Anthony Galczak
January 28, 2019

Question 1

a)(i) In normal SGD, our update function would not involve m, beta1, or (1 - beta1). m and beta1 introduce some level of “smoothing” to our updates. We base part of our model updates upon the gradient we just calculated and some of it on our rolling momentum, m.

There is some intuition behind why this could help for learning. “When you are going fast, you want to keep going fast; when going slow, you want to keep going slow.” In other words, when you first start training and your gradients become large, we want to keep moving fast. If there is a random gradient that is small at epoch 8, but the last 7 epochs had huge gradients, it is still likely we want to move quickly in the space.

a)(ii)

When the gradient is small, the squared term in v will be even smaller. Since we are dividing our weight update by sqrt(v), when the gradient is small, v will be small and the weight update will in turn be increased. This could help speed up convergence when we have tiny gradients. Doing 10x more epoch’s to get from 98% accuracy to 98.4% accuracy isn’t ideal.

b)(i) Since we have turned off p_drop amount of units, we need to scale h_drop with a corresponding offset. To scale h_drop to h, we need to divide h_drop by 1 - p_drop. In other words, gamma = 1 / (1 - p_drop).

b)(ii)

Two reasons to not apply dropout during evaluation.

First, we want to avoid stochasticity at prediction time. We can use a streaming image CV task as an illustrative example. If we drop 50% of the neurons at each frame, we might continuously change our prediction of a streaming image(cat -> cat -> dog -> cat -> dog, etc.). We want the predictions to be consistent for similar images, not stochastic.

Second, we are using dropout for regularization and preventing feature co-adaptation at training time. At test time, we want all of our features to work together. All of the neurons learned something at some point during the training and we want to use all of the neurons in order to make our predictions.

Question 2.

a)

Goes until buffer is empty and stack size is 1

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]	[]		SHIFT
[ROOT, parsed]	[]	parsed->correctly	RIGHT-ARC
[ROOT]	[]	ROOT->parsed	RIGHT-ARC

b) A sentence containing n words will have at most n shifts at most n arcs. Thus, we can parse n words in 2n time, i.e. O(n).

e) Ran for 20 epochs, best test UAS was 88.93, best dev UAS was 88.79

f)
(i) Verb phrase attachment error
Incorrect: wedding -> fearing
Correct: disembarked -> fearing
(ii) Coordination attachment error
Incorrect: makes -> rescue
Correct: rush -> rescue
(iii) Prepositional phrase attachment error
Incorrect: named -> Midland
Correct: guy -> Midland
(iv) Modifier attachment error
Incorrect: elements -> most
Correct: crucial -> most