UNIVERSITY
OF OSLO



IN4310 Back Propagation and Optimization

Tobias Müller, slides adapted from Ali Ramezani-Kebrya

ali@uio.no, tobiasmu@ifi.uio.no



Mandatory Exercise 1

Mandatory Exercise 1 is posted

Deadline: Friday, March 20, 11:59

If you need support: Attend the group sessions on Thursdays

Extensions up to 4 days possible, contact Ali or Artur via email or Mattermost

Learning Goals

Empirical risk minimization

Directional derivatives and gradient

Back propagation for neural networks

Stochastic gradient descent and mini batching

SGD with momentum

Variations of SGD

Outline

Empirical risk minimization

Gradient

Backpropagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

Reminder: Empirical Risk Minimization (ERM)

Our setting:

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

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

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

Goal for today: Solve this optimization problem.

$R_n(\theta)$ is a function of the parameters.

Parameters

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

Reminder: Empirical Risk Minimization (ERM)

Our setting:

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

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

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

Goal for today: Solve this optimization problem.

$R_n(\boldsymbol{\theta})$ is a function of the parameters.

Parameters

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

Outline

Empirical risk minimization

Gradient

Backpropagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

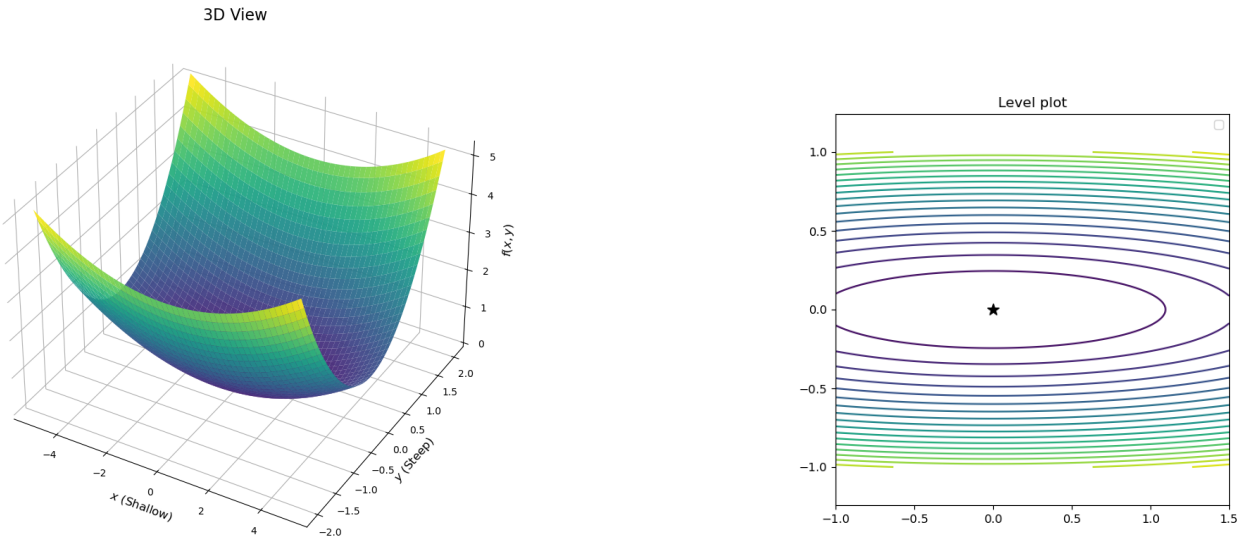
Adaptive variants of SGD

Gradient Descent

Abstract optimization problem:

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

We want to find x^* with $f(x^*) = \min_{x \in \mathbb{R}^d} f(x)$



Example function $f(x_1, x_2) = \alpha x_1^2 + \beta x_2^2$. Level plot on the right.

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 points into the direction of the steepest ascent

Gradient Descent for Training

Use this idea iteratively to always "go" into the direction of the steepest descent.

Gradient Descent Framework

Initialize $\boldsymbol{\theta}_0$ and a learning rate schedule $(\alpha_t)_{t\in\mathbb{N}} \subset \mathbb{R}$

For $t = 0, 1, 2, \ldots$, until some stopping criterion reached

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

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

Stopping criteria:

- Fixed number of iterations
- Change in training loss falls under some threshold
- $\|\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)\| \approx 0$

Matrix Calculus: Gradient

GOAL: We want to compute $\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)$ as efficiently as possible.

Taking gradients is a linear operation, therefore

$$\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) \right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i).$$

We need the gradients for each individual sample (x_i, y_i).

Matrix Calculus: Gradient

GOAL: We want to compute $\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)$ as efficiently as possible.

Taking gradients is a linear operation, therefore

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t) &= \nabla_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) \right) \\ &= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i).\end{aligned}$$

We need the gradients for each individual sample (x_i, y_i).

Matrix Calculus: Gradient

GOAL: We want to compute $\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t)$ as efficiently as possible.

Taking gradients is a linear operation, therefore

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} R_N(\boldsymbol{\theta}_t) &= \nabla_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) \right) \\ &= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i).\end{aligned}$$

We need the gradients for each individual sample (x_i, y_i).

