Assignment 3

Marius Seritan

January 24, 2019

1. **Machine Learning & Neural Networks**

(a) **Adam Optimizer**

i. **Momentum**

The momentum is a rolling average of the gradient and is used to update the parameters. Taking the average reduces the variance and will help in the following two cases:

- when away from the minimum the average will dampen "sideways" noise and concentrate the movement towards the bottom.
- when oscillating around the minimum successive gradients will point in opposite directions and will cancel out reducing the variance

ii. **Magnitude**

The learning rate is divided by an estimate of the square root of the magnitude of the momentum.

When the momentum is high this will reduce the magnitude of the updates and it can help, for example, in the final layers that may otherwise experience exploding gradients.

When the momentum is small, under 1, the small updates will be amplified. This can help in the initial layers that may experience a vanishing gradient.

This simple formula also provides a way to do auto-tuning.

Note that since this is applied in vector form the different parameters will be increased or decreased based on their own individual behaviors.

(b) **Dropout**

i. **Value of constant γ.**

The constant γ in $h_{drop} = \gamma d \circ h$ should be set to $1 - p_{drop}$. The rationale for this is that when units are dropped the remaining units get trained with bigger weights in order to be able to generate the same output.

ii. **When to apply dropout**

One of the rationale of applying dropout is that it generates an exponential number of thinned networks to be trained. Since these networks are trained separately there is less correlation between their units.

At evaluation time these separate networks are used as an ensemble in order to provide a more robust generalization. The ensemble is simply implemented by not using the dropout during evaluation.

2. **Neural Transition-Based Dependency Parsing**

(a) **Transition table for sample sentence**

We have the sentence

`ROOT I parsed this sentence correctly`

The list of successive states for a *transitioned-based parser* is

Stack	Buffer	New Dependency	Transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial configu
[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]	[ ]		`SHIFT`
[ROOT, parsed]	[ ]	parsed → correctly	`RIGHT - ARC`
[ROOT]	[ ]	ROOT → parsed	`RIGHT - ARC`

(b) **Step numbers for a sentence of size** n

We want to estimate the number of steps required to parse a sentence with n words. For each word we will have to generate two transition: `SHIFT` and `* - ARC`. So the total numbers of steps will be $2 * n$, or if you want to count the initial state $2 * n + 1$.