Linear regression

Supervised learning example

$$a(x) \rightarrow 13$$

Supervised learning

x_i — example

y_i — target value

$x_i = (x_{i1}, \dots, x_{id})$ — features

$X = \big((x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\big)$ — training set

$a(x)$ — model, hypothesis

$$x \longrightarrow a(x) \longrightarrow y^{pred}$$

Regression and classification

$y_i \in \mathbb{R}$ — regression task

- Salary prediction
- Movie rating prediction

Regression and classification

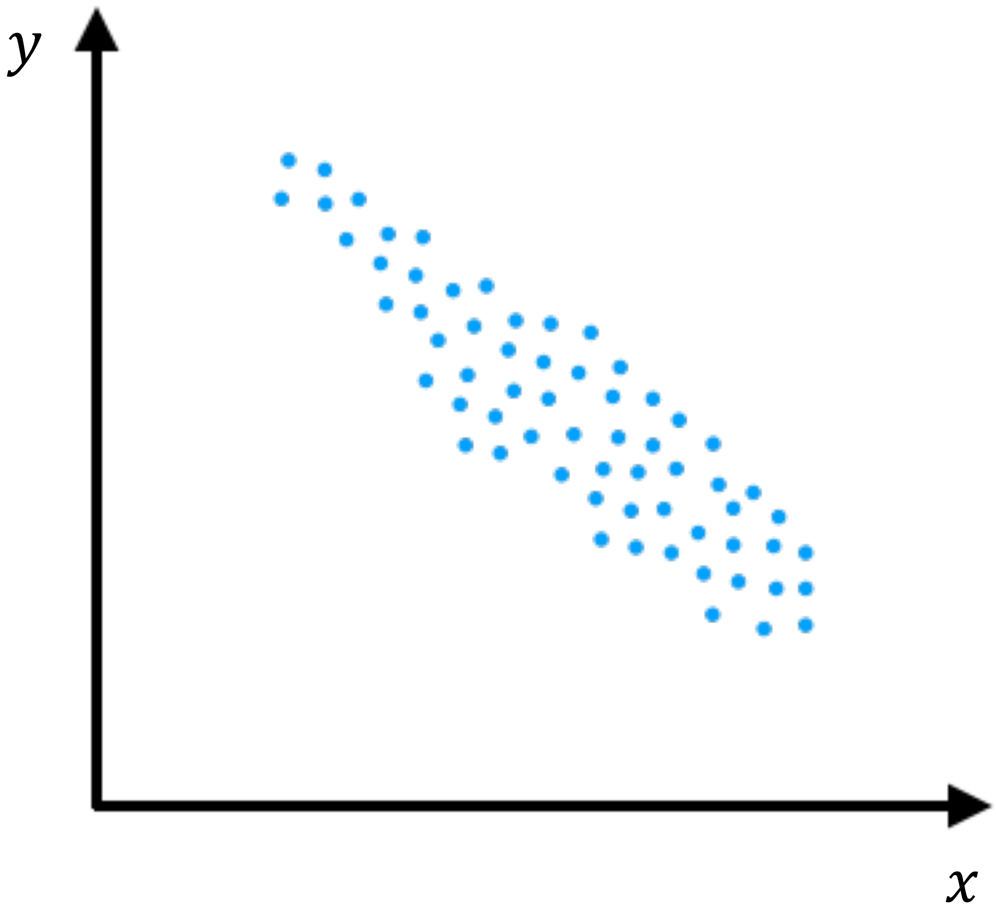
$y_i \in \mathbb{R}$ — regression task

- Salary prediction
- Movie rating prediction

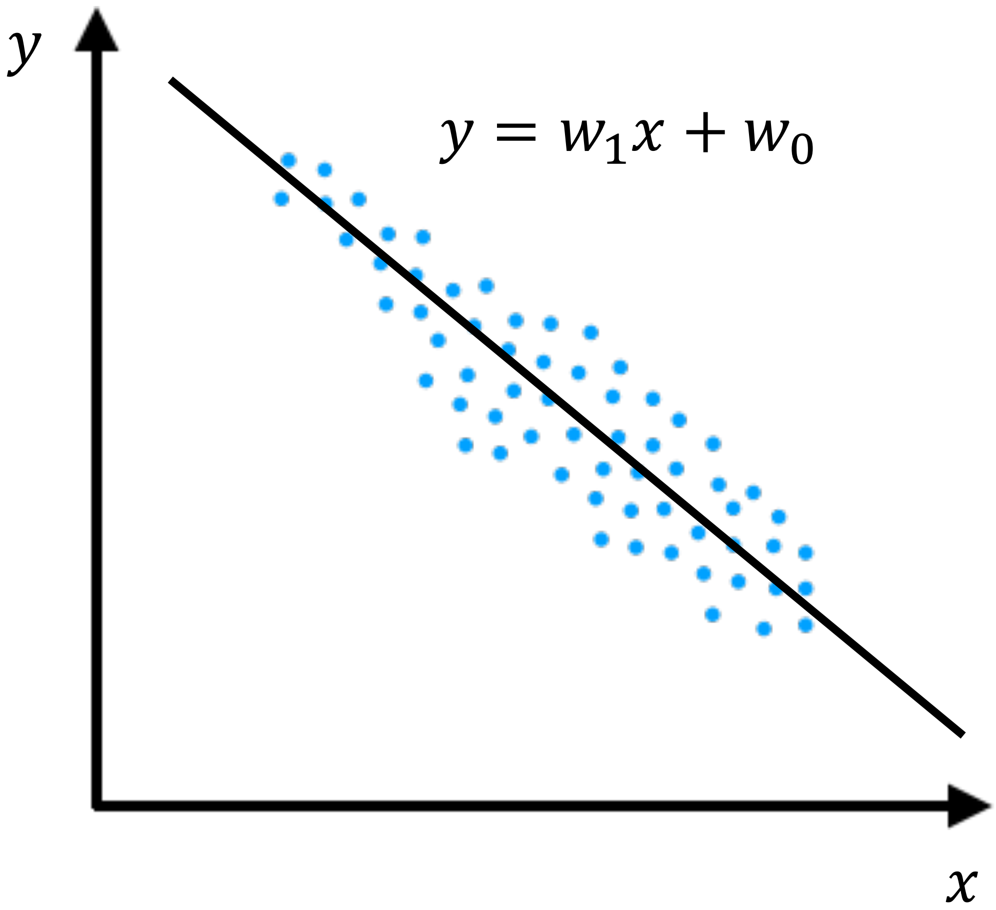
y_i belongs to a finite set — classification task

- Object recognition
- Topic classification

Linear model for regression example



Linear model for regression example



Linear model for regression

$$a(x) = b + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

- $w_1, \dots, w_d$ — coefficients (weights)
- b — bias
- $d + 1$ parameters
- To make it simple: there's always a constant feature

Linear model for regression

Vector notation:

$$a(x) = w^T x$$

For a sample X:

$$a(X) = Xw$$

$$X = \begin{pmatrix} x_{11} & \dots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{\ell 1} & \dots & x_{\ell d} \end{pmatrix}$$

Loss function

How to measure model quality?

Mean squared error:

$$L(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} (w^T x_i - y_i)^2$$

$$= \frac{1}{\ell} \|Xw - y\|^2$$

Training a model

Fitting a model to training data:

$$L(w) = \frac{1}{\ell} \|Xw - y\|^2 \to \min_w$$

Training a model

Fitting a model to training data:

$$L(w) = \frac{1}{\ell} \|Xw - y\|^2 \to \min_{w,}$$

Exact solution:

$$w = (X^T X)^{-1} X^T y$$

But inverting a matrix is hard for high-dimensional data!

Summary

- Linear models are very simple
- MSE can be used as a loss function
- There is an analytical solution, but we need more generic and scalable learning method

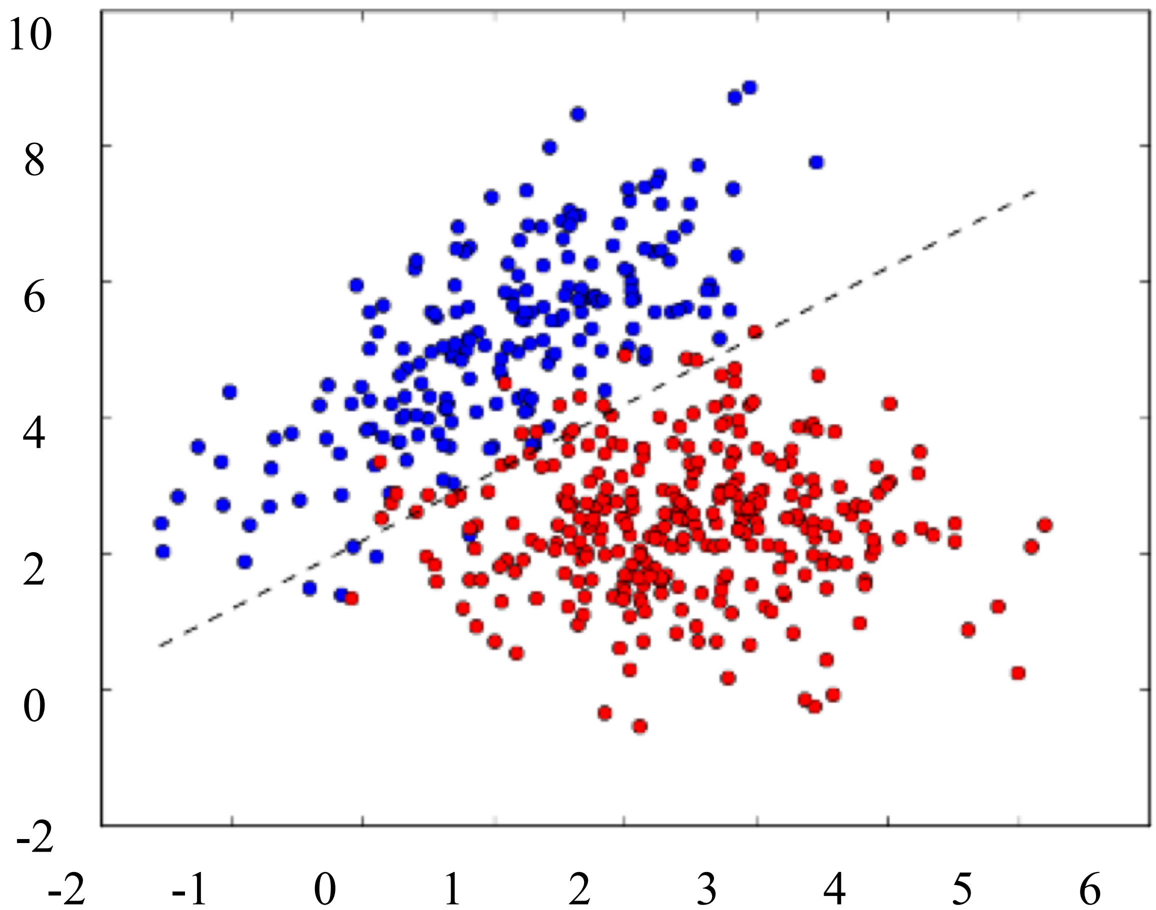
Linear model for classification

Binary classification ($y \in \{-1, 1\}$):

$$a(x) = \text{sign}(w^T x)$$

Number of parameters: d ($w \in \mathbb{R}^d$)