Outline

Empirical risk minimization

Gradient

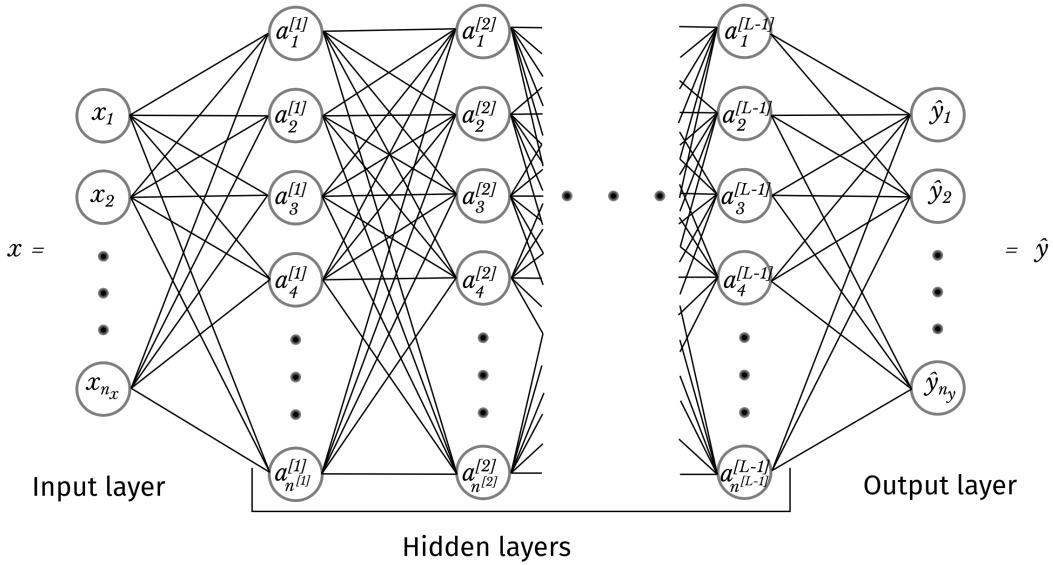
Backpropagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

General Formulation of a Neural Network



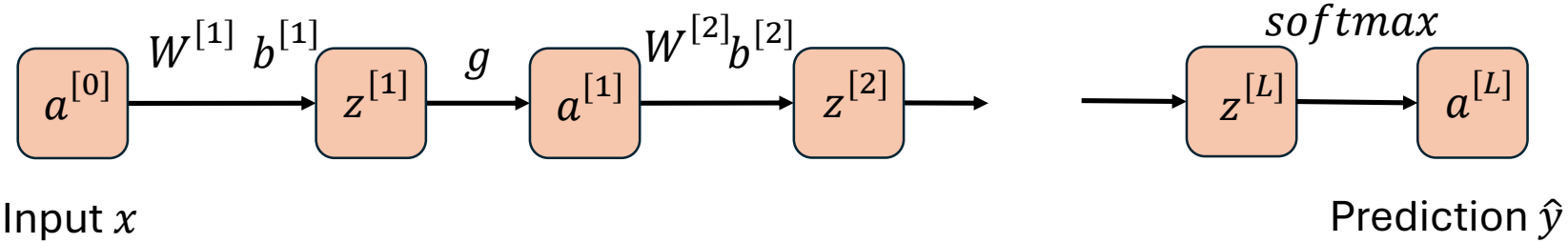
$$a_k^{[l]} = g\left(\mathbf{w}_k^{[l]} \cdot \mathbf{a}^{[l-1]} + b_k^{[l]}\right) = g(z_k^{[l]})$$

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]



Forward pass is computed by

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

for $\ell < L$, and

$$\mathbf{a}^{[L]} = softmax(\mathbf{z}^{[l]})$$

Why do we need backpropagation?

Naive method to compute the gradient:

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

Using the new model parameters $\tilde{\boldsymbol{\theta}}$ compute

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

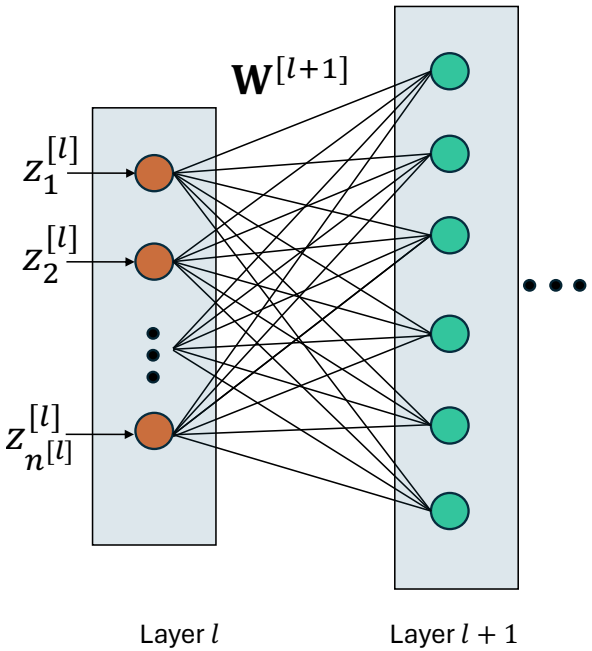
For each $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{j,i}^{[l]}}$ at each iteration, we need $\Theta(d)$ elementary operations

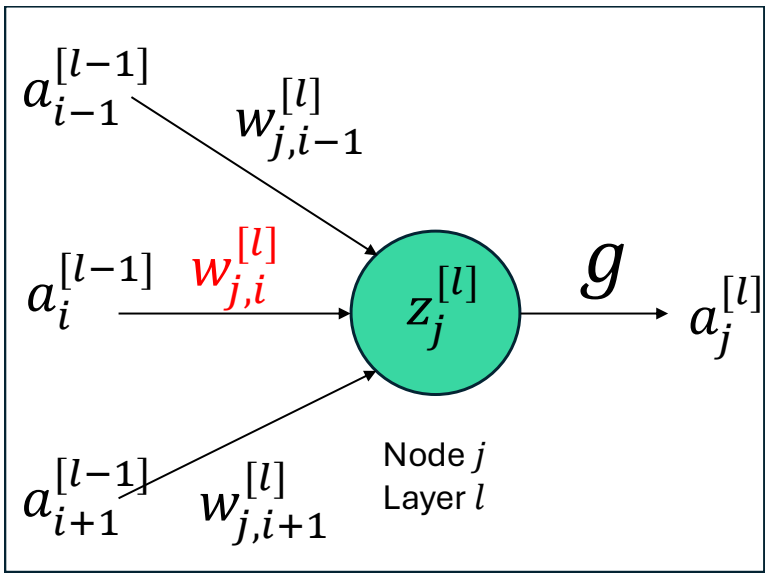
Total complexity per iteration $\Theta(d^2)$ too slow

Use Backpropagation Algorithm [Rumelhart et al., Nature1986]

