Adaptive Learning Rates

Nan Ye

School of Mathematics and Physics
The University of Queensland

Local Geometry and Adaptivity

- Local geometry of the error surface is important for choosing good learning rate
  - Flat error surface $\Rightarrow$ a large learning rate is desirable.
  - Steep error surface $\Rightarrow$ a small learning rate is essential.
  - Near a minimizer $\Rightarrow$ small learning rate to avoid oscillatory behavior.
- Fixed learning rates (such as constant learning rates, or $\frac{1}{t}$) are not able to adapt to the local geometry of the error surface.
- We want to exploit local geometry to adaptively set the learning rates.

Per-dimension adaptivity

- We have seen in earlier lectures that in a deep net, the gradients at different layers often differ significantly.
- Good initialization and input transformation help, but do not solve the problem completely.
- If we can set *per-dimension* adaptive learning rates, it can help us to speed up learning for layers with small gradients, and avoid overshooting for layers with large gradients.

Learning Rate Annealing

- Reduce learning rate when the error plateaus.
  - e.g. reduce from 0.1 to 0.01.
  - Often use the error on a validation set
- Helpful when the algorithm is oscillating around a minimizer due to large learning rate.

Newton's Method

- Newton's method provides a way to better take local geometry into account than vanilla gradient descent.
- Newton's method updates current iterate $\mathbf{w}$ to

$$\mathbf{w}' = \mathbf{w} - H^{-1} g,$$

  where $H = \nabla^2 f(\mathbf{w})$ and g are the Hessian and gradient of f at $\mathbf{w}$.
- This is derived by choosing d to minimize the second order Taylor series approximation

$$f(\mathbf{w} + d) \approx f(\mathbf{w}) + d^\top \nabla f(\mathbf{w}) + \frac{1}{2} d^\top H d.$$

- Newton's method is computationally expensive.

Diagonal approximation

- The diagonal $(h_1, \ldots, h_m)$ of H can be efficiently computed.
- We approximate Newton's method

$$w_i = w_i - \frac{1}{h_i + \epsilon} g_i,$$

where $\epsilon > 0$ is a constant.

AdaGrad

- For each weight w, keep the sum of squared derivative.
- The sum $v^{(t)}$ at iteration t is recursively computed as

$$v^{(t)} = v^{(t-1)} + (g^{(t)})^2,$$

  where $g^{(t)}$ is w's derivative at iteration t.
- At iteration t, current weight $w^{(t)}$ is updated to

$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{v^{(t)} + \epsilon}} g^{(t)},$$

  where $\eta > 0$ is a global learning rate shared by all weights.

Duchi, Hazan, and Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, 2010

- AdaGrad evens out progress for all weights.
  - Weights with small gradients move faster.
  - Weights with large gradients move slower.
- However, the gradients accumulate, and after a while no progress can be made.
- In addition, the method can be sensitive to the initial values.
  - Large initial gradients can make learning too slow.

RMSProp

- RMSProp keeps a moving average of the squared derivative, instead of the sum.
- The moving average $v^{(t)}$ at iteration t is recursively computed as

$$v^{(t)} = \rho v^{(t-1)} + (1-\rho)(g^{(t)})^2.$$

- At iteration t, current weight $w^{(t)}$ is updated to

$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{v^{(t)} + \epsilon}} g^{(t)},$$

  where $\eta > 0$ is a global learning rate shared by all weights.
- ρ is typically close to 1.

- RMSProp evens out progress for all weights as AdaGrad.
- Additionally, it is less sensitive to the initial values, and can keep on making progress after many iterations.

AdaDelta

- An interesting observation
  - GD: unit of change $\propto$ unit of $g \propto \frac{\partial f}{\partial w} \propto \frac{1}{\text{unit of } w}$.
  - Newton's method: unit of change $\propto$ unit of $H^{-1}g \propto \frac{\partial f}{\partial w} / \frac{\partial^2 f}{\partial w^2}$ $\propto$ unit of w.
- AdaDelta is an improvement of RMSProp by adding a scaling factor to each dimension so that the updates have the right units.

- AdaDelta computes the moving average $v^{(t)}$ of the squared derivative as in RMSProp.
- It additionally computes a moving average for the squared updates

$$s^{(t+1)} = \rho s^{(t)} + (1-\rho)(w^{(t+1)} - w^{(t)})^2.$$

- At iteration t, current weight $w^{(t)}$ is updated to

$$w^{(t+1)} = w^{(t)} - \frac{\sqrt{s^{(t)} + \epsilon}}{\sqrt{v^{(t)} + \epsilon}} g^{(t)}.$$

- Note that unit of the update is the same as that of w.

