CS 224D Spring 2016 Assignment #1

SUNet ID: 06106557

Name: Gregory Lambert

1 Softmax

(a) Prove that softmax in invariant to constant offsets in the input, WTS that $softmax(\mathbf{x}+c) = softmax(\mathbf{x})$ for any input vector $\mathbf{x}$ and any constant c.
Proof:

$$\begin{aligned} softmax(\mathbf{x}+c) &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} \\ &= \frac{e^{x_i}e^c}{\sum_j e^{x_j}e^c} \\ &= \frac{e^{x_i}e^c}{e^c\sum_j e^{x_j}} \\ &= \frac{e^{x_i}}{\sum_j e^{x_j}} \end{aligned}$$

which is the definition of $softmax(\mathbf{x})$.

2 Neural Network Basics

(a) Derive the gradients for 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). Recall that the sigmoid function is

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Assuming that x is a scalar then the gradient for the sigmoid function can be expressed by the following

$$\begin{aligned} \sigma'(x) &= \frac{d}{dx}(1+e^{-x})^{-1} \\ &= -(1+e^{-x})^{-2}(-e^{-x}) \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{(1-1)+e^{-x}}{1+e^{-x}}\frac{1}{1+e^{-x}} \\ &= \left(1-\frac{1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) \\ &= (1-\sigma(x))(\sigma(x)) \end{aligned}$$

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

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

where $\mathbf{y}$ is the one-hot label vector, and $\hat{\mathbf{y}}$ is the predicted probability vector of all the classes. Assuming that the i^{th} element of $\mathbf{y}$ is 1 and all other elements are 0, then the gradient of the cross-entropy loss with respect to θ can be expressed by the following

$$\begin{aligned}
\frac{\partial CE}{\partial \theta} &= \frac{\partial}{\partial \theta} - \sum_i y_i log \frac{e^{\theta_i}}{\sum_{c=1}^{C} e^{\theta_c}} \\
&= \frac{\partial}{\partial \theta} - y_i \left(\sum_i log(e^{\theta_i}) - log \left(\sum_{c=1}^{C} e^{\theta_c} \right) \right) \\
&= \frac{\partial}{\partial \theta} \left(-\theta_i + log \left(\sum_{c=1}^{C} e^{\theta_c} \right) \right) \\
&= -\frac{\partial}{\partial \theta} \theta_i + \frac{\partial}{\partial \theta} log \left(\sum_{c=1}^{C} e^{\theta_c} \right) \\
&= -\frac{\partial}{\partial \theta} \theta_i + \frac{\frac{\partial}{\partial \theta} e^{\theta_c}}{\sum_{c=1}^{C} e^{\theta_c}} \\
&= -1 + \hat{\mathbf{y}}
\end{aligned}$$

For the case were the i^{th} element of $\mathbf{y}$ is not 1 then $\frac{\partial CE}{\partial \theta} = \frac{\frac{\partial}{\partial \theta} e^{\theta_c}}{\sum_{c=1}^{C} e^{\theta_c}} = \hat{\mathbf{y}}$

(c) Derive the gradients with respect to the inputs $\mathbf{x}$ to a one-hidden layer neural network (that is, find $\frac{\partial J}{\partial \mathbf{x}}$ 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 that the one-hot label vector is $\mathbf{y}$, and the cross entropy cost is used.

We will use the following definitions for deriving $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial \theta} \frac{\partial \theta}{\partial h} \frac{\partial h}{\partial r} \frac{\partial r}{\partial x}$:
$J = -\sum_i y_i log(\hat{y}_i)$
$r = xW_1 + b_1$
$h = \sigma(r)$
$\theta = hW_2 + b_2$
$\hat{y} = softmax(\theta)$.
We can express $\frac{\partial J}{\partial x}$ in terms of the following partial derivatives
$\frac{\partial J}{\partial \theta} = \hat{y} - y$ (derived from 2(b))
The other partials can be expressed by the following:
$\frac{\partial \theta}{\partial h} = \frac{\partial}{\partial h} hW_2 + b_2 = W_2$
$\frac{\partial h}{\partial r} = \frac{\partial}{\partial r} h = \sigma(r)(1 - \sigma(r))$
$\frac{\partial r}{\partial x} = \frac{\partial}{\partial x} xW_1 + b_1 = W_1$.

