6130

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

(i) $\theta \leftarrow \theta - \alpha m$

Q: How using m stops the updates from varying as much and why this low variance may be helpful to learning, overall.

Sol: If we don't use momentum, each step θ changes by $1 * \nabla_\theta J_{minibatch}(\theta)$; but when we use momentum, only $(1-\beta_1) * \nabla_\theta J_{minibatch}(\theta)$ contributed to the change, another part β_1 comes from last gradient/change.

To see why this helps,

Considering a small rock rolling down a hill, it's very sensitive to the ups and down (low momentum) and is likely to be stuck in a local optimum;

then considering a big rock rolling down the same hill, it's not that sensitive to the ups and downs (high momentum) and are more likely to reach the global optimum.

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

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

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

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

Sol: $\theta_i \leftarrow \theta_i - \alpha \frac{m_i}{\sqrt{v_i}}$ $\quad v_i \leftarrow \beta_2 v_i + (1-\beta_2)(\nabla_{\theta_i} J(\theta))^2$

clearly θ_i with smaller v_i, i.e. smaller rolling average of gradient magnitudes, i.e. θ_i that hasn't changed much will get larger updates, and will get faster to close the optimum, where θ_j that already changed/moved a lot would be more careful near the optimum to avoid divergence.

(b) $h_{drop} = \gamma d \odot h \quad d \in \{0,1\}^{D_h}$

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

(i) What must γ equal in terms of P_{drop}?

sol: $[h_{drop}]_i = \gamma d_i \cdot h_i \quad d_i \sim Ber(1-P_{drop})$

$\Rightarrow E_{P_{drop}}[h_{drop}]_i = E[\gamma d_i h_i] = \gamma h_i E[d_i]$

$= \gamma h_i (1-P_{drop}) = h_i$

$\Rightarrow \gamma = \dfrac{1}{1-P_{drop}} \quad (0 < P_{drop} < 1)$

(ii) We apply dropout during training because we do not want model to solely be confident in / dependent on a few neurons and be overfitting, but after applying dropout and forcing our model to learn from all neurons keeping in mind some might fail, we already have an "ensembled" model, and we don't want to randomly drop some neurons during evaluation which will lead to inconsistent results in same input.