UNIVERSITY
OF OSLO



IN4310 Back Propagation and Optimization

Ali Ramezani-Kebrya

ali@uio.no



Learning Goals

Directional derivatives and gradient

Back propagation for neural networks

Stochastic gradient descent and mini batching

SGD with momentum

Variations of SGD

Outline

Gradient

Back propagation for neural networks

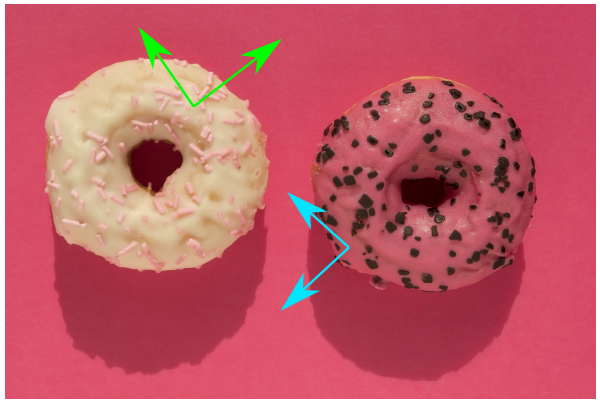
Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

Matrix Calculus: Gradient

$f : \mathbb{R}^n \to \mathbb{R}$ with n-dimensional input $\nabla_{\mathbf{x}} f(\mathbf{x}) = [\frac{\partial f(\mathbf{x})}{\partial x_1} \cdots \frac{\partial f(\mathbf{x})}{\partial x_n}]^\top$



Surface of a donut is two dimensional

At each point, the move along each direction has a slope

Directional derivative how much function value changes along a direction

Directional Derivatives and Gradient

Directional derivative of function f in $\mathbf{x}$ along $\mathbf{v}$

$$\delta_{\mathbf{v}} f(\mathbf{x}) = \lim_{\epsilon \to 0} \frac{f(\mathbf{x} + \epsilon \mathbf{v}) - f(\mathbf{x})}{\epsilon}$$

If the function is differentiable, the directional derivative in $\mathbf{x}$ along $\mathbf{v}$ is inner product of gradient and direction

$$\delta_{\mathbf{v}} f(\mathbf{x}) = \mathbf{v}^\top \nabla f(\mathbf{x})$$

Directional derivative tells how the function grows along a direction under an infinitely small step

Gradient contains information about all directional derivatives

Gradient through Partial Derivatives

Let $\mathbf{e}_i$ be a vector with all elements equal to 0, except the i-th, which is 1

$$\begin{aligned}\mathbf{e}_i^\top \nabla f(\mathbf{x}) = \delta_{\mathbf{e}_i} f(\mathbf{x}) &= \lim_{\epsilon \to 0} \frac{f(\mathbf{x} + \epsilon \mathbf{e}_i) - f(\mathbf{x})}{\epsilon} \\ &= \lim_{\epsilon \to 0} \frac{f(x_1, \ldots, x_i + \epsilon, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{\epsilon} \\ &= \frac{\partial f(\mathbf{x})}{\partial x_i}\end{aligned}$$

Then $\nabla_{\mathbf{x}} f(\mathbf{x}) = [\frac{\partial f(\mathbf{x})}{\partial x_1} \cdots \frac{\partial f(\mathbf{x})}{\partial x_n}]^\top$

Let $\mathbf{v} = [v_1 \cdots v_n]^\top$. We note

$$\delta_{\mathbf{v}} f(\mathbf{x}) = \mathbf{v}^\top \nabla f(\mathbf{x}) = \sum_{i=1}^{n} \frac{\partial f(\mathbf{x})}{\partial x_i} v_i$$

Outline

Gradient

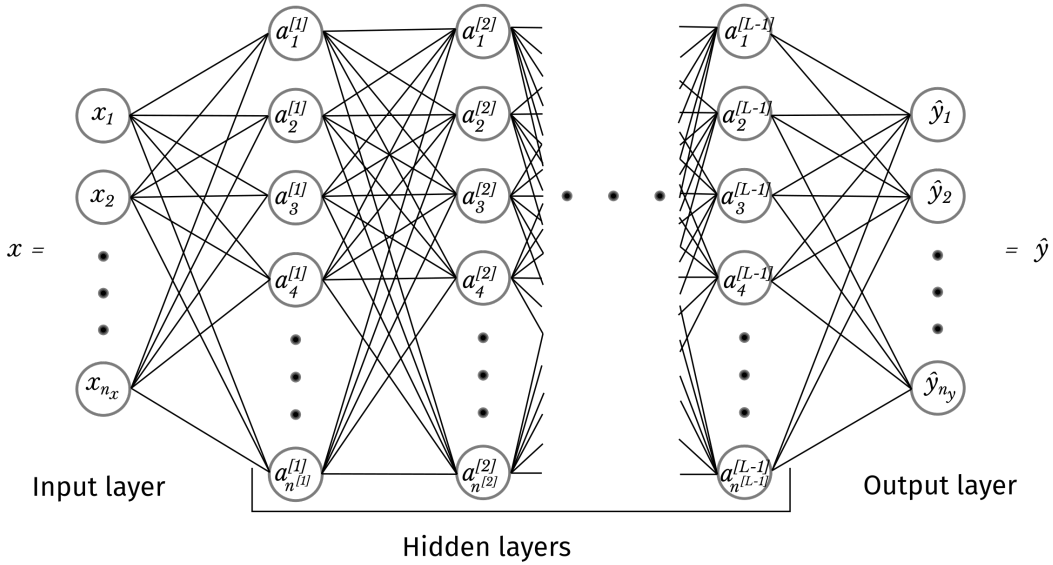
Back propagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

