EC-332 Machine Learning

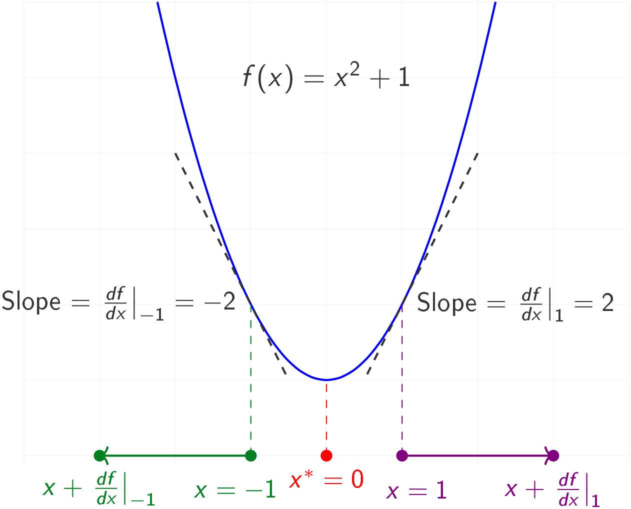
Gradient Descent and its Variations

Nazar Khan
Department of Computer Science
University of the Punjab

So far . . .

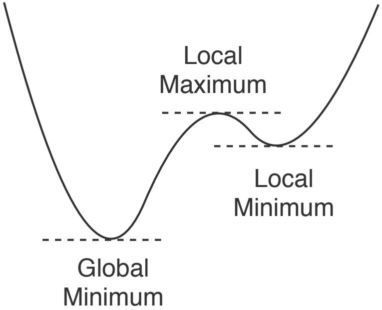
- Neural Networks are universal approximators.
- Backpropagation allows computation of derivatives in hidden layers.
- In this lecture: gradient descent finds weights corresponding to local minimum of loss surface.
- In this lecture: alternative methods of finding local minima of loss surface.
    - Gradient descent
    - First-order methods
        - Rprop
    - Second-order methods
        - Taylor series approximation
        - Newton's method
        - Quickprop
- Next lecture:
    - Momentum-based first-order methods

Minimization



What is the slope/derivative/gradient at the minimizer $x^* = 0$?

Minimization
Local vs. Global Minima



- *Stationary point*: where derivative is 0.
- A stationary point can be a minimum or a maximum.
- A minimum can be local or global. Same for maximum.

Gradient Descent

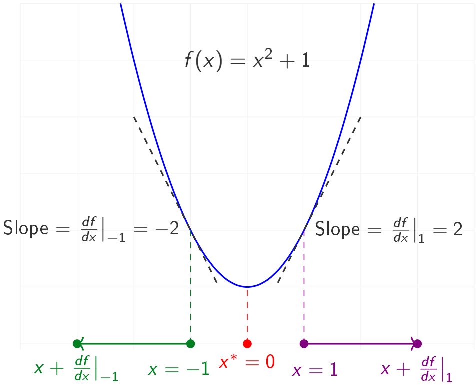
- Gradient is the direction, in input space, of maximum rate of increase of a function.

$$f\left(x + \frac{df}{dx}\right) \geq f(x)$$

- To minimize function $f(x)$ with respect to x, move in negative gradient direction.

$$x^{\text{new}} = x^{\text{old}} - \left.\frac{df}{dx}\right|_{x^{\text{old}}}$$

- Try it! Start from $x^{\text{old}} = -1$. Do you notice any problem?



Minimization via Gradient Descent

- To minimize loss $L(\mathbf{w})$ with respect to weights $\mathbf{w}$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L(\mathbf{w})$$

  where scalar $\eta > 0$ controls the step-size. It is called the *learning rate*.
- Also known as *gradient descent*.

> Repeated applications of gradient descent find the closest local minimum.

Gradient Descent

1. Initialize $\mathbf{w}^{\text{old}}$ randomly.
2. do
   2.1 $\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \, \nabla_{\mathbf{w}} L(\mathbf{w})\big|_{\mathbf{w}^{\text{old}}}$
3. while $\left| L(\mathbf{w}^{\text{new}}) - L(\mathbf{w}^{\text{old}}) \right| > \epsilon$

- Learning rate η needs to be reduced gradually to ensure *convergence to a local minimum*.
- If η is too large, the algorithm can *overshoot* the local minimum and keep doing that indefinitely *(oscillation)*.
- If η is too small, the algorithm will take too long to reach a local minimum.

