Assignment 4 Report

Tomer Gill  
318459450  
gilltom

**The paper** I chose to implement is "**Learning to Compose Task-Specific Tree Structures**" by Jihun Choi, Kang Min Yoo & Sang-goo Lee that was published in November 2017.

Link to the paper: <https://arxiv.org/pdf/1707.02786.pdf>

# The Paper's Method

The paper describes a model that learns how to represent a sentence using tree LSTM for a specific task (in our case, SNLI). Because of this I will first describe the general sentence representation method and then I'll describe the SNLI predictions.

## Tree LSTM

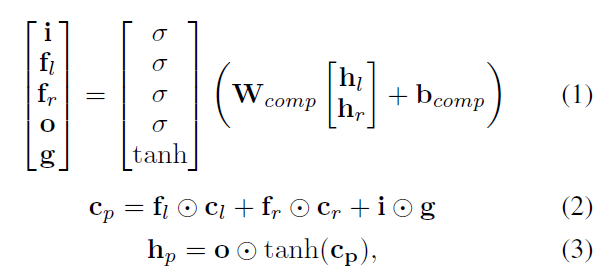
The model is based as an Tree LSTM – The LSTM part means for each represented node (which can be one or more words with a single [h, c] representation. Will be described later) we will have both outer and inner representation which are 2 vectors: **h** and **c** (respectively), which are like the "y"-s and "s"-s of a LSTM RNN. The tree part means we will build our representation of the whole sentence layer-by-layer from the bottom up – at each step the model will choose adjacent nodes from the layer and unite them to a single node (with a single [h, c] representation).

### Leaf Representation

At the beginning, each word is given its own [h, c] representation by encoding it using a linear layer, or alternatively the paper suggests using a LSTM RNN to give each representation info about the previous nodes. This can also be upgraded to a BiLSTM.  
The represented words are the leaf nodes of out tree LSTM.

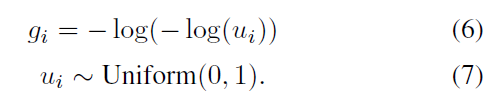
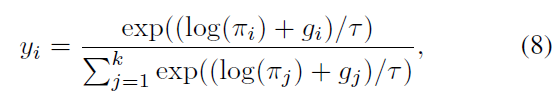
### Parents Representation

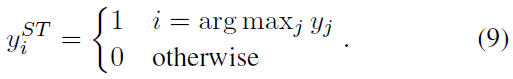
At each step, the model will represent all the possible parents of 2 adjacent nodes. This is done by concatenating the **h** representations of the left node on top of the right node's **h**, then going through a linear layer and different activation functions, we will get 5 gate vectors (as a LSTM does) and with them we will create the [**h**p, **c**p] representation of the parent node:



### Gumbel Softmax

Inspired by the Gumbel Max trick, the Gumbel Softmax uses the differentiable soft max function. The GS "adds" a *Gumbel Noise* a set of unnormalized probabilities π1, · · · , πk, a sample yk, …, yk will be:

And the Straight-Through version will be: 

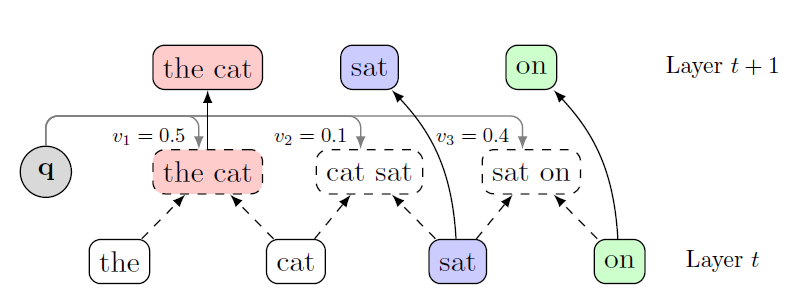
### Parents Selection

To choose which nodes will be united to a single node, each possible parent is scored: using a trainable *query vector* is dot-multiplied with each possible parent. Then those scores are going through a softmax function.

In the testing phase the parent with the maximal score is chosen, but in the training phase the scores are going through the Gumbel Softmax function thus giving us the 2 samples: **y** and the one-hot version **yST**. In the forward pass **yST** is used for fast computing, but in the backward pass to update the parameters **y** is used. In DyNet this can be done like this:

dynet.nobackprop**(**y\_st **-** y**) +** y

The parent with the best probability will replace the couple of nodes in the next layer:

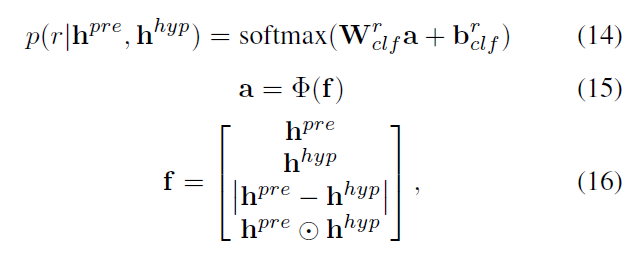


### Output

The merging of nodes goes on until there is only one, who represents the whole sentence. The **h** vector is the output of the tree.

## SNLI

Given the premise and the hypothesis sentences, **h**pre and **h**hyp are the outputs of the above model on those sentences respectively. Those are concatenated and inputted to an MLP with ReLu activation function. Then for option r (that can be either entailment, contradiction or neutral) the probability will be (Φ is the MLP):



Meaning there is a linear layer for each possible r, and their outputs are softmaxed to get the class.

# Why I Chose this Paper?

What first got my attention when I passed over the list of papers, is the use of a tree LSTM which we mentioned in class but didn't go into details, so I thought it will be a good opportunity to learn it "outside" of class. Additionally, the paper mentions that sentences are more tree shaped then shaped in a linear way, so using trees felt right to solve this kind of problems. Then when I initially read the paper I saw how to model is trained to create LSTM trees for given problems, so the model is rally general in it's nature and the SNLI problem is just a specific problem that can be solved with it.

# The Paper's Results

The paper did 3 experiments, with different parameters for each. The one I tried to replicate used GloVe 300D pre-trained vectors as the embedding matrix (which isn't trained with the rest of the model), the size of the **h** and **c** representation vectors are set to 300 also, using dropout at probability 0.01 to inputs, the MLP input and its output.

Their model converged quickly, after 2-3 epochs only, and it reached 84.4% accuracy with 2.3 million parameters in 3.1 hours with linear encoding of the leaves, and 85.6% with 2.9 million parameters in 1.6 hours with LSTM based leaf encoding.

# My Attempt

First, sadly, I didn't manage to replicate the results.

I ran all the tries on a google cloud virtual machine with 8 cores, 52GB RAM and a GPU (which I didn't have enough time to use it ☹)

At first, I tried to implement the paper as is, so it will run on a single input each time. Due to the simpleness of this implementation a run took a lot of time (20+ hours and didn't finish 10% of the training set), and my results wasn't good (though I didn't let it finish a single epoch due to the time).

Then I have tried to implement a better version of this model – I implemented the optimizations written at the end of the paper (using masks / gates in matrix multiplication) and also tried to use dynet's autobatching. This version was full of bugs that took a lot of time to fix, which was hard because it was on the google cloud computer.

I noticed that this was still slow and hard for the computer to process. My attempt using autobatching wasn't successful, because it didn't seem to help at all and all the processing was made on a single core. Furthermore, the optimizations definitely improved the time but sadly not enough.

The model couldn't train well because of what seems like float overflow that together with the gates optimizations lead to Infinties and NaNs in my vectors. After more than a week of bug-fixing and rewriting my code I ran out of time.