CS-568 Deep Learning

Nazar Khan

PUCIT

Gradient Descent Variations

So far . . .

- Neural Networks are universal approximators.
- Backpropagation allows computation of derivatives in hidden layers.
- Gradient descent finds weights corresponding to local minimum of loss surface.
- In this lecture: alternative methods of finding local minima of loss surface.
    - First-order methods
        - Rprop
    - Second-order methods
        - Quickprop
    - Momentum-based first-order methods
        - Momentum
        - Nesterov Accelerated Gradient
        - RMSprop
        - ADAM

Gradient Descent in Higher Dimensions

- Let $\Delta \mathbf{w}^{\tau+1}$ denote the step[1] at time $\tau + 1$.

$$w^{\tau+1} = w^{\tau} + \Delta w^{\tau+1}$$

- For gradient descent

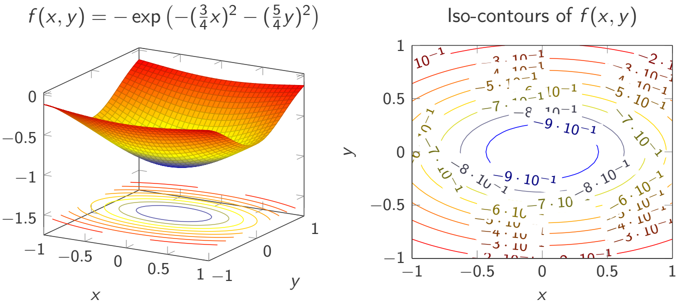
$$\Delta \mathbf{w}^{\tau+1} = -\eta \nabla_{\mathbf{w}}^{\tau} L$$

- For gradient descent in $1D$,

$$\Delta w^{\tau+1} = -\eta \left. \frac{dL}{dw} \right|_{\tau}$$

The only issue is determining step size η.

[1] Step $\neq$ step size.



A function that changes faster in y-direction.

- In higher dimensions, if $\left|\frac{\partial L}{\partial w_i}\right| >> \left|\frac{\partial L}{\partial w_j}\right|$ then using the same η *can* result in overshooting in the direction of w_i and very slow convergence in the direction of w_j.
- Solution: separate step size η_i for each direction w_i.

Resilient Propagation (Rprop)

- In Rprop[2], each direction is handled independently.
- Increase step size for direction i if current derivative has same sign as previous derivative.
- Otherwise, you just overshot a minimum.
  - So go back to previous location.
  - Decrease step size for that direction.
  - Update parameter with this smaller step.

$$\eta_i = \begin{cases} \min(\alpha\eta_i, \eta_{\max}) & \text{if } \left.\frac{\partial L}{\partial w_i}\right|_{\tau} * \left.\frac{\partial L}{\partial w_i}\right|_{\tau-1} > 0 \\ \max(\beta\eta_i, \eta_{\min}) & \text{if } \left.\frac{\partial L}{\partial w_i}\right|_{\tau} * \left.\frac{\partial L}{\partial w_i}\right|_{\tau-1} < 0 \\ \eta_i & \text{otherwise} \end{cases}$$

[2] Riedmiller and Braun, 'A direct adaptive method for faster backpropagation learning: The RPROP algorithm'.

Resilient Propagation (Rprop)

- *Hyperparameters* should follow the constraint $\alpha > 1 > \beta$.
- Typical values are $\alpha = 1.2$ and $\beta = 0.5$.
  - Increase step size slowly but decrease quickly when you overshoot.
- Step sizes are bounded via η_{min} and η_{max}.
- Rprop converges much faster than gradient descent.
- But it works well when derivatives are accumulated over large batches.

