Assignment 3 [written]

(a) (4 points) Adam Optimizer
Recall the standard Stochastic 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[1] uses a more sophisticated update rule with two additional steps.[2]

i. (2 points) First, Adam uses a trick called *momentum* by keeping track of $\mathbf{m}$, a rolling average of the gradients:

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

where β_1 is a hyperparameter between 0 and 1 (often set to 0.9). Briefly explain (you don't need to prove mathematically, just give an intuition) how using $\mathbf{m}$ stops the updates from varying as much and why this low variance may be helpful to learning, overall.

This trick uses the exponential weighted average, the m calculated by this formula is approximately equal to $\frac{1}{1-\beta_1} gradient$. Simple derivation:

$$m_n = \sum_{i=0}^{n} 0.1 \times 0.9^i \times gradient_{n-i}$$

The newer the gradient, the greater the proportion. We now that the gradient obtained each time is not always in the direction of the optimal value, however if it is decomposed, a large part of the gradient is towards the optimal value. If we directly use the gradient obtained each step, the optimization process will be very tortuous and fluctuate all the time. If several gradients obtained in the past are averaged, the components in other directions will be offset, and the components towards the optimal value will be larger and the update speed will be faster.

ii. (2 points) Adam also uses *adaptive learning rates* by keeping track of $\mathbf{v}$, a rolling average of the magnitudes of the gradients:

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

where $\odot$ and $/$ denote elementwise multiplication and division (so $\mathbf{z} \odot \mathbf{z}$ is elementwise squaring) 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?

According to the formula, we can know that the final update is divided by $\sqrt{v}$. Therefore, the smaller the $\sqrt{v}$, the bigger the update step. v is obtained by the exponential weighted average of the square of gradient, so the parameter with smaller gradient will get smaller v value, which will get larger update. On the contrary, the parameters with larger gradient will get smaller update.

(b) (4 points) Dropout[3] 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 γ. 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 $(1 - p_{\text{drop}})$. γ is chosen such that the expected value of $\mathbf{h}_{\text{drop}}$ is $\mathbf{h}$:

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

for all $i \in \{1, \ldots, D_h\}$.

i. (2 points) What must γ equal in terms of p_{drop}? Briefly justify your answer.

ii. (2 points) Why should we apply dropout during training but not during evaluation?

1. $h_{drop} = \gamma(1 - p_{drop}) \circ h$, so $\gamma = \frac{1}{1-p_{drop}}$

2. Dropout is a regularization technique. We use dropout in training to avoid over fitting and make the model more generalized.