Deep Learning for NLP (236602) - HW1

Segev Arbiv

1 Softmax

Let us recall the softmax formula:

$$softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{1.1}$$

for $j = 1 \dots k$.
Hence, a translated version will be:

$$\begin{aligned} softmax(x+c)_j &= \frac{e^{(x+c)_j}}{\sum_{k=1}^{K} e^{(x+c)_k}} && \text{(1.2a)} \\ &= \frac{e^{x_j} \cdot e^{c}}{\sum_{k=1}^{K} e^{x_k} \cdot e^{c}} && \text{(1.2b)} \\ &= \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} && \text{(1.2c)} \\ &= softmax(x)_j && \text{(1.2d)} \end{aligned}$$

2 Neural Network Basics

2.1 a

Let us recall the *Sigmoid* function:

$$s(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

And calculate the derivative:

$$\begin{aligned} s'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} && \text{(2.2a)} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} && \text{(2.2b)} \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) && \text{(2.2c)} \\ &= s(x) \cdot (1 - s(x)) && \text{(2.2d)} \end{aligned}$$

2.2 b

Recalling that y is a one-hot-vector, only the $k'th$ component of it is 1, while the rest is 0. Hence, $CE(y, \hat{y}) = -log(\hat{y_k})$ where $\hat{y_k} = softmax(\theta)_k$. Calculating the derivative:

$$\begin{aligned} \frac{\partial CE}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}\left(-\log \frac{e^{\theta_k}}{\sum_{k=1}^{K} e^{\theta_k}}\right) && \text{(2.3a)} \\ &= \frac{\partial}{\partial \theta_i}(-\theta_k) + \frac{\partial}{\partial \theta_i}\left(\log \sum_{k=1}^{K} e^{\theta_k}\right) && \text{(2.3b)} \\ &= \frac{\partial}{\partial \theta_i}(-\theta_k) + \frac{\frac{\partial}{\partial \theta_i}\sum_{k=1}^{K} e^{\theta_k}}{\sum_{k=1}^{K} e^{\theta_k}} && \text{(2.3c)} \end{aligned}$$

We should now distinguish between the cases where $i \neq k$ and $i = k$ when taking derivative by θ_i.
By noticing that the right term is exactly the derivation of $softmax(y)_i$, we get:

$$\frac{\partial CE}{\partial \theta_i} = \begin{cases} \hat{y}_i & \text{i} \neq \text{k} \\ \hat{y}_i - 1 & \text{i} = \text{k} \end{cases} \tag{2.4}$$

Since y in a one-hot vector, we can rewrite and get a vector notation: $\frac{\partial CE}{\partial \theta} = \hat{y} - y$.

2.3 c

Assuming $\mathbb{1}_{\{J>0\}}$ is true at all times, its similar to the derivation of S, which is actually deriving the cost function CE.
By denoting $z^1 = W^1 x + b^1$, we can see that $\theta = W^2 \sigma(z^1) + b^2$. Taking the derivative:

$$\frac{\partial CE}{\partial x} = \frac{\partial CE}{\partial \theta}\frac{\partial \theta}{\partial x} \tag{2.5}$$

$$= (\hat{y} - y) \cdot \frac{\partial \theta}{\partial x} \tag{2.6}$$

$$= (\hat{y} - y) \cdot \frac{\partial}{\partial x}(W^2 \sigma(z^1) + b^2) \tag{2.7}$$

$$= (\hat{y} - y) \cdot \frac{\partial}{\partial x}(W^2 \sigma(W^1 x + b^1) + b^2) \tag{2.8}$$

$$= (\hat{y} - y) \cdot \sigma'(W^1 x + b^1) \cdot W^2 W^1 \tag{2.9}$$

$$= \delta^{(1)} W^1 \tag{2.10}$$

Where 2.6 is due to 2.2, and 2.10 is from lecture definitions, as $\delta^{(2)} = \hat{y} - y$.
Note, some of the above are Hadamard multiplications (that doesn't add up in the math).

