CS 224n Assignment #3: Dependency Parsing

Jai Gupta

January 29, 2019

Q (1.a.i): Adam Optimizer - Explanation for β_1
Solution:
Simple Gradient descent uses the formula:

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

After including the parameter β_1, we have:

$$\begin{aligned} m &\leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta J_{minibatch}(\theta) \\ \theta &\leftarrow \theta - \alpha m \end{aligned}$$

A basic difference between the two is that instead of directly using the gradient from current state, we are using a linear combination of the cached rolling gradient and the new gradient. Since the sum of the weights is 1, the returned value is always between the two combined values thereby dampening any sudden changes in the value of the gradient.
If gradient descent is compared with a ball rolling down the slope, we can see that in each step, we do not completely forget the previous speed. Instead the new speed is a linear combination of the previous speed and the speed due to current gradient. This can be referred to as the momentum of the ball being preserved.
Generally during gradient descent, we may overshoot the minima and hence keep oscillating between the two sides of the minima if the two sides are pretty steep. But after we add momentum, oscillations will be reduced since the sudden change in the velocity is dampened due to the momentum effect.
Additionally, since oscillating overshoots are reduced, the learning rate (alpha) can also be kept a bit higher. The gradient descent speeds up as it accumulates velocity throughout the descent. This leads to faster convergence of the model.

Q (1.a.ii): Adam Optimizer - Explanation for β_2
Solution:
The updated set of equations after adding the parameter β_2 are:

$$\begin{aligned} m &\leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta J_{minibatch}(\theta) \\ v &\leftarrow \beta_2 v + (1-\beta_2)(\nabla_\theta J_{minibatch}(\theta) \odot \nabla_\theta J_{minibatch}(\theta)) \\ \theta &\leftarrow \theta - \alpha \odot m/\sqrt{v}xz \end{aligned}$$

v has a pretty similar update structure as m except that it uses square of the gradient. This ensures that if there are oscillating steps, they don't cancel out. Since v is the weighted sum (with total weight of 1) of square of the past gradients, its square root will represent the rolling absolute gradient.
Since the step sizes are proportional to the gradient, v will also be proportional to the absolute value of the step sizes being taken for each parameter.
By changing our parameter update to be inversely proportional to v, we are essentially decreasing the step size for parameters that have been taking very large steps (have steep slope). Additionally, it also increases the step size for parameters that have been taking small steps (have flat slope).

Q (1.b.i): Dropout - What must γ be equal to in terms of p_{drop}? Briefly justify your answer.
Solution:

$$\mathbf{h}_{drop} = \gamma \mathbf{d} \odot \mathbf{h}$$

Given that **d** is a vector, where each element is 0 with probability p_{drop} and 1 with probability $1 - p_{drop}$, expected value of i_{th} element of **d** will be:

$$\begin{aligned} E_{p_{drop}}[d_i] &= 0 * p_{drop} + 1 * (1 - p_{drop}) \\ &= 1 - p_{drop} \end{aligned}$$

$$\begin{aligned} E_{p_{drop}}[\gamma d \odot h]_i &= \gamma(E_{p_{drop}}[d]_i * E_{p_{drop}}[h]_i) \\ &= \gamma(1 - p_{drop}) * h_i \end{aligned}$$

Given that we want the following to hold:

$$E_{p_{drop}}[h_{drop}]_i = h_i$$

Therefore, we can say that:

$$\begin{aligned} E_{p_{drop}}[h_{drop}]_i &= h_i \\ E_{p_{drop}}[\gamma d \odot h]_i &= h_i \\ \gamma(1 - p_{drop}) h_i &= h_i \\ \gamma(1 - p_{drop}) &= 1 \\ \gamma &= \frac{1}{1 - p_{drop}} \end{aligned}$$

This is pretty intuitive to think of. Since there is just $1 - p_{drop}$ probability that a given element will not be dropped, the expected value of h_i is multiplied by that probability. Hence, we need to scale the numbers with the inverse of the same probability to keep the expected value same.

Q (1.b.ii): Dropout - Why should we apply dropout during training but not during evaluation?
Solution:
Drop is a regularization technique that intentionally produces wrong values with an aim to reduce over-fitting in the model. By randomly dropping these weights, the trained network generalizes better and is more resilient to bad features since it tries to depend on multiple features for its inference.

At the time of inference:

- We don't want our network to add such deliberate erroneous values.

- We would like our model to make use of as much information as it can. Dropping some of the internal features randomly does not help in any way.
- Since we are not performing back-prop, error received in the output will not be used in any way to improve the model.

Q (2.a): Parsing "I parsed this sentence correctly"
Solution:

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

Q (2.b): A sentence containing n words will be parsed in how many steps (in terms of n)? Briefly explain why.
Solution:
We know that:

- We need one SHIFT step per token to move it from buffer to stack. This means we need a total of n SHIFT steps.
- Each reduce step reduces the token count by 1. We have $n+1$ token and we need to reduce it to 1. Hence, n REDUCE steps will be needed. Another way to think of the same thing is that a tree with $n+1$ nodes needs n edges, and each reduce step adds one edge in the dependency tree. So a total of n steps will be needed.

Hence, total number of steps needed = $2n$ (n SHIFT and n REDUCE). Note that this does not count the "*Initial Configuration*" step in the above table.

Q (2.f): Incorrect Dependencies
Solution:
i) **Error Type:** Verb Phrase Attachment Error
Incorrect Dependency: wedding → fearing
Correct Dependency: heading → fearing

ii) **Error Type:** Coordination Attachment Error
Incorrect Dependency: rescue → and
Correct Dependency: rush → and

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