Machine Learning : Part 2

- 主讲教师: 陈佳伟, sleepyhunt@zju.edu.cn
  - https://jiawei-chen.github.io/
- TA: 陈思睿, chenthree@zju.edu.cn

Grading Policy

- Literature review (60%)
  - Written assignment (>3000 words)
- 2 homework (30%)
- 2 In-class quiz (10%)

Grading Policy

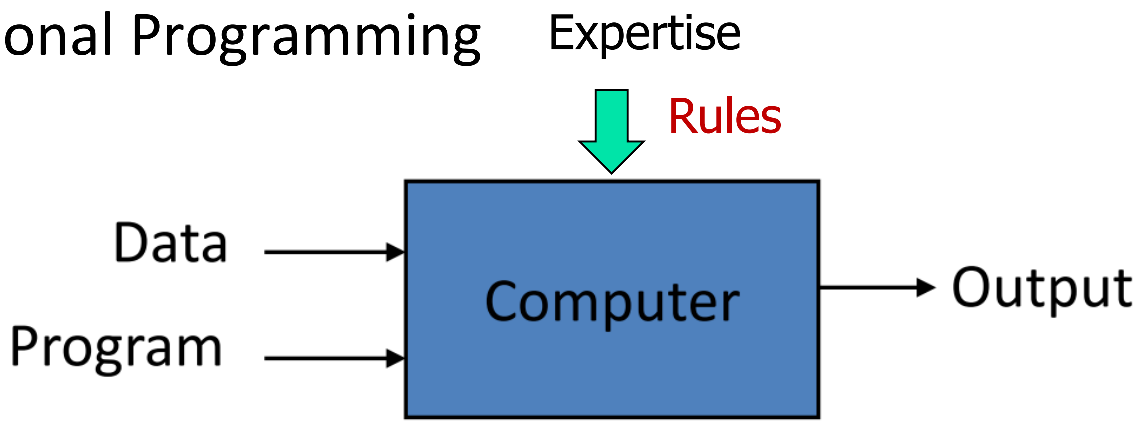
- Literature review (60%)
  - Writing a review paper on any topic related to data mining. Please structure the paper as suggested in class.
  - Requirements:
    - Written assignment (>3000 words).
    - Covering at least 15 core papers.
    - Giving a taxonomy of the chosen area.
    - Directly using GPT for generation is prohibited
  - Submitted through the "Learning from ZJU" Platform (https://courses.zju.edu.cn/)
  - DDL: Nov. 23, 2024

Grading Policy

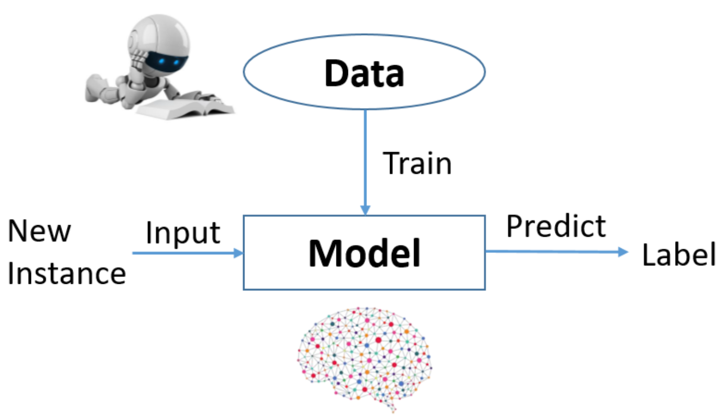
- 2 homework (30%)
  - Homework 1 (15%):
    - PAC Bound
    - Laplacian matrix
  - Homework 2 (15%):
    - Let’s LLM+X
      - Considering and Writing about Utilizing Large Language Models (LLMs) in Your Research Area
      - Requirements (more than 500 words)
  - Submitted through the "Learning from ZJU" Platform (https://courses.zju.edu.cn/)
  - DDL: Nov. 23, 2024

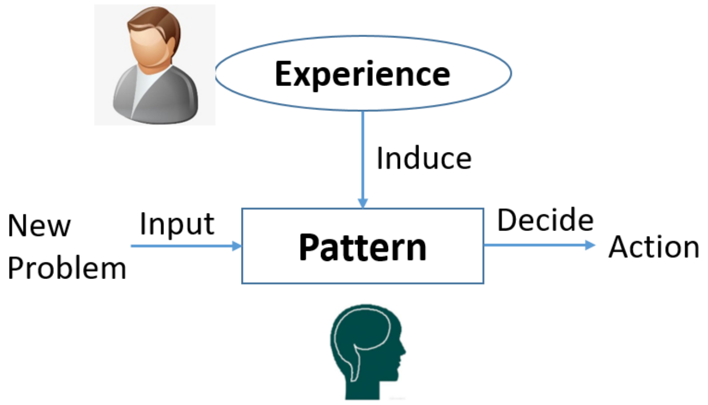
Why machine Learning?

- Traditional Programming



- Machine Learning
  - Intelligence: does not need human explicit programming
  - Universality: can handle complex data and diverse tasks





Foundation of machine learning

- The Statistical Learning Framework
  - Domain set $\mathcal{X}$, the set of objects that we may wish to label, e.g., a set of points represented by a vector of features
  - Label set $\mathcal{Y}, \mathcal{Y}$ can be $\{0,1\}$ for two-side classification or $\mathcal{R}$ for regression
  - Training data: $S = \left((x_1, y_1) \ldots (x_m, y_m)\right)$ is a finite sequence of pairs in $\mathcal{X} \times \mathcal{Y}$ sampled from the training distribution D
  - *Learner:* learning a prediction function $h: \mathcal{X} \to \mathcal{Y}, h \in \mathcal{H}$

Foundation of machine learning

- The Statistical Learning Framework
  - *Loss function: measures the error between the prediction and the label* $l: \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to [0,1]$
    - $l(h,x,y) = \mathbf{I}[h(x) \neq y]$ for classification
    - $l(h,x,y) = (h(x) - y)^2$ for regression
  - True risk: how likely the learned h to make an error when labeled data are randomly drawn **according** to D
    - $L_D(h) = \underset{(x,y)\sim D}{\mathbf{E}}[l(h,x,y)]$
  - Objective of ML: find a predictor $h: \mathcal{X} \to \mathcal{Y}$ that minimizes the true risk $L_D(h)$

Foundation of machine learning

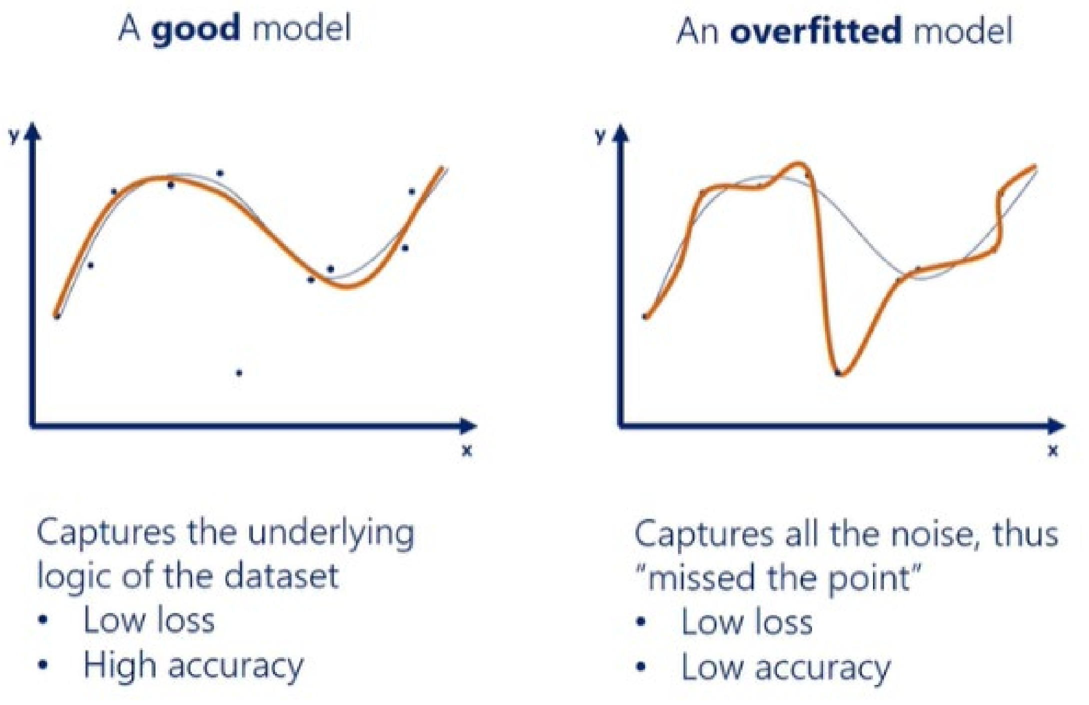
- Empirical risk:
  - Since data distribution D is not available, the model is learned on training data with optimizing:

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} l(h, x_i, y_i)$$

  - Note that $L_S(h) \neq L_D(h)$
  - $L_S(h) = 0$ does not suggests good performance.