General Formulation



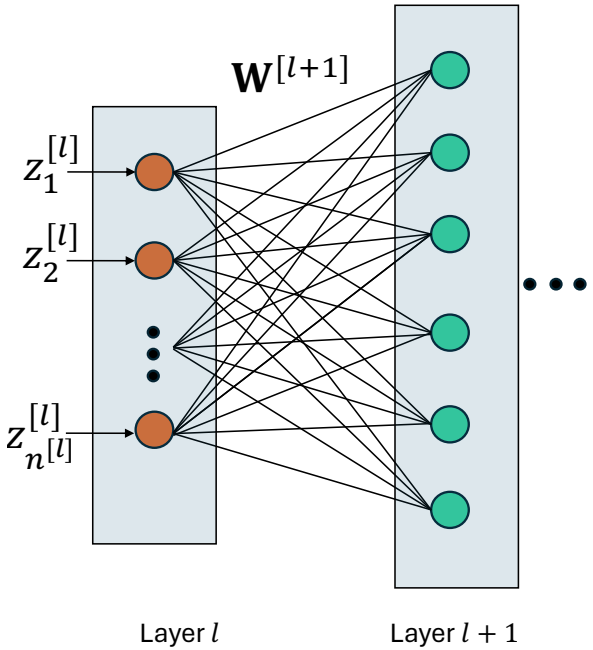
$$a_k^{[l]} = g\left(\mathbf{w}_k^{[l]} \cdot \mathbf{a}^{[l-1]} + b_k^{[l]}\right) = g(z_k^{[l]})$$

Concatenate all outputs $a_k^{[l]}$ of layer l and apply g element-wise

$$\mathbf{a}^{[l]} = g\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}\right)$$

$\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}$ is a matrix-vector multiplication

Forward Direction [1, 2]



Input $\mathbf{a}^{[0]} = \mathbf{x}$ Output $\mathbf{a}^{[L]} = \hat{\mathbf{y}}$

$$\mathbf{a}^{[l]} = g\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}\right)$$

$$\mathbf{a}^{[0]} \xrightarrow{\mathbf{W}^{[1]},\mathbf{b}^{[1]}} \mathbf{z}^{[1]} \xrightarrow{g} \mathbf{a}^{[1]} \xrightarrow{\mathbf{W}^{[2]},\mathbf{b}^{[2]}} \mathbf{z}^{[2]} \xrightarrow{g} \mathbf{a}^{[2]} \ldots \mathbf{z}^{[L]} \xrightarrow{\text{Softmax}} \mathbf{a}^{[L]}$$

ERM with Cross-entropy Loss

Use stochastic gradient descent to optimise the cross-entropy loss

$$\boldsymbol{\theta}^{\star} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{\mathbf{y}}_n, \mathbf{y}_n | \boldsymbol{\theta})$$

Model's predicted probabilities $\hat{\mathbf{y}}_n \in [0,1]^{n_y}$

Input sample $\mathbf{x}_n \in \mathbb{R}^{n_x}$ with true class label $\mathbf{y}_n \in [0,1]^{n_y}$

Parameters

$$\begin{aligned}\boldsymbol{\theta} &= \{w_{i,j}^{[l]}, b_j^{[l]} : i \in \{1,\dots,n^{[l-1]}\}, j \in \{1,\dots,n^{[l]}\}, l \in \{1,\dots,L\}\} \\ &= \{\mathbf{W}^{[l]}, \mathbf{b}^{[l]} : l \in \{1,\dots,L\}\}\end{aligned}$$

Gradient Descent for ERM

Let $h_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ denote a predictor parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$. ERM:

$$\min_{\boldsymbol{\theta}} \left\{ R_N(\boldsymbol{\theta}) := \frac{1}{N} \sum_{n=1}^{N} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_n), \mathbf{y}_n) \right\}$$

where $\ell(h_{\boldsymbol{\theta}}(\mathbf{x}_n), \mathbf{y}_n)$ is the loss on sample n, i.e., $(\mathbf{x}_n, \mathbf{y}_n)$.

Initialize $\boldsymbol{\theta}_0$

For $t = 0, 1, 2, \ldots$

Compute $\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)$

Update $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)$

Continue until $\|\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)\| \approx 0$

Stochastic Gradient Descent for ERM

Gradient Descent: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\alpha_t}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \ell_n(\boldsymbol{\theta}_t)$

Computational complexity per update: $\mathcal{O}(Nd)$

N can be very large

Stochastic Gradient Descent:

For $t = 0, 1, 2, \ldots$

Select $n \in \{1, \ldots, N\}$ uniformly at random

$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} \ell_n(\boldsymbol{\theta}_t)$

Computational complexity per update: $\mathcal{O}(d)$

Requires more iterations

Stochastic Gradient Descent for ERM

For simplicity, let's drop iteration index t

In each iteration

Select $(\mathbf{x}_n, \mathbf{y}_n)$ with $n \in \{1, \ldots, N\}$ uniformly at random

Compute $\ell_n(\boldsymbol{\theta})$

Compute $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}}$ for all i, j, l

$$w_{i,j}^{[l]} = w_{i,j}^{[l]} - \alpha \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}}$$

Number of parameters (edges) $E = \sum_{l=1}^{L} (n^{[l-1]} + 1) n^{[l]}$

\# elementary operations in forward pass $\Theta(E)$: Review O, Ω, Θ notations!

Computing Gradient for One Sample

Modify $w_{i,j}^{[l]} \to w_{i,j}^{[l]} + \Delta$ and keep all other parameters unchanged

New model parameters $\tilde{\boldsymbol{\theta}}$

$$\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}} = \lim_{\Delta \to 0} \frac{\ell_n(\tilde{\boldsymbol{\theta}}) - \ell_n(\boldsymbol{\theta})}{\Delta}$$

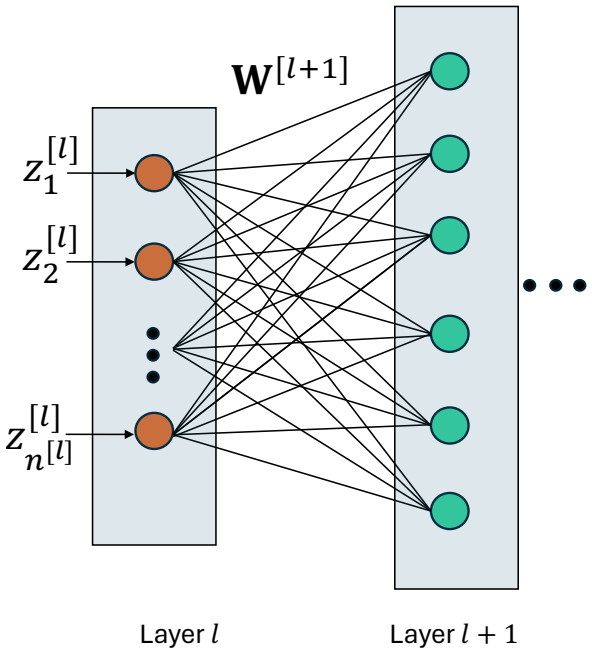
For each $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}}$ at each iteration, we need $\Theta(E)$ elementary operations

