Natural Language Understanding: Assignment 2

Exam number: B095557

28 March 2017

Question 1

Part a

See figure 1.

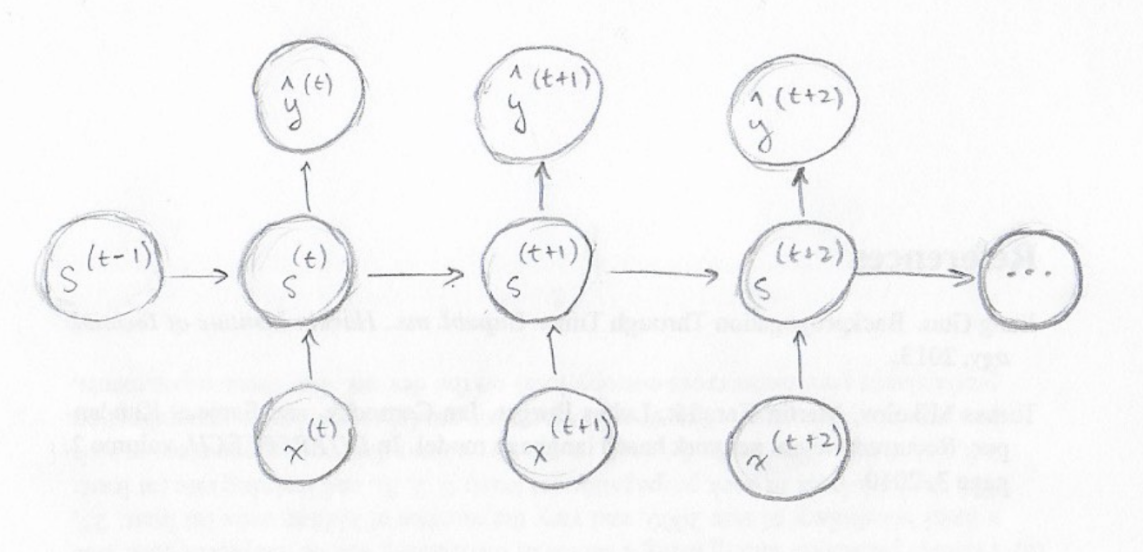

Figure 1

Part b

Softmax is often used as the final layer in multiclass neural network models because it outputs the predicted probabilities of each class label given the inputs. These probabilities are normalized so that they sum to one, which makes intuitive sense since our class labels are mutually exclusive.

The sigmoid function, on the other hand, does not have the restriction that the probabilities of the output classes must sum to 1—and it therefore less appropriate for our model.

Softmax also has nice theoretical properties. From the perspective of information theory, using softmax as our final layer allows our multiclass neural network to minimize *cross-entropy error*. This is equivalent to minimizing the Kullback-Leibler divergence—which means the model's objective is to put its probability mass on the correct class. From the perspective of probability theory, minimizing cross-entropy error is also equivalent to taking the maximum likelihood estimate. Again, using the sigmoid function as our final layer would not grant us these nice properties [1].

Part c

See figure 2.