Foundation of machine learning

- Phenomenon of overfitting:



Captures the underlying logic of the dataset
- Low loss
- High accuracy

Captures all the noise, thus "missed the point"
- Low loss
- Low accuracy

  - Controlling the space of $\mathcal{H}$
  - A theory to build the relation between $L_S(h)$ and $L_D(h)$

Foundation of machine learning

- PAC learning theory:

Theorem 1. For any finite hypothesis space of $\mathcal{H}$, given a training set S sampled i.i.d from D, then for any learned function $\tilde{h} = \mathrm{argmin}_{h\in\mathcal{H}} L_S(h)$, with probability $1-\delta$, satisfies:

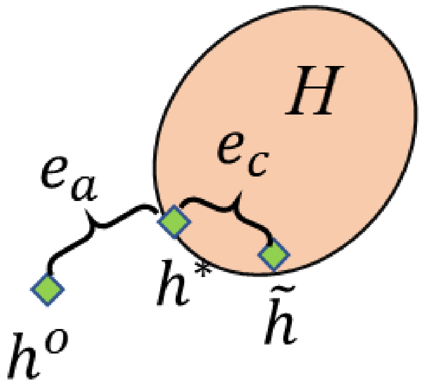
$$L_D(\tilde{h}) \leq L_D(h^*) + \sqrt{\frac{2}{m}\log(\frac{2|\mathcal{H}|}{\delta})}$$

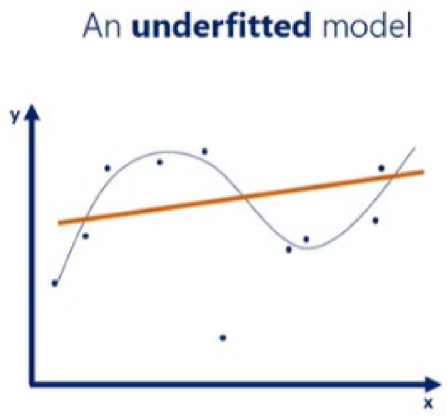
Where h^* denotes the optimal function in $\mathcal{H}$ with $h^* = \mathrm{argmin}_{h\in\mathcal{H}} L_D(h)$

Foundation of machine learning

$$L_D(\tilde{h}) \leq L_D(h^*) + \sqrt{\frac{2}{m}\log(\frac{2|\mathcal{H}|}{\delta})}$$

- *Bias-complexity decomposition*
  - $e_a = L_D(h^*)$, Approximation Error or inductive bias.
  - $e_c = L_D(\tilde{h}) - L_D(h^*)$, Estimation Error.
- When $|\mathcal{H}|$ is small

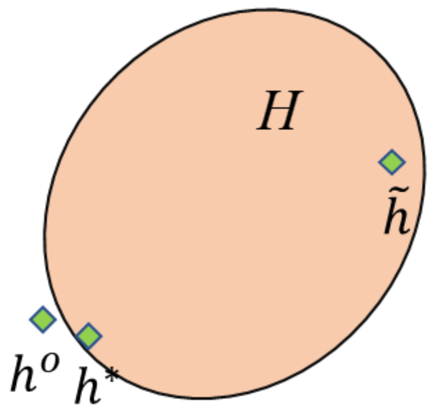


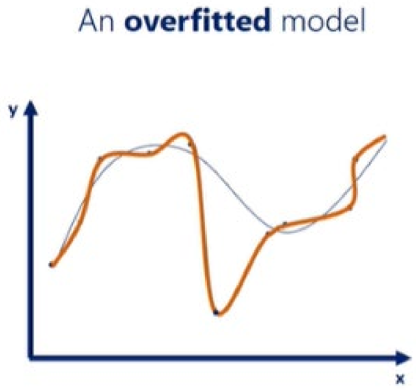


Foundation of machine learning

$$L_D(\tilde{h}) \le L_D(h^*) + \sqrt{\frac{2}{m}\log(\frac{2|\mathcal{H}|}{\delta})}$$

- *Bias-complexity decomposition*
  - $e_a = L_D(h^*)$, Approximation Error or inductive bias.
  - $e_c = L_D(\tilde{h}) - L_D(h^*)$, Estimation Error.
- When $|\mathcal{H}|$ is large



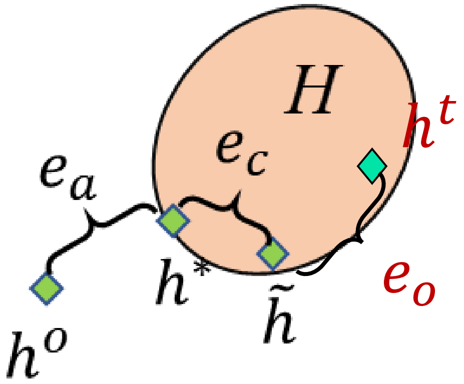


Machine Learning

- How to find $\tilde{h}$?

$$\tilde{h} = \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$$

- Three Components of Machine Learning
  - Models ($\mathcal{H}$)
  - Learning algorithm
  - Loss function ($L_S(h)$)
- All are important

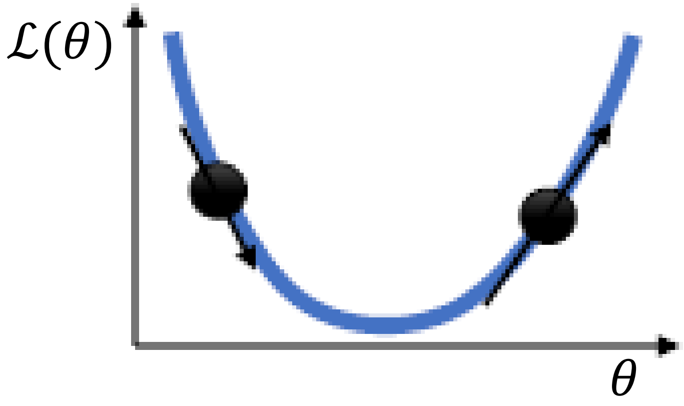


Gradient descent

Gradient descent

- Gradient descent is a method to minimize an objective function $\mathcal{L}(\theta)$.
  - $\theta \in \mathbb{R}^d$: model parameters
  - η: learning rate or step size
  - $\nabla\mathcal{L}(\theta)$: gradient of the $\mathcal{L}(\theta)$ with regard to the parameters, $\nabla\mathcal{L}(\theta) = \begin{pmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{pmatrix}$
  - Update equation: $\theta^t = \theta^{t-1} - \eta\nabla\mathcal{L}(\theta^{t-1})$

Which way does $\mathcal{L}(\theta)$ decrease?



negative slope = go to the right
negative slope = go to the left
In general:
For each dimension, go in the direction **opposite the slope along that dimension.**

Gradient descent

- Why Gradient descent?