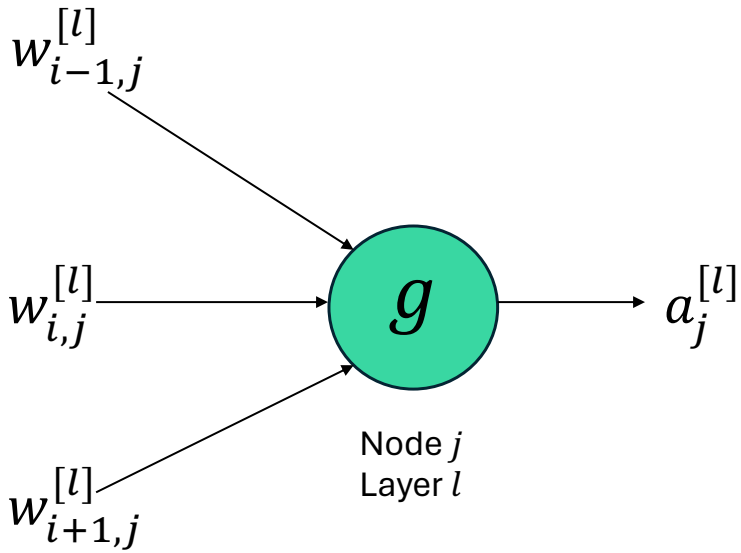
Total complexity per iteration $\Theta(E^2)$ too slow

Back Propagation Algorithm [Rumelhart et al., Nature1986]

Compute all $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}}$ using $\Theta(E)$ elementary operations like forward pass

Back Propagation for Neural Networks





Compute $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}}$ for all i, j, l

Given $z_1^{[l]}, \ldots, z_{n^{[l]}}^{[l]}$, no need to previous layers' to compute what follows

$w_{i,j}^{[l]}$ only affects $z_j^{[l]}$ and given $z_j^{[l]}$, no need to know $w_{i,j}^{[l]}$

Chain Rule

$w_{i,j}^{[l]}$ only affects $z_j^{[l]}$ and given $z_j^{[l]}$, no need to know $w_{i,j}^{[l]}$

$$\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial w_{i,j}^{[l]}}$$

$$\ell_n(\boldsymbol{\theta}) = f(z_1^{[l]}, \ldots, z_j^{[l]}, \ldots, z_{n^{[l]}}^{[l]}, \mathbf{W}^{[l+1]}, \ldots, \mathbf{W}^{[L]})$$

$$\frac{\partial z_j^{[l]}}{\partial w_{i,j}^{[l]}} = a_i^{[l-1]}, \qquad z_j^{[l]} = \sum_k w_{k,j}^{[l]} a_k^{[l-1]}$$

Let define $\delta_j^{[l]} := \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}}$ then $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}} = a_i^{[l-1]} \delta_j^{[l]}$

$a_i^{[l-1]}$: forward message and $\delta_i^{[l]}$: backward message

$w_{i,j}^{[l]}$ connects Node j in Layer l with Node i in Layer $l-1$

How to Compute $\delta_i^{[l]}$: Backward Message

Output layer

$$\delta_j^{[L]} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[L]}} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}}$$

$a_j^{[L]} = (\hat{\mathbf{y}}_n)_j$ Predicted probability of class j in sample n

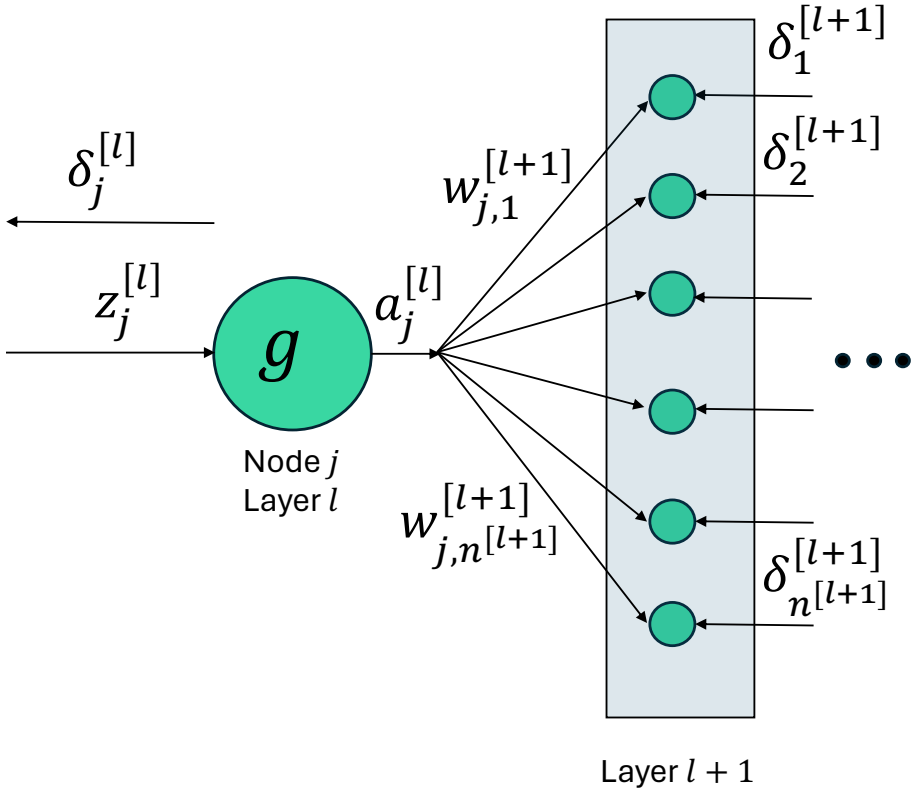
$\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[L]}}$ can be computed using loss. For quadratic loss and regression:

$\ell_n(\hat{y}_n, y_n) = (\hat{y}_n - y_n)^2$. Then $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial \hat{y}_n} = 2(\hat{y}_n - y_n)$

Output layer softmax: $a_j^{[L]} = s(z_j^{[L]})$

$\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \frac{\partial s(z_j^{[L]})}{\partial z_j^{[L]}} = s'(z_j^{[L]})$: easy to compute analytically

