability as a product of factors *without* performing marginalization (i.e. no summations) for a general undirected graphical model? Explain your answer.

It is not possible. Although we can write the joint probability as a product of factors, we also need the partition function which we can only know from marginalization.

1.2 Variable Elimination

1. 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: 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 i.e. 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:

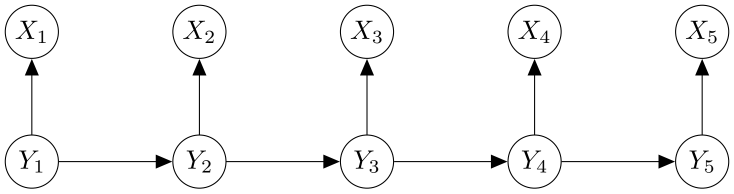
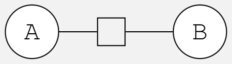
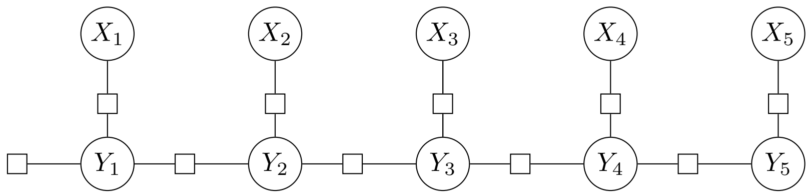


Figure 1.3: Hidden Markov Model

(a) (2 points) Draw the corresponding factor graph for this model.

```
Latex users: If you want to use tikz to draw the factor graph,
here is a sample code snippet for a tiny factor graph:
\tikz[square/.style={regular polygon,regular polygon sides=4}]
{
\node[latent] (A) {A};
\node[latent,right=1.5 cm of A] (B) {B};
\node[square, draw=black, right=0.5 cm of A] (ab) {};
\edge [-] {A} {ab};
\edge [-] {B} {ab};
}
This snippet generates the following graph:
```





(b) (2 points) For this model, write down the joint probability distribution as a product of conditional probability distributions.

$$P(Y_1)\left(\prod_{i=1}^{4} P(X_i \mid Y_i)P(Y_{i+1} \mid Y_i)\right) P(X_5 \mid Y_5)$$

(c) (4 points) 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 factors which will be computed at each step of the elimination process.

Variable Eliminated	**Factor Computed**
Y_1	$m_1(x_1, y_2) = \sum_{y_1} \psi_{Y_1}(y_1)\psi_{X_1,Y_1}(x_1, y_1)\psi_{Y_1,Y_2}(y_1, y_2)$
Y_2	$m_2(x_1, x_2, y_3) = \sum_{y_2} m_1(x_1, y_2)\psi_{X_2,Y_2}(x_2, y_2)\psi_{Y_2,Y_3}(y_2, y_3)$
Y_3	$m_3(x_1, x_2, x_3, y_4) = \sum_{y_3} m_2(x_1, x_2, y_3)\psi_{X_3,Y_3}(x_3, y_3)\psi_{Y_3,Y_4}(y_3, y_4)$
Y_4	$m_4(x_1, x_2, x_3, x_4, y_5) = \sum_{y_4} m_3(x_1, x_2, x_3, y_4)\psi_{X_4,Y_4}(x_4, y_4)\psi_{Y_4,Y_5}(y_4, y_5)$

(d) (1 point) Is it possible to pick a better elimination order for this model?

○ Yes

[✓ No]

(e) (0 points) Do you observe any similarities between the factors computed during variable elimination and the standard forward-backward algorithm for HMMs?

Yes they only differ in terms of normalization.