Therefore our desired gradient can be expressed as the product of the previous partials:
$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial \theta} \frac{\partial \theta}{\partial h} \frac{\partial h}{\partial r} \frac{\partial r}{\partial x} = (\hat{y} - y) W_2 \sigma(r)(1 - \sigma(r)) W_1 = W_1 W_2 (\hat{y} - y) \sigma(r)(1 - \sigma(r))$

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

$W_1 \in \mathbb{R}^{D_x \times H}$, $b_1 \in \mathbb{R}^H$, $W_2 \in \mathbb{R}^{H \times D_y}$, and $b_2 \in \mathbb{R}^{D_y}$

3 word2vec

(a) Derive the gradients with respect to $\mathbf{v_c}$ of the word2vec model

Let $CE(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i)$ and $y_i = \mathbf{u}_i^T \mathbf{v}_c$ and $\hat{y}_i = \frac{e^{y_i}}{\sum_{j=1}^{V} e^{y_j}}$ for the following sections in question 3

The following is the gradient with respect to v_c where $y_i = 1$ if i is the o-th element and 0 otherwise
$\frac{\partial J}{\partial \mathbf{v}_c} = -\sum_{i=1}^{V} \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial \mathbf{v}_c} = \sum_{i=1}^{V} (\hat{\mathbf{y}} - y_i) \frac{\partial}{\partial \mathbf{v}_c} \mathbf{u}_i^T v_c = \sum_{i=1}^{V} (\hat{\mathbf{y}} - y_i) = \sum_{i=1}^{V} (\hat{\mathbf{y}} - y_i) \mathbf{u}_i$

(b) Derive the gradients of all the "output" word vectors

$\frac{\partial J}{\partial \mathbf{u}_i} = \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial \mathbf{u}_i} = \sum_{i=1}^{V} (\hat{\mathbf{y}} - y_i) \frac{\partial}{\partial \mathbf{u_i}} = \sum_{i=1}^{V} (\hat{\mathbf{y}} - y_i) \mathbf{v}_c$

(c) Repeat part (a) and (b) assuming a negative sampling loss for the predicted vector $\mathbf{v}_c$ where $y_k = \mathbf{u}_k^T \mathbf{v}_c$ and $o \notin k$

$$J_{neg-sample}(o, v_c U) = -log(\sigma(\mathbf{u}_o^T \mathbf{v}_c)) - \sum_{k=1}^{K} log(\sigma(-\mathbf{u}_k^T \mathbf{v}_c))$$

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{v}_c} &= \frac{\partial J}{\partial \sigma(y_o)} \frac{\partial \sigma(y_o)}{\partial \mathbf{v}_c} - \sum_{k=1}^{K} \frac{\partial J}{\partial \sigma(-y_k)} \frac{\partial \sigma(-y_k)}{\partial \mathbf{v}_c} \\
&= -\left(1 - \sigma(y_o)\right) \mathbf{u}_o + \sum_{k=1}^{K} \left(1 - \sigma(-y_k)\right) \mathbf{u}_k \\
&= (\sigma(y_o) - 1) \mathbf{u}_o + \sum_{k=1}^{K} \left(1 - \sigma(-y_k)\right) \mathbf{u}_k \\
\frac{\partial J}{\partial \mathbf{u}_i} &= \left(\sigma(y_o) - 1\right) \mathbf{v}_c + \sum_{k=1}^{K} \left(1 - \sigma(-y_k)\right) \mathbf{v}_c
\end{aligned}$$

Negative sampling is less computationally intensive due to the summation of K elements versus over the entire V vocabulary ($K \subset V$).