2.4 d

If there are D_x inputs and H hidden units, W_1 is of size $D_X \times H$. Equivalently, W_2 is of size $D_y \times H$. With the biases of the same size of the layers themselves ($b_1 \in R^{1 \times H}$, $b_2 \in R^{1 \times D_y}$), we get that the total number of parameters is: $HD_x + HD_y + H + D_y$.

3 Word2vec

3.1 a

Again, for one-hot vector $y_i = \delta(i)$. Hence, The cost will be $CE = -\log \hat{y}_i$, where $\hat{y}_i$ is the probability of the i'th word to be predicted (which is softmax in this question). Deriving $J = CE$ we get:

$$J(\hat{r}, \mathbf{w}) = -\log \frac{e^{\mathbf{w}_i^T \hat{r}}}{\sum_{j=1}^{|V|} e^{\mathbf{w}_j^T \hat{r}}} \tag{3.1}$$

$$= -\log e^{\mathbf{w}_i^T \hat{r}} + \log \sum_{j=1}^{|V|} e^{\mathbf{w}_j^T \hat{r}} \tag{3.2}$$

$$= -\mathbf{w}_i^T \hat{r} + \log \sum_{j=1}^{|V|} e^{\mathbf{w}_j^T \hat{r}} \tag{3.3}$$

$$\frac{\partial J(\hat{r}, \mathbf{w})}{\partial \hat{r}} = -\mathbf{w}_i^T + \frac{\sum_{j=1}^{|V|} \mathbf{w}_j \cdot e^{\mathbf{w}_j^T \hat{r}}}{\sum_{j=1}^{|V|} e^{\mathbf{w}_j^T \hat{r}}} \tag{3.4}$$

$$= -\mathbf{w}_i^T + \mathbf{w}_j \sum_{j=1}^{|V|} P(\mathbf{w}_j | \hat{r}, \mathbf{w}) \tag{3.5}$$

3.2 b

Now that $J(\hat{r}, \mathbf{w})$ is written explicitly, it is easy to derive that:

$$\frac{\partial J(\hat{r}, \mathbf{w})}{\partial \mathbf{w}_j} = -\hat{r} \cdot \mathbb{1}_{i=j} + \hat{r} \cdot \frac{e^{\mathbf{w}_j^T \hat{r}}}{\sum_{i=1}^{|V|} e^{\mathbf{w}_i^T \hat{r}}} \tag{3.6}$$

$$= -\hat{r} \cdot \mathbb{1}_{i=j} + \hat{r} \cdot P(\mathbf{w}_j | \hat{r}, \mathbf{w}) \tag{3.7}$$

3.3 c

Now were using negative sampling loss for the predicted vector $\hat{r}$.
The expected output word is w_i, and the negative samples are $w_1 \dots w_k$ where $i \notin 1 \dots k$.

As noted, the negative loss function here is:

$$J(\hat{r}, w_i, w_{1\ldots k}) = -\log(\sigma(w_i^T \hat{r})) - \sum_{k=1}^{K} \log(\sigma(-w_k^T \hat{r})) \tag{3.8}$$

For convenience, let us denote $x_j = w_j^T \hat{r}$, derive by it and then use the chain rule. While the left term of 3.8 only matters when taking derivative by i, we will distinguish between the two cases:

for $j \neq i$:

$$\frac{\partial J}{\partial x_j} = -\frac{\partial}{\partial x_j} \sum_{k=1}^{K} \log(\sigma(-x_k)) \tag{3.9}$$

$$= -\frac{\partial}{\partial x_j} \log(\sigma(-x_j)) \tag{3.10}$$

$$= -\frac{\sigma'(-x_j)}{\sigma(-x_j)} \tag{3.11}$$

$$= \frac{\sigma(-x_j)(1 - \sigma(-x_j)}{\sigma(-x_j)} \tag{3.12}$$

$$= 1 - \sigma(-x_j) \tag{3.13}$$

$$= \sigma(x_j) \tag{3.14}$$

for $j = i$

$$\frac{\partial J}{\partial x_j} = -\frac{\partial}{\partial x_j} \log(\sigma(x_j)) \tag{3.15}$$