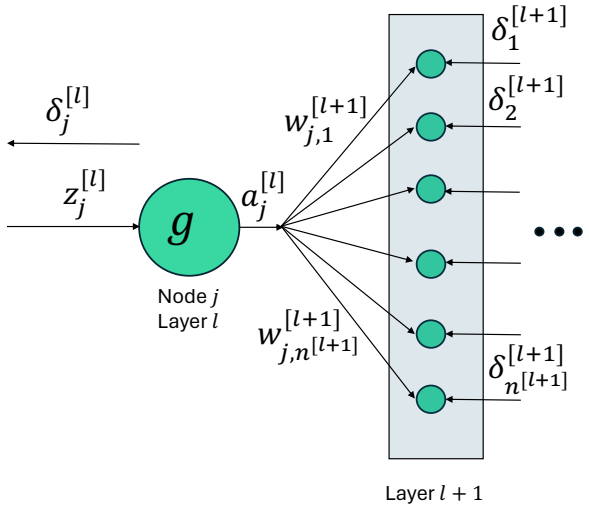
Backward Pass



$$\delta_j^{[l]} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[l]}} \frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} \qquad \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}} = a_i^{[l-1]} \delta_j^{[l]}$$

$$\ell_n(\boldsymbol{\theta}) = f(z_1^{[l]}, \ldots, z_j^{[l]}, \ldots, z_{n^{[l]}}^{[l]}, \mathbf{W}^{[l+1]}, \ldots, \mathbf{W}^{[L]})$$

Backward Pass



$$\delta_j^{[l]} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[l]}} \frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} \qquad \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}} = a_i^{[l-1]} \delta_j^{[l]}$$

$\ell_n(\boldsymbol{\theta}) = h(z_1^{[l+1]}, \ldots, z_{n^{[l+1]}}^{[l+1]}, \mathbf{W}^{[l+2]}, \ldots, \mathbf{W}^{[L]})$ affected by $a_j^{[l]}$

$$\frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} = g'(z_j^{[l]}) \qquad \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[l]}} = \sum_{k=1}^{n^{[l+1]}} \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial a_j^{[l]}}$$

Backward Pass

$$\delta_j^{[l]} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[l]}} \frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} \qquad \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{i,j}^{[l]}} = a_i^{[l-1]} \delta_j^{[l]}$$

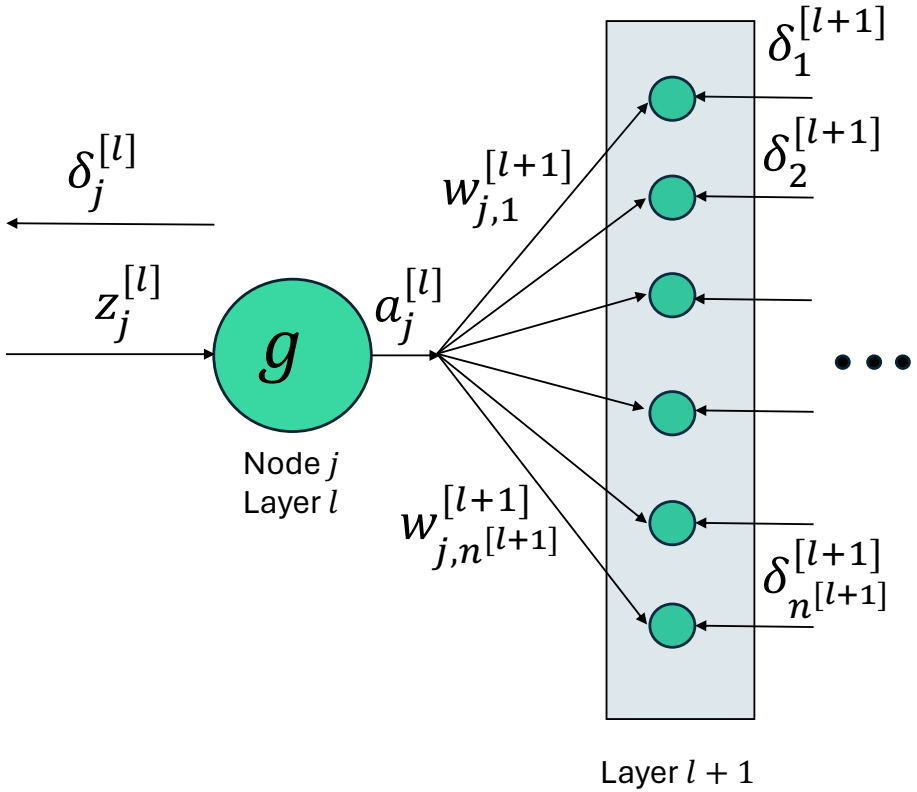
$\ell_n(\boldsymbol{\theta}) = h(z_1^{[l+1]}, \ldots, z_{n^{[l+1]}}^{[l+1]}, \mathbf{W}^{[l+2]}, \ldots, \mathbf{W}^{[L]})$ affected by $a_j^{[l]}$

$$\frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} = g'(z_j^{[l]}) \qquad \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[l]}} = \sum_{k=1}^{n^{[l+1]}} \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial a_j^{[l]}} = \sum_{k=1}^{n^{[l+1]}} \delta_k^{[l+1]} w_{j,k}^{[l+1]}$$

$$\delta_j^{[l]} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}} = \Big(\sum_{k=1}^{n^{[l+1]}} \delta_k^{[l+1]} w_{j,k}^{[l+1]} \Big) g'(z_j^{[l]})$$

$$\boldsymbol{\delta}^{[l]} = \mathbf{W}^{[l+1]} \boldsymbol{\delta}^{[l+1]} \odot g'(\mathbf{z}^{[l]})$$

Backward Pass



$$\boldsymbol{\delta}^{[l]} = \mathbf{W}^{[l+1]}\boldsymbol{\delta}^{[l+1]} \odot g'(\mathbf{z}^{[l]})$$

Forward pass: $\mathbf{a}^{[0]} \xrightarrow{\mathbf{W}^{[1]}} \mathbf{z}^{[1]} \xrightarrow{g} \mathbf{a}^{[1]} \xrightarrow{\mathbf{W}^{[2]}} \mathbf{z}^{[2]} \xrightarrow{g} \mathbf{a}^{[2]} \ldots \mathbf{z}^{[L]} \rightarrow \mathbf{a}^{[L]}$

Backward pass: $\boldsymbol{\delta}^{[L]} \rightarrow \boldsymbol{\delta}^{[L-1]} \rightarrow \ldots \rightarrow \boldsymbol{\delta}^{[1]}$

Interactive Time-out

Join at menti.com | use code 8144 4543

Outline

Gradient

Back propagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

Stochastic Gradient Descent for ERM

Stochastic Gradient Descent:

For $t = 0, 1, 2, \ldots$

Select $n \in \{1, \ldots, N\}$ uniformly at random

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} \ell_n(\boldsymbol{\theta}_t)$$

Computational complexity per update: $\mathcal{O}(E)$

SGD is Unbiased

Let's compute $\mathbb{E}_j[\nabla \ell_j(\mathbf{w})]$ where $j \sim \text{unif}\{1, \ldots, N\}$ and $\mathbf{w} \in \mathbb{R}^d$

$$\begin{aligned}\mathbb{E}_j[\nabla \ell_j(\mathbf{w})] &= \Pr\{j = 1\}\nabla \ell_1(\mathbf{w}) + \cdots + \Pr\{j = n\}\nabla \ell_n(\mathbf{w}) \\ &= 1/N \sum_{j=1}^{N} \nabla \ell_j(\mathbf{w}) = \nabla R_N(\mathbf{w})\end{aligned}$$

It will result in the actual gradient in expectation

SGD for Logistic Regression

$R_N(\mathbf{w}) = 1/N \sum_{j=1}^{N} \ell_j(\mathbf{w}) = 1/N \sum_{j=1}^{N} \log\left(1 + e^{-y_j \mathbf{w}^\top \mathbf{x}_j}\right)$

SGD: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \nabla \ell_j(\mathbf{w}_t), \quad j \sim \text{unif}\{1, \ldots, N\}$

$\nabla_{\mathbf{w}}\left[\log\left(1 + e^{-y_j \mathbf{w}_t^\top \mathbf{x}_j}\right)\right] = \frac{-y_j \mathbf{x}_j}{1 + e^{y_j \mathbf{w}_t^\top \mathbf{x}_j}}$

SGD for Logistic Regression

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t y_j \mathbf{x}_j / \left(1 + e^{y_j \mathbf{w}_t^\top \mathbf{x}_j}\right) \qquad \mathbf{y}_j \in \{-1, +1\}$$

Recall the update rule of Perceptron algorithm

if $(\mathbf{x}_j, y_j)$ is misclassified, $\mathbf{w}_{t+1} = \mathbf{w}_t + y_j \mathbf{x}_j$

if $(\mathbf{x}_j, y_j)$ is correctly classified, $\mathbf{w}_{t+1} = \mathbf{w}_t$

Connection with Perceptron algorithm

SGD for Logistic Regression: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t y_j \mathbf{x}_j / \left(1 + e^{y_j \mathbf{w}_t^\top \mathbf{x}_j}\right)$

Suppose $(\mathbf{x}_j, y_j)$ is misclassified by $\mathbf{w}_t$:

$y_j \neq \text{Sign}(\mathbf{w}_t^\top \mathbf{x}_j)$

$y_j \mathbf{w}_t^\top \mathbf{x}_j < 0$, $\mathbf{w}_t^\top \mathbf{x}_j$: update in Perceptron

Suppose $1 + e^{y_j \mathbf{w}_t^\top \mathbf{x}_j} \approx 1$, so we can ignore the denominator

Update rule for SGD: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t y_j \mathbf{x}_j$

Suppose $(\mathbf{x}_j, y_j)$ is correctly classified by $\mathbf{w}_t$:

$y_j = \text{Sign}(\mathbf{w}_t^\top \mathbf{x}_j)$, i.e., $y_j \mathbf{w}_t^\top \mathbf{x}_j > 0$

$1/\left(1 + e^{y_j \mathbf{w}_t^\top \mathbf{x}_j}\right) \approx 0$

Update rule for SGD: $\mathbf{w}_{t+1} = \mathbf{w}_t$

ERM in Supervised Learning

Training error: $R_N(\mathbf{w}) = 1/N \sum_{j=1}^{N} \ell_j(\mathbf{w})$

Linear Regression: $\ell_j(\mathbf{w}) = (\mathbf{x}_j^\top \mathbf{w} - y_j)^2$

Logistic Regression: $\ell_j(\mathbf{w}) = \log\left(1 + e^{-y_j \mathbf{w}^\top \mathbf{x}_j}\right)$

Goal: find $\mathbf{w}$ to minimize $R_N(\mathbf{w})$

SGD is awesome in terms of computational complexity

SGD may result in substantial variance in gradient updates

How to reduce the variance? $\rightarrow$ mini batch

Compute gradient over a mini batch, i.e., multiple samples

SGD with Mini Batch

SGD with mini batch size $= B$

Epoch: one complete pass through all samples

for epoch $t = 1, 2, \ldots, T$

Randomly permute the orders of training samples

for $j = 1, 2, \ldots, n/B$

Select the next batch of size B

Let $\mathcal{S}_j = \{j_1, \ldots, j_B\}$ denote the indices of selected samples

$\mathbf{w} \leftarrow \mathbf{w} - \alpha/B \sum_{i \in \mathcal{S}_j} \nabla \ell_i(\mathbf{w})$

Check if stopping condition ($\nabla R_N(\mathbf{w}) \approx 0$) is satisfied

Convex vs Nonconvex

Why using a minibatch?

- Having access to multi-core programming and parallel computing
- Obtaining more accurate gradients
- Improving overall training time

Stopping condition, i.e., $\nabla R_N(\mathbf{w}) \approx 0$, is good for convex loss

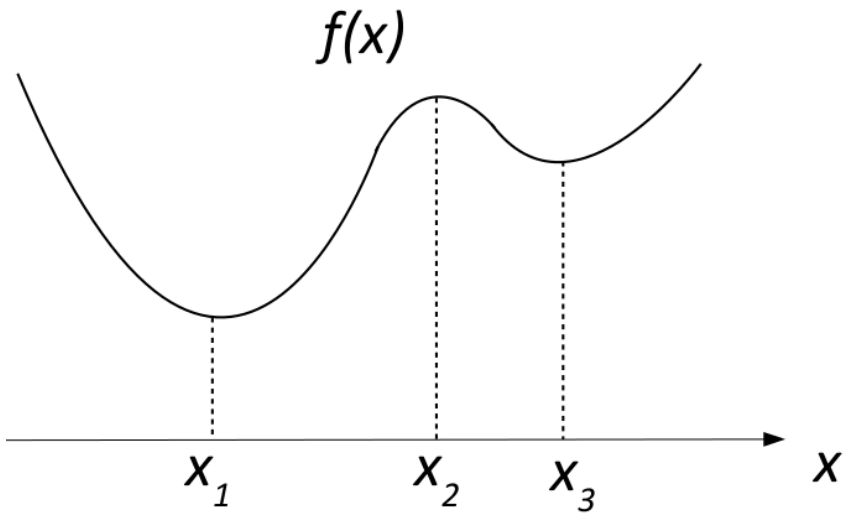
- Linear Regression
- Logistic Regression

