# CS224N Winter 2016 Homework [2]

SUNet ID: [06074217]
Name: [Jiajun Sun]

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

## Problem 1

(a)

(b)

(c) The placeholder variable are created to store input data. There are two placeholder variable one for $X$ the other for label. Feed dictionaries are used for specifying variable for tensorflow placeholder. Here feed dictionaries feed batch data to two placeholders that we created.

(d)

(e) Tensorflow utilises the computation graph user created and automatically computes the gradient against all the tensorflow variable.

# Problem 2

(a) Below table shows the sequence of transitions needed for parsing the sentence.

| stack | buffer | new dependency | transition |
|---|---|---|---|
| [ROOT] | [I, parsed, this, sentence, correctly] | | Initial Configuration |
| [ROOT, I] | [parsed, this, sentence, correctly] | | SHIFT |
| [ROOT, I, parsed] | [this, sentence, correctly] | | SHIFT |
| [ROOT, parsed] | [this, sentence, correctly] | parsed → I | LEFT-ARC |
| [ROOT, parsed, this] | [sentence, correctly] | | SHIFT |
| [ROOT, parsed, this, sentence] | [correctly] | | SHIFT |
| [ROOT, parsed, sentence] | [correctly] | sentence → this | LEFT-ARC |
| [ROOT, parsed] | [correctly] | parsed → sentence | RIGHT-ARC |
| [ROOT, parsed, correctly] | [ ] | | SHIFT |
| [ROOT, parsed] | [ ] | parsed → correctly | RIGHT-ARC |
| [ROOT] | [ ] | ROOT → parsed | RIGHT-ARC |

(b) A sentence contains $n$ words will be parsed in $2n$ steps. Within these $2n$ steps, $n$ steps are used for SHIFT and the other $n$ steps are used for LEFT-ARC or RIGHT-ARC.

(c)

(d)

(e)

(f) First let's derive the expression for the expectation:

$$\mathbb{E}_{p_{drop}}[h_{drop}]_i = \gamma(1 - p_{drop})h_i = h_i$$

Therefore, in order for the equation above to hold:

$$\gamma = \frac{1}{1 - p_{drop}} \tag{1}$$

(g) The $m$ step will merge the immediate updates with previous updates. For example if $\beta_1 = 0.9$, the immediate has a weight of 0.1 and previous has a weight of 0.9. This step help the training to be easier to converge. Normal SGD process usually carries stochastic fluctuation at the end of training.

$\sqrt{\nu}$ is actually a scaling factor for learning rate. When gradient is larger, the learning rate will be scaled down and vice versa. This process can give the training faster converge compared to fix learning rate.

(h) best UAV on dev sets: 88.50
best UAV on test sets: 89.12

# Problem 3

(a) Since $y_j^t$ is a one-hot vector, let's assume its $m$ element is 1, which is $y_{j,m}^t = 1$. Therefore the perplexity and cross-entropy loss becomes:

$$J_\theta^t = -\log \hat{y}_{j,m}^t$$

$$PP^t(y^t, \hat{y^t}) = \frac{1}{\hat{y}_{j,m}^t}$$

When the cross-entropy loss gets its minimal, $\hat{y}_{j,m}^t$ gets its maximal at the same time. As a result, perplexity gets its minimal as well.

When the prediction is random (uniform distribution):

$$\hat{y}_j^t = [1/|V|, 1/|V|, ..., 1/|V|, 1/|V|] \tag{2}$$

Therefore, the cross-entropy loss becomes when $|V| = 10000$:

$$J_\theta^t = -\log 1/10000 = \log 10000 \tag{3}$$

and the perplexity:

$$PP^t(y^t, \hat{y^t}) = \frac{1}{1/10000} = 10000 \tag{4}$$

(b) Given we have derive the gradient for cross-entropy and softmax function in previous assignment, below are some properties we already know:

$$\sigma' = \sigma(1 - \sigma) \tag{5}$$

$$\frac{\partial CE(y, \hat{y})}{\partial softmax(\theta)} \frac{softmax(\theta)}{\partial \theta} = (\hat{y} - y) \tag{6}$$

In the below derivation, I will directly apply above equations:

$$\begin{aligned}
\frac{\partial J^{(t)}}{\partial b_2} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial b_2} \\
&= \sum_i \frac{\partial J^{(t)}}{\partial \hat{y_i}^{(t)}} \frac{\partial \hat{y_i}^{(t)}}{\partial b_{2,j}} = \sum_i \frac{-y_i^{(t)}}{\hat{y_i}^{(t)}} \frac{\partial \hat{y_i}^{(t)}}{\partial b_{2,j}} \\
&= \sum_i \frac{-y_i^{(t)}}{\hat{y_i}^{(t)}} \hat{y_i}^{(t)} (y_j^{(t)} - \hat{y_j}^{(t)}) \mathbf{1} \\
&= -(y_j^{(t)} - \hat{y_j}^{(t)}) \mathbf{1} \\
&= \hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}
\end{aligned} \tag{7}$$

