Author: Benjamin Smidt

Created: October 17th, 2022
Last Updated: October 17th, 2022

CS 224N A3: Dependency Parsing

Note to the reader. This is my work for assignment two of Stanford's course CS 224N: Natural Language Processing with Deep Learning. You can find the lecture Winter 2021 lectures series on YouTube here. This document is meant to be used as a reference, explanation, and resource for the assignment, not necessarily a comprehensive overview of Word Vectors. If there's a typo or a correction needs to be made, feel free to email me at benjamin.smidt@utexas.edu so I can fix it. Thank you! I hope you find this document helpful :).

Contents

1 Machine Learning and Neural Networks

1.1 Adam Optimizer

In our traditional Stochastic Gradient Descent, the update rule is

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

The Adam optimizer modifies SGD such in an effort to improve convergence. The first addition is the use of *momentum*. Adam keeps a rolling average of the gradients instead of using only the current gradient.

$$m \leftarrow \beta_1 m + (1 - \beta_1)\nabla_\theta J_{\text{minibatch}}(\theta) \tag{1}$$

$$\theta \leftarrow \theta - \alpha m$$

(i) *Briefly explain in 2-4 sentences how using m stops the updates from varying as much and why this low variance may be helpful to learning, overall.*

m is a weighted average between all the previous updates, embedded in m, and the current update $\nabla_\theta J_{\text{minibatch}}(\theta)$ (our β_1 parameter specifies the weight to give each term, $\beta_1 = 0.9$ is common). By keeping this weighted average, the update naturally gives higher weight to updating in directions that have been consistent while updates along dimensions that keep switching between positive and negative are given close to no weight. This improves optimization since our updates will minimize steps in dimensions that aren't getting us anywhere meaningful (flipping between positive and negative, can't decide which direction to go in) and maximize steps in dimensions that are getting us somewhere meaningful (nearly all updates have had this direction).

A second addition to Adam is *adaptive learning rates*, which keeps track of v, a rolling average of the magnitude of the gradients.

$$m \leftarrow \beta_1 m + (1 - \beta_1)\nabla_\theta J_{\text{minibatch}}(\theta)$$

$$v \leftarrow \beta_2 v + (1 - \beta_2)(\nabla_\theta J_{\text{minibatch}}(\theta) \odot \nabla_\theta J_{\text{minibatch}}(\theta))$$

$$\theta \leftarrow \theta - \alpha m / \sqrt{v}$$

odot is elementwise multiplication and / is elementwise division. β_2 is our second hyperparameter (often set to 0.99).

(ii) *Since Adam divides the update by $\sqrt{v}$, which of the model parameter will get larger updates? Why might this help with learning?*

If v is quite small, then dividing by $\sqrt{v}$ will make the term $\alpha m/\sqrt{v}$ large. This improve learning because often times we need a large learning rate if our gradient update is naturally small.

The vice versa is also true. When the gradient is very large (v is very large), then we don't need a very large learning rate and often a small learning rate will be better. In this case, by dividing by $\sqrt{v}$, we actually reduce the magnitude of the update to counterbalance the already large gradient.

1.2 Dropout

Dropout is a form of regularization wherein we "drop" random connections within the hidden layers of our network during each update (different connections are dropped for each update). We do this mathematically with the following

$$h_{\text{drop}} = \gamma d \odot h$$

where h is a hidden layer, $d \in \{0, 1\}^{D_h}$ (D_h is the size of h) is a mask vector with each entry being 0 (with probability p_{drop}) or 1 (with probability $1 - p_{\text{drop}}$), and γ is a constant chosen such that the expected value of h_{drop} is h

$$\mathbb{E}_{p_{\text{drop}}}[h_{\text{drop}}]_i = h_i \quad \forall\, i \in \{1, \ldots, D_h\}$$

(i) *What must γ equal in terms of p_{drop}? Briefly justify your answer or show your math derivation using the equations given above*

$$\mathbb{E}_{p_{\text{drop}}}[h_{\text{drop}}] = h$$

$$\mathbb{E}_{p_{\text{drop}}}[\gamma d \odot h] = h$$

$$\gamma\, \mathbb{E}_{p_{\text{drop}}}[d \odot h] = h$$

$$\gamma[h p_{\text{drop}} + (1 - p_{\text{drop}})0] = h$$

$$\gamma\, h p_{\text{drop}} = h$$

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

(ii) *Why should dropout be applied during training? Why should dropout NOT be applied during evaluation?*

Dropout should be applied during training so the network learns different pathways that lead to the same prediction. By closing different connections randomly, the network is forced to produce multiple paths in which data can flow to achieve the correct prediction, theoretically making it more robust.

We wouldn't want to apply dropout during evaluation however because our results would be non-deterministic. Due to the randomness of the dopout connections, it's possible (and may even be likely depending on the network) that evaluating the same input twice yields different predictions. Obviously this is an undesirable trait to have in a machine learning model so we don't apply dropout during evaluation.

2 Neural Transition-Based Dependency Parsing