For convex functions, satisfying $\nabla R_N(\mathbf{w}) \approx 0 \rightarrow$ Global Minimum

No global guarantees for nonconvex functions, typical issues:

- Local minima
- Saddle points
- Flat regions

Local Minima



$f'(x_1) = 0$: global minimum

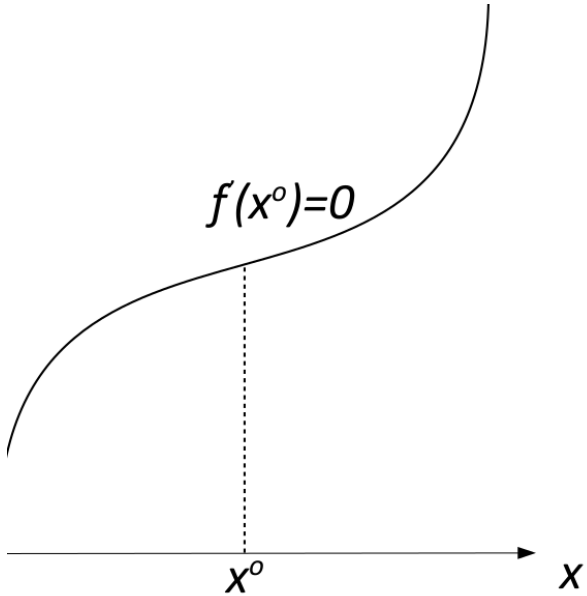
$f'(x_3) = 0$: local minimum

Do not worry about x_2 (local maximum)

In many practical problems, local minimum is OK

Because it works well in terms of test error

Saddle Points



$f(x) = x^3$

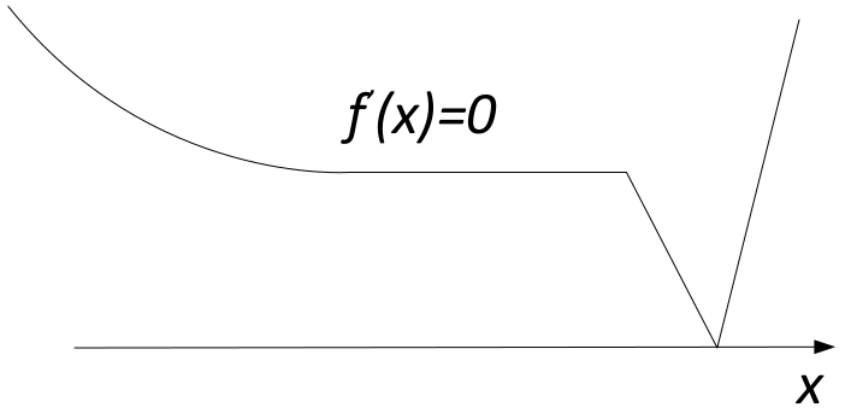
We can avoid saddle points

Momentum methods: by the time gradient becomes small

We have some velocity and can skip the saddle point

This point is not stable

Flat Regions "Plateau Regions"



It is not easy to address it

We can modify stopping criterion

Large number of iterations

Check if $R_N(\mathbf{w})$ is small enough

Outline

Gradient

Back propagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

Momentum Methods

Basic SGD ($B = 1$) $\mathbf{g}_t = \nabla \ell_j(\mathbf{w}_t)$ where $j \sim \text{unif}\{1, \ldots, N\}$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{g}_t$$

SGD with Momentum

$$\mathbf{g}_t = \nabla \ell_j(\mathbf{w}_t), \quad \mathbf{v}_1 = \mathbf{g}_1$$

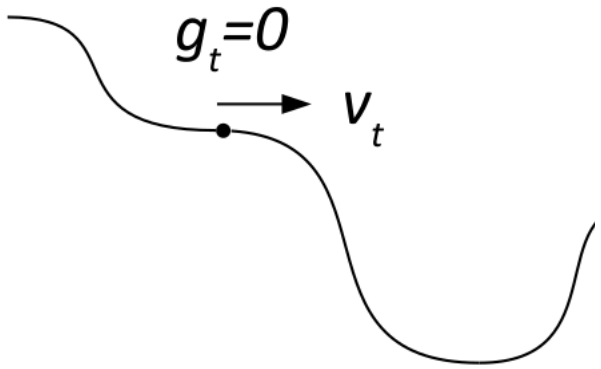
Velocity term: $\mathbf{v}_t = -\alpha \mathbf{g}_t + \mu \mathbf{v}_{t-1}$

- μ: momentum parameter
- Typically $\mu = 0.5, 0.9, 0.99$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

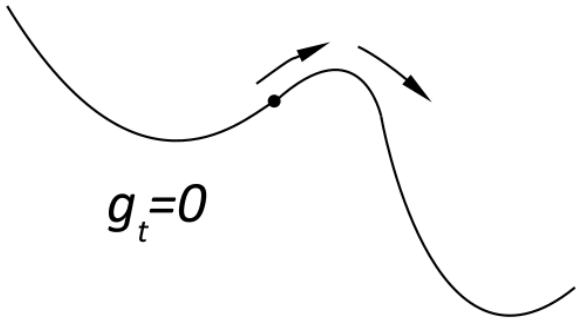
SGD with momentum keeps history of past gradients

