1 Tensorflow Softmax

(c) In Tensorflow, a placeholder represents an input node in the computational graph that you can feed data to when computing the output values of the graph. A feed_dict is a dictionary that defines those input values for all placeholders on every time-step/iteration.

(e) Tensorflow's automatic differentiation removes the need for us to define the gradients in the computation graph. Every node in the computation graph internally both knows how to compute an output value given the required input values, as well as how to compute it's own derivative. Thus the gradient can flow backwards through the computation graph.

2 Deep Networks for NER

(a) First, define the network in standard notation.

$$\begin{aligned}
\mathbf{e}^{(t)} &= [\mathbf{x}_{t-1}\mathbf{L}, \mathbf{x}_t\mathbf{L}, \mathbf{x}_{t+1}\mathbf{L}] \\
\mathbf{a}_1 &= \mathbf{e}^{(t)} \\
\mathbf{z}_2 &= \mathbf{a}_1\mathbf{W} + \mathbf{b}_1 \\
\mathbf{a}_2 &= \mathbf{h} = \tanh(\mathbf{z}_2) \\
\mathbf{z}_3 &= \mathbf{a}_2\mathbf{U} + \mathbf{b}_2 \\
\mathbf{a}_3 &= \hat{\mathbf{y}} = \text{softmax}(\mathbf{z}_3) \\
J(\theta) &= \text{CE}(\mathbf{y}, \hat{\mathbf{y}})
\end{aligned} \tag{1}$$

With $e^{(t)}$ being the $1 \times 3d$ dimensional row vector of 3 concatenated word embeddings, each of length $d = 50$. The network has 100 hidden neurons and 5 output classes, corresponding to the 5 kinds of named entities.

The dimensions of the variables are therefore:

$$\begin{aligned}
\mathbf{a_1}^{1\times150}, \mathbf{a}_2^{1\times100}, \mathbf{a}_3^{1\times5}, \mathbf{z}_2^{1\times100}, \mathbf{z}_3^{1\times5}, \mathbf{e}^{1\times150}, \mathbf{h}^{1\times100}, \hat{\mathbf{y}}^{1\times5} \\
\mathbf{b}_1^{1\times100}, \mathbf{b}_2^{1\times5}, \mathbf{W}^{150\times100}, \mathbf{U}^{100\times5}
\end{aligned} \tag{2}$$

The softmax and cross-entropy are defined as follows.

$$\begin{aligned}
\text{softmax}(\mathbf{x})_i &= \frac{e^{x_i}}{\sum_j e^{x_j}} \\
\text{CE}(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_i y_i \log \hat{y}_i
\end{aligned} \tag{3}$$

From assignment 1 it is known that $\frac{\partial \text{CE}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{x}} = \hat{\mathbf{y}} - \mathbf{y}$ where $\hat{\mathbf{y}}$ is the output of a softmax and $\mathbf{x}$ is the input to that softmax. Furthermore the derivative of tanh can be found in the following way.

$$\begin{aligned}
\frac{d}{dz}\tanh(z) &= \frac{d}{dz}2\sigma(2z) - 1 = 4(\sigma(2z)(1 - \sigma(2z)) \\
&= 4\sigma(2z) - 4\sigma^2(2z) = -(4\sigma^2(2z) - 4\sigma(2z) + 1) + 1 \\
&= -(2\sigma(2z) - 1)^2 + 1 = 1 - \tanh^2(z)
\end{aligned} \tag{4}$$

The gradients w.r.t. all variable terms in the computation graph ($\mathbf{U}, \mathbf{W}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{L}_i$)

can then be found by using the chain rule.

