CS 224D Spring 2016 Assignment #2

SUNet ID: 06106557

Name: Gregory Lambert

1 Tensorflow Softmax

(1a) see *q1_softmax.py*

(1b) see *q1_softmax.py*

(1c) Placeholders allow to construct elements of the network prior to the assignment of data and feed dictionaries map graph layers attributes to data. See *q1_classifier.py*

(1d) see *q1_classifier.py*

(1e) Tensorflows automatic differentiation relieves the user from having to explicitly define gradients. In tensorflow we can define a function and return the derivative of the function without having to explicitly define the gradient of that given function. Running *q1_classifier.py* we show that our implementation has a classification loss of less than 0.50 on synthetic data.

2 Deep Networks for Names Entity Recognition

(2a) Compute the gradients of $J(\theta)$ with respect to all model parameters with corresponding matrix operations. We will first define the following

$tanh^{'}(z^{(2)}) = 1 - tanh^2(z^{(2)})$.
$a^{(1)} = x^{(t)} \in \mathbb{R}^{150\times 1}$
$z^{(2)} = Wx^{(t)} + b_1 \in \mathbb{R}^{100\times 1}$
$a^{(2)} = tanh(z^{(2)}) \in \mathbb{R}^{100\times 1}$
$z^{(3)} = Ua^{(2)} + b_2 \in \mathbb{R}^{5\times 1}$
$a^{(3)} = softmax(z^{(3)}) \in \mathbb{R}^{5\times 1}$
$\delta_3 = a^{(3)} - y \in \mathbb{R}^{5\times 1}$
$\delta_2 = (U)^T\delta_3 \circ tanh^{'}(z^{(2)}) \in \mathbb{R}^{100\times 1}$

(2a continued on next page)

$$\begin{aligned}\frac{\partial J}{\partial U} &= \frac{\partial J}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial U} \\ &= (a^{(3)} - y)(a^{(2)})^T \\ &= \delta_3 (a^{(2)})^T \in \mathbb{R}^{5\times 100}\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b_2} &= \frac{\partial J}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial b_2} \\ &= (a^{(3)} - y) \\ &= \delta_3\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial W} &= \frac{\partial J}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial W} \\ &= \delta_2 (a^{(1)})^T \in \mathbb{R}^{100\times 150}\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b_1} &= \frac{\partial J}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial b_1} \\ &= (U^T \delta_3) \circ 1 - tanh^2(z^{(2)}) \\ &= \delta_2\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial L_i} &= \frac{\partial J}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial L_i} \\ &= W^T \delta_2 \in \mathbb{R}^{150\times 1}\end{aligned}$$

(2b) Compute the gradients with for all the model parameters at a single point in time t.

Since $J_{reg}(\theta)$ only has parameters for the weights (W, U), the updated gradients only need to be evaluated for parameters (W, U).

$tanh^{'}(z^{(2)}) = 1 - tanh^2(z^{(2)})$.
$a^{(1)} = x^{(t)} \in \mathbb{R}^{150\times 1}$
$z^{(2)} = Wx^{(t)} + b_1 \in \mathbb{R}^{100\times 1}$
$a^{(2)} = tanh(z^{(2)}) \in \mathbb{R}^{100\times 1}$
$z^{(3)} = Ua^{(2)} + b_2 \in \mathbb{R}^{5\times 1}$
$a^{(3)} = softmax(z^{(3)}) \in \mathbb{R}^{5\times 1}$
$\delta_3 = a^{(3)} - y \in \mathbb{R}^{5\times 1}$
$\delta_2 = (U)^T \delta_3 \circ tanh^{'}(z^{(2)}) \in \mathbb{R}^{100\times 1}$
(2b continued on next page)

$$
\begin{aligned}
\frac{\partial J}{\partial W} &= \frac{\partial J}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial W} \\
&= ((U)^T\delta_3) \circ tanh^{'}(z^{(2)}) \\
&= (U^T((\hat{y}-y) + \lambda\sum_{ij} W_{ij})) \circ tanh^{'}(z^{(2)}) \\
&= \\
\frac{\partial J}{\partial U} &= \frac{\partial J}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial U} \\
&= ((\hat{y}-y) + \lambda\sum_{ij} U_{ij})(a^{(3)}) \\
&= \delta_3
\end{aligned}
$$

(2c) Implementation of the *xavier_weight_init* can be found in *q2_initialization.py*

(2d) See *q2_NER.py* for hyperparameter selection for name entity recognition implementation. A weighted cross entropy approach was used (tf.nn.weighted_cross_entropy_with_logits) to help deal with the unbalanced class labels inherent in the data to attempt at alleviating fitting problems found on previous runs where only the majority label was predict. The optimal hyperparameters found: learning rate (0.001), dropout (0.9), and L2 regularization (0.5). Validation loss was determined to be 0.154 with a training accuracy of 0.581 on 22 iterations of the model.