SGD with Momentum



We can skip saddle points

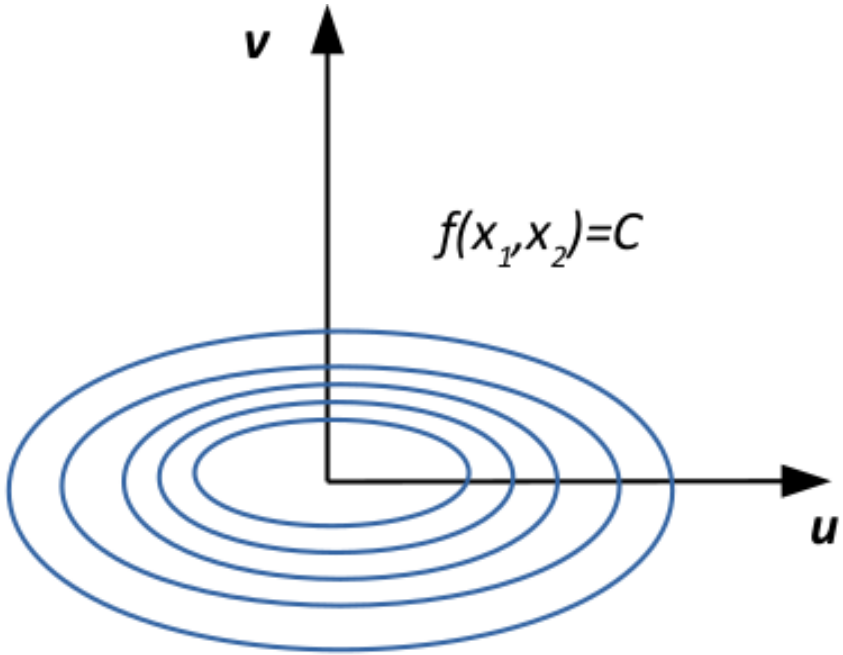
The velocity due to momentum pushes us forward



Shallow local minimum can be skipped

Uncontrolled momentum may result in overshooting effect

Momentum [Polyak, 1964]



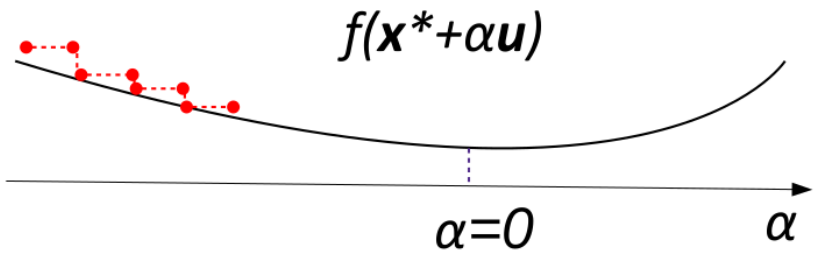
Polyak showed momentum leads to faster convergence for convex functions

No worries about saddle points/local minima in convex case

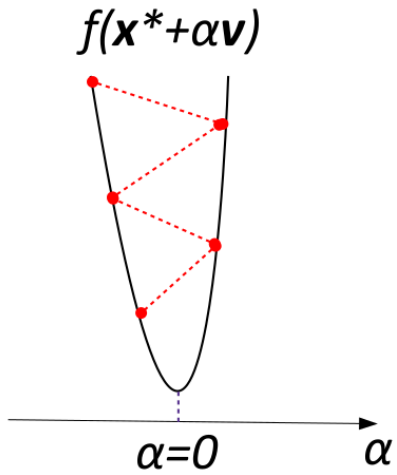
Consider a skewed convex function as above