2. In class, we saw how using variable elimination is more efficient than naively computing the joint probability. 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:

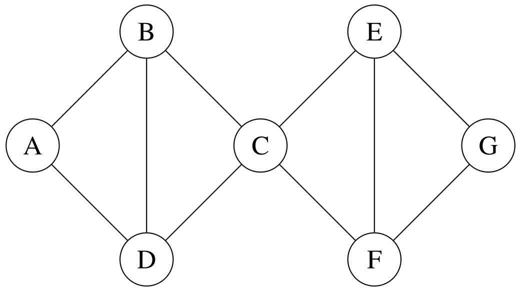
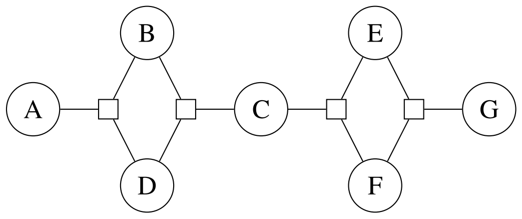
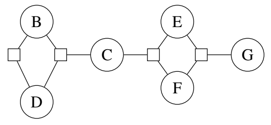
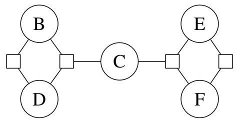
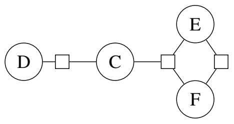
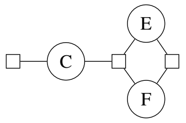
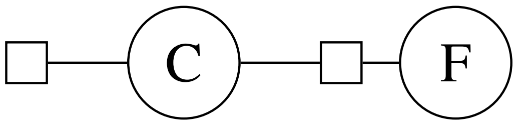
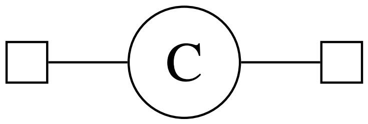
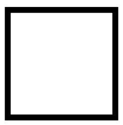


Figure 1.4: Initial graph for variable elimination

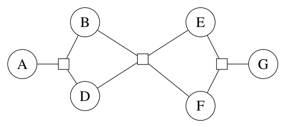
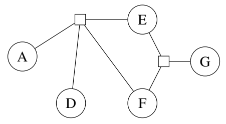
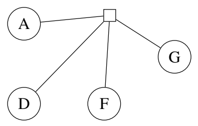
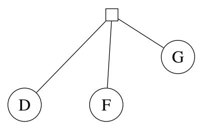
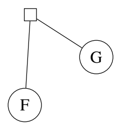
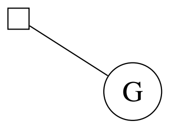
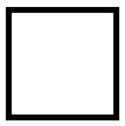
(a) (2 points) Draw a factor graph for this model, with each factor corresponding to a maximal clique in the graph



(b) (4 points) For the variable elimination order $A - G - B - D - E - F - C$, draw the intermediate factor graph at each step

Variable Eliminated	**Intermediate Factor Graph**
A	
G	
B	
D	
E	
F	
C	

(c) (4 points) For the variable elimination order $C - B - E - A - D - F - G$, draw the intermediate factor graph at each step

Variable Eliminated	**Intermediate Factor Graph**
C	
B	
E	
A	
D	
F	
G	

(d) (1 point) Which of the above elimination orders is better and why?

The first elimination order, as the largest intermediate tensor created is 3-dimensional compared to 4-dimensional ones created in the second order, so the overall runtime is asymptotically smaller.

(e) (1 point) Based on your observations, can you think of a way to estimate which elimination order is better without going through the complete process?

At every step, pick the variable with the least number of neighboring factors.

1.3 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

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

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

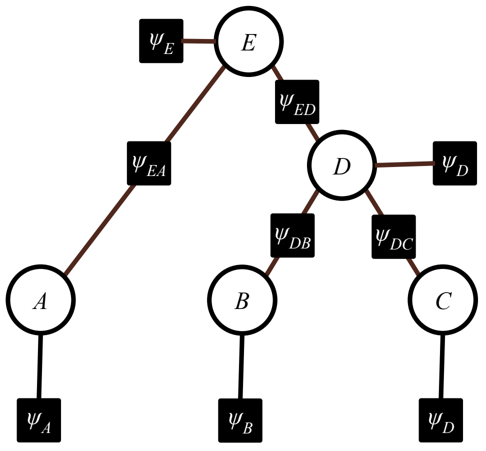


Figure 1.5

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

a	d	$\psi_{ED}(d,e)$
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

Consider the factor graph in Figure 1.5. 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.

