CS 224d: Assignment #1

Due date: 4/16 11:59 PM PST (You are allowed to use three (3) late days maximum for this assignment)

Note: This is the complementary problem set of Assignment 1 for CS 224d. We strongly recommend that you finish these problems before writing the code for the 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 Softmax

Prove that softmax is invariant to constant offsets in the input, that is, for any input vector $\boldsymbol{x}$ and any constant c,

$$\text{softmax}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{x} + c)$$

where $\boldsymbol{x} + c$ means adding the constant c to every dimension of $\boldsymbol{x}$.

Note: In practice, we make use of this property and choose $c = -\max_i x_i$ when computing softmax probabilities for numerical stability (i.e. subtracting its maximum element from all elements of $\boldsymbol{x}$).

2 Neural Network Basics

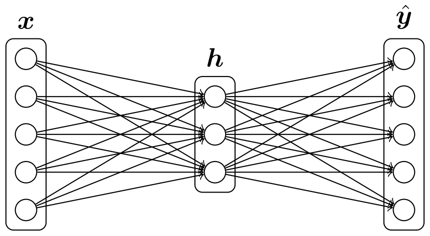
(a) Derive the gradients of the sigmoid function and show that it can be rewritten as a function of the function value (i.e. in some expression where only $\sigma(x)$, but not x, is present). Assume that the input x is a scalar for this question.

(b) Derive the gradient with regard to the inputs of a softmax function when cross entropy loss is used for evaluation, i.e. find the gradients with respect to the softmax input vector $\boldsymbol{\theta}$, when the prediction is made by $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{\theta})$. Remember the cross entropy function is

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

where $\boldsymbol{y}$ is the one-hot label vector, and $\hat{\boldsymbol{y}}$ is the predicted probability vector for all classes. (*Hint: you might want to consider the fact many elements of $\boldsymbol{y}$ are zeros, and assume that only the k-th dimension of $\boldsymbol{y}$ is one.*)

(c) Derive the gradients with respect to the *inputs* $\boldsymbol{x}$ to an one-hidden-layer neural network (that is, find $\frac{\partial J}{\partial \boldsymbol{x}}$ where J is the cost function for the neural network). The neural network employs sigmoid activation function for the hidden layer, and softmax for the output layer. Assume the one-hot label vector is $\boldsymbol{y}$, and cross entropy cost is used. (feel free to use $\sigma'(x)$ as the shorthand for sigmoid gradient, and feel free to define any variables whenever you see fit)



Recall that the forward propagation is as follows

$$\boldsymbol{h} = \text{sigmoid}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1) \qquad\qquad \hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2)$$

Note that here we're assuming that the input vector (thus the hidden variables and output probabilities) is a row vector to be consistent with the programming assignment. When we apply the sigmoid function to a vector, we are applying it to each of the elements of that vector. $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ ($i = 1, 2$) are the weights and biases, respectively, of the two layers.

(d) How many parameters are there in this neural network, assuming the input is D_x-dimensional, the output is D_y-dimensional, and there are H hidden units?

3 word2vec

(a) Assume you are given a predicted word vector $\hat{\boldsymbol{r}}$ (v_{w_I} in the lecture notes in the case of skip-gram), and word prediction is made with the softmax function found in `word2vec` models

$$\Pr(\text{word}_i \mid \hat{\boldsymbol{r}}, \boldsymbol{w}) = \frac{\exp(\boldsymbol{w}_i^\top \hat{\boldsymbol{r}})}{\sum_{j=1}^{|V|} \exp(\boldsymbol{w}_j^\top \hat{\boldsymbol{r}})}$$

where $\boldsymbol{w}_j$ ($j = 1, \ldots, |V|$) are the "output" word vectors for all words in the vocabulary (v'_w in the lecture notes). Assume cross entropy cost is applied to this prediction and word i is the expected word (the i-th element of the one-hot label vector is one), derive the gradients with respect to $\hat{\boldsymbol{r}}$.

(b) In the previous problem, derive gradients for the "output" word vectors $\boldsymbol{w}_j$'s (including $\boldsymbol{w}_i$).

(c) Repeat part (a) and (b) assuming we are using the negative sampling loss for the predicted vector $\hat{\boldsymbol{r}}$, and the expected output word is $\boldsymbol{w}_i$. Assume that K negative samples are drawn, and they are $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_K$, respectively for simplicity of notation ($i \notin \{1, \ldots, K\}$). Recall that the negative sampling loss function in this case is

$$J(\hat{\boldsymbol{r}}, \boldsymbol{w}_i, \boldsymbol{w}_{1\ldots K}) = -\log(\sigma(\boldsymbol{w}_i^\top \hat{\boldsymbol{r}})) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{w}_k^\top \hat{\boldsymbol{r}}))$$

where $\sigma(\cdot)$ is the sigmoid function.

After you've done this, describe with one sentence why this cost function is much more efficient to compute than the softmax-CE loss (you could provide a speed-up ratio, i.e. the runtime of the softmax-CE loss divided by the runtime of the negative sampling loss).

Note: the cost function here is the negative of what Mikolov et al had in their original paper, because we are doing a minimization instead of maximization in our code.

(d) Derive gradients for all of the word vectors for skip-gram and CBOW (optional) given the previous parts, given a set of context words $[\text{word}_{i-C}, \ldots, \text{word}_{i-1}, \text{word}_i, \text{word}_{i+1}, \ldots, \text{word}_{i+C}]$, where C is the context size. You can denote the "input" and "output" word vectors for word_k as $\boldsymbol{v}_{w_k}$ and $\boldsymbol{v}'_{w_k}$ respectively for convenience. (*Hint: feel free to use $F(\boldsymbol{v}_{w_O}|\hat{\boldsymbol{r}})$ as a placeholder for softmax-CE or negative sampling in this part — you'll see that this is a useful abstraction for the coding part.*)

Recall that for skip-gram, the cost for a context is

$$J_{\text{skip-gram}}(\text{word}_{i-C\ldots i+C}) = \sum_{-c\leq j\leq c, j\neq 0} F(\boldsymbol{v}'_{w_{i+j}}|\boldsymbol{v}_{w_i})$$

For (a simpler variant of) CBOW, we sum up the input word vectors in the context

$$\hat{\boldsymbol{r}} = \sum_{-c\leq j\leq c, j\neq 0} \boldsymbol{v}_{w_{i+j}}$$

then the CBOW cost is

$$J_{\text{CBOW}}(\text{word}_{i-C\ldots i+C}) = F(\boldsymbol{v}'_{w_i}|\hat{\boldsymbol{r}})$$

4 Sentiment Analysis

(a) Explain in less than three sentences why do we want to introduce regularization when doing classification (in fact, most machine learning tasks).

(b) Plot the classification accuracy on the dev set with respect to the regularization value, using a logarithmic scale on the x-axis. Briefly explain with less than three sentences what you see in the plot.