See *q2_test.predicted* for a list of predicted labels for the test set.

3 Recurrent Neural Networks: Language Modeling

(3a) From lecture notes 4, let perplexity $= 2^{J^{(t)}}$

$$
\begin{aligned}
2^{J^{(t)}} &= 2^{-\sum_{i=1}^{|V|} y_i^{(t)} log_2 \hat{y}_i^{(t)}} \\
&= \frac{1}{2^{\sum_{i=1}^{|V|} y_i^{(t)} log_2 \hat{y}_i^{(t)}}} \\
&= \prod_{i=1}^{|V|} \left(\hat{y}_i^{(t)}\right)^{y_i^{(t)}} \\
&= \frac{1}{\sum_{i=1}^{|V|} y_i^{(t)} \hat{y}_i^{(t)}} \qquad (one-hot\ vector\ y_i)
\end{aligned}
$$

For a vocabulary of $|\mathbf{V}|$ words and random uniform distribution for the prediction any word V in $\mathbf{V}$ we would be $\frac{1}{|\mathbf{V}|}$. The cross-entropy loss for $|V| = 2000$ is $-\sum_i p(v_i) log_2(p(v_i)) = -|V| log_2(\frac{1}{|V|}) = 21932$ and whe $|V| = 10000$ then $-|V| log_2(\frac{1}{|V|}) = 132877$.

(3a continued on next page)

Show that minimizing the arithmetic mean of the cross-entropy loss is equivalent to minimizing the geometric mean of the perplexity

$$
\begin{aligned}
arithmetic\ mean\ of\ J^{(t)} &= -\frac{1}{T}\sum_{T}\sum_{I} y_{t,i} log_2 \hat{y}_{t,i} \\
&= -\frac{1}{T}\sum_{T} \hat{y}_t^{(\mathbf{m})} \qquad (where\ \mathbf{m}\ is\ the\ vector\ of\ target\ words) \\
2^{-\frac{1}{T}\sum_{T} log_2 \hat{y}_t^{(\mathbf{m})}} &= \sqrt[T]{\prod_{T} \frac{1}{\hat{y}_t}} \qquad (geometric\ mean\ of\ the\ perplexity)
\end{aligned}
$$

(3b) Compute the gradients with for all the model parameters at a single point in time t:

$tanh^{'}(z^{(2)}) = 1 - tanh^2(z^{(2)})$.
$e^{(t)} \in x^{(t)} L$
$z^{(2)} = h^{(t-1)} H + e^{(t)} I + b_1$
$a^{(2)} = h^{(t)} = \sigma(z^{(2)})$
$z^{(3)} = (h^{(t)} U + b_2)$
$a^{(3)} = \hat{y}^{(t)} = softmax(z^{(3)})$
$\delta_3 = a^{(3)} - y$
$\delta_2 = (U)^T \delta_3 \circ \sigma^{'}(z^{(2)})$
$\sigma^{'}(z^{(2)}) = (1 - \sigma(z^{(2)})(\sigma(z^{(2)})$

(3b continued on next page)

$$
\begin{aligned}
\frac{\partial J^{(t)}}{\partial U} &= \frac{\partial J^{(t)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial U} \\
&= (a^{(3)} - y) \left(h^{(t)}\right)^T \\
&= \delta_3 \left(h^{(t)}\right)^T \\
\left.\frac{\partial J^{(t)}}{\partial b_2}\right|_t &= \frac{\partial J^{(t)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial b_2} \\
&= \delta_3 \\
\frac{\partial J^{(t)}}{\partial L_{x^{(t)}}} &= \frac{\partial J^{(t)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial L_{x^{(t)}}} \\
&= \delta_2 (x^{(t)} I)^T \\
\left.\frac{\partial J^{(t)}}{\partial I}\right|_t &= \frac{\partial J^{(t)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial I} \\
&= (U^T \delta_3) \circ \sigma^{'}(z^{(2)}) \\
&= \delta_2 (e^{(t)})^T \\
\left.\frac{\partial J^{(t)}}{\partial H}\right|_t &= \frac{\partial J^{(t)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial H} \\
&= \delta_2 \left(h^{(t-1)}\right)^T \\
\left.\frac{\partial J^{(t)}}{\partial b_1}\right|_t &= \delta_2 \\
\frac{\partial J^{(t)}}{\partial h^{(t-1)}} &= \frac{\partial J^{(t)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(t-1)}} \\
&= \delta_2 H^T
\end{aligned}
$$

(3c) Draw the "unrolled" network for 3 timesteps, and compute the backpropagation-through-time gradients.