1. (1 point) **Numerical answer:** What is the message from A to ψ_{EA}?

   $\begin{bmatrix}1\\2\end{bmatrix}$

2. (1 point) **Numerical answer:** What is the message from ψ_{DB} to B?

   $\begin{bmatrix}114\\66\end{bmatrix}$

3. (1 point) **Numerical answer:** What is the belief at variable A?

   $\begin{bmatrix}98\\196\end{bmatrix}$

4. (1 point) **Numerical answer:** What is the belief at variable B?

$$\boxed{\begin{bmatrix} 228 \\ 66 \end{bmatrix}}$$

5. (1 point) **Numerical answer:** What is the belief at factor ψ_{DB}?

$$\boxed{\begin{bmatrix} 60 & 60 \\ 72 & 36 \end{bmatrix}}$$

6. (1 point) **Numerical answer:** What is the value of the partition function?

$$\boxed{294}$$

1.4 Empirical Questions

The following questions should be completed after you work through the programming portion of this assignment (Section 2).

1. (1 point) **Select one:** If you feed the inputs shown in Figure 1.5 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".)*

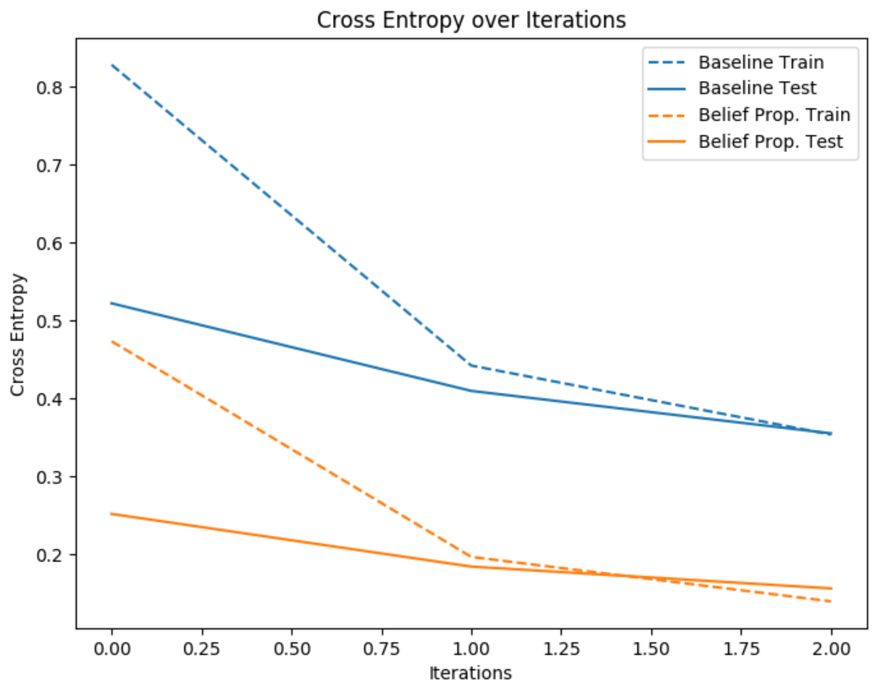
   [✓ Yes]

   ◯ No

2. (10 points) Record your model's performance on the test set and train set in terms of Cross Entropy (CE) , accuracy (AC) and leaf accuracy (LAC). *Note: Round each numerical value to two significant figures.*

Schedule	**Baseline**	**CRF**
Training CE	0.36	0.14
Training AC	0.88	0.96
Training LAC	0.90	0.94
Test AC	0.88	0.95
Test LAC	0.90	0.94

3. (10 points) Plot training and testing cross entropy curves for : *Baseline*, *CRF Model*. Let the x-axis ranges over 3 epochs. *Note: Your plot must be machine generated.*



All metrics for training such as accuracy (all and leaf only) and cross-entropy were averaged across the entire epoch of training while the model was being improved, while test metrics were averaged across all test examples with a frozen model at the end of the epoch.