$$= -\frac{\sigma'(x_j)}{\sigma(x_j)} \tag{3.16}$$

$$= -\frac{\sigma(x_j)(1 - \sigma(x_j)}{\sigma(x_j)} \tag{3.17}$$

$$= \sigma(x_j) - 1 \tag{3.18}$$

Using the chain rule, we can obtain:

$$\frac{\partial J}{\partial \hat{r}} = \sum_{j \in i, 1 \dots K} \frac{\partial J}{\partial x_j} \frac{\partial x_j}{\partial \hat{r}} \tag{3.19}$$

$$= \sum_{j \in i, 1 \dots K} (\sigma(x_j) - \mathbb{1}_{j=i}) \, \mathbf{w}_j \tag{3.20}$$

$$\frac{\partial J}{\partial \mathbf{w}_j} = \frac{\partial J}{\partial x_j} \frac{\partial x_j}{\partial \mathbf{w}_j} \tag{3.21}$$

$$= (\sigma(x_j) - \mathbb{1}_{j=i}) \, \hat{r} \tag{3.22}$$

As we're only summing over the K negative samples, rather then the whole vocabulary - this negative sample loss is much more efficient than the softmax-CE loss.

3.4 d

In the skip-gram model we simply sum the gradients calculated for each context:

$$\frac{\partial J}{\partial \hat{r}} = \sum_{-c \leq j \leq c, j \neq 0} \frac{\partial F(\mathbf{v}'_{w_{i+j}} | \mathbf{v}_{w_i})}{\partial \hat{r}} \tag{3.23}$$

$$\frac{\partial J}{\partial \mathbf{w}_j} = \sum_{-c \leq j \leq c, j \neq 0} \frac{\partial F(\mathbf{v}'_{w_{i+j}} | \mathbf{v}_{w_i})}{\partial \mathbf{w}_j} \tag{3.24}$$

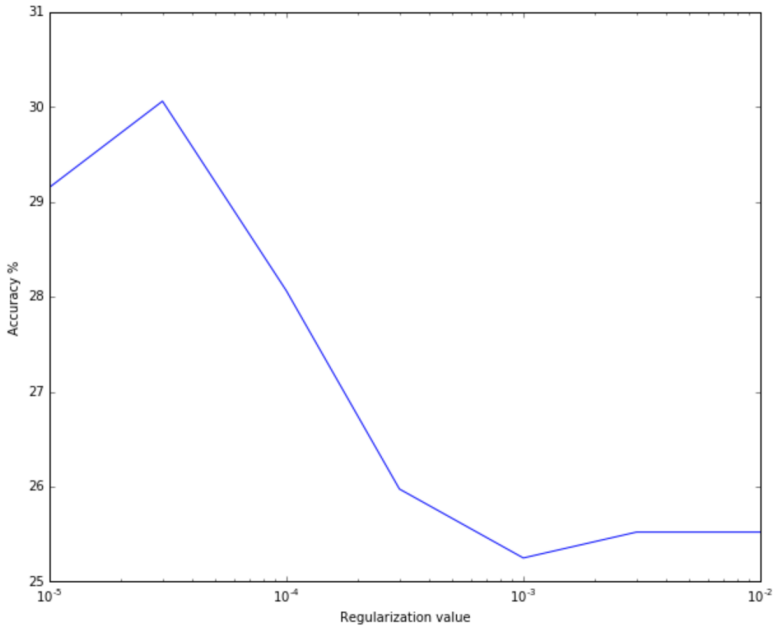
4 Sentiment Analysis

4.1 a

When we introduced regularizations to our classifications tasks, we basically prevented/decreased the effect of *overfitting* in our problem. Otherwise, the parameters would adjust themselves perfectly to our training data, resulting in a bad test data classification results.

4.2 b

Heres a graph specifying the accuracy achieved while testing our method on the dev set, where each time we are using a different regularization term:



We can see that, in general, when the regularization term is bigger - the results on the dev sets reach better accuracy. This fact settles well with our answer in 4.1, meaning that we constrain the parameters not to overfit to the training data.