Linear model for classification example



Linear model for classification

Multi-class classification ($y \in \{1, \dots, K\}$):

$$a(x) = \arg \max_{k \in \{1, \dots, K\}} (w_k^T x)$$

Number of parameters: K*d ($w_k \in \mathbb{R}^d$)

Example:

$z = (7, -7.5, 10)$ — scores

$a(x) = 3$

Classification loss

Classification accuracy:

$$\frac{1}{\ell}\sum_{i=1}^{\ell}[a(x_i) = y_i]$$

- Not differentiable
- Doesn't assess model confidence

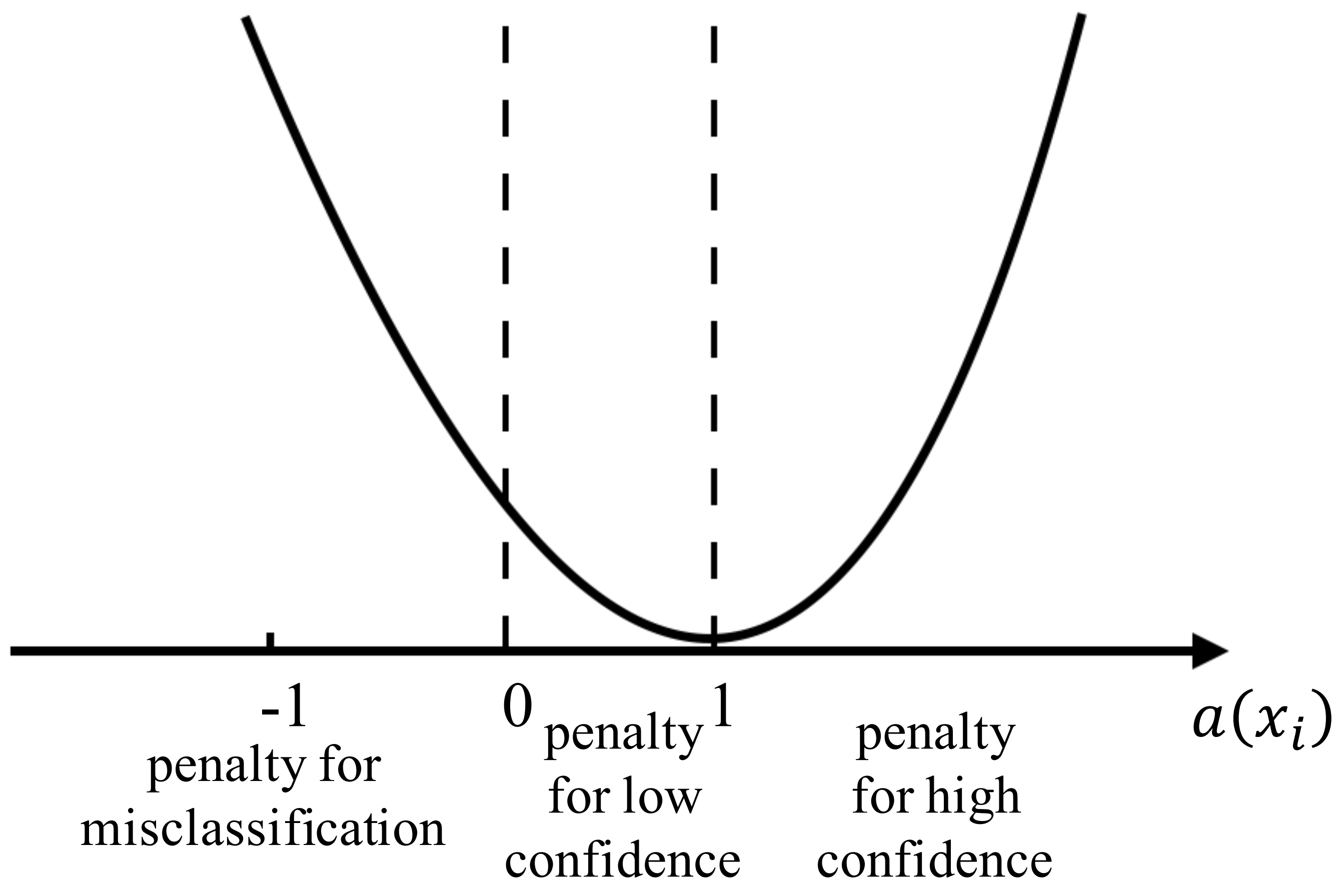
[P] — Iverson bracket:

$$[P] = \begin{cases} 1, & P \text{ is true} \\ 0, & P \text{ is false} \end{cases}$$

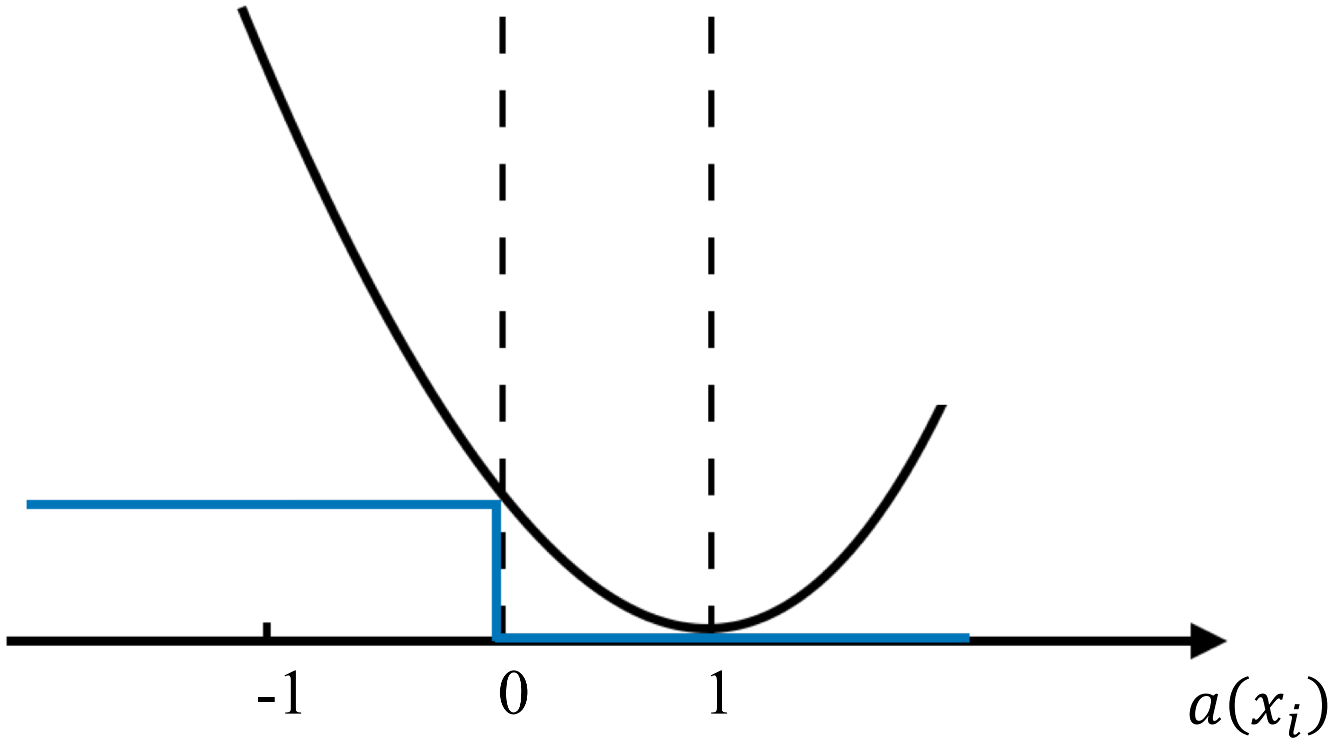
Classification loss

Consider an example x_i such that $y_i = 1$

Squared loss: $(w^T x_i - 1)^2$



Classification loss



Class probabilities

Class scores (**logits**) from a linear model:

$$z = (w_1^T x, \ldots, w_K^T x)$$

$$\downarrow$$

$$(e^{z_1}, \ldots, e^{z_K})$$

$$\downarrow$$

$$\sigma(z) = \left(\frac{e^{z_1}}{\sum_{k=1}^{K} e^{z_k}}, \ldots, \frac{e^{z_K}}{\sum_{k=1}^{K} e^{z_k}} \right)$$

(softmax transform)

Softmax

$$\sigma(z) = \left(\frac{e^{z_1}}{\sum_{k=1}^{K} e^{z_k}}, \ldots, \frac{e^{z_K}}{\sum_{k=1}^{K} e^{z_k}} \right)$$

Example:

$$z = (7, -7.5, 10)$$

$$\sigma(z) \approx (0.05, 0, 0.95)$$

Loss function

Predicted class probabilities (model output):

$$\sigma(z) = \left(\frac{e^{z_1}}{\sum_{k=1}^{K} e^{z_k}}, \dots, \frac{e^{z_K}}{\sum_{k=1}^{K} e^{z_k}}\right)$$