Taylor Series Approximation

- If values of a function $f(a)$ and its derivatives $f'(a), f''(a), \ldots$ are known at a value a, then we can approximate $f(x)$ for x close to a via the *Taylor series expansion*

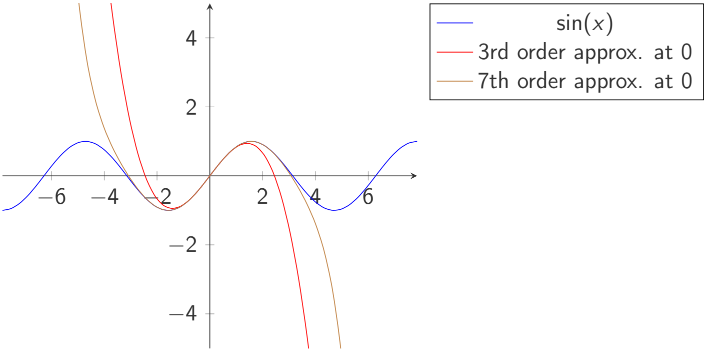
$$f(x) \approx f(a) + (x-a)^1 \frac{f'(a)}{1!} + (x-a)^2 \frac{f''(a)}{2!} + (x-a)^3 \frac{f'''(a)}{3!} + O((x-a)^4)$$

- For example, for x around $a = 0$
  - $\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$
  - $e^x \approx 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$
- Using $\Delta x = x - a$, Taylor series can be equivalently expressed as

$$f(a + \Delta x) \approx f(a) + (\Delta x)^1 \frac{f'(a)}{1!} + (\Delta x)^2 \frac{f''(a)}{2!} + (\Delta x)^3 \frac{f'''(a)}{3!} + O((\Delta x)^4)$$

$$= \sum_{n=0}^{\infty} \frac{1}{n!} f^n(a)(\Delta x)^n$$

Taylor Series Approximation
Not very useful for x not close to a



The sine function (blue) is closely approximated around 0 by its Taylor polynomials. The 7th order approximation is good for a full period centered at 0. However, it becomes poor for $|x-0|>\pi$.

Taylor Series Approximation

- It is often convenient to use the first-order Taylor expansion

$$f(a+\Delta x) \approx f(a) + \Delta x f'(a)$$

  or the second order Taylor expansion

$$f(a+\Delta x) \approx f(a) + \Delta x f'(a) + \frac{1}{2}(\Delta x)^2 f''(a)$$

- In d-dimensional input space

$$f(\mathbf{a}+\Delta\mathbf{x}) \approx f(\mathbf{a}) + \Delta\mathbf{x}^T \nabla f + \frac{1}{2}\Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x}$$

  where $\mathbf{H} \in \mathbb{R}^{d \times d}$ is the Hessian matrix composed from second derivatives.

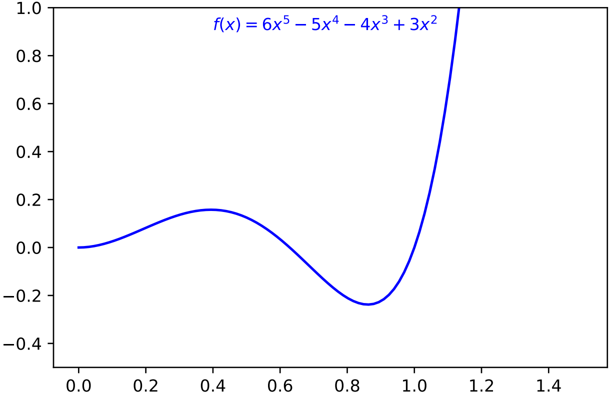
Newton's Method for finding stationary points

- Starting from a_0, we want to find a stationary point of f.
- Instead of actual function f, use a quadratic approximation (second-order Taylor expansion) of f at a_0.
- Find a step Δx such that $a_0 + \Delta x$ minimizes the quadratic approximation of f.

