CS 182 Lecture 4: Optimization — $I(\theta)$

$\theta^* \leftarrow \arg\min_\theta \left(-\sum_i \log p_\theta(y_i|x_i)\right)$ $\quad v_i = -\frac{dI(\theta)}{d\theta_i}$ (gradient vector)

Gradient descent – picks steepest direction, not "best" direction

"Nice" loss surface – all roads lead to Rome.

Not for log. reg. – guaranteed to be convex. Strengthens gradient descent.

Three features of loss landscape: local optimum, the plateau, saddle — all are harmful for GD.

Local optima becomes less of an issue as # of NN params increase...

We need learning rates large enough to get past plateaus.

Hessian: "2-D gradients"

Most critical points in neural net loss landscapes are saddle points
↳ unlikely to have same sign for all diagonal entries in Hessian matrix

$\nabla_\theta \mathcal{L}(\theta_i)$ → gradient
$\nabla^2_\theta \mathcal{L}(\theta_i)$ → Hessian

Newton's Method: able to find better direction for gradient descent

$\theta^* \leftarrow \theta_0 - (\nabla^2_\theta \mathcal{L}(\theta_0))^{-1} \nabla_\theta \mathcal{L}(\theta_0)$ ← Hessian inverse

Great in theory, computationally expensive!

Momentum: Try to tackle oscillation / "slowness" in GD

an aside: Nesterov's accelerated gradient

$\theta_{k+1} = \theta_k - \alpha g_k, \quad g_k = \nabla_\theta \mathcal{L}(\theta_k) + \mu g_{k-1}$ ← "blend in" previous direction

RMSProp: normalize magnitude of gradient along each dimension ← "root-mean-squared"

$s_k \leftarrow \beta s_{k-1} + (1-\beta)(\nabla_\theta \mathcal{L}(\theta_k))^2$ ← "forgetting" property. Better for DL, non-convex probs.

$\theta_{k+1} = \theta_k - \alpha \frac{\nabla_\theta \mathcal{L}(\theta_k)}{\sqrt{s_k}}$ ← this is rough est. of magnitude.

Tuning SGD:

B (batch size) larger better
α (learning rate) decay over time
μ (momentum) 0.99, 0.9 are good
β_1, β_2 (keep defaults)
↑ can tune on validation loss.

AdaGrad: estimate per-dimension cumulative magnitude.

$s_k \leftarrow s_{k-1} + (\nabla_\theta \mathcal{L}(\theta_k))^2$. Good for convex problems.

Adam: combines momentum and RMSProp.

helps "blow up" values for small k, speed it up

$m_k = (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_k) + \beta_1 m_{k-1}$ "first moment estimate" (momentum-like) $\quad \hat{m}_k = \frac{m_k}{1-\beta_1^k}$

$v_k = (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_k))^2 + \beta_2 v_{k-1}$ "second moment estimate". $\quad \hat{v}_k = \frac{v_k}{1-\beta_2^k}$

$\theta_{k+1} = \theta_k - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$ Typically: $\epsilon = 10^{-8}, \alpha = 0.001$, $\beta_1 = 0.9, \beta_2 = 0.999$

Stochastic GD:

1. Sample $B \subset \mathcal{D}$.
2. Estimate $g_k \leftarrow -\nabla_\theta \frac{1}{B} \sum_{i=1}^{B} \log p(y_i|x_i,\theta) \approx \nabla_\theta \mathcal{L}(\theta)$
3. $\theta_{k+1} \leftarrow \theta_k - \alpha g_k$ (or momentum, Adam, etc.)

Each iteration samples diff. B.

Epoch: complete "look-thru" dataset.

Preferably decay learning rate over time w/ SGD(+momentum).

Start with one shuffle for all batches. ↓ saves RAM.