Target values for class probabilities:

$$p = ([y = 1], \dots, [y = K])$$

Similarity between z and p can be measured by the cross-entropy:

$$-\sum_{k=1}^{K} [y = k] \log \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} = -\log \frac{e^{z_y}}{\sum_{j=1}^{K} e^{z_j}}$$

Cross-entropy examples

Suppose $K = 3$ and $y = 1$:

- $-1 * \log 1 - 0 * \log 0 - 0 * \log 0 = 0$
- $-1 * \log 0.5 - 0 * \log 0.25 - 0 * \log 0.25 \approx 0.693$
- $-1 * \log 0 - 0 * \log 1 - 0 * \log 0 = +\infty$

Cross-entropy for classification

Cross-entropy is differentiable and can be used as a loss function:

$$L(w,b) = -\sum_{i=1}^{\ell}\sum_{k=1}^{K}[y_i = k]\log\frac{e^{w_k^T x_i}}{\sum_{j=1}^{K} e^{w_j^T x_i}}$$

$$= -\sum_{i=1}^{\ell}\log\frac{e^{w_{y_i}^T x_i}}{\sum_{j=1}^{K} e^{w_j^T x_i}} \to \min_w$$

Summary

- Linear models can be easily generalized for classification tasks
- There are lots of loss functions for classification
- Cross-entropy is one of the most popular

Loss functions

Linear regression and MSE:

$$L(w) = \frac{1}{\ell} \|Xw - y\|^2$$

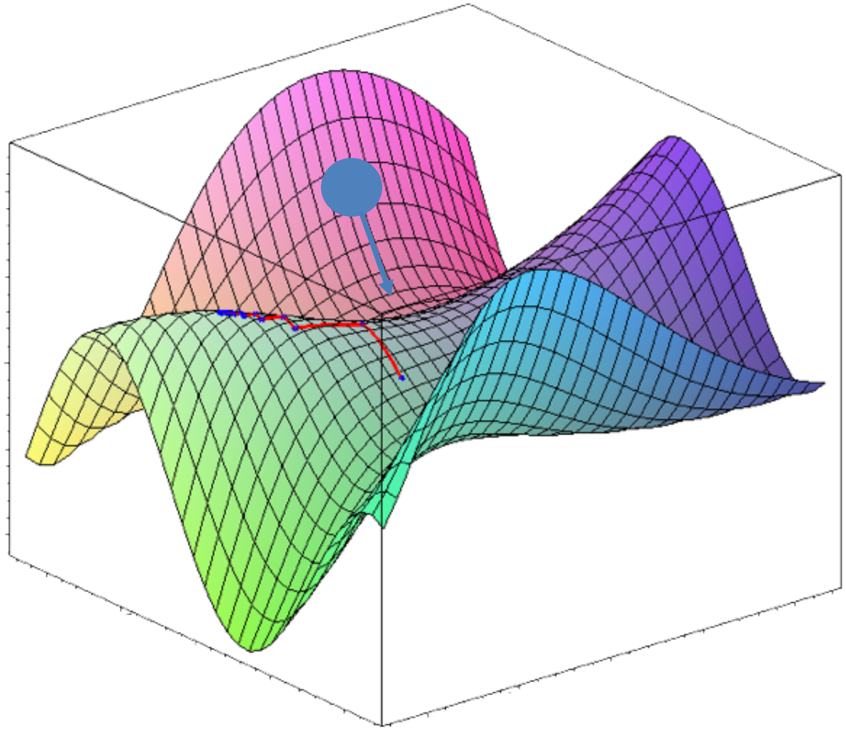
Linear classification and cross-entropy:

$$L(w) = -\sum_{i=1}^{\ell} \sum_{k=1}^{K} [y_i = k] \log \frac{e^{w_k^T x_i}}{\sum_{j=1}^{K} e^{w_j^T x_i}}$$

Gradient descent

Optimization problem: $L(w) \to \min_{w}$

Suppose we have some approximation w^0 — how to refine it?



Gradient descent

Optimization problem: $L(w) \to \min_{w}$

w^0 — initialization

$\nabla L(w^0) = \left(\frac{\partial L(w^0)}{\partial w_1}, \dots, \frac{\partial L(w^0)}{\partial w_n}\right)$ — gradient vector

- Points in the direction of the steepest slope at w^0
- The function has fastest decrease rate in the direction of negative gradient

Gradient descent

Optimization problem: $L(w) \to \min_{w}$

w^0 — initialization

$\nabla L(w^0) = \left(\frac{\partial L(w^0)}{\partial w_1}, \dots, \frac{\partial L(w^0)}{\partial w_n}\right)$ — gradient vector

$w^1 = w^0 - \eta_1 \nabla L(w^0)$ — gradient step

Gradient descent

Optimization problem: $L(w) \to \min_{w}$

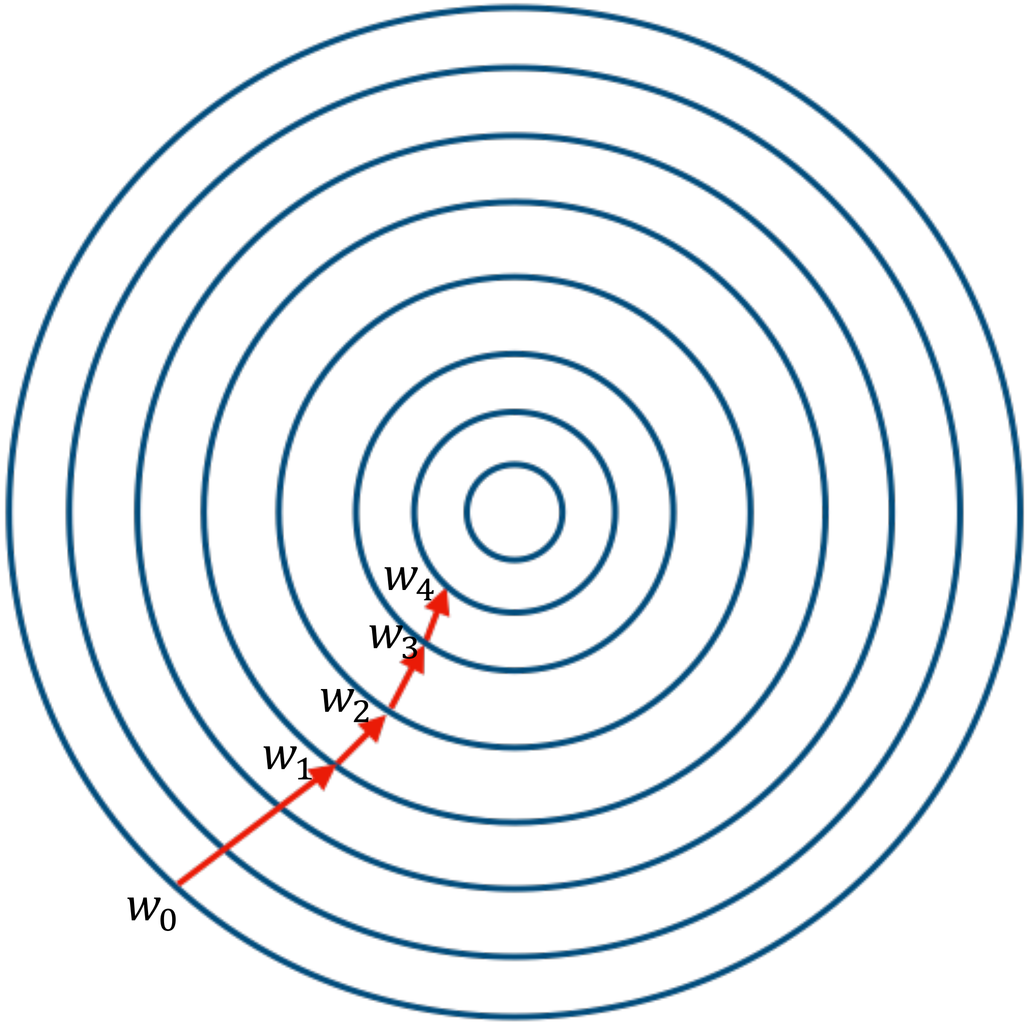
w^0 — initialization

while True:

$$w^t = w^{t-1} - \eta_t \nabla L(w^{t-1})$$

if $\|w^t - w^{t-1}\| < \epsilon$ then break

Gradient descent



Gradient descent

Lots of heuristics:

- How to initialize w^0
- How to select step size η_t
- When to stop
- How to approximate gradient $\nabla L(w^{t-1})$

Gradient descent for MSE

Linear regression and MSE:

$$L(w) = \frac{1}{\ell} \|Xw - y\|^2$$

Derivatives:

$$\nabla L_w(w) = \frac{2}{\ell} X^T (Xw - y)$$

Gradient descent vs analytical solution

Analytical solution for MSE: $w = (X^T X)^{-1} X^T y$

Gradient descent:

- Easy to implement
- Very general, can be applied to any differentiable loss function
- Requires less memory and computations (for stochastic methods)

Summary

- Gradient descent provides a general learning framework
- Can be used both for classification and regression tasks
- Advanced methods — in next lessons