Gradient Descent

- Different types of gradient descent:

Batch	$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L$
Sequential	$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L_n$
Stochastic	same as sequential but n is chosen randomly
Mini-batches	$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} L_{\mathcal{B}}$

- Most common variations are stochastic gradient descent (SGD) and SGD using mini-batches.

Gradient Descent in Higher Dimensions

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

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

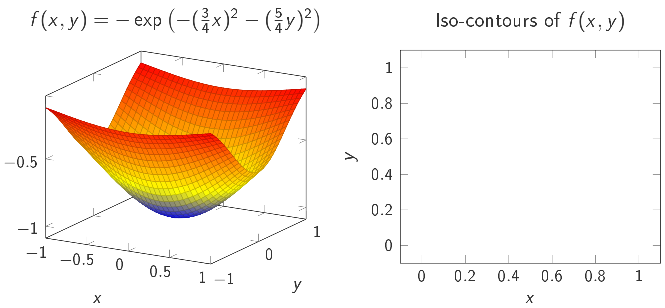
- For gradient descent

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

- For gradient descent in $1D$,

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

The only issue is determining learning rate η.



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 learning rate η_i for each direction w_i.

Resilient Propagation (Rprop)

- In Rprop[1], each direction is handled independently.
- Increase learning rate 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 learning rate for that direction.
  - Update parameter with this smaller step.

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

- *Hyperparameters* should follow the constraint $\alpha > 1$ and $\beta < 1$.

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

Resilient Propagation (Rprop)

- Typical values are $\alpha = 1.2$ and $\beta = 0.5$.
  - Increase learning rate slowly but decrease quickly when you overshoot.
- In practice, learning rates are bounded via η_{min} and η_{max}.

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

- 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 $\underline{x \text{ close to } a}$ via the *Taylor series expansion*

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

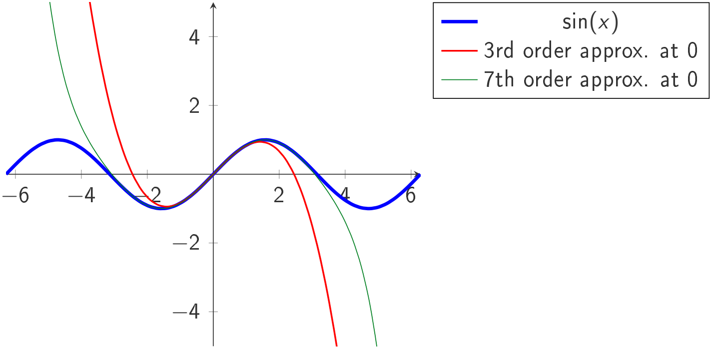
- Using $\Delta x = x - a$, Taylor series can be equivalently expressed as

$$\begin{aligned} f(a+\Delta x) &\approx f(a) + (\Delta x)^1 \frac{f'(a)}{1!} + (\Delta x)^2 \frac{f''(a)}{2!} + (\Delta x)^3 \frac{f'''(a)}{3!} + O((\Delta x)^4) \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} f^n(a)(\Delta x)^n \end{aligned}$$

Taylor Series Approximation
Examples

- For x around $a = 0$
  - $\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$
  - $e^x \approx 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$

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 (green) 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.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d \partial x_d} \end{bmatrix}$$

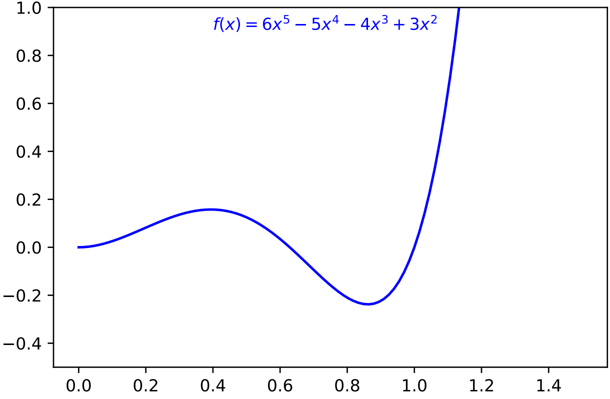
Newton's Method for finding stationary points

- 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.