1.5 Wrap-up Questions

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

   [✓ Yes]

   No

2. (1 point) **Numerical answer:** How many hours did you spend on this assignment?.

   10

2 Programming [30 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 tag the nodes with appropriate POS tag.

Your solution for this section must be implemented in **PyTorch** using the data files we have provided to you. This restriction is because we will be grading your code by hand to check your understanding as well as your model's performance.

2.1 Task Background

Constituency parsing aims to extract a parse tree from a sentence that represents its syntactic structure according to a phrase structure grammar. Non-terminals in the tree are types of phrases, the terminals are the words in the sentence, and the edges are unlabeled. Throughout this assignment, we use nodes to refer to the set of all non terminals.

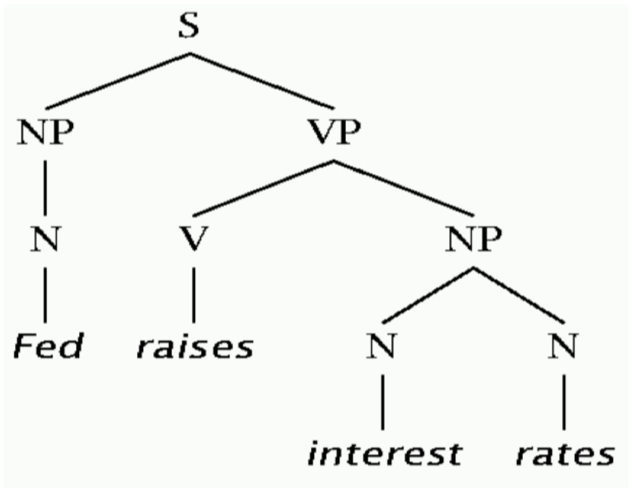


Figure 2.1: A parse tree with eight nodes - four leaves and four intermediate nodes.

Assuming we know the structure of this tree, our goal is to successfully predict the appropriate tag for all its nodes. We will train our model with a Cross Entropy Loss, based on the beliefs of each node after message passing. We define the accuracy of the model as the average accuracy over all examples where each example consists of a tree structure with n nodes. Accuracy for a single tree is defined as

$$\text{Acc} = \frac{\text{number of correctly predicted nodes}}{\text{total number of nodes in the tree}}$$

Note that this accuracy is computed across **all** nodes in the graph. We also define leaf accuracy as

$$\text{Leaf Acc} = \frac{\text{number of correctly predicted leaf nodes}}{\text{total number of leaves in the tree}}$$

2.2 Data

We have provided a preprocessed version of Penn Tree Bank with 49,208 examples. Each line consists of one tree. You are to split the data using a 70/30 ratio for train and test.
We have provided starter code to read each line into an NLTK tree data structure, and a custom tree structure, which you can modify.

Given Input: An input sequence and the associated skeleton of its constituency parse tree.

Goal Output: The labels of the non terminals in the parse tree.

2.3 Baseline

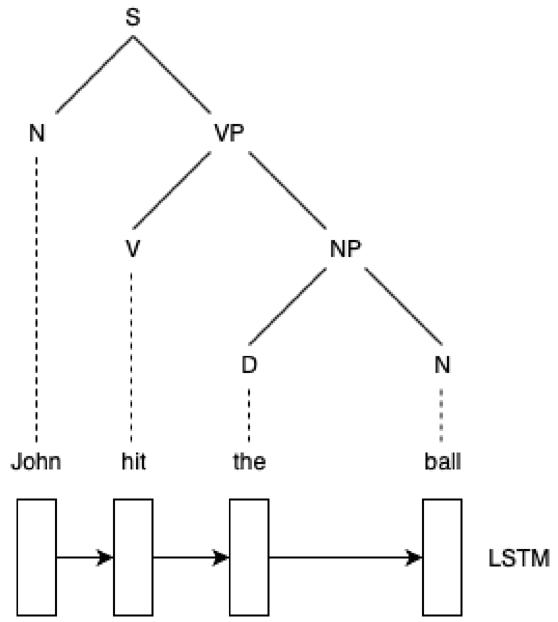


Figure 2.2: Our baseline model using a unidirectional LSTM.