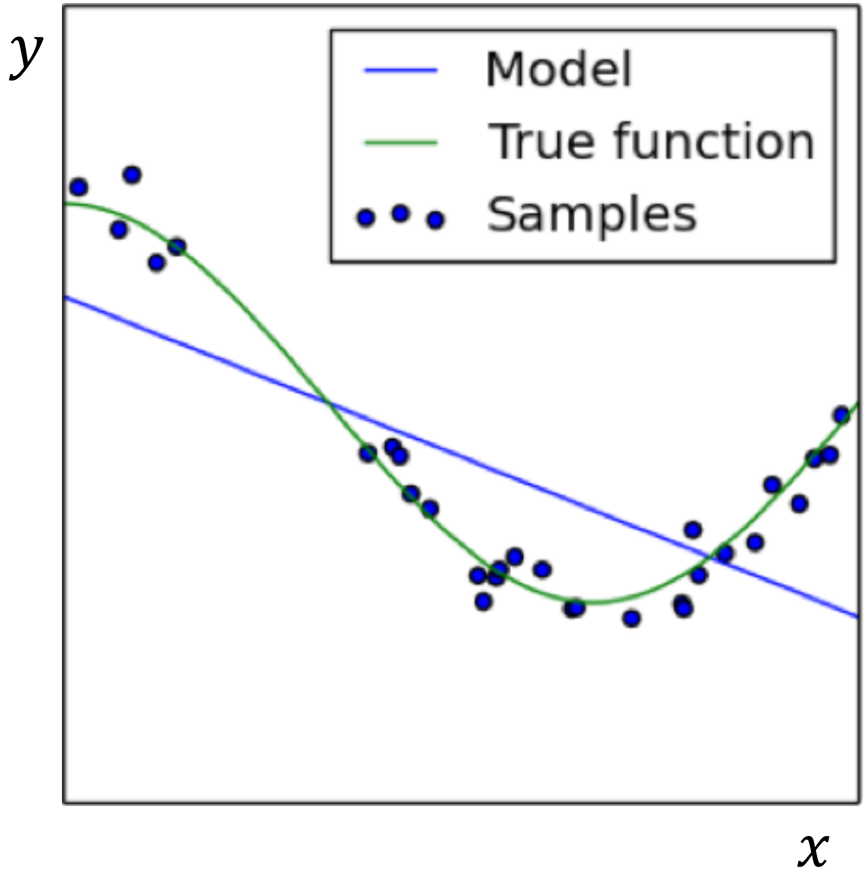
Generalization

- Consider a model with accuracy 80% on training set
- How it will perform on the new data?
- Does the model generalize well?

Underfitting and overfitting example

Training set: $X \subset \mathbb{R}$

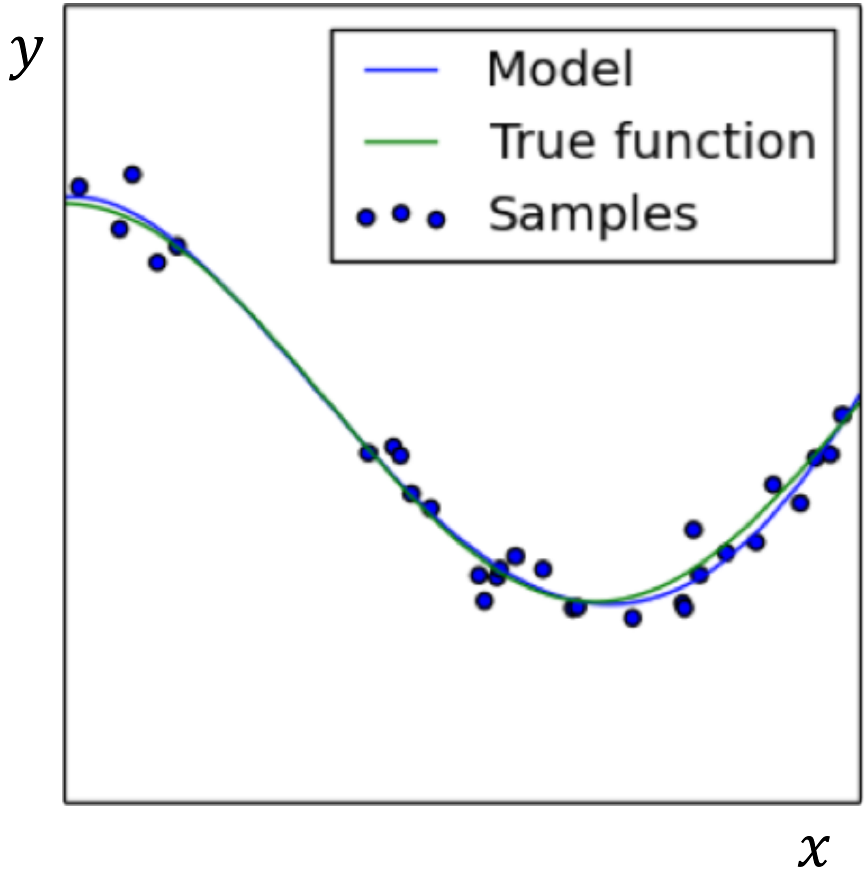
Model: $a(x) = b + w_1 x$



Underfitting and overfitting example

Training set: $X \subset \mathbb{R}$

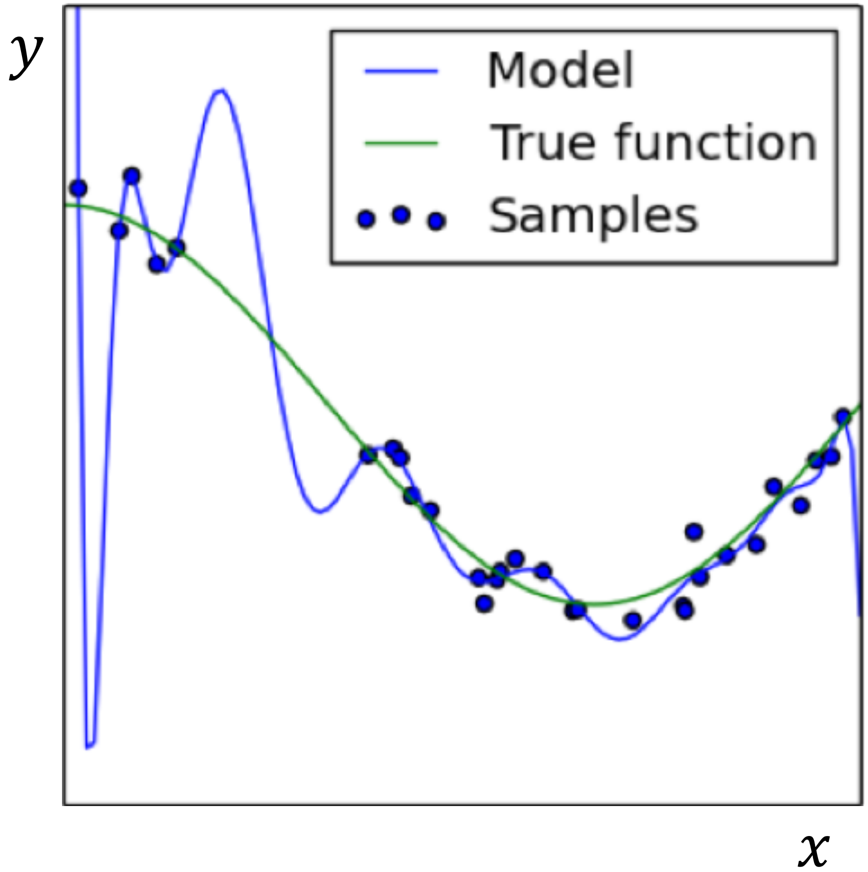
Model: $a(x) = b + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$



Underfitting and overfitting example

Training set: $X \subset \mathbb{R}$

Model: $a(x) = b + w_1 x + w_2 x^2 + \cdots + w_{15} x^{15}$



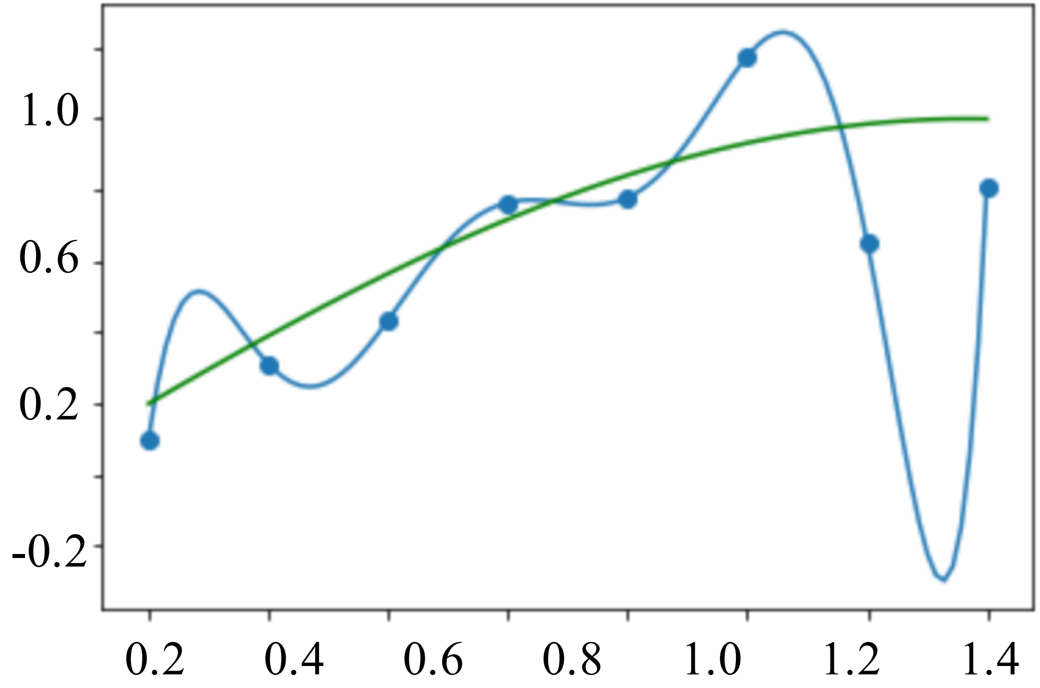
Overfitting example 2

Training set: $\{0.2, 0.4, \ldots, 1.6\}$, $y = \sin(x) + \epsilon$

Model: $a(x) = b + w_1 x + w_2 x^2 + \cdots + w_8 x^8$

Parameters: $(130.0, -525.8, \ldots, 102.6)$

Model just incorporates target into parameters!



