Optimizers for Deep Neural Networks

Yang You

Presidential Young Professor at NUS Computer Science

ai.comp.nus.edu.sg

SGD

- Typical Assumption: Loss Function is L-Smooth

$$\|\nabla L(\theta_2;\xi) - \nabla L(\theta_1;\xi)\|_2 \le L\|\theta_2 - \theta_1\|_2 \ \ \forall\ \theta_1, \theta_2, \xi$$

- Empirical risk minimization: draw samples ξ from a distribution $\mathbb{P}$

$$\nabla L(\theta_t) = \nabla \mathbb{E}[L(\theta_t;\xi)]$$

- Updating rule for SGD

$$\theta_t = \theta_{t-1} - \eta_t * \nabla L(\theta_{t-1})$$

- η_t is a number (learning rate)
- θ_t is a vector (weights or parameters)
- only uses a batch of data to compute gradients each iteration

AdaGrad (Adaptive Gradient) Duchi et al. 2011

$$g_t = \nabla L(\theta_{t-1})$$

$$v_t = v_{t-1} + g_t \odot g_t$$

$$\theta_t = \theta_{t-1} - \eta_t * \frac{g_t}{\sqrt{v_t + \epsilon}}$$

- θ_t, g_t, v_t is a vector; η_t is a number (default value is 0.01)
- It eliminates the need to manually tune the learning rate
- Sparse gradient has a larger update (element-wise updating) and moves faster
- Accumulation of squared gradients in the denominator (weakness):
  - Since every added term is positive, the accumulated sum keeps growing during training
  - This in turn causes the update to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge

RMSprop (Root Mean Square Porp), Hinton et al. 2014

$$g_t = \nabla L(\theta_{t-1})$$

$$v_t = \gamma v_{t-1} + (1 - \gamma) g_t \odot g_t$$

$$\theta_t = \theta_{t-1} - \eta_t * \frac{g_t}{\sqrt{v_t + \epsilon}}$$

- θ_t, g_t, v_t is a vector; η_t, γ is a number
  - Hinton suggests γ to be set to 0.9
- RMSprop is an extension of Adagrad that deals with its aggressively decaying learning rate
- Sparse gradient has a larger learning rate and moves faster

Adam (adaptive moment estimation), Kingma & Ba 2015

$$g_t = \nabla L(\theta_{t-1})$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t$$

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

$$\theta_t = \theta_{t-1} - \eta_t * \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- θ_t, g_t, m_t, v_t is a vector; η_t, β_1, β_2 is a number
    - The authors suggest $\beta_1 = 0.9$, $\beta_2 = 0.999$
- It works well for NLP and reinforcement learning
- It fails to achieve the target accuracy for ImageNet