$$\begin{aligned}
\frac{\partial J^{(t)}}{\partial L_x^{(t)}} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial e^{(t)}} \frac{\partial e^{(t)}}{\partial L_x^{(t)}} \\
&= \sum_i (\frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}})_i \frac{\partial h_i}{\partial e_j^{(t)}} \frac{\partial e_j^{(t)}}{\partial L_x^{(t)}} \\
&= \sum_i (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) U_i^T h_i^{(t)} (1 - h_i^{(t)}) I_{i,j}^T 1 \\
&= (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) U^T \circ h^{(t)} \circ (1 - h^{(t)}) I^T
\end{aligned} \tag{8}$$

$$\begin{aligned}\left.\frac{\partial J}{\partial I}\right|_{(t)} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}}\frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}}\frac{\partial h^{(t)}}{\partial I} \\ &= (\frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}}\frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}})_i\frac{\partial h_i}{\partial I_{i,j}} \\ &= (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)})U_i^T h_i^{(t)}(1-h_i^{(t)})e_j^{(t)} \\ &= e^{(t)^T}(\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)})U^T \circ h^{(t)} \circ (1-h^{(t)})\end{aligned} \tag{9}$$

$$\begin{aligned}\left.\frac{\partial J}{\partial H}\right|_{(t)} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}}\frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}}\frac{\partial h^{(t)}}{\partial H} \\ &= (\frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}}\frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}})_i\frac{\partial h_i}{\partial H_{i,j}} \\ &= (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)})U_i^T h_i^{(t)}(1-h_i^{(t)})h_j^{(t-1)} \\ &= h^{(t-1)^T}(\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)})U^T \circ h^{(t)} \circ (1-h^{(t)})\end{aligned} \tag{10}$$

(c) First denotes $\frac{\partial J^{(t)}}{\partial h^{(t-1)}}$ as $\delta^{(t-1)}$ and:

$$\delta^{(t-1)} = (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)})U^T \circ h^{(t)} \circ (1-h^{(t)})H^T$$

$$\begin{aligned}\frac{\partial J^{(t)}}{\partial L_x^{(t-1)}} &= \frac{\partial J^{(t)}}{\partial h^{(t-1)}}\frac{\partial h^{(t-1)}}{\partial L_x^{(t-1)}} \\ &= (\delta^{t-1} \circ h^{(t-1)} \circ (1-h^{(t-1)}))I^T\end{aligned} \tag{11}$$

$$\begin{aligned}\left.\frac{\partial J^{(t)}}{\partial I}\right|_{(t-1)} &= \frac{\partial J^{(t)}}{\partial h^{(t-1)}}\frac{\partial h^{(t-1)}}{\partial I} \\ &= e^{(t-1)^T}(\delta^{t-1} \circ h^{(t-1)} \circ (1-h^{(t-1)}))\end{aligned} \tag{12}$$

$$\begin{aligned}\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t-1)} &= \frac{\partial J^{(t)}}{\partial h^{(t-1)}}\frac{\partial h^{(t-1)}}{\partial H} \\ &= h^{(t-2)^T}(\delta^{t-1} \circ h^{(t-1)} \circ (1-h^{(t-1)}))\end{aligned} \tag{13}$$

(d) **Forward cost:**
For the forward propogation,

$$\begin{aligned}cost(h) &= o(D_h^2) + o(D_h) \\ cost(\sigma) &= o(D_h|V|) \\ cost(J) &= o(|V|)\end{aligned}$$

Therefore, the total cost for forward propogation:

$$o(D_h^2) + o(D_h) + o(D_h|V|) + o(|V|) \approx o(D_h^2) + o(D_h|V|)$$

If assume the vocabulary size $|V|$ is much greater than hidden layer dimension, we can have:

$$cost \approx o(D_h|V|)$$

**Backward cost:**
The cost for backward propogation is comprised of the following element:

$$cost(\delta^{(t-1)}) = o(D_h|V|) + o(D_h) + o(D_h^2)$$

$$cost(\frac{\partial J^{(t)}}{\partial L_x^{(t-1)}}) = o(D_h) + o(dD_h)$$

$$cost(\left.\frac{\partial J^{(t)}}{\partial I}\right|_{(t-1)}) = o(dD_h)$$

$$cost(\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t-1)}) = o(D_h^2)$$

Therefore, the total cost for backward propogation is:

$$o(D_h|V|) + o(D_h) + o(D_h^2) + o(D_h) + o(dD_h) + o(dD_h) + o(D_h^2) \approx o(D_h|V|) + o(dD_h) + o(D_h^2)$$

If we assume the vocabulary size is much larger than other dimensions:

$$cost \approx o(D_h|V|)$$

When adding multiple steps back in time, the cost matters is for calculating $\delta$:

$$\begin{aligned} cost &\approx n \times cost(\delta) \\ &= o(nD_h|V|) + o(nD_h) + o(nD_h^2) \\ &\approx o(D_h|V|) \end{aligned}$$

(e) We find most the computation are constriants by the dimension of vocabulary. One possible way is to add a layer that rather take work vector into a word class vector. Then take this word class vector into the RNN. This should reduce the cost for both forward and backward propogation.