Holdout set

Training set | Holdout set

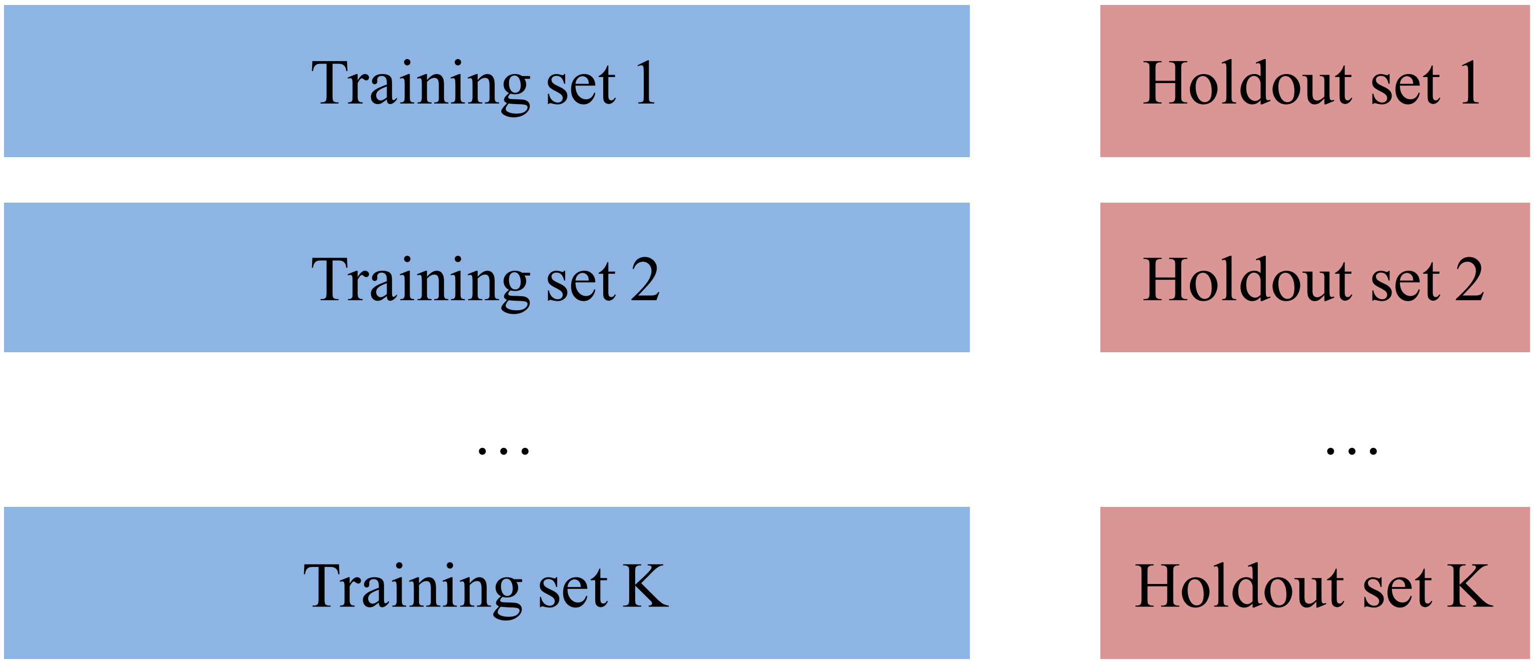
Small holdout set:

- Training set is representative
- Holdout quality has high variance

Large holdout set:

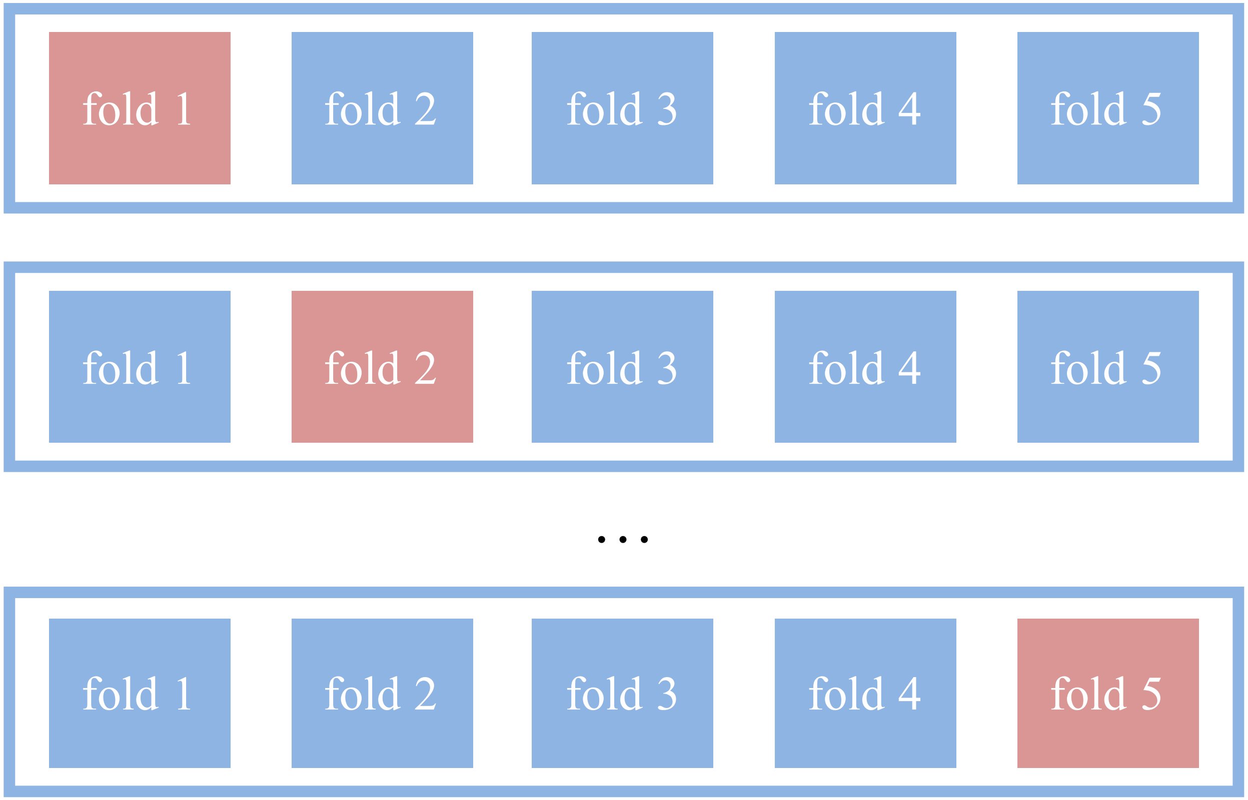
- Holdout quality has low variance
- Holdout quality has high bias

Holdout set



No guarantees that each object will be in holdout part at least once

Cross-validation



Cross-validation

- Requires to train models K times for K-fold CV
- Useful for small samples
- In deep learning holdout samples are usually preferred

Summary

- Models can easily overfit with high number of parameters
- Overfitted model just remembers target values for training set and doesn’t generalize
- Holdout set or cross-validation can be used to estimate model performance on new data

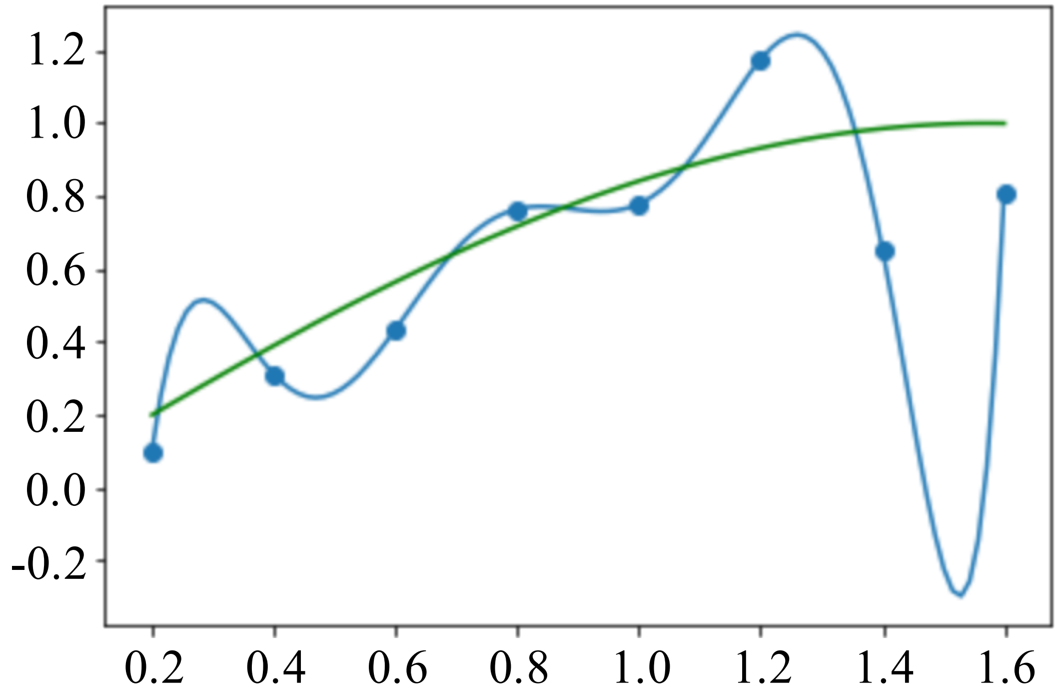
Overfitting example

Training set: $\{0.2, 0.4, \ldots, 1.6\}$, $y = \sin(x) + \epsilon$

Model: $a(x) = b + w_1 x + w_2 x^2 + \cdots + w_8 x^8$

Parameters: $(130.0, -525.8, \ldots, 102.6)$

Model just incorporates target into parameters!