For this section, you must implement a working baseline LSTM model.

Our baseline model uses the hidden state of an LSTM layer to predict the output tag. For the leaf nodes, this computation is trivial. For intermediate nodes, we use a linear layer on the concatenation of the hidden state of the LSTM outputs of the left and the right child to compute distributions over tags.

Your implementation must have a single layer unidirectional LSTM with a hidden dimension of 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. Your program should be able to run on a laptop without GPUs due to the simplicity of our model.

2.4 Adding the CRF layer

For this section, you must implement functions to compute the unary potentials for each node and binary potential for each edge in the tree.

The CRF layer consists of computing unary node potentials and binary edge potentials derived from the LSTM hidden state. You need to compute a unary potential for every node in the graph and an edge potential for every edge. Note here that the dotted line between the non terminals and the terminals do not count as edges.

Recall that these potentials are the scores associated for each of the possible tags, and have to be positive. By simply using a linear layer on the output of the LSTMs, however, we may get negative scores. To account for these negative values, we assume that the output of the linear layer is the **logarithm of potentials**. All further computation is carried out in log space. This gives us the added advantage of numerical stability[2.6].

2.5 Message passing implementation

For this section, you must implement a function for belief propagation.

The sum-product message passing algorithm is defined as follows:

While there is a node x_i ready to transmit to x_j, send the message

$$m_{i\to j} = \sum_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{l\in N(i)/j} m_{l\to i}(x_i)$$

Here, $N(i)/j$ refers to the set of nodes that are neighbors of i, excluding j. After we have computed all messages, we may answer any marginal query over x_i in constant time using the equation.

$$p(x_i) \propto \phi(x_i) \prod_{l\in N(i)} m_{l\to i}(x_i)$$

We will be implementing the asynchronous version of the algorithm in this assignment. This consists of two sets of messages, an upward pass from the leaves to the root, and a downward pass from the root node to the leaf nodes.

2.6 The LogSumExp trick

The LogSumExp trick is a common trick in machine learning to deal with problems of numerical stability. To illustrate the problem, consider the contrived example for our features x_i: [1000, 1000, 1000]. Feeding this sequence into the softmax function should yield a probability distribution of [1/3, 1/3, 1/3] and the log of 1/3 is a reasonable negative number. However, calculating one of the terms of the summation in python yields the following output:

```
>>> import math
>>> math.e**1000
Traceback (most recent call last):
File "", line 1, in
OverflowError: (34, 'Result too large')
```

To deal with the underflow (and similar overflow), we can compute all messages and potentials in the log space. Multiplication operations on messages would then turn to addition of log messages using

$$e^a.e^b = e^{a+b}$$

and

$$\log(ab) = \log(a) + \log(b)$$

Hint: Pytorch has a logsumexp function which can be applied to any dimension of a tensor.

2.7 Test time decoding

During test time decoding, predict the tag with highest marginal probability for each variable. **DO NOT** run the belief propagation here.

2.8 Autolab Submission [30 pts]

You must submit a .tar file named `beliefprop.tar` containing `beliefprop.py`, which contains all of your code.

You can create that file by running:

```
tar -cvf beliefprop.tar beliefprop.py
```

from the directory containing your code.

Some additional tips: **DO NOT** compress your files; you are just creating a tarball. Do not use tar `-czvf`. **DO NOT** put the above files in a folder and then tar the folder. Autolab is case sensitive, so observe that all your files should be named in **lowercase**. You must submit this file to the corresponding homework link on Autolab.

Your code will **not** be autograded on Autolab. Instead, we will grade your code by hand; that is, we will read through your code in order to grade it. As such, please carefully identify major sections of the code via comments.

3 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.

   No

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.

   No

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.

   No