Assignment 2

1 Tensorflow Softmax

(a) See file q1_softmax.py

(b) See file q1_softmax.py

(c) The purpose of placeholder is to hold our input data, and the purpose of feed dictionaries is to feed input values to placeholders. See implementation in file q1_classifier.py

(d) See file q1_classifier.py

(e) See file q1_classifier.py. After the model's *train_op* is called, the prediction *y_hat* is computed in the forward propagation and the gradient of loss with respect to W and b is computed during the back propagation. The variable W and b will be changed.

2 Neural Transition-Based Dependency Parsing

(a) The parsing procedure is as follows:

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

Table. 1: Parsing Procedure

(b) A sentence containing n words will be parsed in $2 \times n$ steps. This is true because every word will be shifted into the stack once, and will be removed from the stack once, therefore, the total number of steps is $2 \times n$.

(c) See file q2_parser_transitions.py

(d) See file q2_parser_transitions.py

(e) See file q2_initialization.py

(f) The constant γ can be expressed as:

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

This is true because the expected value of h_{drop} is $(1 - p_{drop})h$.

(g)

(i) By using m, the amount of steps we take at each update will now become the running average of gradients of all times. And the current gradient calculated only contribute a little to the m, therefore it is more stable and can stop the updates from varying too much.

(ii) The parameters that have smaller gradient will get larger updates. This helps the learning because it will smooth the updates we make and try to update all parameters equally.

(h) See file q2_parser_model.py and predictions in file q2_test.predicted.pkl

The best UAS my model achieves on the dev set is 88.70, and the UAS achieves on the test set is 88.87.

3 Recurrent Neural Networks: Language Modeling

(a)

(i) Assuming k is the right word at position $t+1$, then we have:

$$\begin{aligned}
\frac{1}{\exp\Big(-J^{(t)}(\theta)\Big)} &= \frac{1}{\exp\Big(\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}\Big)} \\
&= \frac{1}{\hat{y}_k^t} \\
&= \frac{1}{\bar{P}(\boldsymbol{x}_{pred}^{(t+1)} = \boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \\
&= \text{PP}\Big(y^{(t)}, \hat{y}^{(t)}\Big)
\end{aligned}$$

(ii) By using the result above, we have:

$$\begin{aligned}
\log\left(\prod_{t=1}^{T} PP(y^{(t)}, \hat{y}^{(t)})\right)^{1/T} &= \log\left(\frac{1}{\exp\big(-\sum_{t=1}^{T} J^{(t)}(\theta)\big)}\right)^{1/T} \\
&= \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)
\end{aligned}$$

For any positive function, minimizing its logarithm is equivalent to minimizing f itself, therefore minimizing the geometric mean perplexity is the same as minimizing the function above, which is the arithmetic mean cross-entropy loss.

(iii) For a single word, the perplexity is $|V|$. When $|V| = 10000$, the cross-entropy loss is $\log 10000 \approx 9.21$

(b) Let's define:

$$\begin{aligned}
z^{(t)} &= W_h h^{(t-1)} + W_e e^{(t)} + b_1 \\
\theta^{(t)} &= U h^{(t)} + b_2 \\
\delta_1^{(t)} &= \frac{\partial J}{\partial \theta^{(t)}} \\
\delta_2^{(t)} &= \frac{\partial J}{\partial z^{(t)}} = \delta_1^{(t)} \frac{\partial \theta^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial z^{(t)}}
\end{aligned}$$

We then have:

$$
\begin{aligned}
\delta_1^{(t)} &= \hat{y}^{(t)} - y^{(t)} \\
\frac{\partial J^{(t)}}{\partial U} &= \delta_1^{(t)} {h^{(t)}}^T \\
\frac{\partial J^{(t)}}{\partial h^{(t)}} &= U^T \delta_1^{(t)} \\
\delta_2^{(t)} &= (U^T \delta_1^{(t)}) \odot \big(z^{(t)} \odot (1 - z^{(t)})\big) \\
\frac{\partial J^{(t)}}{\partial W_e}|_{(t)} &= \delta_2^{(t)} {e^{(t)}}^T \\
\frac{\partial J^{(t)}}{\partial W_h}|_{(t)} &= \delta_2^{(t)} {h^{(t-1)}}^T \\
\frac{\partial J^{(t)}}{\partial h^{(t-1)}} &= {W_h}^T \delta_2^{(t)}
\end{aligned}
$$

(c)

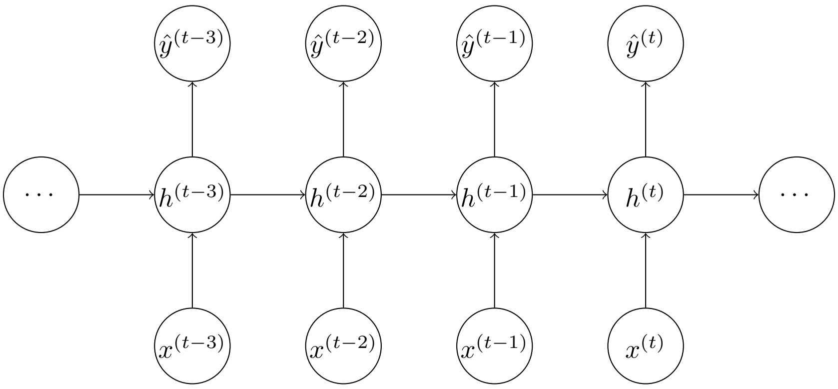

Figure. 1: The "Unrolled" Network for 3 Timesteps

The gradients can be calculated as follows:

$$
\begin{aligned}
\frac{\partial J^{(t)}}{\partial e^{(t-1)}} &= {W_e}^T \Big(\gamma^{(t-1)} \odot \big(z^{(t-1)} \odot (1 - z^{(t-1)})\big)\Big) \\
\frac{\partial J^{(t)}}{\partial W_e}|_{(t-1)} &= \delta_2^{(t)} {e^{(t)}}^T + \Big(\gamma^{(t-1)} \odot \big(z^{(t-1)} \odot (1 - z^{(t-1)})\big)\Big) {e^{t-1}}^T \\
\frac{\partial J^{(t)}}{\partial W_h}|_{(t-1)} &= \delta_2^{(t)} {h^{t-1}}^T + \Big(\gamma^{(t-1)} \odot \big(z^{(t-1)} \odot (1 - z^{(t-1)})\big)\Big) {h^{t-2}}^T
\end{aligned}
$$

(d) Given $h^{(t-1)}$, first we need $O(d \times |V| + D_h \times D_h + D_h \times d + |V| \times D_h)$ to do forward propagation. Then we need $O(|V| \times D_h + |V| \times D_h + D_h \times d + D_h \times D_h + D_h \times D_h)$ to do back propagation.

(e) The total number of operations we need to do forward propagation is simply $O(T \times \big(d \times |V| + D_h \times D_h + D_h \times d + |V| \times D_h\big))$, and the total number of operations we need to do back propagation is:

$$O(3 \times T \times \big(|V| \times D_h + D_h \times d + D_h \times D_h\big))$$

(f) The term $|V| \times D_h$ is likely to be the largest if we use a large corpus, and it's from the output layer (the one that calculates the output according to current hidden state) of the RNN.