CS 224n Assignment #3: Dependency Parsing

1. Machine Learning & Neural Networks

(a) Adam Optimizer
Recall the standard Stochatic Gradient Descent update rule:

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

where $\boldsymbol{\theta}$ is a vector containing all of the model parameters, J is the loss function, $\nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta})$ is the gradient of the loss function with respect to the parameters on a minibatch of data, and α is the learning rate. Adam Optimization uses a more sophisticated update rule with two additional steps.

i. First, Adam uses a trick called $momentum$ by keeping track of $\boldsymbol{m}$, a rolling average of the gradients:

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

where β_1 is a hyperparameter between 0 and 1 (often set to 0.9). Briefly explain how using $\boldsymbol{m}$ stops the updates from varing as much and why this low variance may be helpful to learning, overall.

Solution:

The actual update difference of each parameter depends on the weighted average of the recent gradient. When the gradient direction of a parameter in the most recent period of time is inconsistent, its real parameter update change becomes smaller (generally at the end of the iteration to improve the stability); On the other hands, when the gradient direction in the recent period of time is consistent, its real parameter update variance becomes larger, which accelerates learning (generally at the beginning to quick reach the optimum).

ii. Adam also uses $adaptive\ learning\ rates$ by keeping track of **v**, a rolling average of the magnitudes of the gradients:

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta})$$
$$\mathbf{v} \leftarrow \beta_2 \mathbf{v} + (1 - \beta_2) \left(\nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta})\right)$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \odot \mathbf{m} / \sqrt{\mathbf{v}}$$

where $\odot$ and $/$ denote elementwise multiplication and division, and β_2 is a hyperparameter between 0 and 1 (often set to 0.99). Since Adam divides the update by $\sqrt{\mathbf{v}}$, which of the model parameters will get larger updates? Why might this help with learning?

Solution:
Because convergence rate is different based on different each parameter scale, we can set learning rate based on the convergence situation of each parameter. When the rolling average of one partial derivative is small, the learning rate of this parameter will become larger to speed up learning process.

(b) Dropout is a regularization technique. During training, dropout randomly sets units in the hidden layer $\mathbf{h}$ to zero with probability p_{drop} (dropping different units each minibatch), and then multiplies $\mathbf{h}$ by a constant τ. We can write this as

$$\mathbf{h}_{\text{drop}} = \gamma \mathbf{d} \circ \mathbf{h}$$

where $\mathbf{d} \in \{0, 1\}^{D_h}$ (D_h is the size of $\mathbf{h}$) is a mask vector where each entry is 0 with probability p_{drop} and 1 with probability (1 - p_{drop}). τ is chosen such that the expected value of $\mathbf{h}_{drop}$ is $\mathbf{h}$:

$$\mathbb{E}_{p_{\text{drop}}}\left[\mathbf{h}_{drop}\right]_i = h_i$$

for all $i \in \{1, ..., D_h\}$
i. What must tau equal in terms of p_{drop}?
ii. Why should we apply dropout during training but not during evaluation?
Solution:
i. $\tau = \frac{1}{1-p_{drop}}$

ii. From the view of ensemble learning, dropping some neural nets means sample a sub network from the original network. Every training iteration equals to train a different sub network, which share the parameters of the original network. The final network could be a total model integrated with several sub networks. If we still dropout during evaluation, the model will lose the original purpose to have generalization ability.