$$\theta^t = \theta^{t-1} - \eta \nabla \mathcal{L}(\theta^{t-1})$$

- Which direction can reduce the loss?

$$\min_\epsilon \mathcal{L}(\theta + \epsilon)$$

$$\mathcal{L}(\theta + \epsilon) = \mathcal{L}(\theta) + \epsilon^T \nabla \mathcal{L}(\theta) + O(|\epsilon|^2)$$

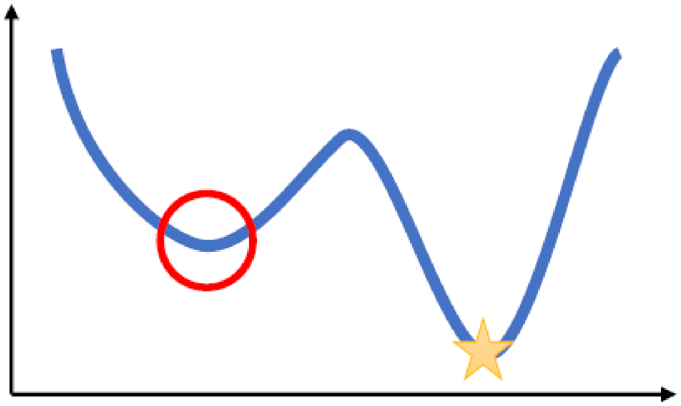
- The negative gradient direction

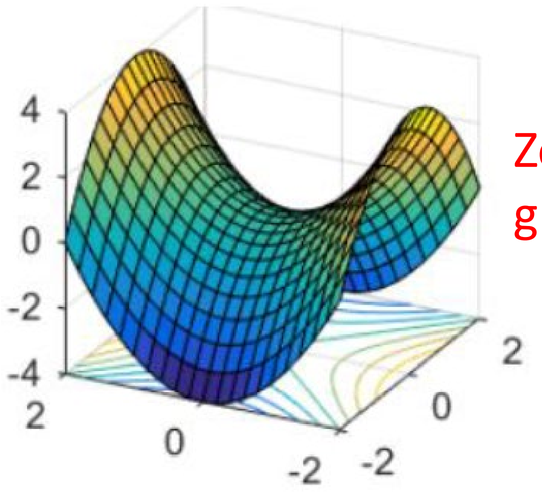
$$\epsilon \propto - \nabla \mathcal{L}(\theta)$$

Gradient descent

- Stochastic Gradient Descent (SGD)
  - Motivation: The high cost of running gradient descent over the full training set at once.
  - Method: Computes update for each datapoint (x_i, y_i) or a batch, usually uniformly sampled from the training dataset.
  - Update equation: $\theta^t = \theta^{t-1} - \eta\nabla\mathcal{L}(\theta^{t-1}; x_i; y_i)$

But SGD has pitfalls too....


The local optimum



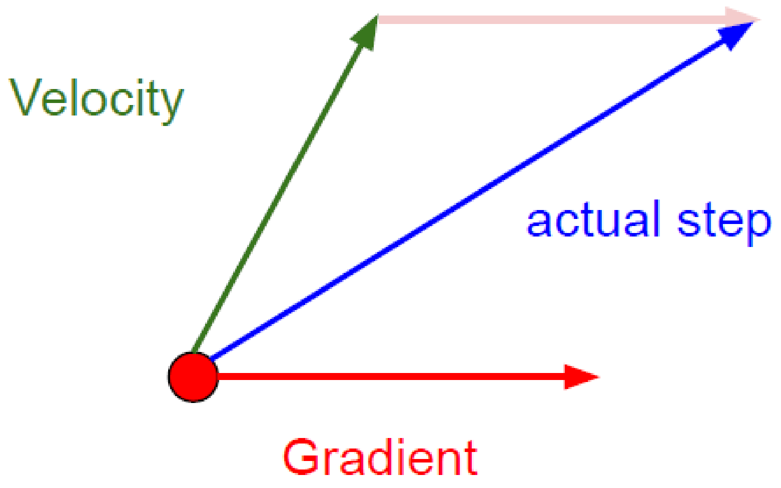
Saddle point

Zero gradient,
gradient descent gets stuck

Gradient descent

- Gradient descent with momentum
  - Motivation: help gradient descent navigate local optimum or saddle points
  - introduce velocity variable v to save proportion of previous movements.

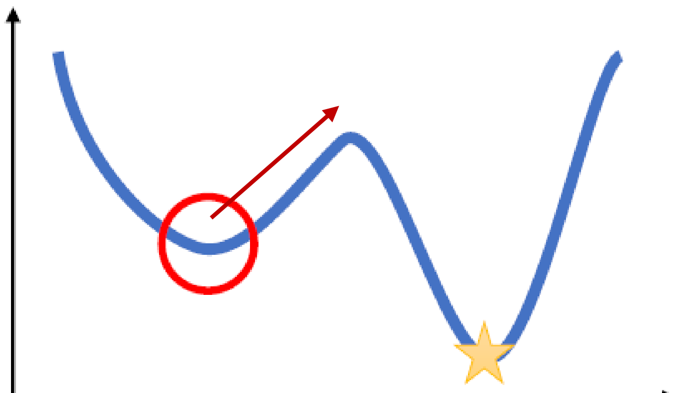
Intuition: if successive gradient steps point in **similar** directions, we should **go faster** in that direction



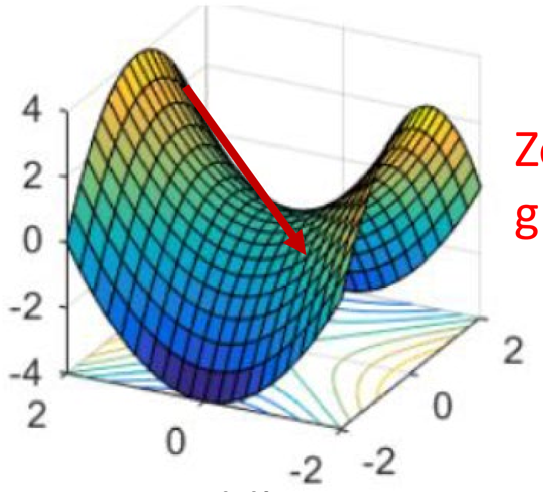
Combine gradient at current point with velocity to get step used to update weights

Gradient descent

SGD with momentum:



The local optimum



Saddle point

Zero gradient,
gradient descent gets stuc

Gradient descent

- Comparison of SGD and SGD + momentum

SGD

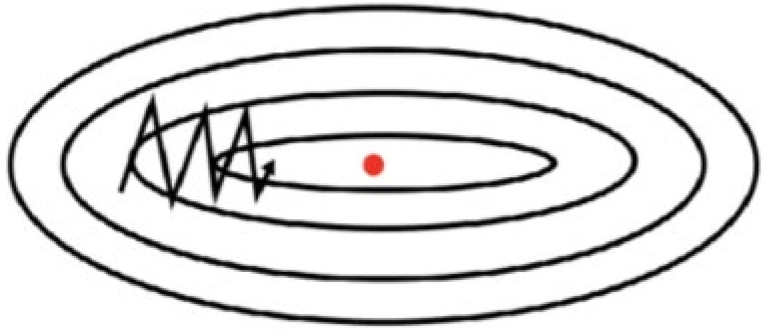
$$\theta^t = \theta^{t-1} - \eta \nabla \mathcal{L}(\theta^{t-1}; x_i; y_i)$$