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

Backpropagation for Neural Networks





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

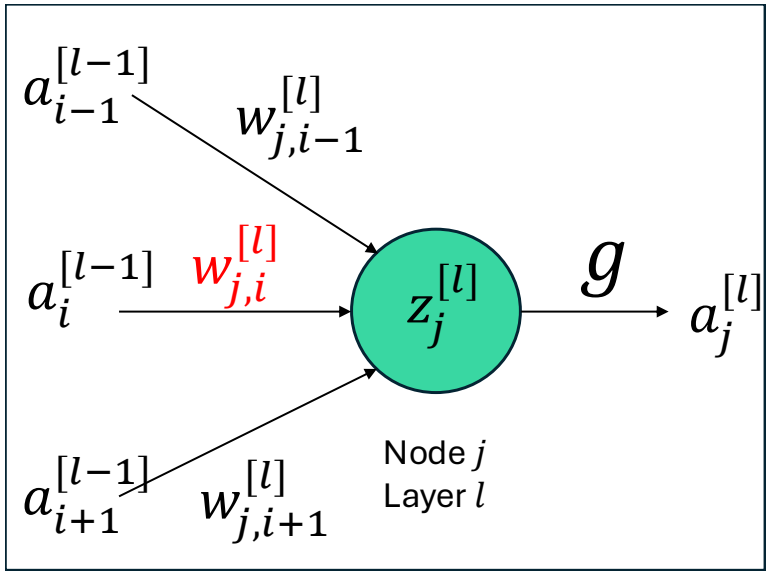
Given $z_1^{[l]}, \ldots, z_{n^{[l]}}^{[l]}$, no need to previous layers' to compute what follows

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

Derivative with Respect to Weights

$w_{j,i}^{[l]}$ only affects $z_j^{[l]}$. Therefore, the chain rule tells us that

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



Compute the derivative:

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

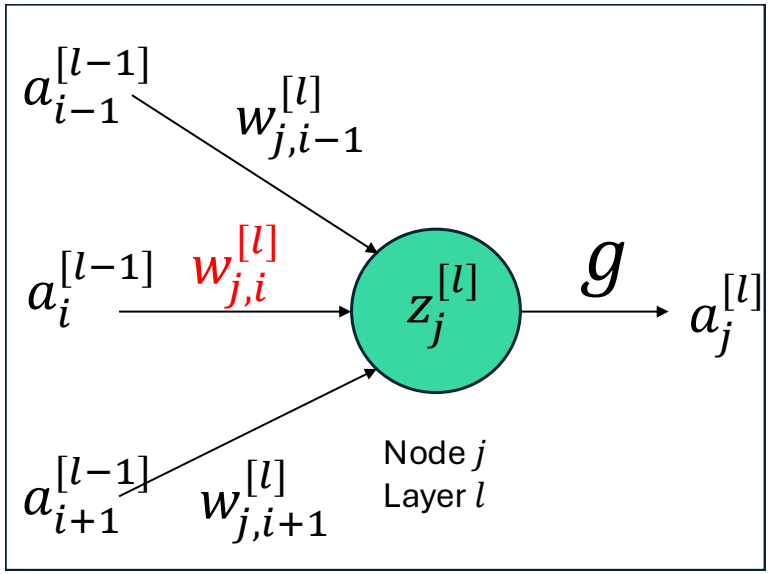
Define $\delta_j^{[l]} := \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}}$ then $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial w_{j,i}^{[l]}} = a_i^{[l-1]} \delta_j^{[l]}$

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

Derivative with Respect to Biases

$b_j^{[l]}$ only affects $z_j^{[l]}$. Therefore, the chain rule tells us that

$$\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial b_j^{[l]}} = \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}}$$



Compute the derivative:

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

As before $\delta_j^{[l]} := \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial z_j^{[l]}}$ then $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial b_j^{[l]}} = \delta_j^{[l]}$

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

How to Compute $\delta_i^{[L]}$: Backward Message for Output Layer

The error signal $\delta_j^{[L]}$ is computed by the chain rule:

$$\delta_j^{[L]} = \frac{\partial \ell_n}{\partial z_j^{[L]}} = \underbrace{\frac{\partial \ell_n}{\partial a_j^{[L]}}}_{\text{Loss Deriv.}} \cdot \underbrace{\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}}}_{\text{Activ. Deriv.}}$$

Regression (Quadratic)

Uses MSE loss and identity activation.

$$\begin{aligned} \ell_n &= (\hat{y}_n - y_n)^2 \\ a_j^{[L]} &= z_j^{[L]} \\ \implies \delta_j^{[L]} &= 2(a_j^{[L]} - y_n) \end{aligned}$$

Classification (Softmax)

Uses Cross-Entropy and Softmax.

$$\begin{aligned} \ell_n &= -\sum_{k=1}^{n^{[L]}} y_k \log(\hat{y}_k) \\ a_k^{[L]} &= \text{softmax}(z^{[L]})_k \end{aligned}$$

Will show: $\delta_j^{[L]} = a_j^{[L]} - y_j$

Derivation of $\delta^{[L]}$ for Classification with CE Loss and Softmax

Loss and softmax are

$$\ell_n(a^{[L]}) = -\sum_{k=1}^{n^{[L]}} y_k \log(a_k^{[L]}), \qquad a_k^{[L]} = \text{softmax}(z^{[L]})_k = \frac{e^{z_k^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_k^{[L]}}}$$

We get

$$\begin{aligned}
\ell_n(z^{[L]}) &= -\sum_{k=1}^{n^{[L]}} y_k \log\left(\frac{e^{z_k^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}}\right) \\
&= -\sum_{k=1}^{n^{[L]}} y_k \log\left(e^{z_k^{[L]}}\right) + \sum_{k=1}^{n^{[L]}} y_k \log\left(\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}\right) \\
&= -\sum_{k=1}^{n^{[L]}} y_k z_k^{[L]} + \log\left(\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}\right)
\end{aligned}$$

Taking the derivative with respect to $z_j^{[L]}$

$$\delta_j^{[L]} = \frac{\partial \ell_n}{\partial z_j^{[L]}} = -y_j + \frac{e^{z_j^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}} = a_j^{[L]} - y_j$$

