Summary of Optimization Algorithms in Deep Learning

Abdul Samad Khan

Overview

This note summarizes key optimization methods relevant to training deep learning models, including those applicable to volatility forecasting tasks using Transformer-GNN architectures.

1. Stochastic Gradient Descent (SGD)

SGD updates model parameters using noisy gradients computed on mini-batches:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

While simple, SGD is sensitive to learning rate and often slow to converge.

2. SGD with Momentum

Momentum improves SGD by adding a velocity term:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned}$$

Here, γ (typically 0.9) controls the contribution of past gradients, smoothing updates and accelerating convergence.

3. Nesterov Accelerated Gradient (NAG)

NAG adds a *look-ahead* step before computing the gradient:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta_t - \gamma v_{t-1}) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned}$$

This anticipatory behavior improves stability and convergence, especially in non-convex landscapes.

4. Adagrad

Adagrad adapts the learning rate per parameter based on the sum of squared past gradients:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta L(\theta_t)$$

Good for sparse data, but learning rates shrink too much over time.

5. RMSprop

RMSprop fixes Adagrad's decay problem using an exponential moving average:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Performs well on non-stationary problems; was designed for RNNs.

6. Adam

Adam combines RMSprop and Momentum:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Default optimizer for most deep learning tasks due to robustness and speed.

7. BFGS (Quasi-Newton)

BFGS approximates second-order information without computing the Hessian:

$$\theta_{t+1} = \theta_t - H_t^{-1} \nabla_\theta L(\theta_t)$$

Too computationally expensive for deep networks but useful in convex optimization problems (e.g., classical ML or econometric models).

Recommendation

For training large-scale models like Transformer-GNN hybrids in financial time-series:

- Use **Adam or AdamW** as default.
- Use **SGD + Nesterov Momentum** for better generalization on large datasets.
- Avoid Adagrad/BFGS unless solving niche or small-scale convex problems.