SGD + momentum

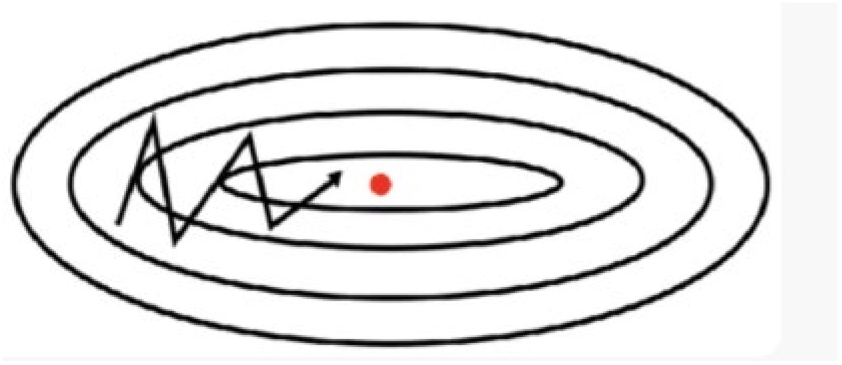
$$v^t = \gamma v^{t-1} + \eta \nabla \mathcal{L}(\theta^{t-1}; x_i; y_i)$$

$$\theta^t = \theta^{t-1} - v^t$$

(γ:usually about 0.9)



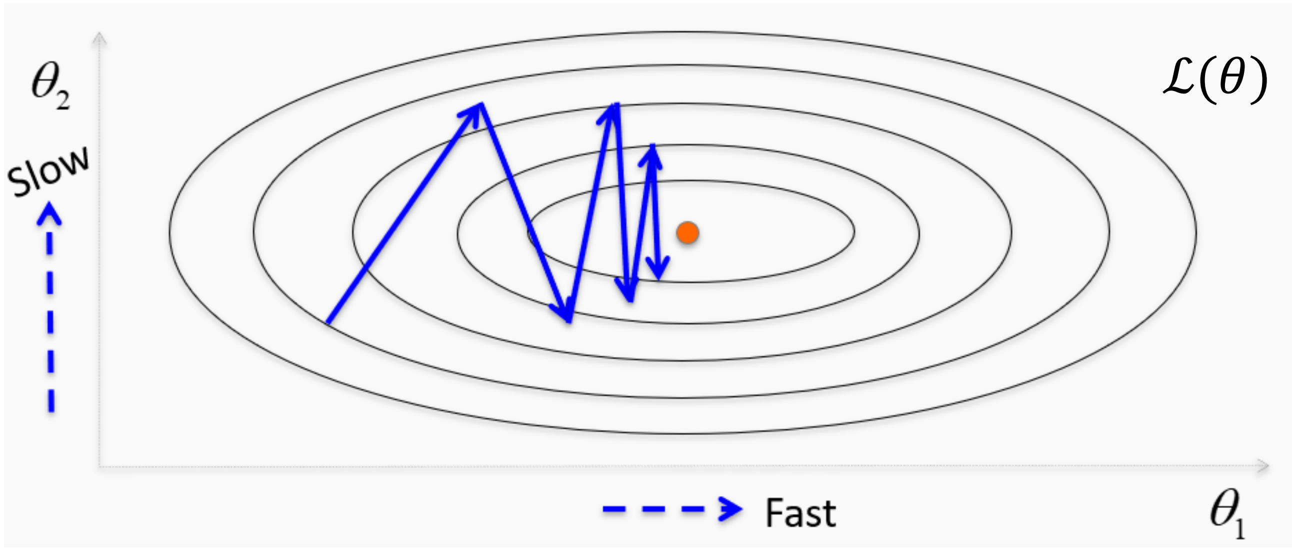
SGD without momentum



SGD with momentum

Gradient descent

- Adagrad(Adaptive Gradient)
  - Motivation: previous methods use **same learning rate** η for all parameters θ.



➢ If $\nabla\mathcal{L}(\theta_1) \gg \nabla\mathcal{L}(\theta_2)$,then a large learning rate leads divergence on θ_1 while a small one make too little progress on θ_2

➢ We need a learning rate adapts to each dimension:

Gradient descent

- Adagrad(Adaptive Gradient)
  - Method: **Increase the learning rate** of parameters that have not changed much so far, and **decrease the learning rate** of parameters that have much changed so far.

  At time t , Accumulate the squared sum of historic gradients:

$$G^t = G^{t-1} + (\nabla\mathcal{L}(\theta^{t-1}))^2$$

  Update parameters:

$$\theta^t = \theta^{t-1} - \frac{\eta}{\sqrt{G^t + \epsilon}}\nabla\mathcal{L}(\theta^{t-1})$$

ϵ: smoothing term to avoid division by zero

Gradient descent

- RMSprop
  - Motivation: Adagrad decays the learning rate that may lead slow progress at end
  - RMSprop use exponentially weighted average for gradient accumulation, updates parameters in the same way as Adagrad

$$G^t = \beta G^{t-1} + (1-\beta)(\nabla\mathcal{L}(\theta^{t-1}))^2$$
$$= (1-\beta)\sum_{\tau=1}^{t}(\nabla\mathcal{L}(\theta^{\tau}))^2$$

(β is the decay rate, usually set to 0.9)

- Compare to Adagrad: In the updating process of RMSprop, the learning rate of each parameter **does not decrease monotonically**; it can increase or decrease.

Gradient descent

- Adam
  - Motivation: wants both momentum and adapted learning rate
  - Method:

    Estimate first moment(similar to momentum):

    $$v^t = \beta_1 v^{t-1} + (1-\beta_1)\nabla\mathcal{L}(\theta^{t-1})$$

    Estimate second moment(similar to RMSprop):

    $$G^t = \beta_2 G^{t-1} + (1-\beta_2)(\nabla\mathcal{L}(\theta^{t-1}))^2$$

    (v^t and G^t are initialized as 0, for this reason they are biased towards 0)

    Compute bias-corrected first and second moment estimates:

    $$\hat{v}^t = \frac{v^t}{1-\beta_1^t} \qquad \hat{G}^t = \frac{G^t}{1-\beta_2^t}$$

    Update parameters:

    $$\theta^t = \theta^{t-1} - \frac{\eta}{\sqrt{\hat{G}^t + \epsilon}}\hat{v}^t$$

Matrix Differentiation

Why We Need Matrix Differentiation

- Optimization involves the gradient with a vector:

$$\nabla\mathcal{L}(\theta^{t-1})$$

- How to calculate it?
- The rationale behind Py-Torch?

https://pytorch.org/

How to define

Today, we only consider the scaler value function with respect to vector & matrix variable:

$$f: \mathbf{R}^n \to \mathbf{R} \text{ and } f: \mathbf{R}^{n \times m} \to \mathbf{R}$$

We define the differentiation as an **element-wise** operator:

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \text{ and } \frac{\partial f}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{n1}} & \cdots & \frac{\partial f}{\partial x_{nm}} \end{bmatrix}$$

Matrix Differentiation Rule

Chain Rule is All You Need

Reconstruct Computation as

Matrix Multiplication

is All You Need

Vector Differentiation

Now, we firstly consider **scalar-value function** with respect to **vector variable**:

$$f: \mathbf{R}^n \to \mathbf{R}$$

The matrix differentiation $\frac{df}{d\mathbf{x}}$ is actually associated with the multi-variable differentiation:

$$\begin{aligned} df &= \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \cdots + \frac{\partial f}{\partial x_n} dx_n \\ &= (\frac{\partial f}{\partial \mathbf{x}})^T \mathbf{dx} \end{aligned}$$

How about Matrix

As the differentiation is an element-wise operator, what we have is:

$$\mathrm{d}f = \frac{\partial f}{\partial x_{11}}\mathrm{d}x_{11} + \frac{\partial f}{\partial x_{12}}\mathrm{d}x_{12} + \cdots + \frac{\partial f}{\partial x_{nm}}\mathrm{d}x_{nm}$$

This is just :

$$\mathrm{d}f = sum\left(\begin{bmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{n1}} & \cdots & \frac{\partial f}{\partial x_{nm}} \end{bmatrix} \odot \begin{bmatrix} \mathrm{d}x_{11} & \cdots & \mathrm{d}x_{1m} \\ \vdots & \ddots & \vdots \\ \mathrm{d}x_{n1} & \cdots & \mathrm{d}x_{nm} \end{bmatrix}\right)$$