Derivation of $\delta^{[L]}$ for Classification with CE Loss and Softmax

Loss and softmax are

$$\ell_n(a^{[L]}) = -\sum_{k=1}^{n^{[L]}} y_k \log(a_k^{[L]}), \qquad a_k^{[L]} = \text{softmax}(z^{[L]})_k = \frac{e^{z_k^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_k^{[L]}}}$$

We get

$$\begin{aligned}
\ell_n(z^{[L]}) &= -\sum_{k=1}^{n^{[L]}} y_k \log\left(\frac{e^{z_k^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}}\right) \\
&= -\sum_{k=1}^{n^{[L]}} y_k \log\left(e^{z_k^{[L]}}\right) + \sum_{k=1}^{n^{[L]}} y_k \log\left(\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}\right) \\
&= -\sum_{k=1}^{n^{[L]}} y_k z_k^{[L]} + \log\left(\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}\right)
\end{aligned}$$

Taking the derivative with respect to $z_j^{[L]}$

$$\delta_j^{[L]} = \frac{\partial \ell_n}{\partial z_j^{[L]}} = -y_j + \frac{e^{z_j^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}} = a_j^{[L]} - y_j$$

Derivation of $\delta^{[L]}$ for Classification with CE Loss and Softmax

Loss and softmax are

$$\ell_n(a^{[L]}) = -\sum_{k=1}^{n^{[L]}} y_k \log(a_k^{[L]}), \qquad a_k^{[L]} = \text{softmax}(z^{[L]})_k = \frac{e^{z_k^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_k^{[L]}}}$$

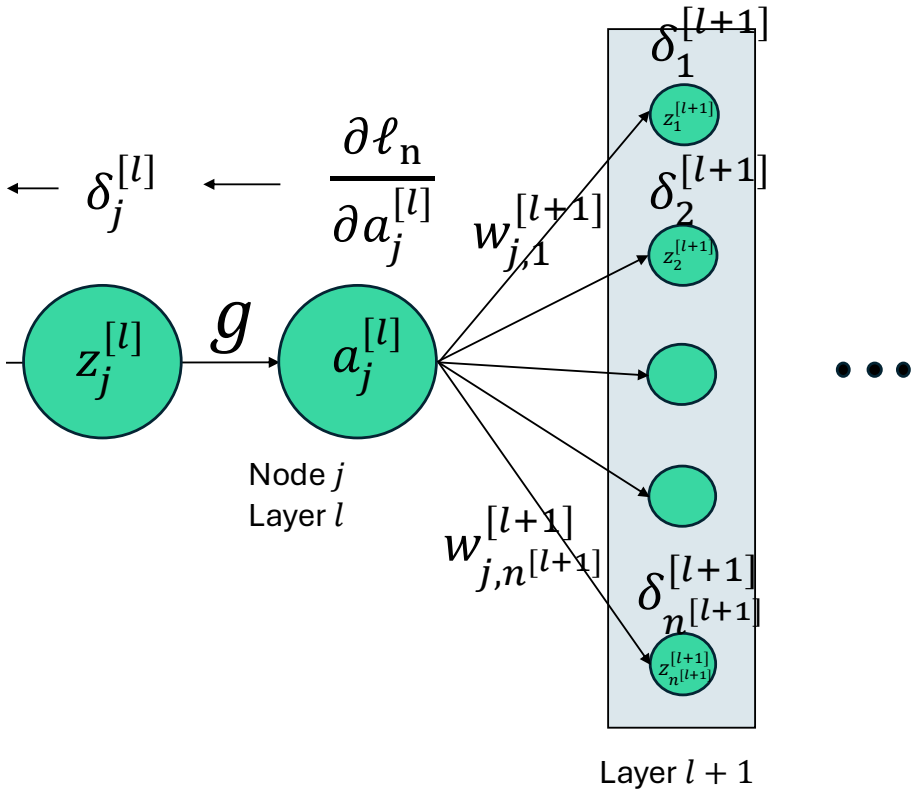
We get

$$\begin{aligned}
\ell_n(z^{[L]}) &= -\sum_{k=1}^{n^{[L]}} y_k \log\left(\frac{e^{z_k^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}}\right) \\
&= -\sum_{k=1}^{n^{[L]}} y_k \log\left(e^{z_k^{[L]}}\right) + \sum_{k=1}^{n^{[L]}} y_k \log\left(\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}\right) \\
&= -\sum_{k=1}^{n^{[L]}} y_k z_k^{[L]} + \log\left(\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}\right)
\end{aligned}$$

Taking the derivative with respect to $z_j^{[L]}$

$$\delta_j^{[L]} = \frac{\partial \ell_n}{\partial z_j^{[L]}} = -y_j + \frac{e^{z_j^{[L]}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L]}}} = a_j^{[L]} - y_j$$

Backward Pass



First step

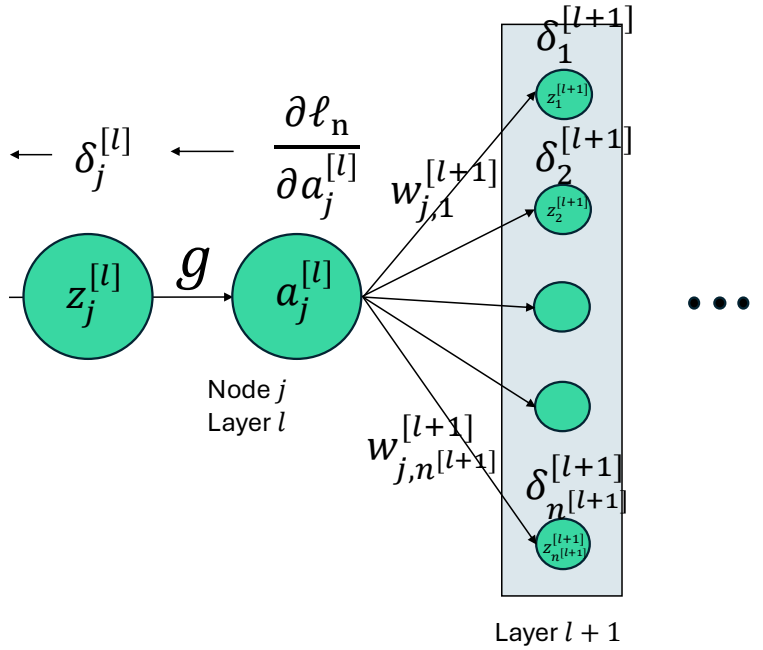
By chain rule

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

Backward Pass

We need $\frac{\partial \ell_n(\boldsymbol{\theta})}{\partial a_j^{[l]}}$. The chain rule tells us that