Zeiler, ADADELTA: an adaptive learning rate method, 2012

- AdaDelta overcomes the sensitivity to the hyperparameter selection in methods like RMSProp.
- AdaDelta appears to be robust to noisy gradient information, and is insensitive to the choice of the hyperparameter ϵ.

Adam

- Adam combines RMSProp with standard momentum.
- For each weight w, it computes (biased) 1st moment $m^{(t)}$ and 2nd moment $v^{(t)}$ at iteration t as follows

$$m^{(t)} = \rho_1 m^{(t-1)} + (1 - \rho_1) g^{(t)},$$
$$v^{(t)} = \rho_2 m^{(t-1)} + (1 - \rho_2)(g^{(t)})^2.$$

- The total weights of the derivatives are not 1, and a bias correction is applied

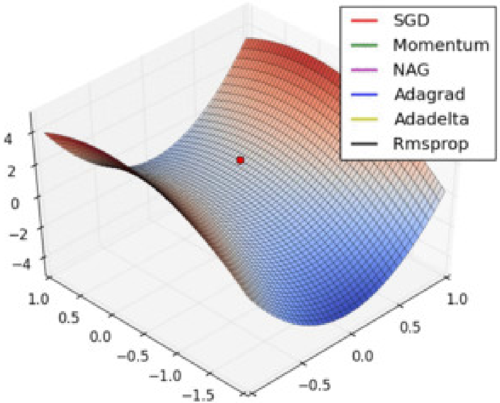
$$\hat{m}^{(t)} = m^{(t)}/(1 - \rho_1^t),$$
$$\hat{v}^{(t)} = m^{(t)}/(1 - \rho_2^t).$$

- At iteration t, current weight $w^{(t)}$ is updated to

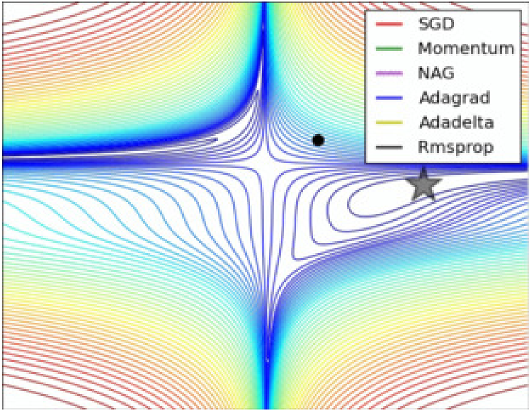
$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)} + \epsilon}} \hat{m}^{(t)}.$$

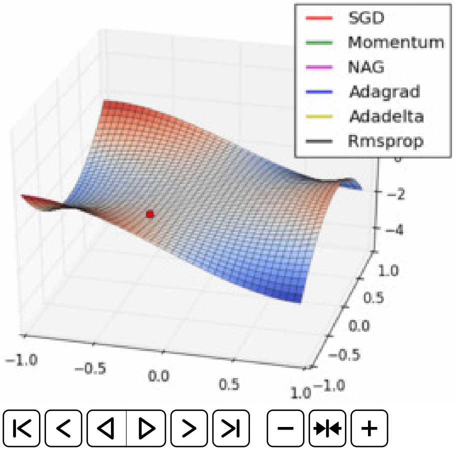
Kingma and Ba, Adam: A method for stochastic optimization, 2015

Visualizing Optimization Algorithms



https://imgur.com/a/Hqolp#2dKCQHh


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Your Turn

Which of the following statement is correct? (Multiple choice)

(a) Having a learning rate per dimension for gradient-based algorithm can possibly lead to better convergence behavior.

(b) AdaGrad, RMSProp and AdaDelta can all be seen as gradient-based methods that adaptively define learning rates for each dimension.

(c) Adam combines RMSProp with standard momentum.

Numerical Optimization for Machine Learning

- The error surface is often nonconvex and nonsmooth (local minima, saddle points, plateaus...)
- Some commonly used techniques
  - Acceleration using a momentum term (standard momentum, Nesterov)
  - Good initialization (Xavier, He)
  - Normalization tricks (input normalization, weight-dependent normalization for non-input layers)
  - Adaptive learning rates (Adagrad, RMSProp, AdaDelta, Adam)

Debugging Your Neural Net

- Architecture has enough but not too much capacity?
- Input normalized?
- Good initialization?
- Suitable loss function?
- Suitable optimization algorithm with suitable hyperparameters?
- Trained for long enough?