1 Machine Learning & Neural Networks

1. (4 points) Adam Optimizer
Recall the standard Stochastic Gradient Descent update rule:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J_{minibatch}(\theta) \tag{1}$$

where θ is a vector containing all of the model parameters, J is the loss function, $_\theta J_{minibatch}(\theta)$ is the gradient of the loss function with respect to the parameters on a minibatch of data, and α is the learning rate. Adam Optimization uses a more sophisticated update rule with two additional steps
i. (2 points) First, Adam uses a trick called momentum by keeping track of m, a rolling average of the gradients:

$$m \leftarrow \beta_1 m + (1 - \beta_1) \nabla_\theta J_{minibatch}(\theta) \tag{2}$$

$$\theta \leftarrow \theta - \alpha m \tag{3}$$

where $_1$ is a hyperparameter between 0 and 1 (often set to 0.9). Briey explain (you don't need to prove mathematically, just give an intuition) how using m stops the updates from varying as much and why this low variance may be helpful to learning, overall.

My answer:

m is a moving average of gradients which consists a lot of history gradients. As variance occurs randomly, it can be eliminated by calculating the average value. Lower variance can make the gradient decent path more directly towards the minimal value, because variance will cause the path deviating from the best one.

ii. (2 points) Adam also uses *adaptive learning rates* by keeping track of v, a rolling average of the magnitudes of the gradients.

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_\theta J_{minibatch}(\theta) \tag{4}$$

$$\mathbf{v} \leftarrow \beta_2 \mathbf{m} + (1 - \beta_2) \nabla_\theta J_{minibatch}(\theta) \odot \nabla_\theta J_{minibatch} \tag{5}$$

$$\theta \leftarrow \theta - \alpha \odot m / \sqrt{v} \tag{6}$$

Since Adam divides the update by $\sqrt{v}$, which of the model parameters will get larger updates? Why might this help with learning?

My answer:

Parameters with smaller gradients will get larger updates.
Because this enforces those parameters that may not change in several consecutive rounds to continuously update.

2. (4 points) Dropout is a regularization technique. During training, dropout randomly sets units in the hidden layer $\mathbf{h}$ to zero with probability p_{drop} (dropping dierent units each minibatch), and then multiplies $\mathbf{h}$ by a constant γ. We can write this as

$$h_{drop} = \gamma d \circ h$$

where $\mathbf{d} \in 0, 1^{D_h}$ (D_h is the size of $\mathbf{h}$ is a mask vector where each entry is 0 with probability p_{drop} and 1 with probability $(1p_{drop})$. γ is chosen such that the expected value of $\mathbf{h}_{drop} is \mathbf{h}$.
$i.(2points)Whatmust\gamma$ equal in terms of p_drop? Briey justify your answer.
ii. (2 points) Why should we apply dropout during training but not during evaluation?

My answer

i. γ should equal to $1/(1-p_{drop})$ According to the definition of drop out and its mask matrix, if no gamma applies, the expectation value of each element in $\mathbf{h}$ will be $1-p_{drop}$. Thus the value of γ is confirmed.
ii. In training, we want to stop some parameters from updating to avoid over-fitting. So we randomly mask some parameter's forward propagation.
In inference, we believe that each parameter is useful. If we just randomly abandon some of them, then the accuracy will decrease heavily.

2 Neural Transition-Based Dependency Parsing (42 points)

In this section, you'll be implementing a neural-network based dependency parser, with the goal of maximizing performance on the UAS (Unlabeled Attachemnt Score) metric.

Before you begin please install PyTorch 1.0.0 from https://pytorch.org/get-started/locally/ with the CUDA option set to None. Additionally run pip install tqdm to install the tqdm package – which produces progress bar visualizations throughout your training process.

A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between head words, and words which modify those heads. Your implementation will be a transition-based parser, which incrementally builds up a parse one step at a time. At every step it maintains a partial parse, which is represented as follows:

- A *stack* of words that are currently being processed.
- A *buffer* of words yet to be processed.
- A list of *dependencies* predicted by the parser.

Initially, the stack only contains ROOT, the dependencies list is empty, and the buffer contains all words of the sentence in order. At each step, the parser applies a transition to the partial parse until its buffer is empty and the stack size is 1. The following transitions can be applied:

- `SHIFT`: removes the first word from the buffer and pushes it onto the stack.
- `LEFT-ARC`: marks the second (second most recently added) item on the stack as a dependent of the first item and removes the second item from the stack.
- `RIGHT-ARC`: marks the first (most recently added) item on the stack as a dependent of the second item and removes the first item from the stack.

a (6 points)

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,sentence]	[correctly]	parsed → scentence	`RIGHT-ARC`
[ROOT,parsed,sentence,correctly]	[]		`SHIFT`
[ROOT,parsed]	[]	parsed → correctly	`RIGHT-ARC`
[ROOT]	[]	ROOT → parsed	`RIGHT-ARC`

(b) (2 points)

A sentence containing n words will be parsed in n steps (excluding `ROOT` here). Because at each parsing step one word is parsed, so the total number of parsing steps of a sentence equals to the number of words in this sentence.

(c) (6 points)

Please see homework code

(d) (6 points)

Please see homework code

(e) (6 points)

Please see homework code

(f) (12 points)

i.

- **Error type**: Verb Phrase Attachment Error
- **Incorrect dependency**: wedding → fearing
- **Correct dependency**: heading → wedding

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