10-418: HW2 Recitation Notes

Austin Dill

October 4th, 2019

1 Introduction

These notes are meant to introduce the topics and concepts needed to work through homework 2, including constituency parsing, belief propagation on factor graphs, and incorporating neural networks into CRFs. The best source for this information can be found by rewatching the lecture videos and going through the reading material.

2 Constituency Parsing

First we'll consider an extremely ambiguous sentence and see how we could disambiguate the meaning.

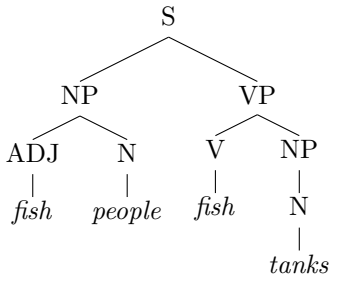
Fish people fish tanks. (1)

This sentence could be interpreted two ways.

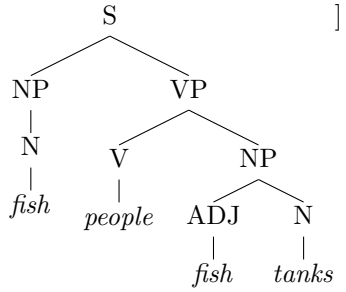
1. Fish-People hybrids fish in tanks.
2. Fish populate fish tanks. (This uses a less common meaning of people meaning "to populate".)

How could we build in structure to distinguish between these two meanings? One way is to build and label a parse tree.

1. Fish-People hybrids fish in tanks.



2. Fish populate fish tanks.



Both of these options can be summarized by a simple set of rules called a *grammar*. If you've taken a compiler course, you'll know and hate this term.

$$\begin{aligned} \mathbf{S} &\rightarrow \mathbf{NP\ VP} \\ \mathbf{NP} &\rightarrow \mathbf{ADJ\ N} \mid \mathbf{N} \\ \mathbf{VP} &\rightarrow \mathbf{V\ NP} \end{aligned}$$

The takeaway from this is that if we could somehow generate these tree-labelings, we would be much closed to pulling *meaning* from a sequence of words before further processing. For example, this can be a useful first step in a dialogue system where disambiguating meaning is crucial.
So now that we've established the context for our problem of interest, let's set it aside for a little while.

3 Message Passing Review

Remember that for belief propagation on factor graphs, we have two types of messages we can pass.

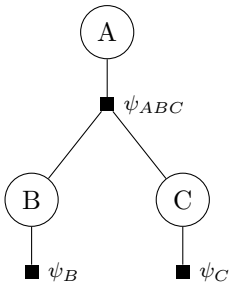
Variable to Factor:

$$\mu_{x\to f}(x) = \prod_{g\in \mathrm{Ne}(x)/f} \mu_{g\to x}(x)$$

Factor to Variable:

$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f/x} \psi_f(\mathcal{X}_f) \prod_{y\in\{\mathrm{Ne}(f)/x\}} \mu_{y\to f}(y)$$

We'll consider applying these rules to the following undirected graphical model.



3.1 Representation

First, let's consider the case where each potential function is tabular, variables B and C are binary, and variable A has N possible settings.

Q: How would we represent this as a NumPy array?

A: Use an array where each axis i represents all the values variable i can take.

Q: What would be the size of ψ_b?

A: $(2,)$

Q: What would be the size of ψ_{abc}?

A: $(N, 2, 2)$

Q: How would we marginalize out a variable in a potential function in this scenario?

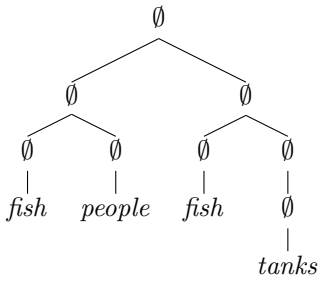
A: Sum over the axis that represents the variabel you want to marginalize.

Q: How would we express each of the types of messages in code?

A: This is a straightforward combination of element wise multiplication and summing over your axis of interest.

4 Incorporating Neural Networks

Using the same graphical model as above, let's consider the situation where instead of tabular potential functions, we have neural networks providing the scores.
Let's step through a parse tree to see what we want to happen in your code. Note that for this assignment we are given the *structure* of the tree but not the labels on each node.



Let's start by examining a baseline for this task. What if we only considered the sequence of words to build these labels?

4.1 Baseline Model

This is a pain to draw in LaTeX, so I'll just write it out in words. What we want to do is use the hidden layer from an LSTM to determine the tag output. We'll divide this into two situations: When our node of interest in a leaf node and when it's an internal node.
For these steps, we will assume we have already calculated the LSTM hidden layers for each time step.

Leaf Node:

1. Run the current node's hidden state through a linear layer so you have $|V|$ outputs.
2. Run softmax on this to get probabilities.
3. Return the tag with the highest probability.

Internal Node:

1. Get the LSTM hidden layers for the children of this node.
2. Concatenate these hidden layers.
3. Run this concatenated vector through a linear layer so you have $|V|$ outputs.
4. Run softmax on this to get probabilities.
5. Return the tag with the highest probability.

4.2 LSTM+CRF

Now our model will be slightly more complicated, because we don't just need to find the argmax of our outputs. We want to run belief propagation, so first we'll need to define our potential functions.

Unary Potentials - Leaf Node:

1. Pass the hidden state of the LSTM through a linear layer so you get an output of size $|V|$.

Unary Potentials - Intermediate Node:

1. Concatenate the hidden states of this node's children.
2. Run the resulting vector through a linear layer to get a "hidden layer" for this node.
3. Pass the hidden state of this node through a linear layer so you get an output of size $|V|$.

Edge Potentials:

1. Concatenate the hidden states of the nodes adjacent to this edge.
2. Pass the resulting vector though a linear layer so you get an output of size $|V| \times |V|$. (Why this size?)

Now we have potential tables that we can use to run BP just as explained in lecture! The only significant difference is we're asking that you do BP in log-space. (Think about how you could write messages if you wanted log messages.)

Okay...but after running BP, what do we output? We output the argmax of the *belief* at each node. Intuitively this should give us better results than our baseline because it takes into account the setting of every tag in the tree.