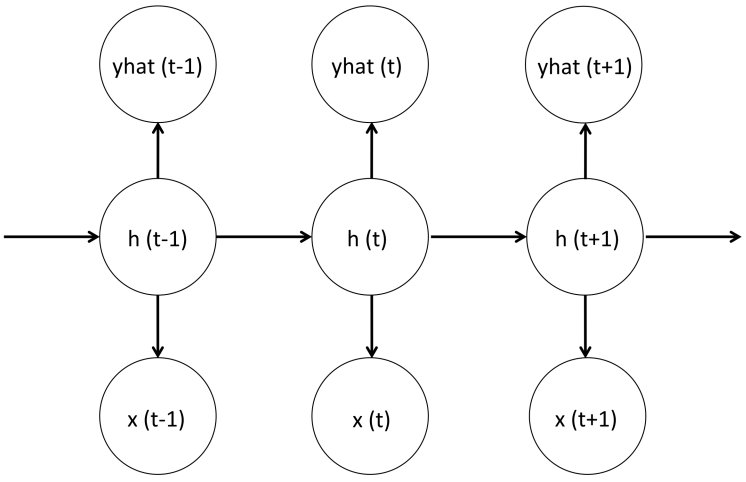

Figure 1: network for 3 timesteps

(3c continued on next page)

$$
\begin{aligned}
\frac{\partial J^{(t)}}{\partial L_{x^{(t-1)}}} &= \frac{\partial J^{(t)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial L_{x^{(t-1)}}} \\
&= \delta_3 \left(h^{(t-1)} \right)^T \\
\left. \frac{\partial J^{(t)}}{\partial H} \right|_{t-1} &= \frac{\partial J^{(t)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial H} \\
&= \delta_2 \left(h^{(t-2)} \right)^T \\
\left. \frac{\partial J^{(t)}}{\partial I} \right|_{t-1} &= \frac{\partial J^{(t)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial I} \\
&= \delta_2 \left(e^{(t-1} \right)^T \\
\left. \frac{\partial J^{(t)}}{\partial b_1} \right|_{t-1} &= \delta_2
\end{aligned}
$$

where all δ terms are for the time point $t-1$

(3d) Given h^{t-1}, how many operations are required to perform one step of forward propagation to compute $J^{(t)}(\theta)$?

The operations can be expressed in terms of $|V|$, d, and D_h. $h^{(t-1)}H$ is D_h^3, $x^{(t)}L$ is $D_h^2|V|$, and $e^{(t)}I$ is $D_h|V|$. So for forward propagation we get $O(|V|D_h^2)$ and for t time steps we have $O(t|V|D_h^2)$. Forward propagation is more computationally expensive per timestep where $t > 1$.

(3e) See *q3_RNNLM.py* for code implementation. For training the RNN language model we ran a maximum number of epochs =30, with early stopping =2. Dropout was used for both inputs and outputs to the sigmoid function, but found through training to only be effective for the inputs. We used probability of 0.95 to determine if each element would be included in the sample. An optimal learning rate of 0.015 was used for the final model selection. Additionally a $\epsilon = 0.0009$ was used for numerlical stability during optimization. The following are the perplexity scores that resulted from the previous hyperparameter selection and the saved model parameters

Training perplexity: 106.682
Validation perplexity: 174.819
Test perplexity: 175.060

Saved model parameters for best model:
batch_size = 75
embed_size = 65
hidden_size = 140
num_steps = 13
max_epochs = 30
early_stopping = 2

dropout = 0.95

lr = 0.015

epsilon=0.0009 (tf.train.AdamOptimizer)

(3f) See *q3_RNNLM.py* for code implementation. Generated sentences:

(1)pizza jr. the a they the exhausted according bond mafia and has while quarter 's drop past market drink stock aeronautics well <eos>

(2)kid towel u.s. attacking securities interim <unk> u.s. remembered to showtime demand pay N with but the <unk> quack say also the board pressure testify that N rating a of any battled tough relief <unk> n't would <unk> in <eos>

(3) in palo alto by the it <eos>