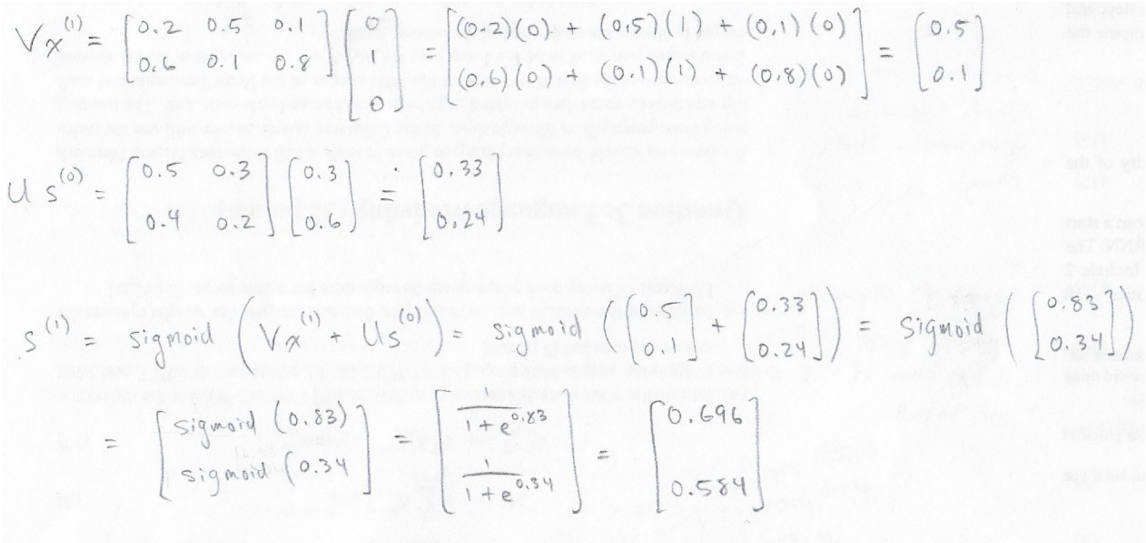

Figure 2

This can also be calculated using the code below.

```
import numpy as np
from rnnmath import *

U = np.array([[0.5, 0.3], [0.4, 0.2]])
V = np.array([[0.2, 0.5, 0.1], [0.6, 0.1, 0.8]])
W = np.array([[0.4, 0.2], [0.3, 0.1], [0.1, 0.7]])
s0 = np.array([0.3, 0.6]).T
x = np.array([0., 1., 0.]).T

s = sigmoid(np.dot(V, x) + np.dot(U, s0))

def array_to_sigfigs(array, sigfigs):
```

```
    format_str = '{:.' + str(sigfigs) + 'g}'
    return [float(format_str.format(i)) for i in array]

print array_to_sigfigs(s, 3)  # [0.696, 0.584]
```

Part d

See figure 3.

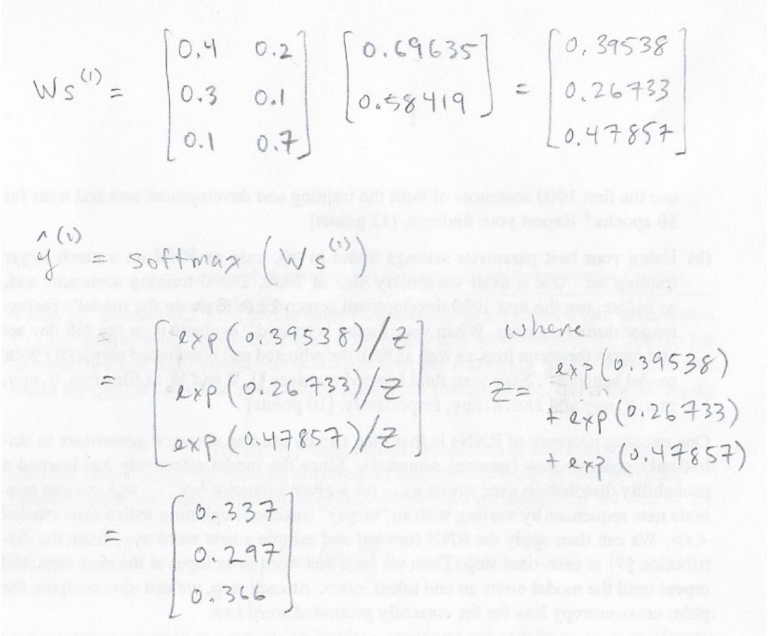

Figure 3

This can also be calculated using the code below.

```
y = softmax(np.dot(W, s))
print array_to_sigfigs(y, 3)  # [0.337, 0.297, 0.366]
```

