CS224n Winter 2019 Homework 3: Dependency Parsing

SUNet ID: 05794739
Name: Luis Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) i We first note that for $\beta_1 > 0$, $\boldsymbol{m}$, as defined, will tend to vary less than the gradient itself, since it will be a weighed average of all previous gradients. For a concrete example, consider the scaler gradients such $\{-10, 10, -10, 15\}$, which we can see vary significantly. The corresponding $\boldsymbol{m}$ values, for a reasonable $\beta_1 = 0.9$, are $\{-1, -0.9 + 1 = 0.1, 0.09 - 0.1 = -0.91, -0.819 + 1.5 = 0.671\}$, which we can immediately see vary significantly less due to their averaging nature. Overall, this low variance in $\boldsymbol{m}$ prevents the changes in the paremeters $\boldsymbol{\theta}$ from jumping around too much, either due to a bad minibatch or simply due to the randomness involved in SGD. As such, this can improve learning by taking smaller, more measured steps in a direction more consistent with the true gradient.

ii We note that $\boldsymbol{v}$ is simply a weighed average of the square of the gradients thusfar, and as such, $\sqrt{\boldsymbol{v}}$ is a weighed average of the magnitude of the gradients. Since the update is being re-scaled by $\sqrt{\boldsymbol{v}}$, this means that parameters for which the gradients have had small magnitudes (ie, small $\sqrt{\boldsymbol{v}}$), will have their changes scaled to be larger. This can help in learning by making sure that even parameters that are receiving small gradient signal can nonetheless still change.

(b) i As presented, we must have the value:

$$\gamma = \frac{1}{1 - p_{\text{drop}}}$$

To see exactly why, we can compute $\mathbb{E}[\boldsymbol{h}_{\text{drop}}]$ as follows:

$$\begin{aligned}
\mathbb{E}_{p_{\text{drop}}}[\boldsymbol{h}_{\text{drop}}]_i &= \gamma \cdot 0 \cdot h_i p_{\text{drop}} + \gamma \cdot 1 \cdot h_i (1 - p_{\text{drop}}) \\
&= \frac{1}{1 - p_{\text{drop}}} \cdot (1 - p_{\text{drop}}) h_i \\
&= h_i
\end{aligned}$$

as desired.

ii We apply dropout during training but not during evaluation because during evaluation, we're taking an ensemble of all of the trained subnetworks (produced by using dropout during training), and with the schema above, by not applying dropout during evaulation, we're having the network compute the expected result of the ensemble which can help generalize and thereby perform better.

Problem 2

(a) We present the entire table in Table 1

Stack	Buffer	New Dependency	Transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial Config
[ROOT, I]	[parsed, this, sentence, correctly]		SHIFT
[ROOT, I, parsed]	[this, sentence, correctly]		SHIFT
[ROOT, parsed]	[this, sentence, correctly]	parsed $\rightarrow$ I	LEFT-ARC
[ROOT, parsed, this]	[sentence, correctly]		SHIFT
[ROOT, parsed, this, sentence]	[correctly]		SHIFT
[ROOT, parsed, sentence]	[correctly]	sentence $\rightarrow$ this	LEFT-ARC
[ROOT, parsed]	[correctly]	parsed $\rightarrow$ sentence	RIGHT-ARC
[ROOT, parsed, correctly]	[]		SHIFT
[ROOT, parsed]	[]	parsed$\rightarrow$ correctly	RIGHT-ARC
[ROOT]	[]	ROOT $\rightarrow$ parsed	RIGHT-ARC

Table 1: Table of transitions for "I parsed this sentence correctly."

(b) For a sentence containing n words, a total of $2n$ steps will be taken to parse the sentence. This is because n SHIFTs will occur (in some order), adding one word to the stack. Since the end condition consists of a stack with a single word, there must be n pops from the stack, and pops can only occur if an *-ARC operation occurs. As such, for a sentence of length n, there are $2n$ steps to parse.