Assignment 3

1a)

i) First, Adam uses a trick called momentum by keeping track of m, a rolling average of gradients
$m \leftarrow \beta_1 m + (1 - \beta_1)\nabla_\theta J_{minibatc}\ (\theta)$

$\theta \leftarrow \theta - \alpha m$

Where β_1 is a hyper parameter between 0 and 1(often set to 0.9). Briefly explain (you don't need to prove mathematically, just give an intuition) how using m stops the update from varying as much and why this low variance may be helpful to learning overall.

Ans) Using m stops the updates from varying as much because $m \leftarrow \beta_1 m + (1 - \beta_1)\nabla_\theta J_{minibatc}\ (\theta)$ since β_1 is often set to 0.9, therefore (1 - β_1) = 0.1, therefore the equation becomes $m \leftarrow 0.9m + 0.1\nabla_\theta J_{minibatc}\ (\theta)$. Therefore, this can be thought of as taking the weighted average of m and ∇_θ and since ∇_θ has a very low weight, m mostly depends on the previous value of m and therefore does not vary much.

This low variance is helpful to learning overall, as the gradients don't vary much and therefore there is not much oscillation in the steps taken during gradient descent and therefore gradient descent converges much faster.

ii) Adam also uses adaptive learning rates by keeping track of v, a rolling average of the magnitudes of the gradients:

$m \leftarrow \beta_1 m + (1 - \beta_1)\nabla_\theta J_{minibatch}(\theta)$

$v \leftarrow \beta_2 v + (1 - \beta_2)(\nabla_\theta J_{minibatc}\ (\theta) * \nabla_\theta J_{minibatch}(\theta))$

$\theta \leftarrow \theta - \alpha m/\sqrt{v}$

Where * and / denote element wise multiplication and division and β_2 is a hyper parameter between 0 and 1(often set to 0.99). Since Adam divides the update by $\sqrt{v}$, which of the model parameters will get larger updates? Why might this help with learning?

Ans) The parameters that have smaller gradients will get a larger learning rate, while the parameters that have larger gradients will get a smaller learning rate. Therefore the updates for all the parameters are normalized. This helps with learning because the algorithm converges faster as there is less oscillation in the steps the algorithm takes.

b) Dropout is a regularization technique. During training, dropout randomly sets units in the hidden layer **h** to zero with probability p_{drop} (dropping different units each minibatch), and then multiplies **h** by a constant γ. We can write this as $h_{drop} = \gamma d * h$ where $d \in \{0,1\}^{D_h}$(D_h is the size of **h**) is a mask vector where each entry is 0 with probability p_{drop} and 1 with probability (1 – p_{drop}). γ is chosen such that the expected value of $\mathbf{h}_{drop}$ is **h**: $\mathrm{E}_{p_{drop}}[h_{drop}]_i = h_i$ for all $i \epsilon \{1, \dots, D_h\}$.

i) What must γ equal in terms of p_{drop}? Briefly justify your answer.

Ans) If scaling is done during training then $\gamma = 1/(1 - p_{drop})$, but if scaling is done at test time then $\gamma = 1/p_{drop}$. This is done so that the expected value of **h** remains the same. During training out of n nodes, $n*p_{drop}$ nodes will be set to zero, therefore only $n*(1-p_{drop})$ nodes will be active, therefore to keep the expected value the same we divide by $(1-p_{drop})$

ii) Why should we apply dropout during training but not during evaluation?

Ans) Dropout is used during training to prevent the model from overfitting to the training set. We do not use it during evaluation because we want our model to perform its best during evaluation and by using dropout we will be removing a few of the nodes of the network, thereby reducing its performance.

2a) Go through the sequence of transitions needed for parsing the sentence "I parsed this sentence correctly". The dependency tree for the sentence is shown below. At each step, give the configuration of the stack and buffer, as well as what transition was applied this step and what new dependency was added (if any). The first three steps are provided below as an example.

ROOT I parsed this sentence correctly

Ans)

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]	Ø		SHIFT
[ROOT, parsed]	Ø	parsed → correctly	RIGHT-ARC
[ROOT]	Ø	ROOT → parsed	RIGHT-ARC

b) A sentence containing n words will be passed in how many steps (in terms of n)? Briefly explain why.

Ans) A sentence containing n words will be passed in 2n steps. There will be n SHIFT operations; each word will be shifted onto the stack once. There will be n ARC operations because each word has only one parent (i.e. it modifies only one word)

f) In this question are four sentences with dependency parses obtained from a dependency parser. Each sentence has one error, and there is one example of each of the four types of errors- Prepositional phrase attachment error, Verb phrase attachment error, Modifier attachment error, Coordination attachment error. For each sentence state the type of error, the incorrect dependency, and the correct dependency.

Ans i) Error type: Verb phrase attachment error

Incorrect dependency: wedding -> fearing

Correct dependency: heading -> fearing

ii) Error type: Coordination attachment error

Incorrect dependency: makes -> rescue

Correct dependency: rush -> rescue

iii) Error type: Prepositional phrase attachment error

Incorrect dependency: named -> Midland

Correct dependency: guy -> Midland

iv) Error type: Modifier attachment error

Incorrect dependency: elements -> most

Correct dependency: crucial -> most