HW3 – Dry

Question 1:

(a) At every iteration of the recursion, we remove at least two edges, since a cycle must have at least two edges and an iteration of recursive call contracts (i.e. removes) the cycle. Every graph with a finite number of nodes must have a finite number of edges, therefore the algorithm must stop after a finite number of steps.

(b) In this section we refer to the pseudocode from the tutorial.

- First, notice that the 'for' loop in lines 4-8 takes exactly $|E|$ operations, since every edge is incoming to exactly one node, eliminating the possibility of going over the same edge twice.
  $$\Rightarrow O(|E|)$$
- Validating a spanning tree means making sure every node is included, therefore the complexity is $O(|V|)$.
  $$\Rightarrow O(|V|)$$
- We know that finding a cycle is $O(|V| + |E|)$.
  $$\Rightarrow O(|V| + |E|)$$
- Contracting a cycle is at worst 2 operations since the worst case of the algorithm is when we eliminate the smallest possible number of nodes and edges at each step.

Therefore, the complexity of the algorithm until the recursive call is:

$$O(|V| + |E| + |V| + |E|) = O(|V| + |E|) \leq O(2|E|) = O(|E|)$$

In addition, cycle contraction removes at least 1 node, meaning there can be at most $|V| - 1$ recursive calls.

Therefore, the overall complexity of the algorithm is:

$$O\big((|V| - 1)|E|\big) = O(|V| \cdot |E|)$$

Question 2:

(a) The forward pass is calculated in the following manner:

$$H = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_3 & w_5 \\ w_2 & w_4 & w_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_1 \end{pmatrix}$$

$$O = \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} = \begin{pmatrix} w_7 & w_9 \\ w_8 & w_{10} \end{pmatrix} \sigma(H) + \begin{pmatrix} b_2 \\ b_2 \end{pmatrix}$$

$$output = \sigma(O)$$

Our input is $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 5 \end{pmatrix}$, therefore the forward pass is:

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \begin{pmatrix} 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \\ 5 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 4.3 \\ 5.3 \end{pmatrix}$$

$$\begin{pmatrix} o_1 \\ o_2 \end{pmatrix} = \begin{pmatrix} 0.7 & 0.9 \\ 0.8 & 0.1 \end{pmatrix} \begin{pmatrix} 0.986 \\ 0.995 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 2.0857 \\ 1.3883 \end{pmatrix}$$

$$output = \sigma \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} = \begin{pmatrix} 0.889 \\ 0.8 \end{pmatrix}$$

$$loss = \frac{1}{2}(0.1 - 0.889)^2 + \frac{1}{2}(0.05 - 0.8)^2 = \frac{1.185}{2}$$

(b) Calculate using the chain rule:

- w_1:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_1} \cdot \frac{\partial o_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} + \frac{\partial L}{\partial o_2} \cdot \frac{\partial o_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial L}{\partial o_1} = \frac{2}{2}\big(y_1 - \sigma(o_1)\big) \cdot \sigma'(o_1) = -\frac{0.142}{2}, \frac{\partial L}{\partial o_2} = \frac{2}{2}\big(y_2 - \sigma(o_2)\big) \cdot \sigma'(o_2) = -\frac{0.2385}{2}$$

$$\frac{\partial o_1}{\partial h_1} = w_7 \cdot \sigma'(h_1) = 0.0238, \qquad \frac{\partial o_2}{\partial h_1} = w_8 \cdot \sigma'(h_1) = 0.0272,$$

$$\frac{\partial h_1}{\partial w_1} = x_1 = 1$$

$$\Rightarrow \frac{\partial L}{\partial w_1} = -\frac{0.0033}{2} - \frac{0.0065}{2} = -\frac{0.01}{2} = -0.005$$

- w_9:

$$\frac{\partial L}{\partial w_9} = \frac{\partial L}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_9}$$

$$\frac{\partial L}{\partial o_1} = \frac{2}{2}\big(y_1 - \sigma(o_1)\big) \cdot \sigma'(o_1) = -\frac{0.142}{2}, \quad \frac{\partial o_1}{\partial w_9} = h_2 = 5.3,$$

$$\Rightarrow \frac{\partial L}{\partial w_9} = -\frac{0.142}{2} \cdot 5.3 = -\frac{0.7526}{2}$$

(c) The vanishing gradient problem occurs in especially deep networks.
Since the gradients are computed using the chain rule and the product of gradients bounded by 1 converges very close to 0, the network cannot learn.

(d) 3 ways to mitigate the vanishing gradient problem:

- LSTM – instead of only holding a hidden state which is constantly rewritten, holds an additional cell state that enables more information to be preserved during propagation through long sequences.
- Sentence length bound – in lieu of a better option, it is possible to have the network preform only on relatively short sequences, therefore alleviating the information loss problem.
- Activation function – the vanishing gradient problem occurs most often when the network gradients are bounded by 1 (for example when using tanh), therefore using a function such as ReLU.

(e) When using vanilla RNNs for sentence-level classification, we take the output from the last cell, meaning we use the information aggregated throughout the forward pass of the sequence through the network.
As in the vanishing gradients problem, this could be a problem when dealing with long sentences, since RNNs preserve from far away very poorly, meaning any information from the beginning of the sentence might not contribute at all to the sentence classification, which is obviously not what we want.

(f) In LSTM architecture, both the 'hidden state' and the 'cell state' are computed for each timestep.
In RNNs, the hidden state at each timestep is identical to the output at each timestep, making it difficult to preserve information about previous timesteps without letting them affect the current output too much.
However, in LSTMs, the output from each timestep is detached from the state of the current timestep, allowing the network to preserve more information at each timestep, while still being able to distill it into the relevant current hidden state.

(g) During each timestep we preform the following:

1. *Forget gate:* controls what is kept vs forgotten, from previous cell state
2. *Input gate:* controls what parts of the new cell content are written to cell
3. *Output gate:* controls what parts of cell are output to hidden state
4. *New cell content*: this is the new content to be written to the cell
5. *Cell state:* erase ("forget") some content from last cell state, and write ("input") some new cell content
6. *Hidden state:* read ("output") some content from the cell

However, computing the values of the input, forget, output gates and the new cell content can be completely parallelized since they are independent.

The cell state is dependent on the value of the new cell content, and the hidden state is dependent on the updated cell state, therefore these must be done sequentially, resulting in a total of 3 sequential operations per timestep (i.e. $O(3n)$ total sequential operations).