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



Compute the derivative:

$$z_k^{[l]} = \sum_{i=1}^{n^{[l]}} w_{k,i}^{[l]} a_i^{[l]} + b_j^{[l]}, \qquad \frac{\partial z_k^{[l]}}{\partial a_j^{[l]}} = w_{k,j}^{[l+1]}$$

Putting that in shows

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

Backward Pass: Completition

Put together all parts

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

In matrix form

$$\delta^{[l]} = W^{[l+1]T} \delta^{[l+1]} \odot g'(z^{[l]})$$

Overview of Backpropagation

Forward pass:

Start with $a^{[0]} = x$

Then iteratively

$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}, \quad a^{[l+1]} = g(z^{[l+1]})$$

Backward pass:

Start by computing $\delta^{[L]}$

Then iteratively

$$\begin{aligned} \delta^{[l]} &= W^{[l+1]T} \delta^{[l+1]} \odot g'(z^{[l]}) \\ \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial b^{[l]}} &= \delta^{[l]} \\ \frac{\partial \ell_n(\boldsymbol{\theta})}{\partial W^{[l]}} &= \delta^{[l]} a^{[l-1]T} \end{aligned}$$

Outline

Empirical risk minimization

Gradient

Backpropagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

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

SGD for Logistic Regression

Logistic regression problem: function $h_w(x) = w^T x$, labels $y \in \{-1, 1\}$
Loss function $l(f(x), y) = \log(1 + e^{-yf(x)}) = \log(1 + e^{-yw^T x})$

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

Recall the update rule of Perceptron algorithm

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

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 $\mathbf{w}_t$:

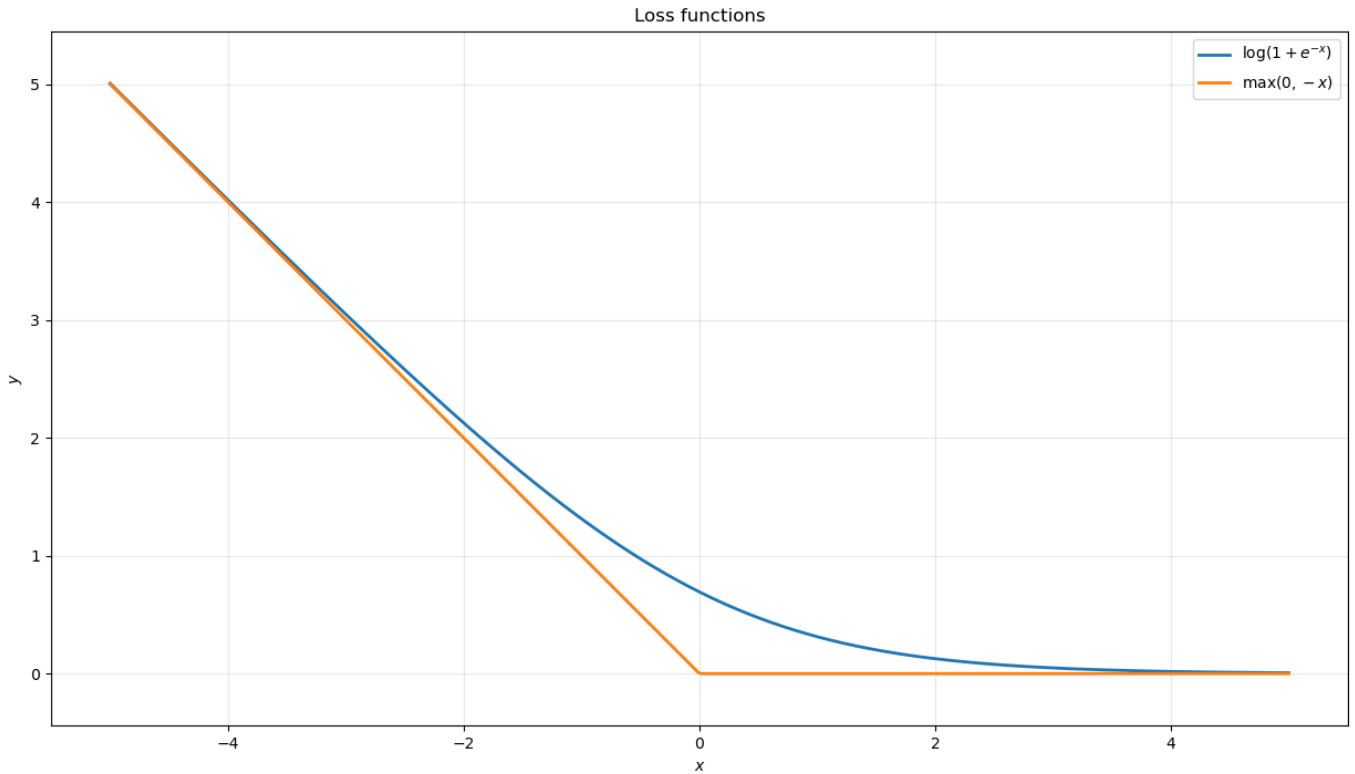
- $y_j = \text{Sign}(\mathbf{w}_t^\top \mathbf{x}_j)$, i.e., $y_j \mathbf{w}_t^\top \mathbf{x}_j > 0$
- $1/\left(1 + e^{y_j \mathbf{w}_t^\top \mathbf{x}_j}\right) \approx 0$
- Update rule for SGD: $\mathbf{w}_{t+1} = \mathbf{w}_t$

What is the Difference Between Them?