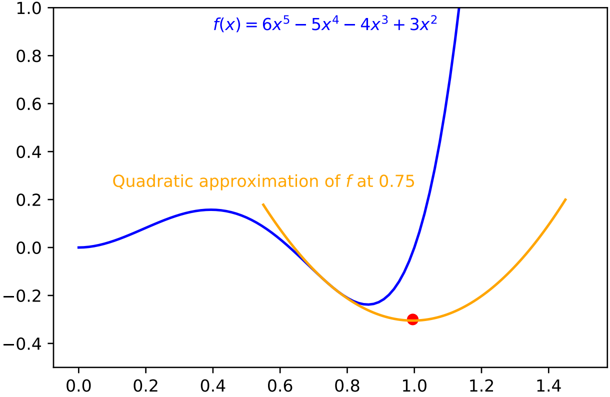
$$\begin{aligned} \frac{d}{d\Delta x}\left(f(a_0) + f'(a_0)\Delta x + \frac{1}{2}f''(a_0)(\Delta x)^2\right) &= 0 \\ f'(a_0) + f''(a_0)\Delta x &= 0 \\ \Delta x &= -\frac{f'(a_0)}{f''(a_0)} \end{aligned}$$

- Move to $a_1 = a_0 + \Delta x$ and repeat the process at a_1.
- Continue until convergence to a stationary point a_n.

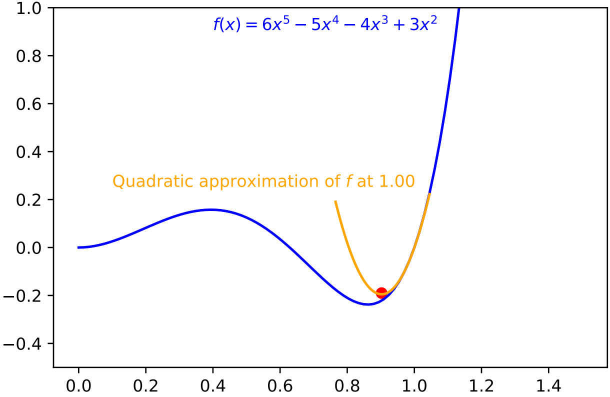
Newton's Method for finding stationary points



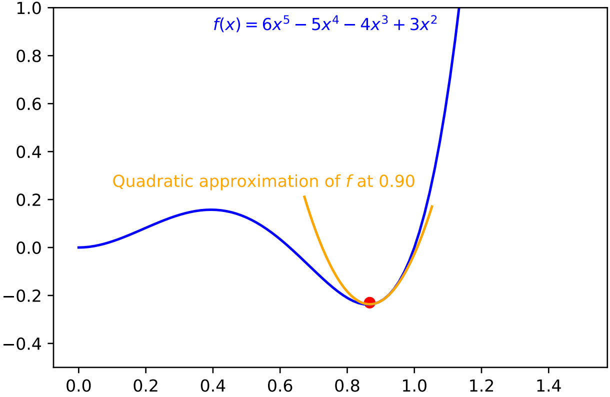
Newton's Method for finding stationary points



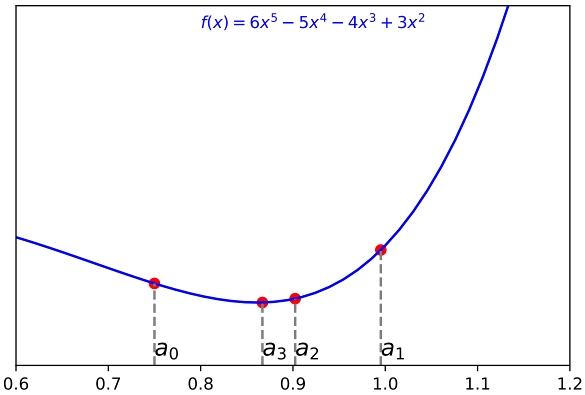
Newton's Method for finding stationary points



Newton's Method for finding stationary points



Newton's Method for finding stationary points



Newton's Method

- For weights of a neural network, Newton's update corresponds to

$$w^{\tau+1} = w^{\tau} - \left(\frac{\partial^2 L}{\partial w^2}\right)^{-1} \frac{\partial L}{\partial w}$$

- In other words, gradient descent step size η corresponds to inverse of 2nd-derivative.
- Division by 2nd-derivative can also be viewed as normalizing the gradient.
- In higher dimensions

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \mathbf{H}^{-1}\nabla_{\mathbf{w}} L$$

  The inverse Hessian matrix normalizes the gradient vector.
- Complete Hessian matrix is rarely used because of its size and computational cost.
  - Common assumption: diagonal Hessian matrix.