Trace Trick

The element-wise multiplication can be easily calculated as:

$$\mathrm{sum}(A \odot B) = \sum_{j=1}^{m}\sum_{i=1}^{n} A_{ij}B_{ij} = \sum_{j=1}^{m}\sum_{i=1}^{n} A_{ji}^{T}B_{ij} = \mathrm{tr}(A^{T}B)$$

It just uses matrix multiplication rule. What we want to emphasize is as follows:

$$df = \mathrm{tr}(\left(\frac{\mathrm{d}f}{\mathrm{d}\mathbf{X}}\right)^{T} \mathrm{d}\mathbf{X})$$

Vector Differentiation

Some useful formulas:

$$d(X \pm Y) = dX \pm dY$$

$$d(XY) = (dX)Y + XdY$$

$$d(X^T) = (dX)^T$$

$$d(X \odot Y) = dX \odot Y + X \odot dY$$

$$d\sigma(X) = \sigma'(X) \odot dX$$

Quadratic Example

All you need is construct the same structure as follows.

$$\mathrm{d}f = (\frac{\partial f}{\partial \mathbf{x}})^T \mathrm{d}\mathbf{x}$$

$$df = \mathrm{tr}(\left(\frac{\mathrm{d}f}{\mathrm{d}\mathbf{X}}\right)^T \mathrm{d}\mathbf{X})$$

Consider the quadratic:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x}$$

Ok, let us use the vector differentiation rule as:

$$\mathrm{d}f = \mathrm{d}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathrm{d}\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{A}\mathrm{d}\mathbf{x}$$

$$= (\mathbf{A}\mathbf{x})^T\mathrm{d}\mathbf{x} + \mathbf{0} + \left(\mathbf{x}^T\mathbf{A}\right)\mathrm{d}\mathbf{x}$$

We can get the result as, $\frac{\partial f}{\partial \mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{A}^T\mathbf{x} = \left(\mathbf{A} + \mathbf{A}^T\right)\mathbf{x}$

Inverse Matrix Example

What about inverse matrix $\mathrm{d}\mathbf{X}^{-1}$ 🤔

Damn, bro! This is even not a scalar value function? 🤔

Calm down. Calm down. 🤗 Let us derive as follows:

$$\mathbf{X}\mathbf{X}^{-1} = I$$

$$\mathrm{d}\mathbf{X}\,\mathbf{X}^{-1} + \mathbf{X}\,\mathrm{d}\mathbf{X}^{-1} = \mathrm{d}I$$

$$\mathrm{d}\mathbf{X}\,\mathbf{X}^{-1} + \mathbf{X}\,\mathrm{d}\mathbf{X}^{-1} = 0$$

$$\mathrm{d}\mathbf{X}^{-1} = -\mathbf{X}^{-1}\,\mathrm{d}\mathbf{X}\,\mathbf{X}^{-1}$$

Linear Regression Example

The linear regression is one of the most normal problems.

Given n-size batch of data, in which each sample is consisted of m features:

$$\mathbf{X} \in R^{n\times m} = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ & \vdots & \\ - & \mathbf{x}_n^T & - \end{bmatrix} \text{ and } \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \text{ and } \boldsymbol{w} \in R^m$$

What we want to do is find the weight vector which minimize the difference with label:

$$\mathcal{L}(w) = (\mathbf{X}w - \mathbf{y})^T(\mathbf{X}w - \mathbf{y})$$

In-class Quiz

- Please give the gradient of $\frac{\partial \mathcal{L}}{\partial w}$:

$$\mathcal{L}(w) = (\mathbf{X}w - \mathbf{y})^T(\mathbf{X}w - \mathbf{y})$$

$$\mathbf{X} \in R^{n\times m} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \text{ and } \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \text{ and } \boldsymbol{w} \in R^m$$

- Requirements:
  - 15-minute time limit
  - Giving the detailed derivations

Linear Regression Example

We can derive that:

$$\mathcal{L}(w) = (\mathbf{X}w - \mathbf{y})^T(\mathbf{X}w - \mathbf{y})$$

$$= \left(w^T\mathbf{X}^T - \mathbf{y}^T\right)(\mathbf{X}w - \mathbf{y})$$

$$= w^T\mathbf{X}^T\mathbf{X}w - 2\mathbf{y}^T\mathbf{X}w + \mathbf{y}^T\mathbf{y}$$

$$\mathbf{d}\mathcal{L} = \mathrm{d}w^T\mathbf{X}^T\mathbf{X}w + w^T\mathbf{X}^T\mathbf{X}\mathrm{d}w - 2\mathbf{y}^T\mathbf{X}\mathrm{d}w$$

$$= \left(\mathbf{X}^T\mathbf{X}w\right)^T dw + w^T\mathbf{X}^T\mathbf{X}\mathrm{d}w - 2\mathbf{y}^T\mathbf{X}\mathrm{d}w$$

$$\frac{\partial\mathcal{L}}{\partial w} = 2\mathbf{X}^T\mathbf{X}w - 2\mathbf{X}^T y$$

Determinant Example

Now, let us consider the determinant:

$$f(\mathbf{X}) = |\mathbf{X}| \text{ and } f(\mathbf{X}) = \log(|\mathbf{X}|)$$

🤔 It seems that we cannot easily construct the trace form.

Reviewing what we have learned in your linear algebra class:

$$|\mathbf{X}| = \begin{vmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{vmatrix} = x_{i1}A_{i1} + x_{i2}A_{i2} + \cdots + x_{in}A_{in}$$

Determinant Example

Therefore, we have:

$$\frac{\partial f}{\partial \mathbf{X}} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \dots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} = (\mathbf{X}^*)^{\boldsymbol{T}}$$

$\mathbf{X}^*$ is the associated matrix:

$$XX^* = \begin{bmatrix} |\mathbf{X}| & 0 & \dots & 0 \\ 0 & |\mathbf{X}| & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & |\mathbf{X}| \end{bmatrix} \rightarrow \boldsymbol{X}^* = |\boldsymbol{X}|\boldsymbol{X}^{-1}$$

Determinant Example

With $\mathbf{X}^* = |\mathbf{X}|\mathbf{X}^{-\mathbf{1}}$ and $\frac{\partial f}{\partial \mathbf{X}} = (\mathbf{X}^*)^{\boldsymbol{T}}$, we could have:

$$\mathrm{d}|\mathbf{X}| = \mathrm{tr}\left(\left(\frac{\partial f}{\partial \mathbf{X}}\right)^T \mathrm{d}\mathbf{X}\right) = \mathrm{tr}(\mathbf{X}^* \mathrm{d}\mathbf{X}) = \mathrm{tr}\left(|\mathbf{X}|\mathbf{X}^{-\mathbf{1}}\mathrm{dX}\right)$$

Now, we could easily derive the derivative of $f(\mathbf{X}) = \log(|\mathbf{X}|)$ **:**

$$df = \mathrm{tr}(df) = tr(\frac{1}{|\mathbf{X}|}\mathrm{d}|\mathbf{X}|) = \frac{1}{|\mathbf{X}|}\mathrm{tr}\left(|\mathbf{X}|\mathbf{X}^{-\mathbf{1}}\mathrm{dX}\right) = \mathrm{tr}(\boldsymbol{X}^{-\mathbf{1}}\mathrm{dX})$$

Then:

$$\frac{\partial \log(|\mathbf{X}|)}{\partial \mathbf{X}} = (\mathbf{X}^{-1})^T$$

Multi-Variable Gaussian Example

In many machine-learning applications, we would use multi-variable Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$:

$$f(\mathbf{x}) = \frac{1}{\left(\sqrt{2}\right)^{n} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^{T} \Sigma^{-1} (\mathbf{x} - \mu)\right)$$