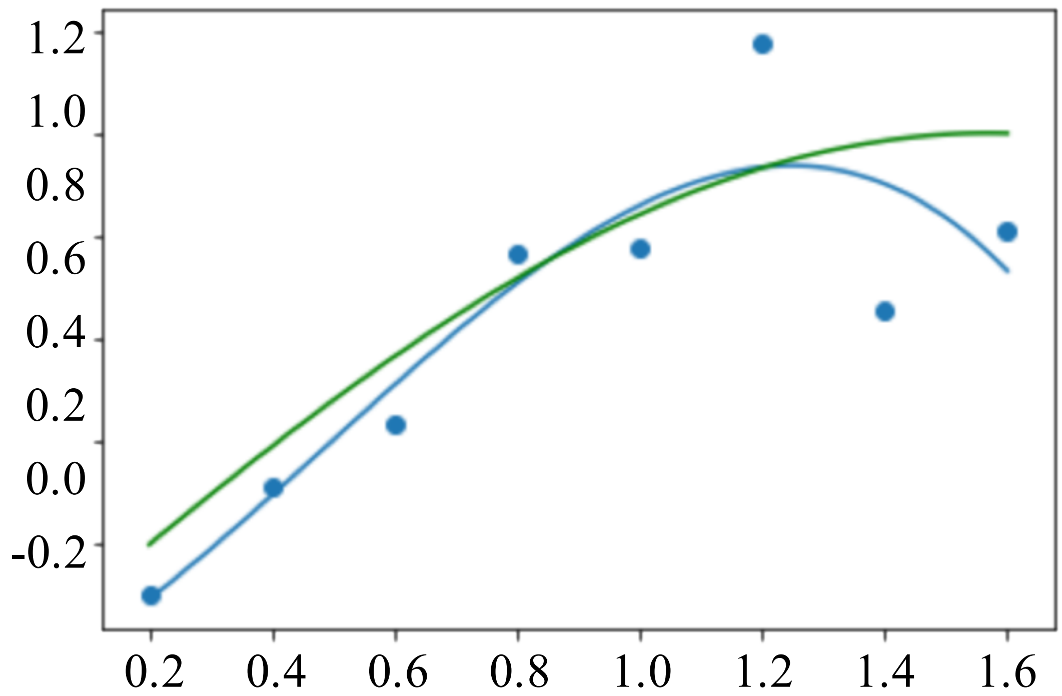


Overfitting example

Training set: $\{0.2, 0.4, \ldots, 1.6\}$, $y = \sin(x) + \epsilon$

Model: $a(x) = b + w_1 x + w_2 x^2 + w_3 x^3$

Parameters: $(0.634, 0.918, -0.626)$



Regularization

Good model weights: $(0.634, 0.918, -0.626)$

Overfitted model weights: $(130.0, -525.8, \ldots, 102.6)$

Weight penalty

$$L_{reg}(w) = L(w) + \lambda R(w) \to \min_{w}$$

- $L(w)$ — loss function (MSE, log-loss, etc.)
- $R(w)$ — regularizer (e.g. penalizes large weights)
- λ — regularization strength

L2 penalty

$$L_{reg}(w) = L(w) + \lambda\|w\|^2 \to \min_w$$

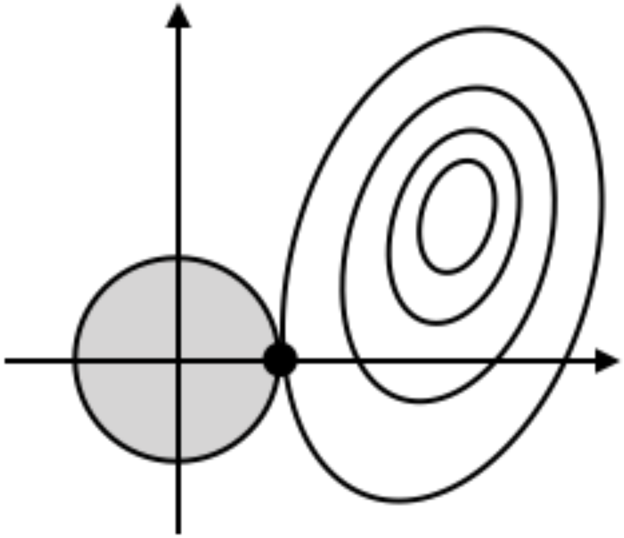
- $\|w\|^2 = \sum_{j=1}^{d} w_j^2$
- Drives all weights **closer** to zero
- Can be optimized with gradient methods

L2 penalty

$$L_{reg}(w) = L(w) + \lambda \|w\|^2 \to \min_w$$

The optimization problem is equivalent to

$$\begin{cases} L(w) \to \min\limits_w \\ \text{s.t. } \|w\|^2 \leq C \end{cases}$$



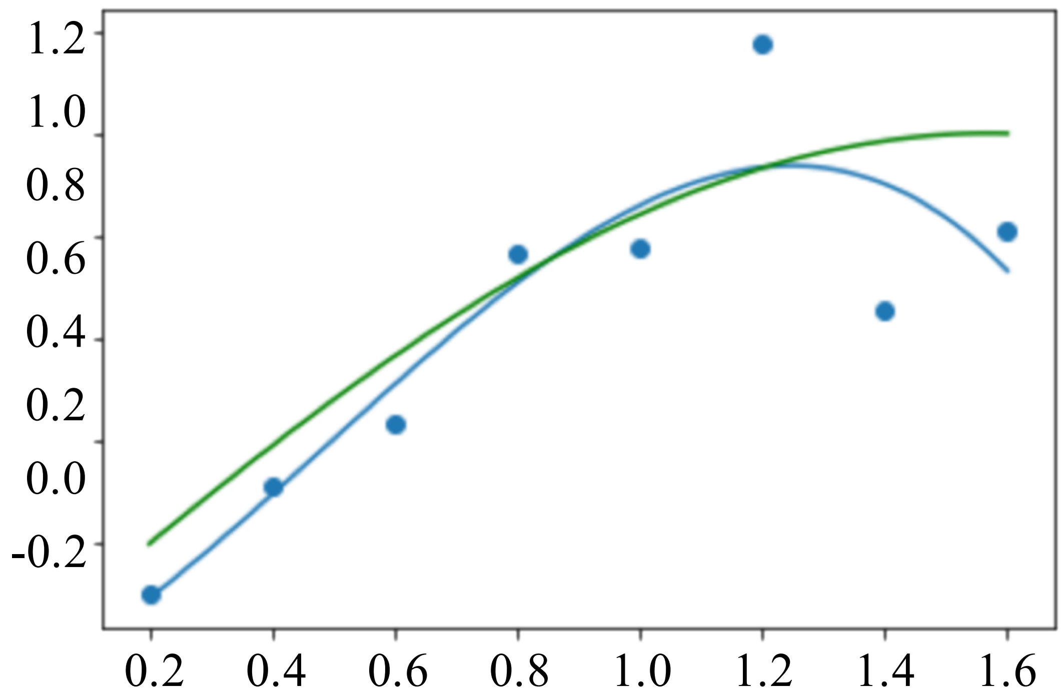
L2 penalty

$$L_{reg}(w) = L(w) + \lambda \|w\|^2 \to \min_w$$

Training set: $\{0.2, 0.4, \dots, 1.6\}$, $y = \sin(x) + \epsilon$

Model: $a(x) = b + w_1 x + w_2 x^2 + \cdots + w_8 x^8$

Parameters: $(0.166, 0.168, 0.13, 0.075, 0.014, -0.04, -0.05, 0.018)$



L1 penalty

$$L_{reg}(w) = L(w) + \lambda \|w\|_1 \to \min_w$$

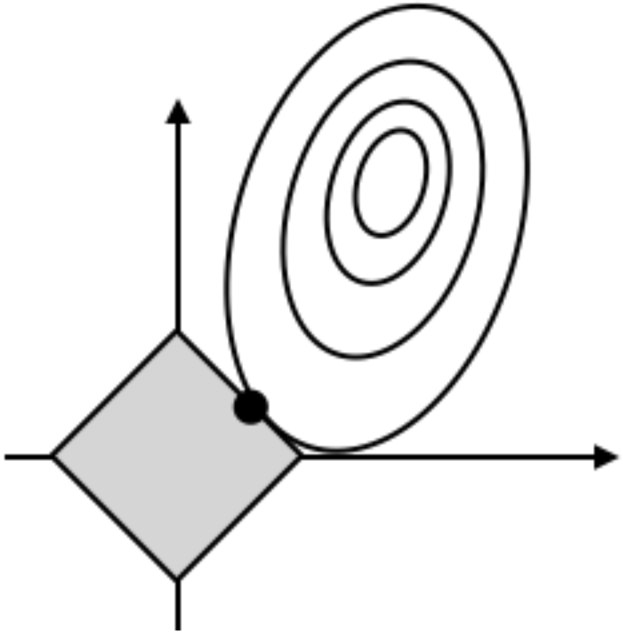
- $\|w\|_1 = \sum_{j=1}^{d} |w_j|$
- Drives some weights **exactly** to zero
- Learns sparse models
- Cannot be optimized with simple gradient methods

L1 penalty

$$L_{reg}(w) = L(w) + \lambda \|w\|_1 \to \min_w$$

The optimization problem is equivalent to

$$\begin{cases} L(w) \to \min\limits_w \\ \text{ s.t. } \|w\|_1 \le C \end{cases}$$