Quickprop

- Decouple all directions.
- Perform Newton updates in each direction.

$$w^{\tau+1} = w^{\tau} - \left(\frac{\partial^2 L}{\partial w^2}\right)^{-1} \frac{\partial L}{\partial w}$$

- Approximate 2nd-derivative by finite difference of 1st-derivatives.

$$\frac{\partial^2 L}{\partial w_i^2} \approx \frac{\left.\frac{\partial L}{\partial w_i}\right|_{\tau} - \left.\frac{\partial L}{\partial w_i}\right|_{\tau-1}}{\Delta w_i^{\tau-1}}$$

- Leads to very fast convergence.
- Some instability where loss is non-convex since everything is based on assumptions of convexity[3].

[3] Quadratic approximation in Newton's method
Fahlman, *An empirical study of learning speed in back-propagation networks*.

Momentum Updates

Basic idea

- Keep track of oscillating directions.
- Increase step size in directions that converge smoothly.
- Decrease step size in directions that overshoot and oscillate.

Steps

1. Compute gradient step $-\eta \left.\nabla_{\mathbf{w}} L\right|_{\mathbf{w}^\tau}$ at the current location $\mathbf{w}^\tau$.
2. Add the scaled previous step $\beta\Delta\mathbf{w}^\tau$ to obtain a running average of the step

$$\Delta\mathbf{w}^{\tau+1} = \beta\Delta\mathbf{w}^\tau - \eta \left.\nabla_{\mathbf{w}} L\right|_{\mathbf{w}^\tau}$$

   Typically $\beta = 0.9$.

3. Update parameters by the running average of the step

$$\mathbf{w}^{\tau+1} = \mathbf{w}^\tau + \Delta\mathbf{w}^{\tau+1}$$

Why does momentum work?

- Directions that oscillate will cancel each other out in the running average.
  - So the running average will be small in magnitude.
  - So the steps for oscillating directions will be smaller.
- Directions that are consistently converging will be reinforced.
  - So the running average will be large in magnitude.
  - So those directions will gain *momentum* by having larger and larger steps.

Nesterov Accelerated Gradient

Same idea as momentum updates *but with steps 1 and 2 swapped.*

1. Extend the previous scaled step.

$$\hat{\mathbf{w}} = \mathbf{w}^\tau + \beta \Delta \mathbf{w}^\tau$$

2. Compute gradient step at resultant location $\hat{\mathbf{w}}$.

$$-\eta \left. \nabla_{\mathbf{w}} L \right|_{\hat{\mathbf{w}}}$$

3. Add previous scaled step and new gradient step to obtain the running average of the step

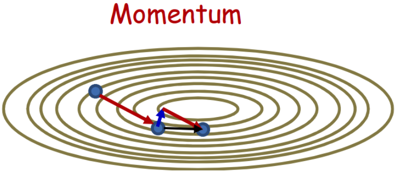
$$\Delta \mathbf{w}^{\tau+1} = \beta \Delta \mathbf{w}^\tau - \eta \left. \nabla_{\mathbf{w}} L \right|_{\hat{\mathbf{w}}}$$

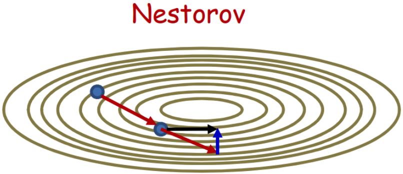
4. Update parameters by the running average of the step

$$\mathbf{w}^{\tau+1} = \mathbf{w}^\tau + \Delta \mathbf{w}^{\tau+1}$$

Nesterov's method has been shown to converge faster than momentum updates.