Choice of the loss function!

Logistic regression: $l(w^T x, y) = \log(1 + e^{-w^T x})$

Perceptron: $l(w^T x, y) = \max(0, -w^T x)$



Drawback of SGD

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 not one sample, but a mini batch of 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 $\mathcal{S}$ of size B
Let $\mathcal{S} = \{j_1, \ldots, j_B\}$ denote the indices of selected samples
Compute $\nabla_{\mathcal{S}} R(\boldsymbol{\theta}) := 1/B \sum_{i \in \mathcal{S}} \nabla \ell_i(\boldsymbol{\theta})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\mathcal{S}} R(\boldsymbol{\theta})$

Check if stopping condition (e.g., $\nabla R_{\mathcal{S}}(\boldsymbol{\theta}) \approx 0$) 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., $\nabla R_{\mathcal{S}}(\boldsymbol{\theta}) \approx 0$, is good for convex loss

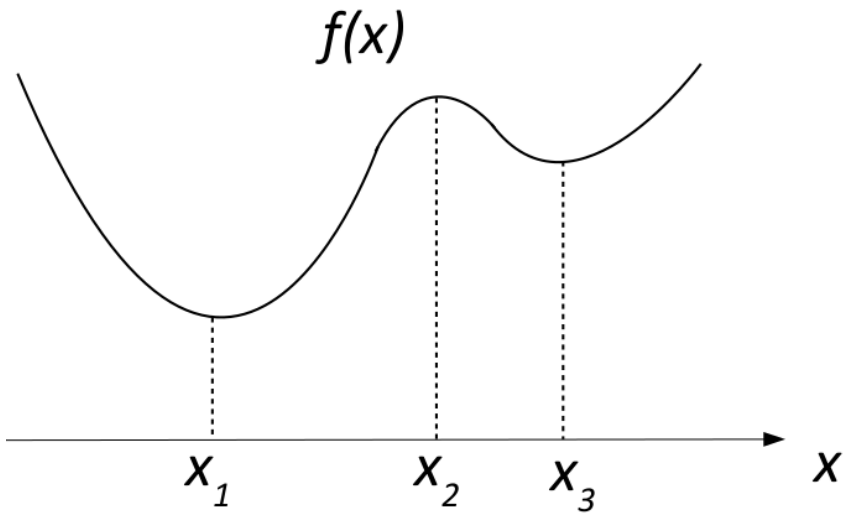
- Linear Regression
- Logistic Regression

For convex functions, satisfying $\nabla R_{\mathcal{S}}(\boldsymbol{\theta}) \approx 0 \rightarrow$ Global Minimum

No global guarantees for nonconvex functions, typical issues:

- Local minima
- Saddle points
- Flat regions

Local Minima



x_1: global minimum

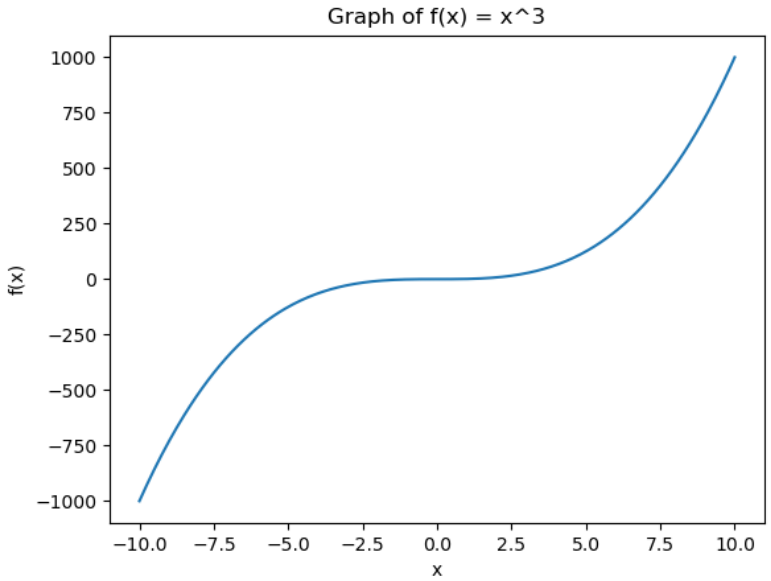
x_3: local minimum

Do not worry about x_2 (local maximum)

In many practical problems, local minimum is OK

Because it tends to work well in terms of test error

Saddle Points



$f(x) = x^3$

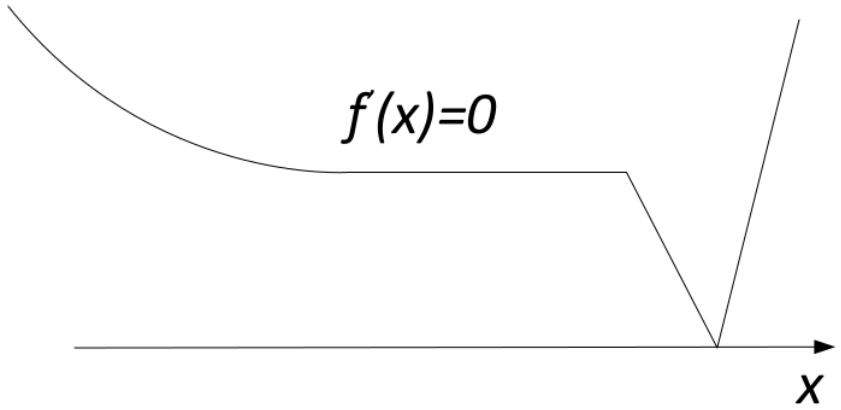
We can avoid saddle points

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 $R_N(\boldsymbol{\theta})$ is small enough

Outline

Empirical risk minimization

Gradient

Backpropagation for neural networks

Stochastic gradient descent and mini batching

SGD with Momentum

Adaptive variants of SGD

Momentum Methods

Basic SGD $\mathbf{g}_t = \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t)$