Elliptical level sets where $f(x_1, x_2) = C$ for a given C

Function with Asymmetric Curvature

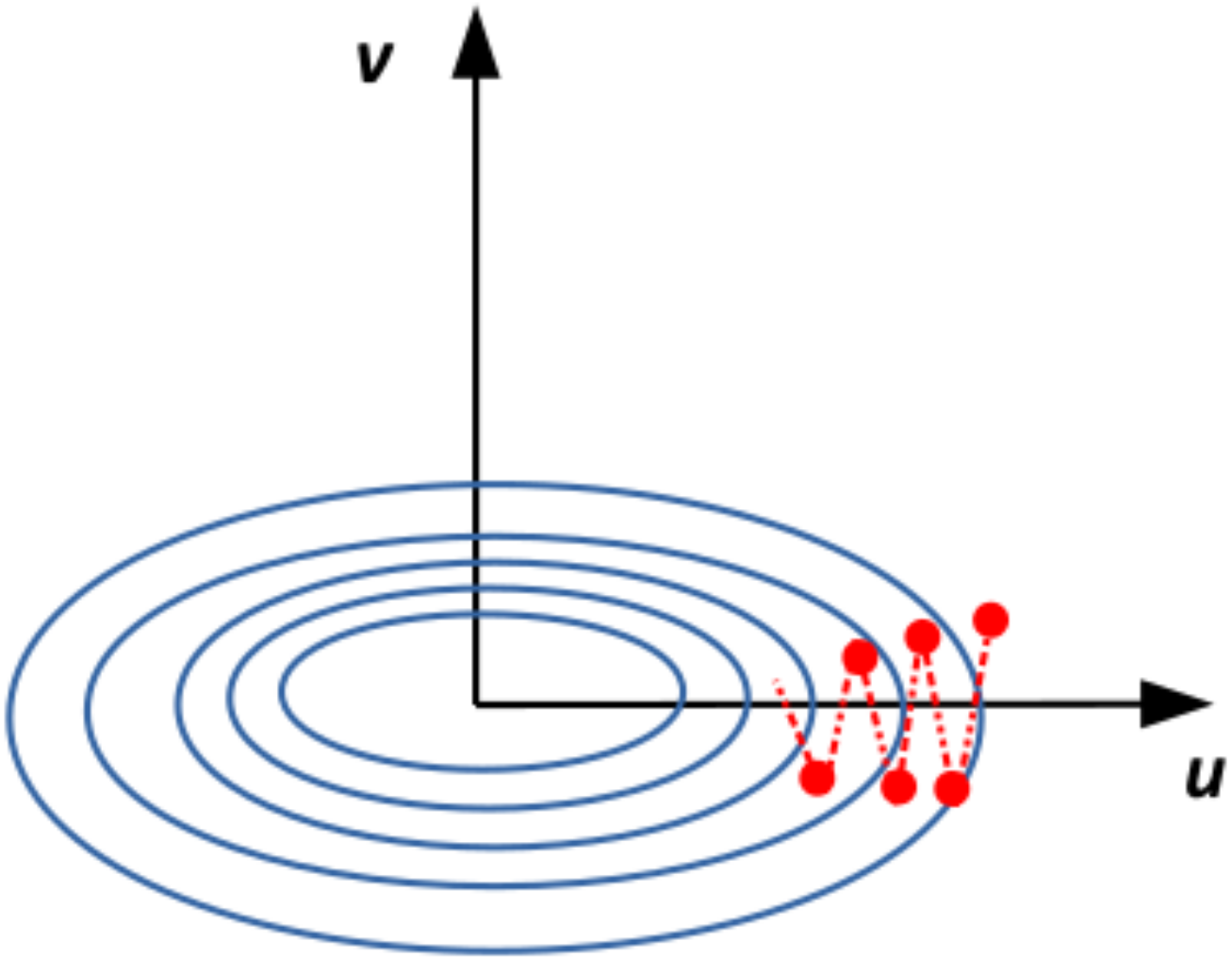


Little progress along u direction



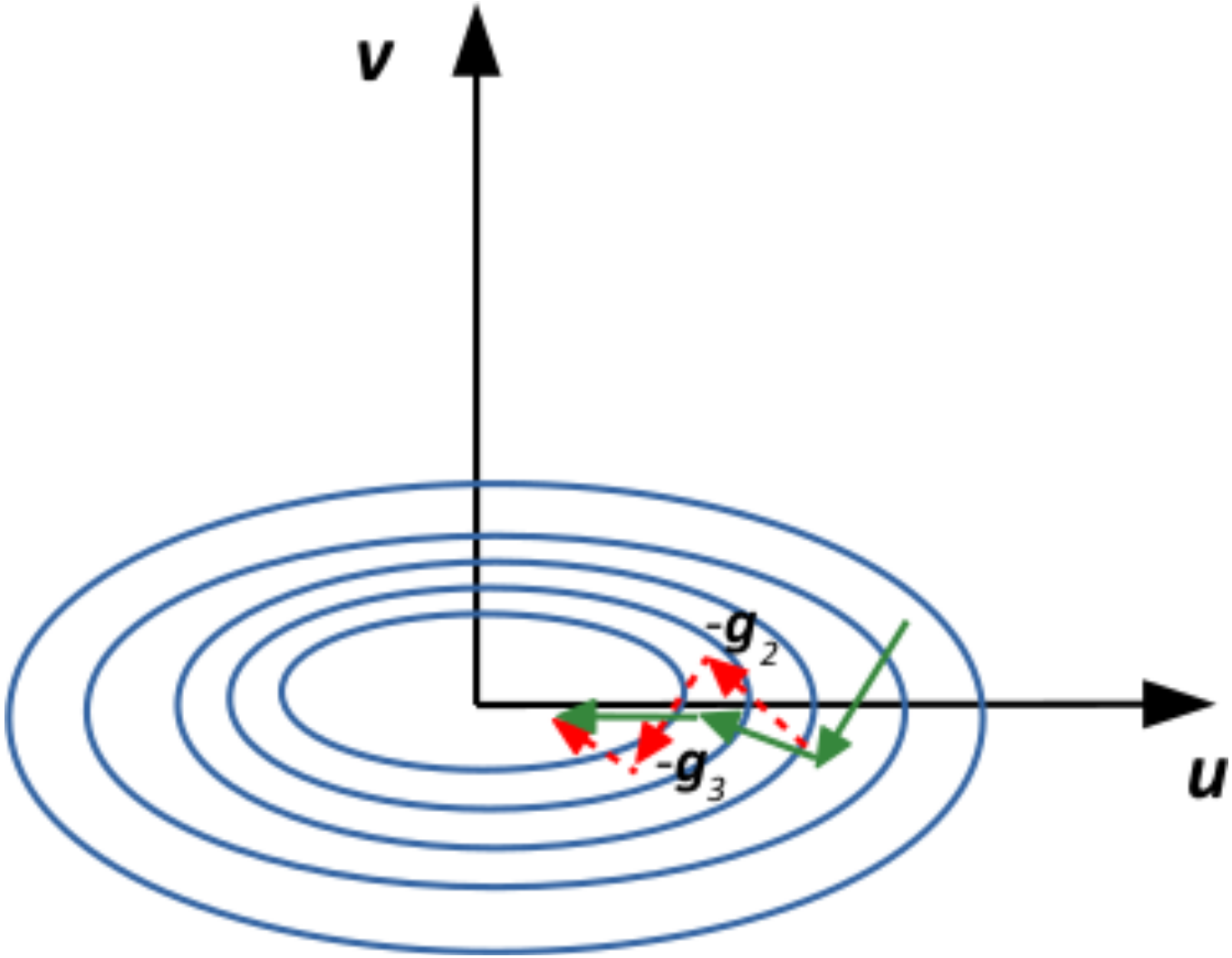
Overshoot along v direction

SGD without Momentum



Convergence is slow

SGD with Momentum



Gradient and velocity are combined and we get a smooth way

Convergence is fast

SGD and Momentum Summary

$-\nabla f(\mathbf{x})$: direction that leads to steepest descent

SGD is awesome in terms of computational complexity

- We should control variance in gradient updates

For convex functions, $\nabla R_N(\mathbf{w}) \approx 0 \rightarrow$ Global Minimum

Typical issues in nonconvex optimization:

- Local minima
- Saddle points
- Flat regions

Skip saddle points and shallow local minima using momentum

Outline

Gradient

Back propagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

Adaptive Learning Rate and Ada Grad [Duchi et al., JMLR11]

Curvature varies across coordinates

Ada Grad

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \nabla R_N(\mathbf{w}_t) \odot \nabla R_N(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla R_N(\mathbf{w}_t) \odot \frac{1}{\sqrt{\mathbf{c}_t + 10^{-6}}}$$

Keep track of running sum of squares of each coordiante

$$\mathbf{c}_t(j) = \sum_{j=1}^{t} \Big(\nabla R_N(\mathbf{w}_t)(j) \Big)^2 \quad j = 1, \ldots, d$$

Small learning rate in directions that function is very steep

After few iteration, $\mathbf{c}_t$ may grow very large!

RMSProp and ADAM [Knigma and Ba, ICLR15]

RMSProp

$$\mathbf{c}_t = \beta \mathbf{c}_{t-1} + (1-\beta) \nabla R_N(\mathbf{w}_t) \odot \nabla R_N(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla R_N(\mathbf{w}_t) \odot \frac{1}{\sqrt{\mathbf{c}_t + 10^{-6}}}$$

Scale down previous contributions by β

ADAM: RMSProp + Momentum

$$\mathbf{c}_t = \beta \mathbf{c}_{t-1} + (1-\beta) \nabla R_N(\mathbf{w}_t) \odot \nabla R_N(\mathbf{w}_t)$$

$$\mathbf{v}_t = \mu \mathbf{v}_{t-1} - \alpha \nabla R_N(\mathbf{w}_t) \odot \frac{1}{\sqrt{\mathbf{c}_t + 10^{-6}}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from data*. AMLBook New York, 2012.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.