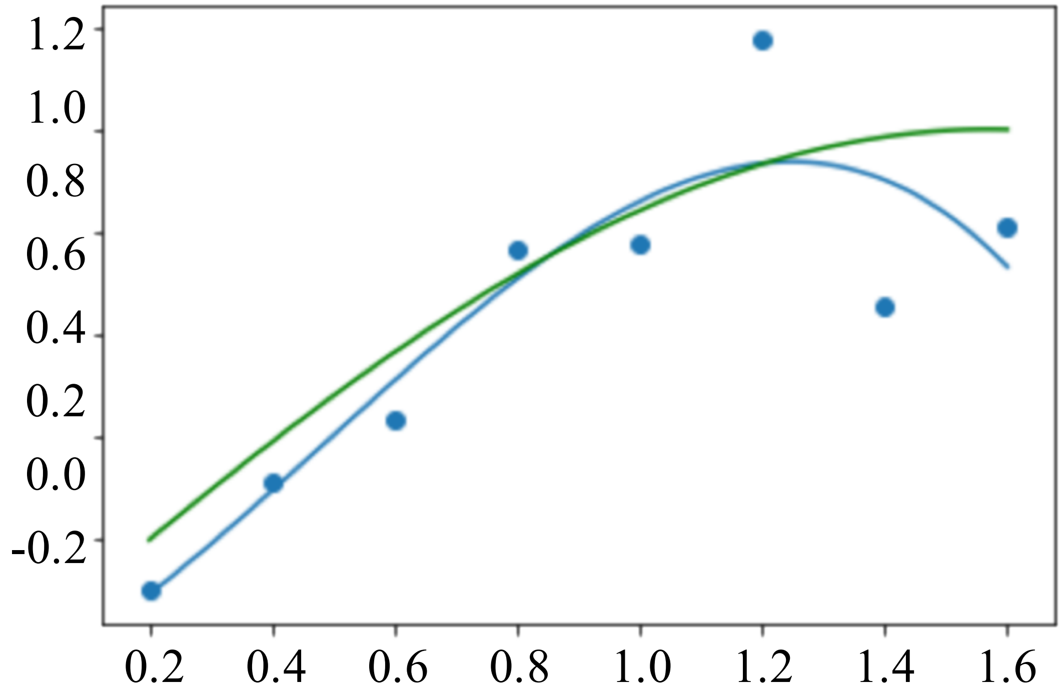
L1 penalty

$$L_{reg}(w) = L(w) + \lambda \|w\|_1 \to \min_w$$

Training set: $\{0.2, 0.4, \dots, 1.6\}$, $y = \sin(x) + \epsilon$

Model: $a(x) = b + w_1 x + w_2 x^2 + \cdots + w_8 x^8$

Parameters: (for $\lambda = 0.01$): (0.78, 0.03, **0**, **0**, **0**, −0.016, −0.01, **0**)



Other regularization techniques

- Dimensionality reduction
- Data augmentation
- Dropout
- Early stopping
- Collect more data

Summary

- One should restrict model complexity to prevent overfitting
- Common approach: penalize large weights
- Other approaches: next modules

Stochastic gradient descent

Gradient descent

Optimization problem:

$$L(w) = \sum_{i=1}^{\ell} L(w; x_i, y_i) \to \min_w$$

w^0 — initialization

while True:

$$w^t = w^{t-1} - \eta_t \nabla L(w^{t-1})$$

if $\|w^t - w^{t-1}\| < \epsilon$ then break

Gradient descent

Mean squared error:

$$\nabla L(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} \nabla (w^T x_i - y_i)^2$$

- ℓ gradients should be computed on each step
- If the dataset doesn't fit in memory, it should be read from the disk on every GD step

Stochastic gradient descent

Optimization problem:

$$L(w) = \sum_{i=1}^{\ell} L(w; x_i, y_i) \to \min_w$$

w^0 — initialization

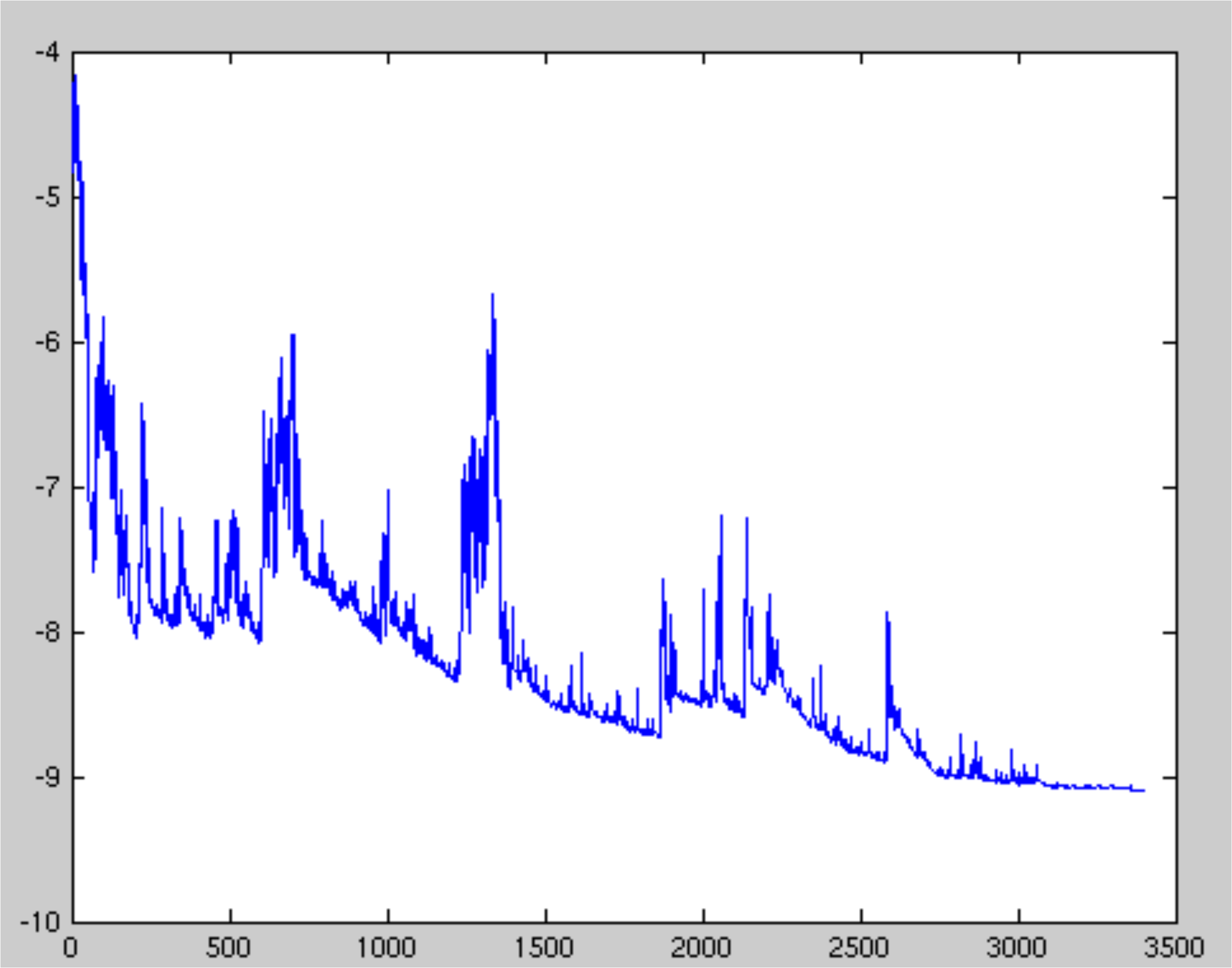
while True:

i = random index between 1 and ℓ

$w^t = w^{t-1} - \eta_t \nabla L(w^{t-1}; x_i; y_i)$

if $\|w^t - w^{t-1}\| < \epsilon$ then break

Stochastic gradient descent



Joe pharos, https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Stochastic gradient descent

- Noisy updates lead to fluctuations
- Needs only one example on each step
- Can be used in online setting
- Learning rate η_t should be chosen very carefully

Mini-batch gradient descent

Optimization problem:

$$L(w) = \sum_{i=1}^{\ell} L(w; x_i, y_i) \to \min_w$$

w^0 — initialization

while True:

$i_1, \ldots, i_m$ = random indices between 1 and ℓ

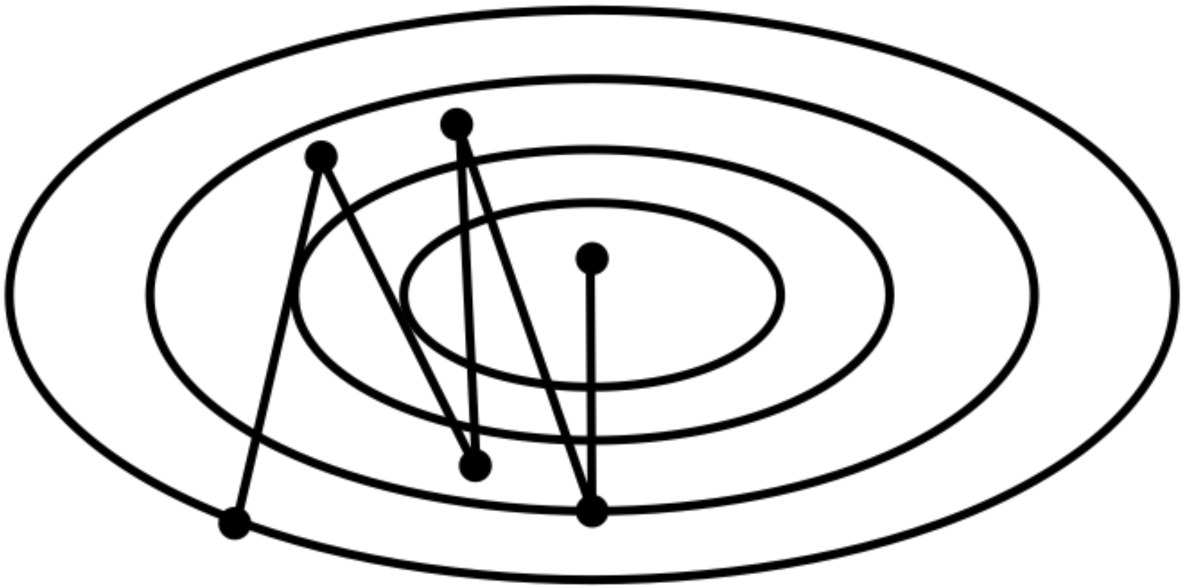
$w^t = w^{t-1} - \eta_t \frac{1}{m} \sum_{j=1}^{m} \nabla L\left(w^{t-1}; x_{i_j}; y_{i_j}\right)$

if $\|w^t - w^{t-1}\| < \epsilon$ then break

Mini-batch gradient descent

- Still can be used in online setting
- Reduces the variance of gradient approximations
- Learning rate η_t should be chosen very carefully