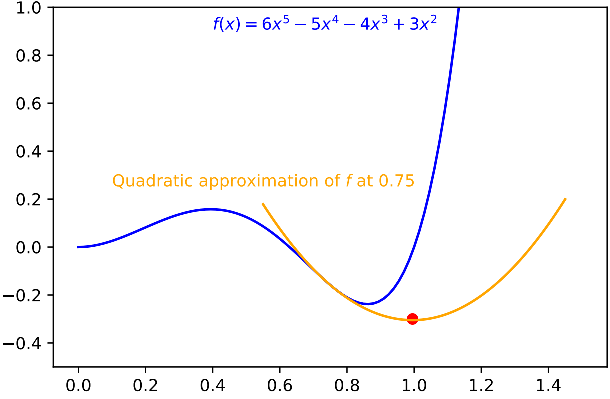
$$\begin{aligned}
\frac{d}{d\Delta x}\left(f(a_0) + f'(a_0)\Delta x + \frac{1}{2}f''(a_0)(\Delta x)^2\right) &= 0 \\
f'(a_0) + f''(a_0)\Delta x &= 0 \\
\Delta x &= -\frac{f'(a_0)}{f''(a_0)}
\end{aligned}$$

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

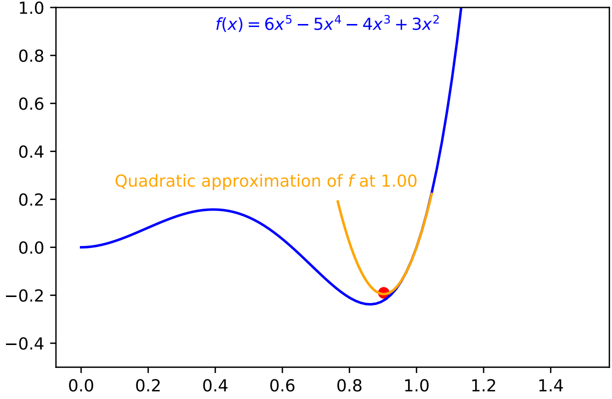
Newton's Method for finding stationary points



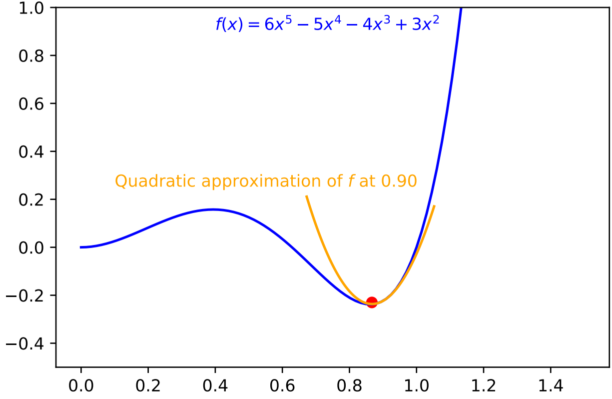
Newton's Method for finding stationary points



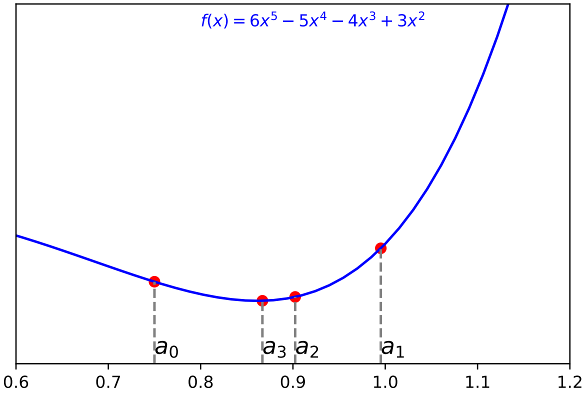
Newton's Method for finding stationary points



Newton's Method for finding stationary points



Newton's Method for finding stationary points



Newton's Method
Role of the 2nd-derivative

- 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 learning rate η corresponds to inverse of 2nd-derivative.
- Division by 2nd-derivative can also be viewed as normalising the gradient.
- In higher dimensions

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

  The inverse Hessian matrix normalises the gradient vector.

Newton's Method

Role of the 2nd-derivative

- Complete Hessian matrix is rarely used because of its size and computational cost of inverting it.
- Common assumption: diagonal Hessian matrix.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & 0 & \dots & 0 \\ 0 & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\partial^2 f}{\partial x_d \partial x_d} \end{bmatrix}$$

- Inverse of diagonal matrix is cheap (reciprocal of entries on the diagonal).

Quickprop

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

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

- Approximate 2nd-derivative *numerically* 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 (quadratic approximation in Newton's method).

Fahlman, *An empirical study of learning speed in back-propagation networks*.

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 learning rates.
- Adaptive learning rates are very important.
- Next lecture: momentum-based first-order methods.