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

SGD with Momentum

$$\mathbf{g}_t = \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t), \quad \mathbf{v}_1 = \mathbf{g}_1$$

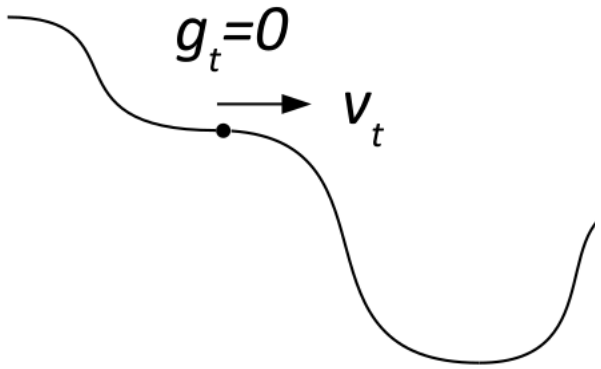
Velocity term: $\mathbf{v}_t = -\alpha \mathbf{g}_t + \mu \mathbf{v}_{t-1}$

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

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

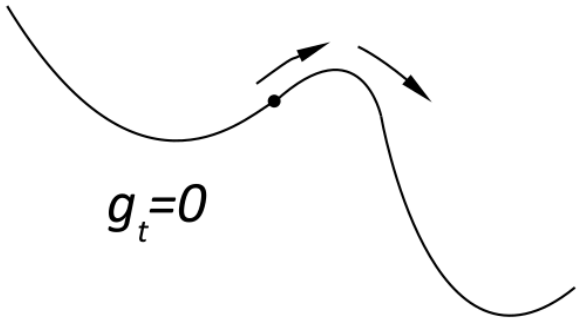
SGD with momentum keeps history of past gradients

SGD with Momentum



We can skip saddle points

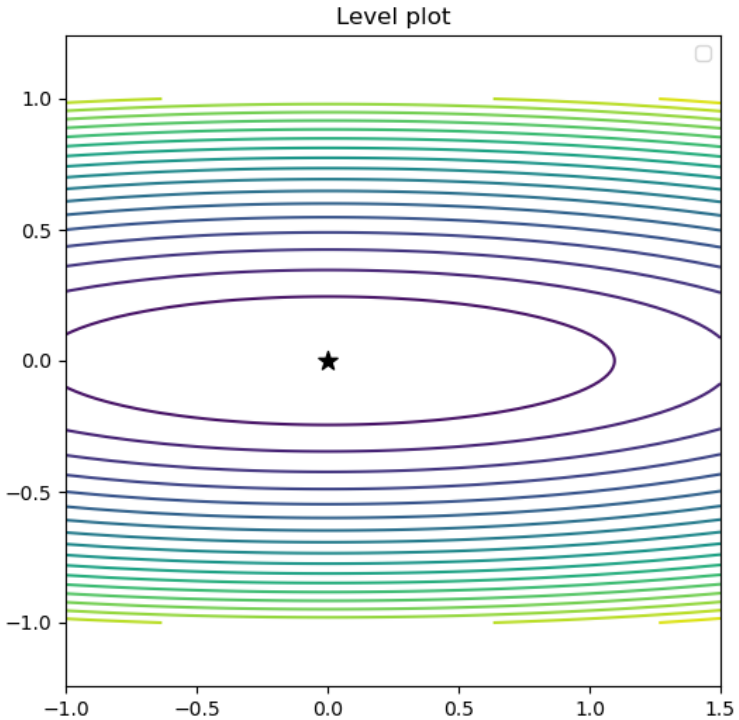
The velocity due to momentum pushes us forward



Shallow local minimum can be skipped

Uncontrolled momentum may result in overshooting effect

Momentum [Polyak, 1964]

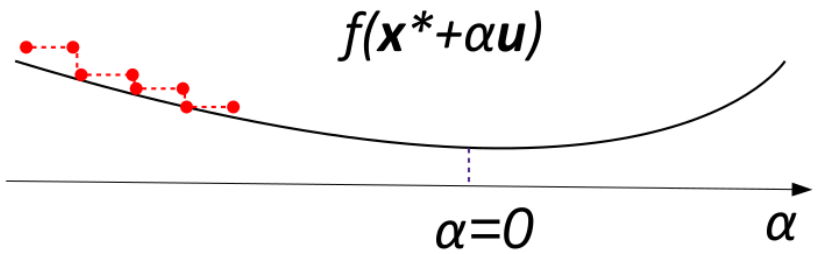


Polyak showed momentum leads to faster convergence for convex functions

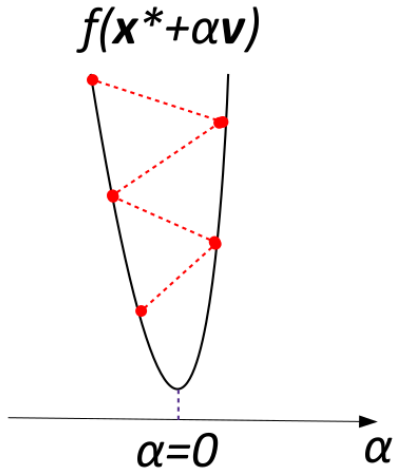
No worries about saddle points/local minima in convex case