Difficult function

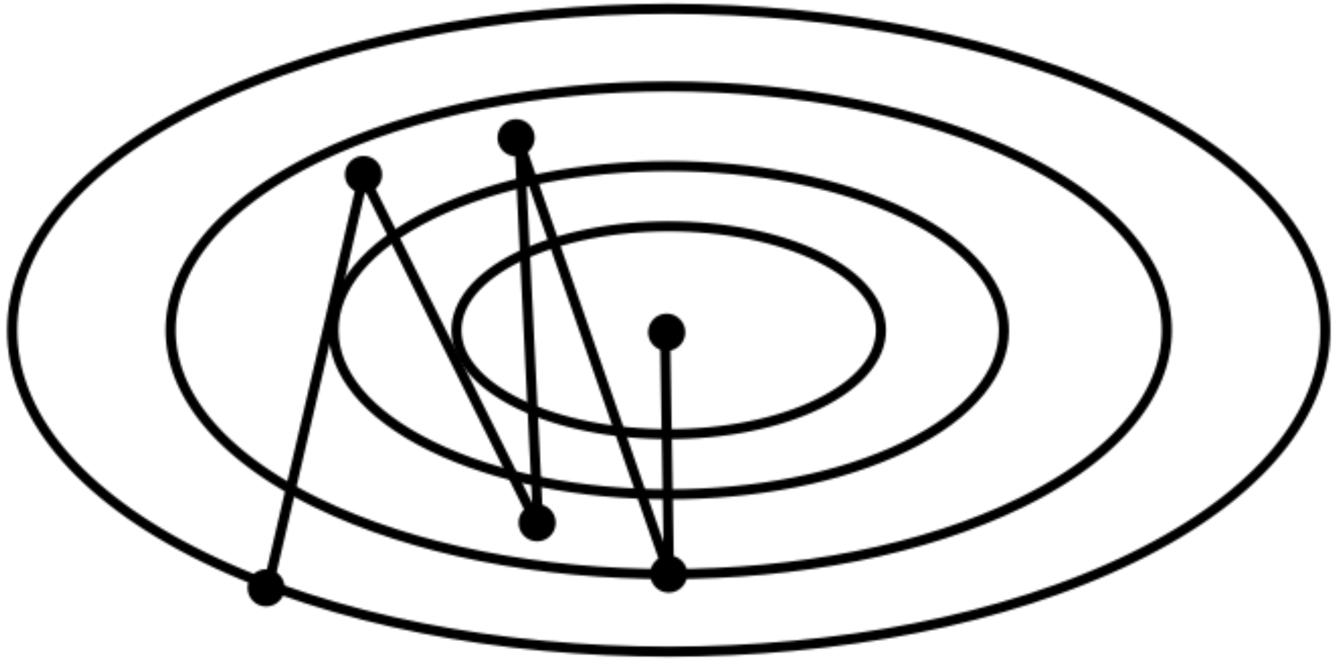


Summary

- Gradient descent is infeasible for large training sets
- Stochastic and mini-batch descents use gradient approximations t speed up computations
- Learning rate is quite hard to select
- Methods can be optimized for difficult functions

Gradient descent extensions

Difficult function



Mini-batch gradient descent

w^0 — initialization

while True:

$i_1, \dots, i_m$ = random indices between 1 and ℓ

$g_t = \frac{1}{m} \sum_{j=1}^{m} \nabla L\left(w^{t-1}; x_{i_j}; y_{i_j}\right)$

$\boldsymbol{w^t = w^{t-1} - \eta_t g_t}$

if $\|w^t - w^{t-1}\| < \epsilon$ then break

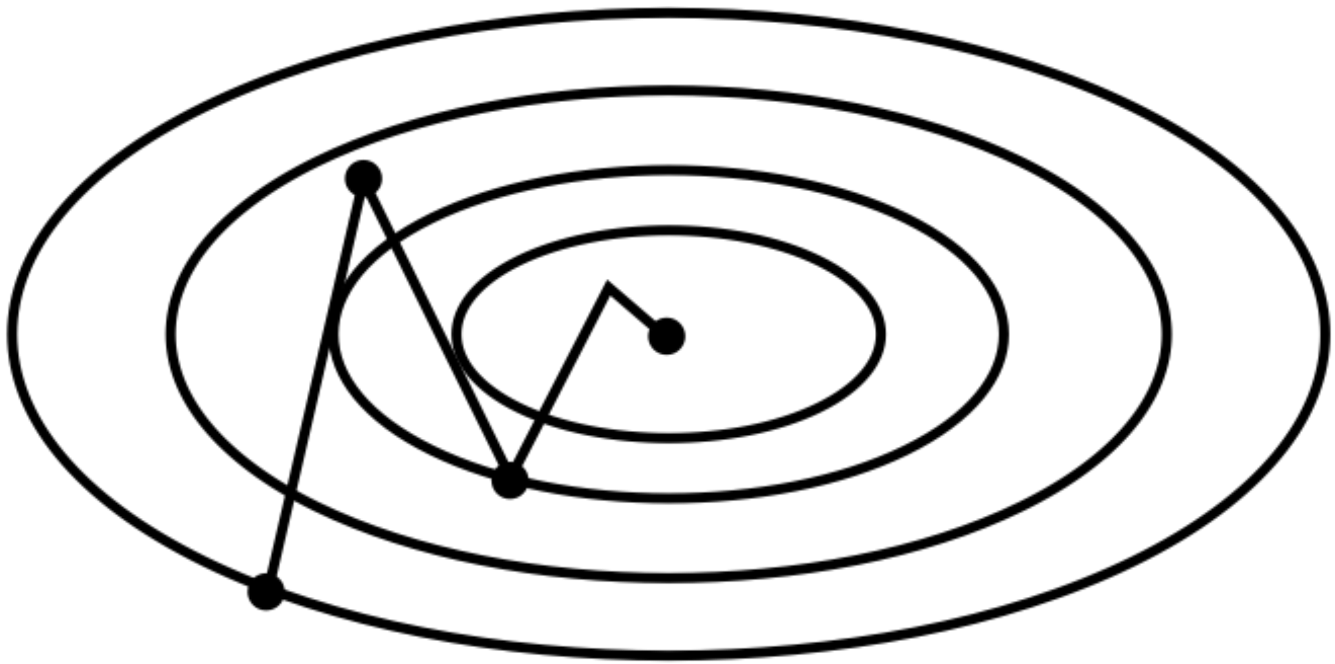
Momentum

$$h_t = \alpha h_{t-1} + \eta_t g_t$$

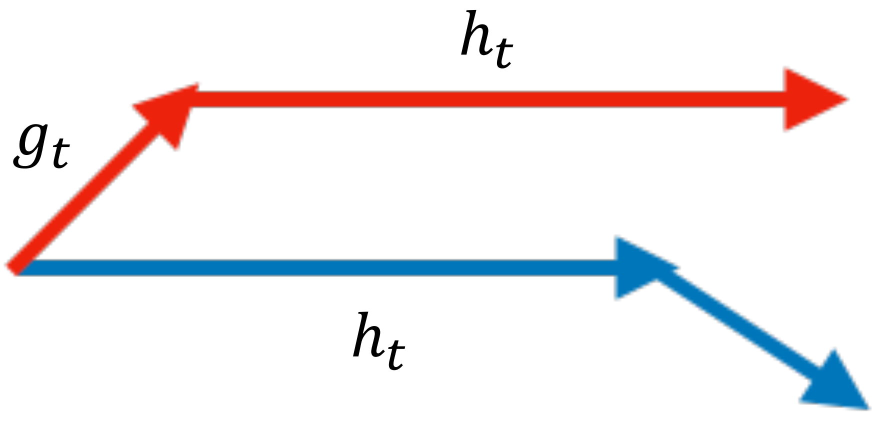
$$w^t = w^{t-1} - h_t$$

- Tends to move in the same direction as on previous steps
- h_t accumulates values along dimensions where gradients have the same sign
- Usually: $\alpha = 0.9$

Momentum



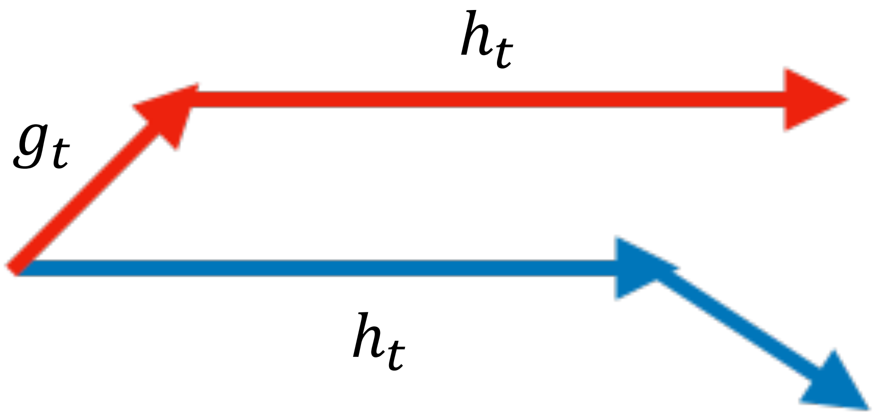
Nesterov momentum



Nesterov momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla L(w^{t-1} - \alpha h_{t-1})$$

$$w^t = w^{t-1} - h_t$$



AdaGrad

$$G_j^t = G_j^{t-1} + g_{tj}^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} g_{tj}$$

- g_{tj} — gradient with respect to j-th parameter
- Separate learning rates for each dimension
- Suits for sparse data
- Learning rate can be fixed: $\eta_t = 0.01$
- G_j^t always increases, leads to early stops

RMSprop

$$G_j^t = \alpha G_j^{t-1} + (1-\alpha) g_{tj}^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} g_{tj}$$

- α is about 0.9
- Learning rate adapts to latest gradient steps

Adam

$$v_j^t = \frac{\beta_2 v_j^{t-1} + (1-\beta_2) g_{tj}^2}{1-\beta_2^t}$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{v_j^t + \epsilon}}\, g_{tj}$$

Adam

$$m_j^t = \frac{\beta_1 m_j^{t-1} + (1-\beta_1) g_{tj}}{1-\beta_1^t}$$

$$v_j^t = \frac{\beta_2 v_j^{t-1} + (1-\beta_2) g_{tj}^2}{1-\beta_2^t}$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{v_j^t} + \epsilon} m_j^t$$

Combines momentum and individual learning rates

Summary

- Momentum methods smooth gradients and speed up convergence
- Adaptive methods eliminate sensitive learning rate
- Adam combines both approaches