Assignment3

April 19, 2020

1 Machine Learning & Neural Networks

1.1 (a) Adam Optimizer

(i) m is the exponentially decaying average of past gradients. When $J(\theta)$ is in a ravine for one dimension, its gradient mean is small. In this case $abs(m)$'s would be small, or it would on the opposite direction compared with $\nabla_\theta J(\theta)$. Then m would decrease the updating step for those directions whose mean is small or the new gradient's is on the opposite direction of the exponentially decaying mean. Then the updating step on such ravine would be smaller, then lead to smaller updating variance. The concept is similar like Inertia: a resistance of an object to any change in its velocity.

SGD will oscillate across the slopes of the ravine if we have a constant learning rate. But with m, for the directions that gradients' directions are unchanged, m will increase gradually. So the momentum m helps the learning progress by preferring larger steps on more stable directions.

(ii) Parameters with smaller accumulated updated gradients will get larger updates. It might help with learning because for those parameters that have large v, it means the neural network already gained enough information within recent learning steps, so we should decrease its updating step size. But for those parameters that have small v, its updating frequency is low. So we should increase their learning rate. One issue of such scaling is that, the $|v|$ is a monotonic increasing function. So learning rate would $\rightarrow 0$.

1.2 (b) Dropout

(i) $h_{drop} = \gamma d \odot h$ where $d \in \{0,1\}^{D_h}$ is a mask vector where each entry is 0 with probability p_{drop} and 1 with probability $1 - p_{drop}$. To ensure that $\mathbb{E}_{p_{drop}}.[h_{drop}]_i = h_i$, $\gamma = \frac{1}{1-p_{drop}}$. Because $\mathbb{E}_{p_{drop}}[h_{drop}]_i = \gamma[0(p_{drop}) + h_i(1 - p_{drop})] = h_i$

(ii) With limited training data, many parameters learned from the neural network will be the result of sampling noise. They will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting issue. So if we apply dropout during evaluation, the noise in training dataset will still be learned by the parameters, and dropout hidden unit at evaluation time will cause difference between expected output during training and evaluating output, thus causing bias.

2 Neural Transition-Based Dependency Parsing

(a)

Stack	Buffer	New dependency	Transition
[ROOT, parsed, this]	[sentence, correctly]	parsed→ I	SHIFT
ROOT, parse	[sentence, correctly]	parsed→ I, parsed→ this	RIGHT-ARC
ROOT, parse, sentence	[correctly]	parsed→ I, parsed→ this	SHIFT
ROOT, parse	[correctly]	parsed→ I, parsed→ this, parsed→ sentence	RIGHT-ARC
ROOT, parse, correctly	[]	parsed→ I, parsed→ this, parsed→ sentence	SHIFT
ROOT, correctly	[]	parsed→ I, parsed→ this, parsed→ sentence, correctly → parse	RIGHT-ARC
ROOT	[]	parsed→ I, parsed→ this, parsed→ sentence, correctly → parse, ROOT → correctly	RIGHT-ARC

(b) n words will lead to n arcs in the final dependency set A. And each word need a $SHIFT$ action for being moved from buffer to stack, and an ARC action for being moved from stack to the final set. So in total $2n$ steps.