Question 2

Part a

$$
\begin{aligned}
J^{(t)}(\theta) &= -\sum_{j=1}^{|V|} d_j^{(t)} \log\, \hat{y}_j^{(t)} \\
J^{(1)}(\theta) &= -\sum_{j=1}^{3} d_j^{(1)} \log\, \hat{y}_j^{(1)} \\
&= -\Big((d_1^{(1)})(log\ \hat{y}_1^{(1)}) + (d_2^{(1)})(log\ \hat{y}_2^{(1)}) + (d_3^{(1)})(log\ \hat{y}_3^{(1)})\Big) \\
&= -\Big((0)(\log(0.337)) + (0)(\log(0.297)) + (1)(\log(0.337))\Big) \\
&\approx 1.00 \\
J^{(2)}(\theta) &= -\sum_{j=1}^{3} d_j^{(2)} \log\, \hat{y}_j^{(2)} \\
&= -\Big((0)(\log(0.168)) + (0)(\log(0.229)) + (1)(\log(0.603))\Big) \\
&\approx 0.506 \\
J^{(3)}(\theta) &= -\sum_{j=1}^{3} d_j^{(3)} \log\, \hat{y}_j^{(3)} \\
&= -\Big((0)(\log(0.475)) + (1)(\log(0.317)) + (0)(\log(0.208))\Big) \\
&\approx 1.15
\end{aligned}
\tag{1}
$$

This can also be calculated using the code below.

```
d = np.array([0., 0., 1.]).T
y2 = np.array([0.168, 0.229, 0.603]).T
d2 = np.array([0., 0., 1.]).T
y3 = np.array([0.475, 0.317, 0.208]).T
d3 = np.array([0., 1., 0.]).T

calc_J = lambda d, y: -sum(d * np.log(y))
format_J = lambda x: '{:.3g}'.format(x)

print format_J(calc_J(d, y))  # 1.00
print format_J(calc_J(d2, y2))  # 0.506
print format_J(calc_J(d3, y3))  # 1.15
```

Part d

The recursive update means that the gradient is back-propagated as if the network were unrolled (like in Question 1 Part a). It is similar to the back propagation of most deep networks, except that weights are shared between layers. In contrast, simple back propagation treats the network more or less as if it were rolled up and each time step stood on its own.

An advantage of BPTT can be that it treats the model more completely, taking into account the relationship between time steps. Disadvantages of BPTT include (1) increased algorithm complexity;[1] (2) longer training time due to the additional updates to ΔV and ΔU;[2] and (3) possible issues with local minima [2].

Question 3

Part a

The best model has 25 hidden dimensions, 0 steps for BPTT and an initial learning rate of 0.5, giving a loss of 5.0607 after 10 epochs. Generally, the models seemed most sensitive to learning rate within the values tested, with 0.5 performing consistently well (figure 4). Models with 25 hidden units were particularly susceptible to changes in learning rate.

The terminal output for the chosen model is below.

```
Training model for 10 epochs
training set:  1000 sentences (batch size 100)
Optimizing loss on 1000 sentences
Vocab size:  2000
Hidden units:  25
Steps for back propagation:  0
Initial learning rate set to 0.5, annealing set to 5

calculating initial mean loss on dev set:  8.01793038639

epoch 1, learning rate 0.5000 instance 1000 epoch done in 36.26 seconds new loss:  6.97122529865
epoch 2, learning rate 0.4167 instance 1000 epoch done in 32.70 seconds new loss:  5.63657580191
epoch 3, learning rate 0.3571 instance 1000 epoch done in 31.46 seconds new loss:  5.91074281875
epoch 4, learning rate 0.3125 instance 1000 epoch done in 31.31 seconds new loss:  5.22497799733
epoch 5, learning rate 0.2778 instance 1000 epoch done in 31.38 seconds new loss:  5.23584390218
epoch 6, learning rate 0.2500 instance 1000 epoch done in 31.26 seconds new loss:  5.15444291545
epoch 7, learning rate 0.2273 instance 1000 epoch done in 31.59 seconds new loss:  5.10624397674
epoch 8, learning rate 0.2083 instance 1000 epoch done in 31.22 seconds new loss:  5.0999730556
```

[1] This includes need to store the gradients from the previous time step.