$$
\begin{aligned}
\frac{\partial J}{\partial \mathbf{z}_3} &= (\hat{\mathbf{y}} - \mathbf{y}) = \delta_3 \\
\frac{\partial J}{\partial \mathbf{U}} &= \frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial \mathbf{z}_3}{\partial \mathbf{U}} = \mathbf{a}_2^T \delta_3 \\
\frac{\partial J}{\partial \mathbf{b}_2} &= \frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial \mathbf{z}_3}{\partial \mathbf{b}_2} = \delta_3 \\
\frac{\partial J}{\partial \mathbf{z}_2} &= \frac{\partial J}{\partial \mathbf{z}_3}\frac{\partial \mathbf{z}_3}{\partial \mathbf{z}_2} = (\delta_3 \mathbf{U}^T) \circ (1 - \tanh^2(\mathbf{z}_2)) = \delta_2 \\
\frac{\partial J}{\partial \mathbf{W}} &= \frac{\partial J}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{W}} = \mathbf{a}_1^T \delta_2 \\
\frac{\partial J}{\partial \mathbf{b}_1} &= \frac{\partial J}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{b}_1} = \delta_2 \\
\frac{\partial J}{\partial \mathbf{L}_i} &= \frac{\partial J}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{L}_i} = \delta_2 \mathbf{W}^T_{id:(i+1)d}
\end{aligned}
\tag{5}
$$

(b) A regularization term in the cost function can help against overfitting.

$$
J(\theta)_{full} = J(\theta) + \frac{\lambda}{2}[\sum_{i,j} \mathbf{W}_{ij}^2 + \sum_{i',j'} \mathbf{U}_{i'j'}^2] \tag{6}
$$

This influences the gradient terms. The gradient terms of this updated cost function w.r.t. $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{L}_i$ stay the same because the regularization cost does not depend on them. However, the gradients w.r.t. $\mathbf{W}$ and $\mathbf{U}$ have to be updated.

$$
\begin{aligned}
\frac{\partial J_{full}}{\partial \mathbf{U}} &= \frac{\partial J}{\partial \mathbf{U}} + \frac{\partial J_{reg}}{\partial \mathbf{U}} = \mathbf{a}_2^T \delta_3 + \lambda \mathbf{U} \\
\frac{\partial J_{full}}{\partial \mathbf{W}} &= \frac{\partial J}{\partial \mathbf{W}} + \frac{\partial J_{reg}}{\partial \mathbf{W}} = \mathbf{a}_1^T \delta_2 + \lambda \mathbf{W}
\end{aligned}
\tag{7}
$$

3 Recurrent Neural Networks: Language Modeling

(a) Perplexity is defined as follows.

$$\mathrm{PP}^{(t)}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \hat{y}_j^{(t)}} \tag{8}$$

This score can be derived from the cross-entropy loss function. Considering that the ground-truth $\mathbf{y}^{(t)}$ is a one-hot vector with index i denoting the index of the 1 element, both the cross-entropy and the perplexity can be rewritten.

$$\begin{aligned} \mathrm{PP}^{(t)}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) &= \frac{1}{\hat{y}_i^{(t)}} \\ \mathrm{CE}^{(t)}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) &= -\log y_i^{(t)} = \log \frac{1}{y_i^{(t)}} \end{aligned} \tag{9}$$

Thus, the cross-entropy loss is simply the log of the perplexity.

For a completely random initial prediction, the probability distribution $\hat{\mathbf{y}}^{(t)}$ would just be uniform with each element having a probability of $\frac{1}{|V|}$. Therefore $E[\hat{y}_i] = \frac{1}{|V|}$ and $E[PP(\mathbf{y}, \hat{\mathbf{y}})] = |V|$. For $|V| = 10000$ words, an initial cross-entropy of $\log 10000 = 9.21$ could then be expected.

(b) Denoting the model in standard notation for a particular time-step t.

$$\begin{aligned} \mathbf{a}_1^{(t)} &= \mathbf{e}^{(t)} = \mathbf{x}^{(t)}\mathbf{L} \\ \mathbf{z}_2^{(t)} &= \mathbf{h}^{(t-1)}\mathbf{H} + \mathbf{e}^{(t)}\mathbf{I} + \mathbf{b}_1 \\ \mathbf{a}_2^{(t)} &= \mathbf{h}^{(t)} = \sigma(\mathbf{z}_2^{(t)}) \\ \mathbf{z}_3^{(t)} &= \mathbf{h}^{(t)}\mathbf{U} + \mathbf{b}_2 \\ \mathbf{a}_3^{(t)} &= \hat{\mathbf{y}}^{(t)} = \mathrm{softmax}(\mathbf{z}_3^{(t)}) \\ J^{(t)} &= \mathrm{CE}^{(t)}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) \end{aligned} \tag{10}$$