(d) Derive all of the word vectors for skip-gram and CBOW

$$J_{skip-gram}(word_{c-m,\ldots,c+m}) = \sum_{-m \le j \le m,\ j \neq 0} F(w_{c+j}, V_c)$$

$\frac{\partial J_{skip-gram}}{\partial \mathbf{v}_c} = \sum_{-m \le j \le m,\ j \neq 0} \frac{\partial F(\mathbf{w}_{c+j}, \mathbf{v}_c)}{\partial \mathbf{v}_c}$

$\frac{\partial J_{skip-gram}}{\partial \mathbf{w}_{c+j}} = \sum_{-m \le j \le m,\ j \neq 0} \frac{\partial F(\mathbf{w}_{c+j}, \mathbf{v}_c)}{\partial \mathbf{w}_{c+j}}$

$\frac{\partial J_{CBOW}}{\partial \hat{\mathbf{v}}} = \sum_{-m \le j \le m,\ j \neq 0} \frac{\partial F(\mathbf{w}_{c+j}, \hat{v})}{\partial \hat{\mathbf{v}}}$

$\frac{\partial J_{CBOW}}{\partial \mathbf{w}_{c+j}} = \sum_{-m \le j \le m,\ j \neq 0} \frac{\partial F(\mathbf{w}_{c+j}, \hat{\mathbf{v}})}{\partial \mathbf{w}_{c+j}}$

4 Sentiment Analysis

(b) Explain in fewer than three sentences why we want to introduce regularization when doing classification.

Regularization is a common modeling component in statistical and machine learning algorithms for helping with overfitting, by penalizing your loss function. In terms of neural networks and deep neural networks it can be especially critical to adjust model parameters to account for overfitting due to natural complexity of the modeling framework.

(c)What value did you select? Report your train, dev, and test accuracy.

First larger regularization parameters were chosen (e,g. 0.9, 0.5, 0.1, 0.01, and 0.001) but found to give insufficient accuracy for train, dev, and test sets. The final list of regularization parameters selected for the sentiment analysis: [1×10^{-4}, 1×10^{-6}, 1×10^{-7}, 1×10^{-8}, 1×10^{-9}]. Small regularization parameters were selected due to previous results, small sample size, and the use of a linear classifier (softmax regression). The best regularization value: 1×10^{-6} with a test accuracy (%): 17.918552 was selected.

regularization	1×10^{-4}	1×10^{-6}	1×10^{-7}	1×10^{-8}	1×10^{-9}
Train accuracy (%)	28.815543	29.143258	29.143258	29.143258	29.143258
Dev accuracy (%)	28.065395	30.881017	30.881017	30.881017	30.881017

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

Conducting the regularization search displays that there is an increase in both training and dev accuracy as the regularization parameter decreases. As displayed in Figure 1, values smaller than 1×10^{-6} show no improvement in train and dev accuracy. For this reason our selected regularization parameter value was 1×10^{-6}.

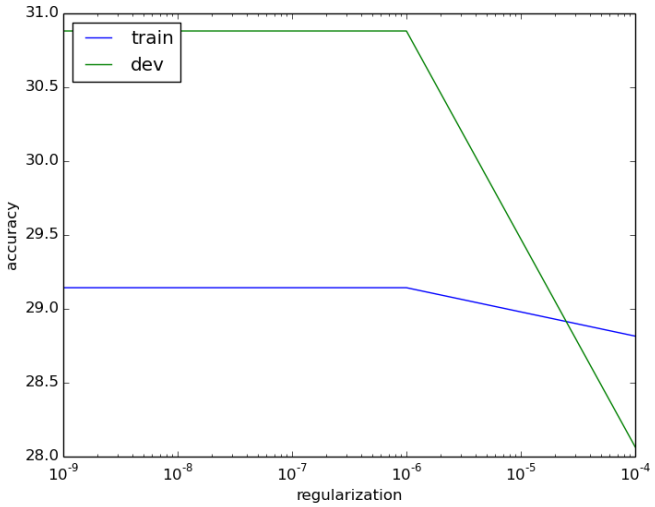

Figure 1: Classification accuracy