[2] In Question 3 Part a, when the number of steps backward was 5, the network took nearly 50% longer to train per epoch than when the number of steps was 0.

Hidden units	Steps back	Learning rate	Loss
25	5	0.5	5.0896
25	5	0.1	5.2774
25	5	0.05	5.5208
25	2	0.5	5.0684
25	2	0.1	5.2849
25	2	0.05	5.4872
25	0	0.5	5.0607
25	0	0.1	5.2936
25	0	0.05	5.4711
50	5	0.5	5.0686
50	5	0.1	5.2277
50	5	0.05	5.3876
50	2	0.5	5.1141
50	2	0.1	5.2453
50	2	0.05	5.3923
50	0	0.5	5.2343
50	0	0.1	5.2207
50	0	0.05	5.4502

Figure 4

```
epoch 9, learning rate 0.1923 instance 1000 epoch done in 31.19 seconds new loss:  5.07773564283
epoch 10, learning rate 0.1786 instance 1000 epoch done in 31.34 seconds new loss:  5.06070020875
```

Part b

The best observed loss was 4.45. The unadjusted and adjusted perplexities were 93.201 and 155.912, respectively.

The output from the terminal is below.

```
Retained 2000 words from 38444 (84.00% of all tokens)

Training model for 10 epochs
training set:  25000 sentences (batch size 100)
Optimizing loss on 1000 sentences
Vocab size:  2000
Hidden units:  25
Steps for back propagation:  0
Initial learning rate set to 0.5, annealing set to 5
```

```
calculating initial mean loss on dev set:  8.00925208467

epoch 1, learning rate 0.5000 instance 25000 epoch done in 473.93 seconds new loss:  4.84608603637
epoch 2, learning rate 0.4167 instance 25000 epoch done in 460.09 seconds new loss:  4.72123471813
epoch 3, learning rate 0.3571 instance 25000 epoch done in 460.01 seconds new loss:  4.64464891614
epoch 4, learning rate 0.3125 instance 25000 epoch done in 452.45 seconds new loss:  4.63983196416
epoch 5, learning rate 0.2778 instance 25000 epoch done in 472.26 seconds new loss:  4.5667536256
epoch 6, learning rate 0.2500 instance 25000 epoch done in 468.75 seconds new loss:  4.53374122262
epoch 7, learning rate 0.2273 instance 25000 epoch done in 573.48 seconds new loss:  4.55377171879
epoch 8, learning rate 0.2083 instance 25000 epoch done in 485.61 seconds new loss:  4.49315698463
epoch 9, learning rate 0.1923 instance 25000 epoch done in 555.29 seconds new loss:  4.46742907606
epoch 10, learning rate 0.1786 instance 25000 epoch done in 485.75 seconds new loss:  4.45303803143

training finished after reaching maximum of 10 epochs
best observed loss was 4.45303803143, at epoch 10
setting U, V, W to matrices from best epoch
Unadjusted:  93.201
Adjusted for missing vocab:  155.912
```

Part c

Some example sentences include:

- <s> in the rose for all , up the companies to u.s. DG.DG , the her show activity will be columbia asia employee for a deal has himself official . </s> (loss = 160, perplexity = e^{160})
- <s> it friday in golden making that flight more , but where a make DGDG years for $ DGDGDG.DG million , communications he said . </s> (loss = 121, perplexity = e^{121})
- <s> jaguar over this yen , we mr. or many of DGDGDG.DG : over a) . </s> (loss = 89.3, perplexity = $e^{89.3}$)

This is with `maxLength = 50`. The `UUUNKKK` terms set to a probability of zero in the $\hat{y}$ vector so that they were not chosen by `multinomial_sample`.

References

[1] J. J. Fei-Fei Li, Andrej Karpathy, "Cs231n: Convolutional neural networks for visual recognition." Stanford University course. Website available at http://cs231n.stanford.edu/. Accessed 13 Mar 2016.

[2] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.