CS 224d: Assignment #3

Due date: 5/22 11:59 PM PST (You are allowed to use three (3) late days maximum for this assignment)

This handout consists of several homework problems, as well as instructions on the "deliverables" associated with the coding portions of this assignment.

These questions require thought, but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. However, each student must finish the problem set and programming assignment individually, and must turn in her/his assignment. We ask that you abide by the university Honor Code and that of the Computer Science department, and make sure that all of your submitted work is done by yourself.

Please review any additional instructions posted on the assignment page at http://cs224d.stanford.edu/assignments.html. When you are ready to submit, please follow the instructions on the course website.

1 RNN's (Recursive Neural Network)

Welcome to SAIL (Stanford Artificial Intelligence Lab): Congrats! You have just been given a Research Assistantship in SAIL. Your task is to discover the power of Recursive Neural Networks (RNNs). So you plan an experiment to show their effectiveness on Positive/Negative Sentiment Analysis. In this part, you will derive the forward and backpropogation equations, implement them, and test the results.

Our RNN has one ReLU layer and one softmax layer, and uses Cross Entropy loss as its cost function. We follow the parse tree given from the leaf nodes up to the top of the tree and evaluate the cost at each node. During backprop, we follow the exact opposite path. Figure 1 shows an example of such a RNN applied to a simple sentence "I love this assignment". These equations are sufficient to explain our model:

$$CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i y_i \log(\hat{y}_i)$$

where $\boldsymbol{y}$ is the label represented as a one-hot row vector, and $\hat{\boldsymbol{y}}$ is a row vector containing the predicted probabilities for all classes. In our case, $y \in \mathbb{R}^{1\times 5}$ and $\hat{y} \in \mathbb{R}^{1\times 5}$ to represent our 5 sentiment classes: Really Negative, Negative, Neutral, Positive, and Really Positive. Furthermore,

$$h^{(1)} = \max(\left[\ h^{(1)}_{Left}, h^{(1)}_{Right}\ \right] W^{(1)} + b^{(1)}, 0)$$

$$\hat{y} = \text{softmax}(h^{(1)}U + b^{(s)})$$

where $h^{(1)}_{Left}$ is the vector representation of the left subtree (possibly be a word vector), the $h^{(1)}_{Right}$ of the right subtree. For clarity, $L \in \mathbb{R}^{|V|\times d}$, $W^{(1)} \in \mathbb{R}^{2d\times d}$, $b^{(1)} \in \mathbb{R}^{1\times d}$, $U \in \mathbb{R}^{d\times 5}$, $b^{(s)} \in \mathbb{R}^{1\times 5}$.

(a) (20 points) Follow the example parse tree in Figure 1 in which we are given a parse tree and truth labels y for each node. Starting with Node 1, then to Node 2, finishing with Node 3, write the update rules for $W^{(1)}$,$b^{(1)}$, U, $b^{(s)}$, and L after the evaluation of $\hat{y}$ against our truth, y. This means for at each node, we evaluate:

$$\delta_3 = \hat{y} - y$$

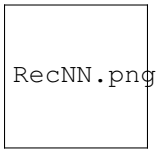

Figure 1: RNN (Recursive Neural Network) example

as our first error vector and we backpropogate that error through the network, aggregating gradient at each node for:

$$\frac{\partial J}{\partial U} \quad \frac{\partial J}{\partial b^{(s)}} \quad \frac{\partial J}{\partial W^{(1)}} \quad \frac{\partial J}{\partial b^{(1)}} \quad \frac{\partial J}{\partial L_i}$$

Points will be deducted if you do not express the derivative of activation functions (ReLU) in terms of their function values (as with Assignment 1 and 2) or do not express the gradients by using an "error vector" (δ_i) propagated back to each layer. Tip on notation: δ_{below} and δ_{above} should be used for error that is being sent down to the next node, or came from an above node. This will help you think about the problem in the right way. Note you should not be updating gradients for L_i in Node 1. But error should be leaving Node 1 for sure!

Solution: We derive all the gradients for an arbitrary tree below:

For root node: $\delta_{above} = 0$

For any arbitrary node:

$$\delta_3 = \hat{y} - y$$
$$\frac{\partial J}{\partial U} = {h^{(1)}}^T \delta_3$$

If internal node:

$$\delta_2 = (\delta_3 U^T + \delta_{above}) \odot 1\{h^{(1)} > 0\}$$
$$\delta_{below} = \delta_2 W^T$$
$$\left[\ \delta_{Left\_above}, \delta_{Right\_above} \ \right] = \delta_2 W^T$$
$$\frac{\partial J}{\partial W^{(1)}} = \left[\ h^{(1)}_{Left}, h^{(1)}_{Right} \ \right]^T \delta_2$$

Else (leaf node with word corresponding to index i):

$$\frac{\partial J}{\partial L_i} = \delta_3 U^T + \delta_{above}$$

(b) (80 points) Implementation time! We have simplified the problem to reduce training time by binarizing the sentiment labels. This means that all the sentences are either positive or negative. The internal nodes however, can be positive negative or neutral. While training, the cost function includes predictions over all nodes that have a sentiment associated with them and ignores the neutral nodes. While testing, we are only interested in the performance at the full sentence level. This is all provided for you in the starter code.

- (a) Download, unzip, and have a look at the code base.
- (b) From the command line, run `./setup.sh` to download the labeled parse tree dataset.
- (c) Begin implementing the model in `rnn.py` by filling out the outline.

- (d) Run the model and show us the resulting loss plot, final training accuracy, final validation accuracy. You should be able to achieve an accuracy of 75%.
- (e) (10 points extra credit) Tune the parameters of the model to improve performance and report your new loss plot, final training accuracy, final validation accuracy. Explain why the changes you made helped achieve the improvement.
- (f) Ensure that your code is using the best model config and the trained weights are present in the weights folder. Run `./prepare_submission.sh` and submit the resulting zip file.