We want to use **Maximum Likelihood Estimation** to estimate Σ**.**

Multi-Variable Gaussian Example

Given a series of data point, we use the likelihood function l (omit the constant) as:

$$\sum_{i=1}^{N} \log\left(f\left(\mathbf{x}_i\right)\right) = -\frac{N}{2}\log(|\Sigma|) - \frac{1}{2}\sum_{i=1}^{N}(\mathbf{x}_i - \mu)^T\Sigma^{-1}(\mathbf{x}_i - \mu)$$

We can derive that:

$$\mathrm{d}l = \mathrm{tr}(\mathrm{d}l)$$

$$= -\frac{\mathrm{N}}{2}\mathrm{tr}(\Sigma^{-1}d\Sigma) - \mathrm{tr}\left(\frac{1}{2}\sum_{i=1}^{N}(\mathbf{x}_i - \mu)^T(-\Sigma^{-1}\,\mathrm{d}\Sigma\,\Sigma^{-1})(\mathbf{x}_i - \mu)\right)$$

Multi-Variable Gaussian Example

Now we give rule that $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, we could have:

$$\mathrm{tr}\Big((\mathbf{x}_i - \mu)^T (-\Sigma^{-1}\,\mathrm{d}\Sigma\,\Sigma^{-1})(\mathbf{x}_i - \mu)\Big)$$

$$= \mathrm{tr}\Big((-\Sigma^{-1}\,\mathrm{d}\Sigma\,\Sigma^{-1})(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T\Big)$$

$$= \mathrm{tr}\Big((\Sigma^{-1})(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T - \Sigma^{-1}\,\mathrm{d}\Sigma\Big)$$

Let $S = \frac{1}{N}\sum_{i=1}^{N}(x-\mu)(x-\mu)^T$. Then, we have:

$$\mathrm{d}l = \mathrm{tr}(\mathrm{d}l) = \frac{\mathrm{N}}{2}\Big(\mathrm{tr}(\Sigma^{-1}d\Sigma) - \mathrm{tr}(\Sigma^{-1}S\Sigma^{-1}\,\mathrm{d}\Sigma)\Big)$$

We could have: $\frac{\partial l}{\partial \Sigma} = (\Sigma^{-1} - \Sigma^{-1}S\Sigma^{-1})^{\mathrm{T}} = 0 \rightarrow \Sigma = S$

Loss function

Classification

- Evaluation metrics
  - Error rate: the proportion of wrong predictions among the total number of cases examined.
  - Accuracy: the proportion of correct predictions among the total number of cases examined.

Error rate

$$E(f;D) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f\left(\boldsymbol{x}_i\right) \neq y_i\right)$$

Accuracy

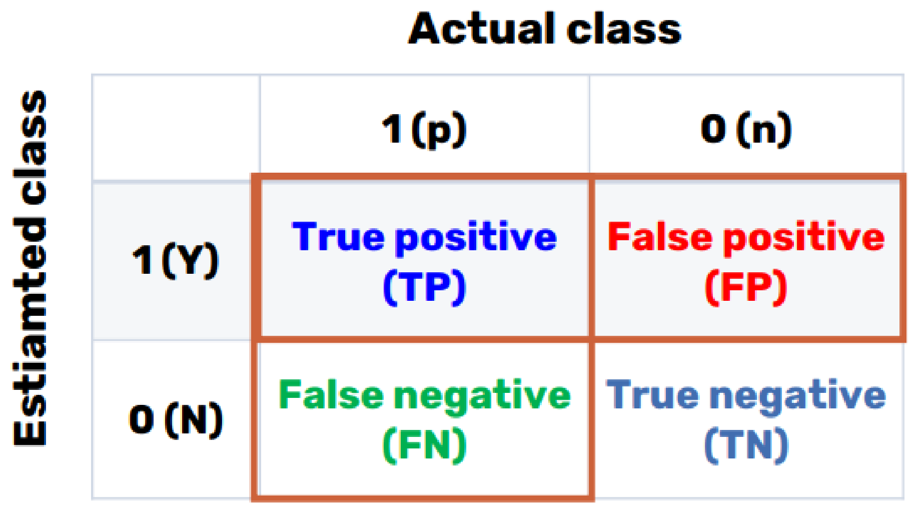
$$\begin{aligned}\operatorname{acc}(f;D) &= \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f\left(\boldsymbol{x}_i\right) = y_i\right) \\ &= 1 - E(f;D)\ .\end{aligned}$$

Classification

- Evaluation metrics

  Suppose that y=1 in presence of a class that we want to detect.

  - Confusion matrix



		Actual class	
Estiamted class		1 (p)	0 (n)
	1 (Y)	True positive (TP)	False positive (FP)
	0 (N)	False negative (FN)	True negative (TN)

  - Precision(How accurate the positive predictions are)

$$P = \frac{TP}{TP + FP}$$

  - Recall(How well the model identifies all positive instances)

$$R = \frac{TP}{TP + FN}$$

Classification

- Evaluation metrics
  - F1-score: combine precision and recall

$$F1 = \frac{2 \times P \times R}{P + R}$$

	Precision	Recall	Average	F1-score
Model1	0.5	0.4	0.45	$\boxed{0.444}$
Model2	0.7	0.1	0.4	0.175
Model3	0.02	1.0	$\boxed{0.51}$	0.0392

The Best is model1

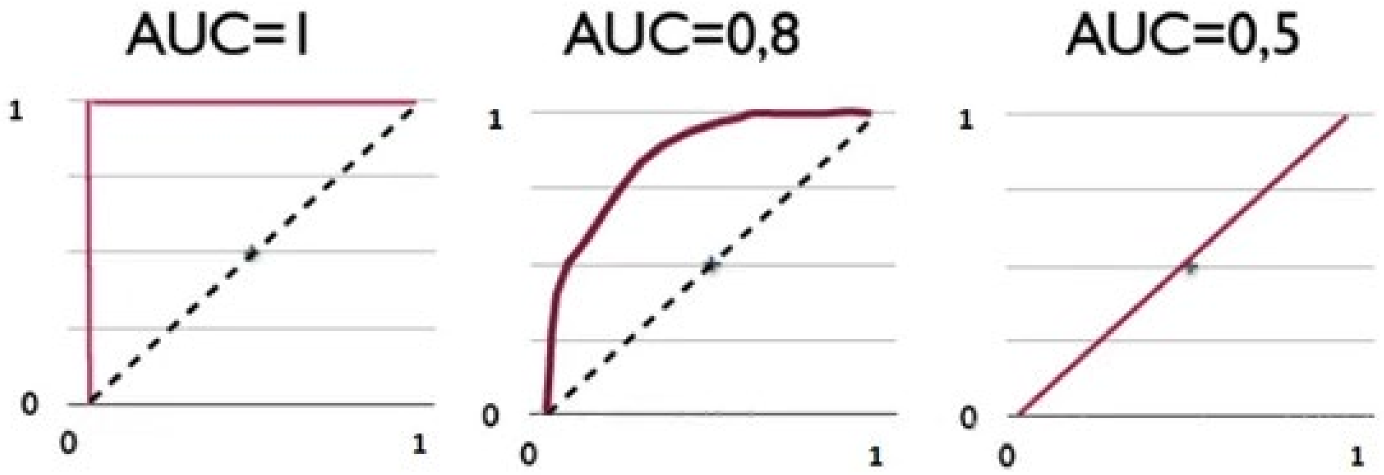
The average does not correctly indicate that Algorithm 3 is the best

Classification

- Evaluation metrics
  - AUROC(Area under Receiver operating characteristics curve) are widely used for imbalanced binary classification tasks.
  - Define the true positive rate (TPR) and false positive rate (FPR):

$$TPR = \frac{TP}{TP + FN} \qquad \mathrm{F}PR = \frac{FP}{TN + FP}$$

  - The ROC curve plots the TPR against FPR as the discrimination threshold is varied from 0 to 1.