Consider a skewed convex function as above

Function with Asymmetric Curvature

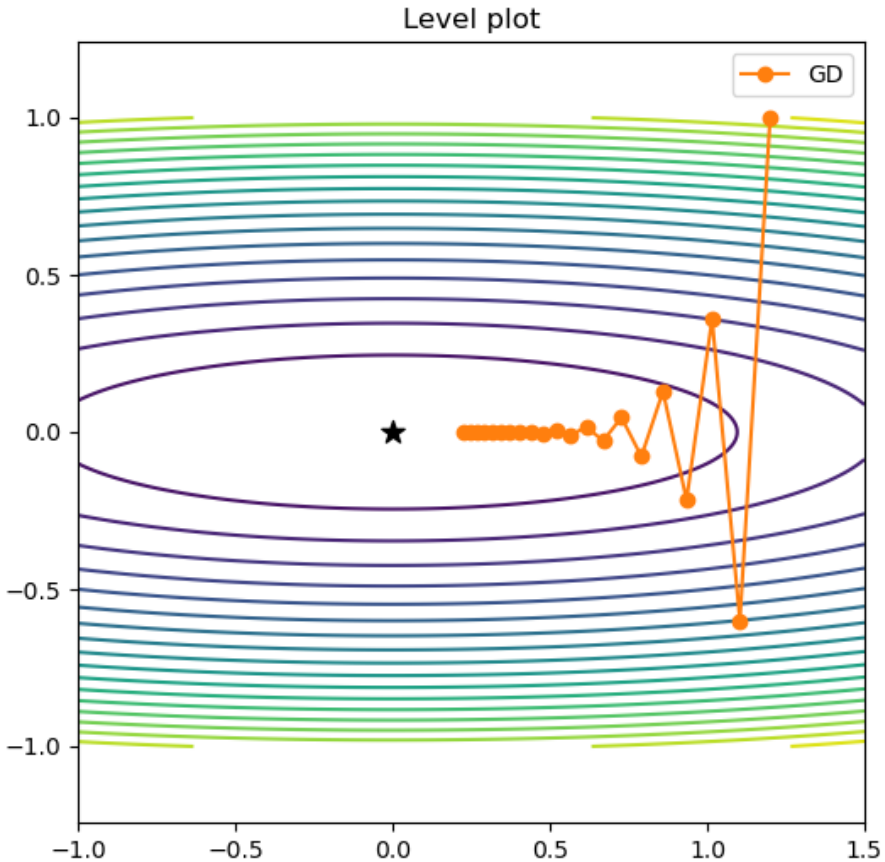


Little progress along u direction



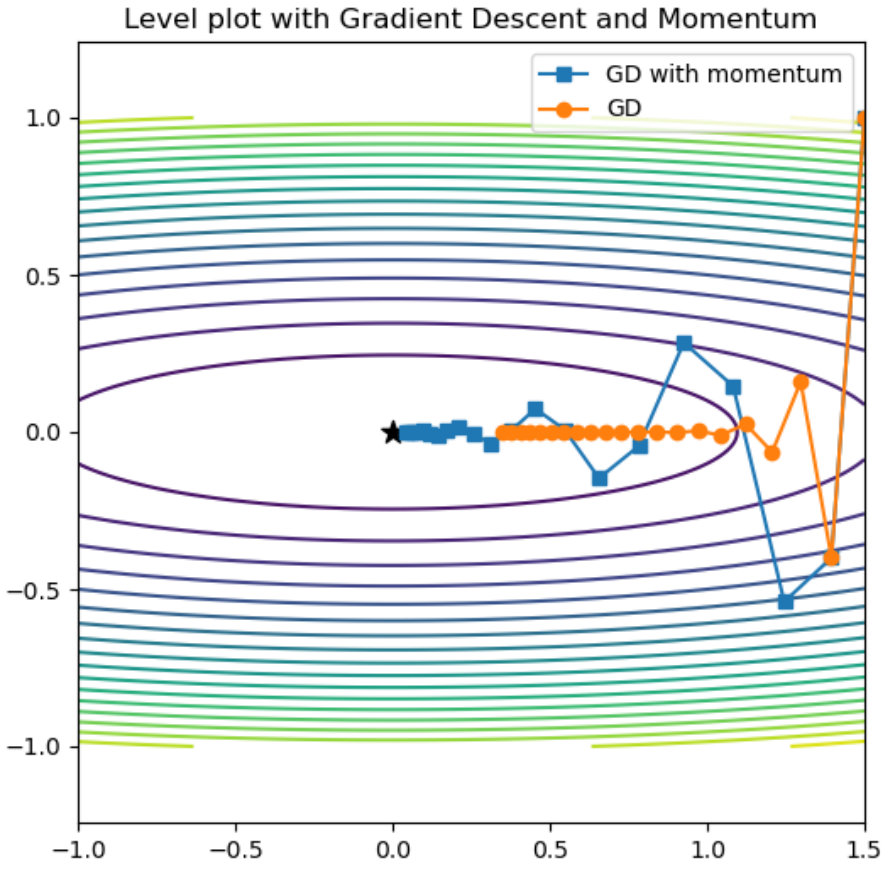
Overshoot along v direction

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, $\nabla R_N(\boldsymbol{\theta}) \approx 0 \rightarrow$ Global Minimum

Typical issues in nonconvex optimization:

- Local minima
- Saddle points
- Flat regions

Skip saddle points and shallow local minima using momentum

Outline

Empirical risk minimization

Gradient

Backpropagation 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

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t) \odot \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t) \odot \frac{1}{\sqrt{\mathbf{c}_t + 10^{-6}}}$$

Keep track of running sum of squares of each coordiante

$$\mathbf{c}_t(j) = \sum_{j=1}^{t} \Big(\nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t)(j) \Big)^2 \quad j = 1, \dots, d$$

Small learning rate in directions that function is very steep

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

RMSProp

$$\mathbf{c}_t = \beta \mathbf{c}_{t-1} + (1-\beta) \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t) \odot \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t) \odot \frac{1}{\sqrt{\mathbf{c}_t + 10^{-6}}}$$

Scale down previous contributions by β

ADAM [Kingma and Ba, ICLR15]

ADAM: RMSProp + Momentum

$$\mathbf{c}_t = \beta_1 \mathbf{c}_{t-1} + (1-\beta_1)\nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t) \odot \nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\nabla R_{\mathcal{S}}(\boldsymbol{\theta}_t)$$

$$\hat{\mathbf{c}}_t = \frac{\mathbf{c}_t}{1-\beta_1^t}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \hat{\mathbf{v}}_t \odot \frac{1}{\sqrt{\hat{\mathbf{c}}_t + 10^{-6}}}$$

Take moving averages of gradients ($\mathbf{v}_t$), and of squared gradients ($\mathbf{c}_t$)

$\hat{\mathbf{c}}_t$ and $\hat{\mathbf{v}}_t$ are bias-corrected scalings, ensure that early iterations work well.

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.