Momentum vs. Nesterov Momentum





Nesterov – Sometimes it is better make a correction after making an error.
Source: Bhiksha Raj

RMSprop

- Decouple each direction.
- We can compute the running average of the *squared* 1st-derivative in direction i as

$$\bar{v}_i^\tau = \gamma \bar{v}_i^{\tau-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_i}\right)^2$$

  with initialization $\bar{v}_i^0 = 0$.
- Root-mean-squared (RMS) value $\sqrt{\bar{v}_i^\tau} + \epsilon$ represents average magnitude of 1st-derivative for direction i.
  - High value indicates oscillating derivatives. So reduce step size.
  - Low value indicates flat region. So increase step size.
- So divide step size by this average before performing gradient descent.

$$w_i^\tau = w_i^{\tau-1} - \frac{\eta}{\sqrt{\bar{v}_i^\tau} + \epsilon}\frac{\partial L}{\partial w_i}$$

Rprop vs RMSprop

Rprop

Multiplicatively increase step size when derivative retains its sign.

$$\eta \leftarrow \alpha\eta$$

Multiplicatively decrease step size when derivative oscillates.

$$\eta \leftarrow \beta\eta$$

RMSprop

Multiplicatively increase/decrease step size when average derivative magnitude decreases/increases.

$$\eta \leftarrow \frac{\eta_0}{\sqrt{\bar{v}} + \epsilon}$$

Fixed multiplicative factors α and β in Rprop are replaced by *adaptive* factor $\frac{1}{\sqrt{\bar{v}}+\epsilon}$ in RMSprop.

ADAM

RMSprop+Momentum

- RMSprop uses the current derivative.
- ADAM[5] replaces current derivative by its running average.

$$\bar{m}_i^{\tau} = \delta \bar{m}_i^{\tau-1} + (1-\delta)\frac{\partial L}{\partial w_i}$$

- *Currently the most popular flavor of gradient descent.*
- Statistics terminology:
  - Average of random variable x is also called its 1st statistical moment $E[x]$.
  - Average of the square of a random variable is also called its 2nd *uncentered* statistical moment $E[x^2]$.
  - Average of the square of a centered random variable is also called its 2nd statistical moment $E[(x-\mu)^2]$ or variance.

[5] Kingma and Ba, 'ADAM: A Method for Stochastic Optimization'.

ADAM
RMSprop+Momentum

- Initialize moments $\bar{m}_i^0 = 0$ and $\bar{v}_i^0 = 0$.
- Compute 1st moment and 2nd uncentered moment of derivative

$$\bar{m}_i^\tau = \delta \bar{m}_i^{\tau-1} + (1-\delta)\frac{\partial L}{\partial w_i}$$

$$\bar{v}_i^\tau = \gamma \bar{v}_i^{\tau-1} + (1-\gamma)\left(\frac{\partial L}{\partial w_i}\right)^2$$

- Correct for bias of initial moments $(=0)$ by scaling up in early iterations.

$$\bar{m}_i^\tau = \frac{\bar{m}_i^\tau}{1-\delta^\tau} \text{ and } \bar{v}_i^\tau = \frac{\bar{v}_i^\tau}{1-\gamma^\tau}$$

- Perform update

$$w_i^\tau = w_i^{\tau-1} - \frac{\eta}{\sqrt{\bar{v}_i^\tau} + \epsilon}\bar{m}_i^\tau$$

- Proposed hyperparameter values: $\eta = 10^{-3}, \delta = 0.9, \gamma = 0.999, \epsilon = 10^{-8}$.

Summary

- For complex and non-convex loss functions of deep networks, vanilla gradient descent can get stuck in poor local minima and saddle points.
- It can also converge very slowly.
- Different directions require different step sizes.
- Adaptive step sizes are very important.
- Most useful technique is to adapt step size based on *recent trend* of 1st-derivative.