Classification

- Logistic regression: apply regression model to classification

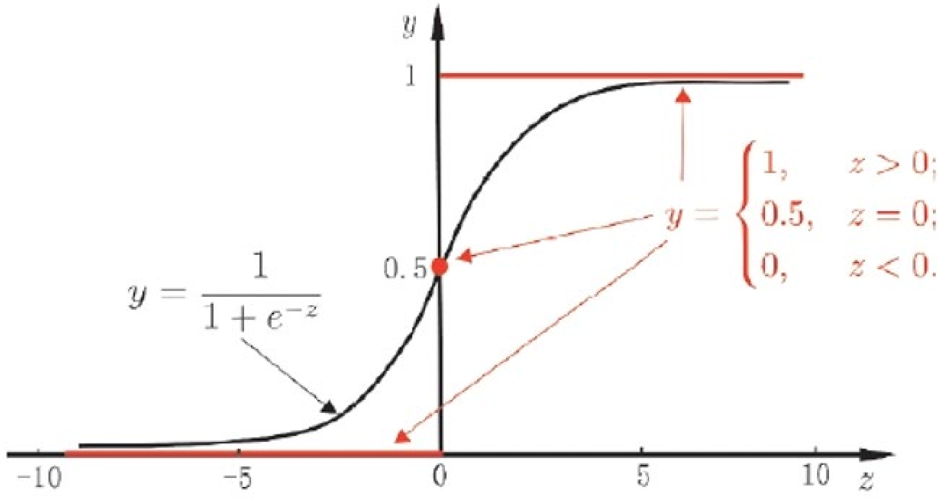
$$E(f;D) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f\left(\boldsymbol{x}_i\right) \neq y_i\right)$$

Linear regression models output continuous value: $z = w^T x + b$

Expect output of classification: $y \in \{0,1\}$

Ideal unit-step function:

$$y = \begin{cases} 0, z < 0 \\ 1, z \geq 0 \end{cases}$$



It’s non-differentiable, discontinuous and non-monotonic, we need a surrogate function.

$$y = \frac{1}{1+e^{-z}}$$

Logistic function, also known as sigmoid function

Classification

- Logistic regression: apply regression model to classification

Using the logistic function as a surrogate:

$$y = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w^T x+b)}}$$

$$\Rightarrow \ \ln \boxed{\frac{y}{1-y}} = \ln \frac{p(y=1|x)}{p(y=0|x)} = w^T x + b$$

Odds, reflecting the relative likelihood of x being a positive example.

Using maximum likelihood estimation, we have the cost function:

$$\mathcal{L}(w,b) = \sum_{i=1}^{m} \ln p(y_i|x_i;w;b)$$

where $p(y_i|x_i;w;b) = y_i p(y=1|x_i;w;b) + (1-y_i) p(y=0|x_i;w;b)$

Ranking Loss

Early Point-wise Loss

It inherits the thought from supervised learning.

The user preferred items are labeled with 1, while others are labeled with 0:

$$\mathcal{L}_{MSE} = \sum_{i \in \mathcal{P}_u, j \in \mathcal{N}_u} (\sigma(s_{ui}) - 1)^2 + \left(\sigma\left(s_{uj}\right) - 0\right)^2$$

Early Point-wise Loss

It also inherits the thought from supervised learning.

However, we view the activated score as probability, and minimize the cross entropy:

$$\mathcal{L}_{BCE} = -\left[\sum_{i \in \mathcal{P}_u, j \in \mathcal{N}_u} \log(\sigma(s_{ui})) + \log\left(1 - \sigma(s_{uj})\right) \right]$$

Learning to Rank

Recommendation is essentially an ranking problem.

We need to teach computer learn to rank.

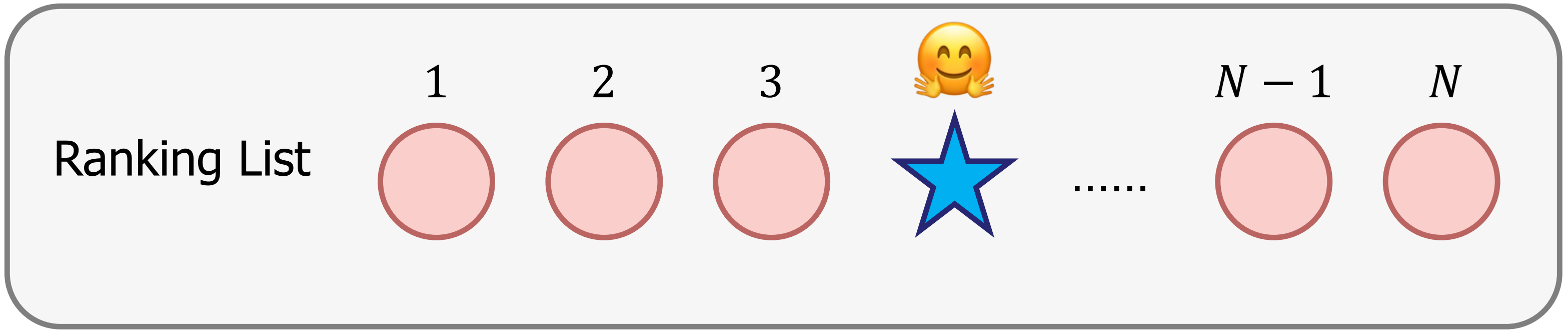
(What?🤔 Computers of course can rank! we have various sort algorithm!)

Learning to rank means, computer should give higher scores to items which people would like (human performance).

Learning to Rank

So, how do we quantify the ranking of an item i?

$$\text{Rank} = \sum_{j \in \mathcal{I}} \mathbb{I}(s_{uj} > s_{ui})$$

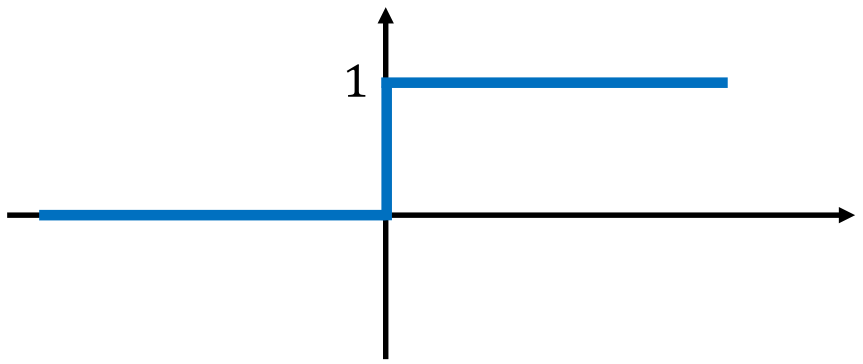


Various Issues

What we want to do is maximize the ranking of items that users prefer:

$$\min \sum_{i \in \mathcal{P}_u} \sum_{j \in \mathcal{I}} \mathbb{I}(s_{uj} > s_{ui})$$

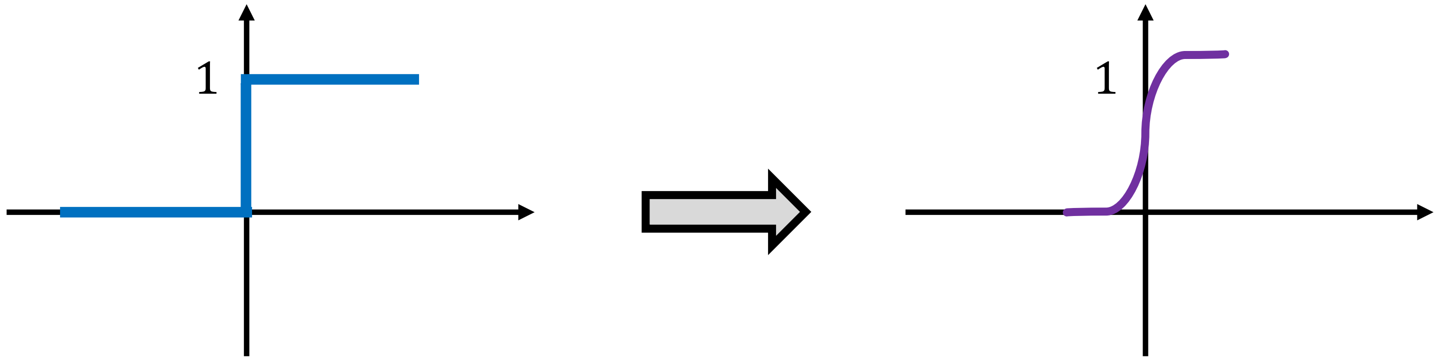
The Indicator/Heaviside function is not differentiable:



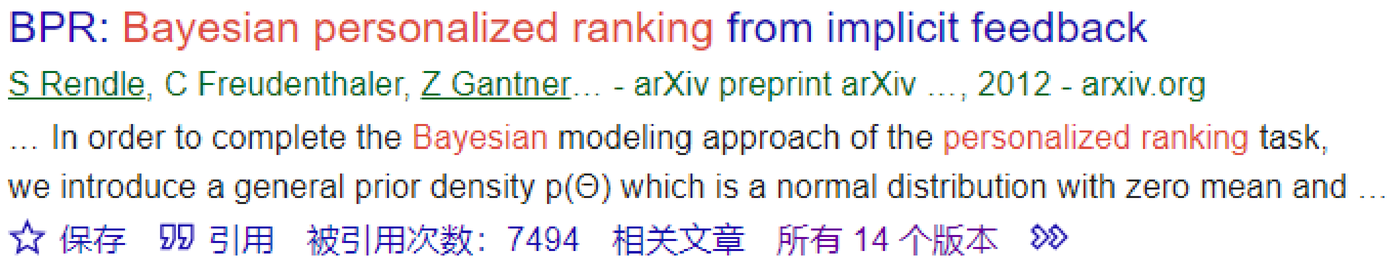
Solutions

- The Indicator/Heaviside function is not differentiable

Find differentiable functions to surrogate it



$$\sum_{i\in\mathcal{P}_u}\sum_{j\in\mathcal{I}}\mathbb{I}(s_{uj} > s_{ui}) \quad \Longrightarrow \quad \sum_{i\in\mathcal{P}_u, j\in\mathcal{N}_u} \ln\sigma(s_{uj} - s_{ui})$$


BPR: Bayesian personalized ranking from implicit feedback

S Rendle, C Freudenthaler, Z Gantner… - arXiv preprint arXiv …, 2012 - arxiv.org

… In order to complete the Bayesian modeling approach of the personalized ranking task, we introduce a general prior density p(Θ) which is a normal distribution with zero mean and …

☆ 保存 99 引用 被引用次数：7494 相关文章 所有 14 个版本 »

Solutions

- More conventional metric:

$$\boldsymbol{DCG} = \sum_{i \in \mathcal{P}_u} \frac{1}{\log(1 + rank_{ui})}$$

How to optimize NDCG

How to optimize NDCG

The NDCG can be bounded as:

$$-\log(NDCG) + \log(|\mathcal{P}_u|) = -\log\left(\frac{1}{|\mathcal{P}_u|}\sum_{i\in\mathcal{P}_u}\frac{1}{\log(1+\pi_{ui})}\right)$$

▷ Jensen Inequality $$\leq \frac{1}{|\mathcal{P}_u|}\sum_{i\in\mathcal{P}_u}\log(\log(1+\pi_{ui}))$$

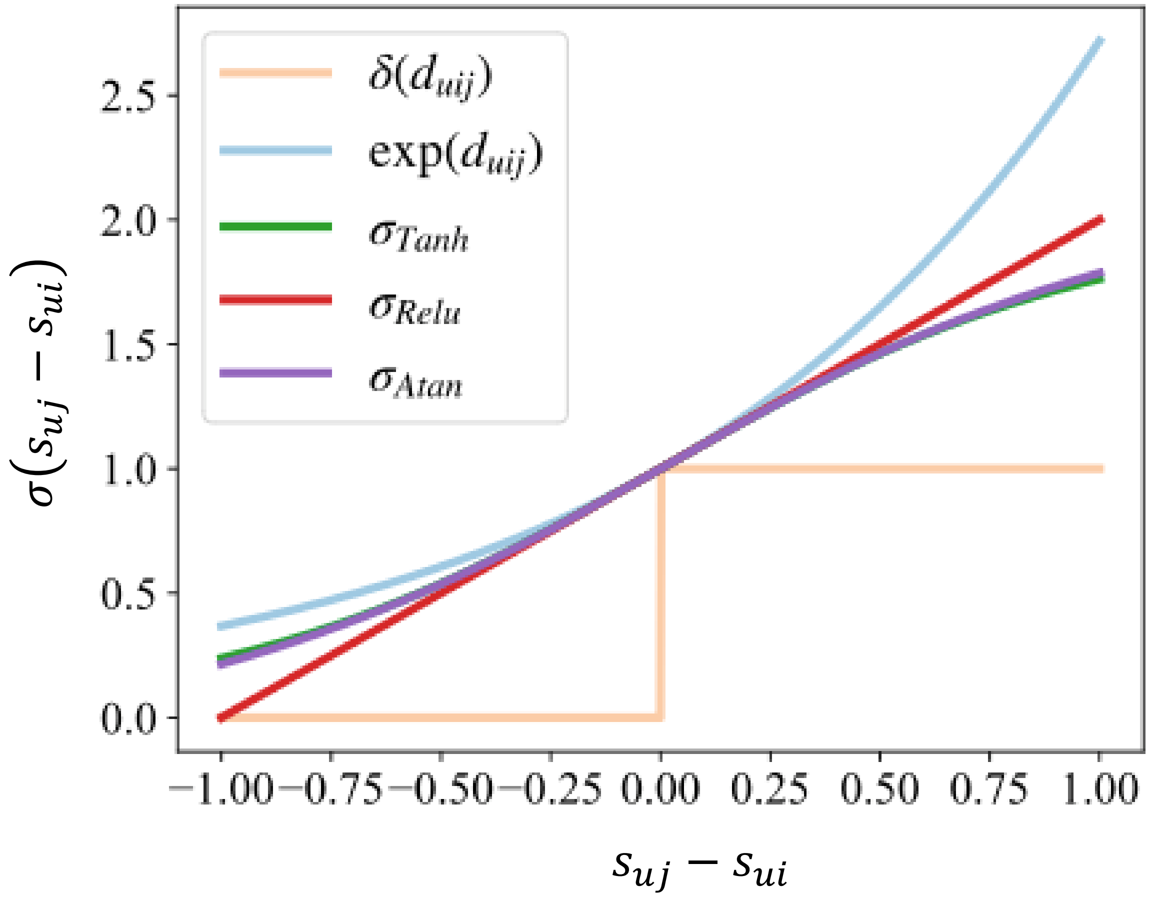
▷ $\log(1+x)\leq x$ $$\leq \frac{1}{|\mathcal{P}_u|}\sum_{i\in\mathcal{P}_u}\log(\pi_{ui})$$

$$= \frac{1}{|\mathcal{P}_u|}\sum_{i\in\mathcal{P}_u}\log\left(\sum_{j\in\mathcal{I}}\mathbb{I}(s_{uj}>s_{ui})\right)$$

$$\leq ? \frac{1}{|\mathcal{P}_u|}\sum_{i\in\mathcal{P}_u}\log\left(\sum_{j\in\mathcal{I}}\boldsymbol{\sigma}(s_{uj}>s_{ui})\right)$$

How to optimize NDCG

Choice two: Using other functions $\sigma(\cdot)$ to surrogate the upper bound:



How to optimize NDCG

The form of these functions can be summarized as:

$$\log\left(\sum_{j\in\mathcal{J}}\sigma\left(s_{uj}-s_{ui}\right)^{\frac{1}{\tau}}\right)$$

This is our recently published work: *PSL: Rethinking and Improving Softmax Loss from Pairwise Perspective for Recommendation (NeurIPS 2024).* 🔥 🔥 🔥

THANK YOU!