Given that d is the dimension of each word embedding, $|V|$ is the size of the vocabulary and D_h is the hidden layer dimension, the following dimensions hold for the variables.

$$\mathbf{e}^{1\times d}, \mathbf{h}^{1\times D_h}, \hat{\mathbf{y}}^{1\times |V|}, \mathbf{L}^{|V|\times d}, \mathbf{H}^{D_h\times D_h}, \mathbf{I}^{d\times D_h}, \mathbf{U}^{D_h\times |V|}, \mathbf{b}_1^{1\times D_h}, \mathbf{b}_2^{1\times |V|} \tag{11}$$

The derivatives for the time-step t can then be expressed as follows.

$$
\begin{aligned}
\frac{\partial J^{(t)}}{\partial \mathbf{z}_3^{(t)}} &= \hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)} = \delta_3^{(t)} \\
\frac{\partial J^{(t)}}{\partial \mathbf{U}} &= \mathbf{a}_2^{(t)T} \delta_3^{(t)} \\
\frac{\partial J^{(t)}}{\partial \mathbf{b}_2^{(t)}} &= \delta_3^{(t)} \\
\frac{\partial J^{(t)}}{\partial \mathbf{z}_2^{(t)}} &= (\delta_3^{(t)} \mathbf{U}^{(t)T}) \circ \dot{\sigma}(\mathbf{z}_2^{(t)}) = \delta_2^{(t)} \\
\left.\frac{\partial J^{(t)}}{\partial \mathbf{I}}\right|_{(t)} &= \mathbf{e}^{(t)T} \delta_2^{(t)} \\
\left.\frac{\partial J^{(t)}}{\partial \mathbf{H}}\right|_{(t)} &= \mathbf{h}^{(t-1)T} \delta_2^{(t)} \\
\left.\frac{\partial J^{(t)}}{\partial \mathbf{b}_1}\right|_{(t)} &= \delta_2^{(t)} \\
\frac{\partial J^{(t)}}{\partial \mathbf{L}_{x^{(t)}}} &= \delta_2^{(t)} \mathbf{I}^{(t)T} \\
\frac{\partial J^{(t)}}{\partial \mathbf{h}^{(t-1)}} &= \delta_2^{(t)} \mathbf{H}^{(t)T} = \delta^{(t)}
\end{aligned}
\tag{12}
$$

(c) Using the gradient delta $\delta^{(t)}$ of $J^{(t)}$ w.r.t. $\mathbf{h}^{(t-1)}$ the gradients for $\mathbf{I}$, $\mathbf{H}$ and $\mathbf{b}_1$ in the previous time step $t-1$ can then be calculated.

$$
\begin{aligned}
\frac{\partial J^{(t)}}{\partial \mathbf{z}_2^{(t-1)}} &= \delta^t \circ \dot{\sigma}(\mathbf{z}_2^{(t-1)}) = \delta_2^{(t-1)} \\
\left.\frac{\partial J^{(t)}}{\partial \mathbf{I}}\right|_{(t-1)} &= \frac{\partial J^{(t)}}{\partial \mathbf{z}_2^{(t-1)}} \frac{\partial \mathbf{z}_2^{(t-1)}}{\partial \mathbf{I}} = \mathbf{e}^{(t-1)T} \delta_2^{(t-1)} \\
\left.\frac{\partial J^{(t)}}{\partial \mathbf{H}}\right|_{(t-1)} &= \frac{\partial J^{(t)}}{\partial \mathbf{z}_2^{(t-1)}} \frac{\partial \mathbf{z}_2^{(t-1)}}{\partial \mathbf{H}} = \mathbf{h}^{(t-2)T} \delta_2^{(t-1)} \\
\left.\frac{\partial J^{(t)}}{\partial \mathbf{b}_1}\right|_{(t-1)} &= \frac{\partial J^{(t)}}{\partial \mathbf{z}_2^{(t-1)}} \frac{\partial \mathbf{z}_2^{(t-1)}}{\partial \mathbf{b}_1} = \delta_2^{(t-1)}
\end{aligned}
\tag{13}
$$

This method can then be extended to arbitrary network lengths.