Machine Learning Cheat Sheet – Part 1

October 14, 2025

Supervised Learning

Naive Bayes

Generative model: model the prior and class-conditional distribution (each class has a particular distribution of features) by observation / assumption to use MLE on class conditional probability and Bayes decision rule (BDR) to get target probability:

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y) \cdot p(y)}{p(x)} = \frac{p(x|y) \cdot p(y)}{\sum_y p(x|y)} \tag{1}$$

Naive Bayes: assume features are independent, i.e. $p(\boldsymbol{x} = (x_1, x_2)|y) = p(x_1|y)p(x_2|y)$.

(class) **Model** $p(y)$. *Bernoulli distribution*: $p(y) = \pi^{\mathbb{1}(y=1)} \cdot (1-\pi)^{\mathbb{1}(y=0)}$.

MLE: $\pi^* = \arg\max \sum_{i=1}^{N} \log p(y_i) \Rightarrow \pi = \frac{\#(y=1)}{\#(y=0) + \#(y=1)}$.

(observation) **Model** $p(x|y)$. ① *Gaussian*: $p(x|y=c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{(-\frac{1}{2\sigma_c^2}(x-\mu_c)^2)}$.

MLE: $\mu_c^* = \frac{1}{N}\sum_{i=1}^{N} x_i, {\sigma_c^*}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_c^*)^2$. $(x_i|y=c)$

② *Poisson*: $p(x|y=c) = \frac{1}{x!}e^{-\mu_c}{\mu_c}^x, x \in \{0,1,2,\cdots\}, (\mu_c = \bar{x}|y=c)$.

MLE: $\mu_c^* = \frac{1}{N}\sum_{i=1}^{N} x_i$. $(x_i|y=c)$

Laplace smoothing (smoothed MLE): $\pi_j = \frac{N_j+\alpha}{N+2\alpha}$ *to prevent overfitting.*

Naive Bayes Example – SMS spam

Bag-of-Words (BoW) model: 1. build vocabulary $\mathcal{V}$. 2. text t to vector $x \in \mathbb{R}^V$
`sklearn.feature_extraction.text.`
① count occurrence to vectorize: `CountVectorizer(stop_words=xx, max_features=xx)`
② *Term-Frequency (TF)*: $x_j = \frac{w_j}{|D|} = \frac{\text{\# word } j}{\text{\# words in document}}$.
③ *TF Inverse Document Frequency (TF-IDF)*: $x_j = \frac{w_j}{|D|}\log\frac{N}{N_j}$
$IDF(j) = \log\frac{N}{N_j} = \frac{\text{\# documents}}{\text{\# documents with word } j}$. `TfidfVectorizer`
④ `HashingVectorizer`

NB Gaussian $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}}}$
MLE: $\boldsymbol{\mu}^* = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i, \boldsymbol{\Sigma}^* = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$.

NB Multinomial: $p(\boldsymbol{x}|y) = \frac{(\sum_j x_j)!}{\prod_j x_j!}\left(\prod_j \pi_{j,y}^{x_j}\right)$, where x_j is the frequency of word j ($\sum_j x_j = 1$), $\pi_{j,y}$ is the probability of word w_j in class y ($\sum_{j=1}^{V} \pi_{j,y} = 1$).

L_1 & L_2

L_1 regularization: 1. treat all errors equally 2. encourage more sparsity 3. decision is more likely to be aligned with the coordinate axis

L_1 error: robust to outlier, no preference to reduce the larger errors.

Generative → Discriminative

Generative: class-conditional distribution (CCD) $\xrightarrow{\text{Bayes rule}}$ classifier $p(y|x)$, can be used to small dataset.
Discriminative: classifier $p(y|x)$ directly

$$\frac{p(y=1|x)}{p(y=2|x)} > 1 \Leftrightarrow \log\frac{p(y=1|x)}{p(y=2|x)} \overset{\text{Bayes}}{=} \log\frac{p(x|y=1)p(y=1)}{p(x|y=2)p(y=2)} > 0 \tag{1}$$

sharing σ^2: $\log\frac{p(x|y=1)}{p(x|y=2)} = \log\frac{\prod_{i=1}^{D}\mathcal{N}(x_i|\mu_i,\sigma^2)}{\prod_{i=1}^{D}\mathcal{N}(x_i|\nu_i,\sigma^2)} = \frac{1}{2\sigma^2}\sum_{i=1}^{D}\left[2(\mu_i - \nu_i)x_i - \mu_i^2 + \nu_i^2\right]$

$$\frac{p(y=1|x)}{p(y=2|x)} > 1 \Leftrightarrow \sum_{i=1}^{D}\underbrace{\frac{1}{\sigma^2}(\mu_i - \nu_i)}_{w_i}x_i + \underbrace{\frac{1}{2\sigma^2}\sum_{i=1}^{D}(\nu_i^2 - \mu_i^2) + \log\frac{\pi_1}{\pi_2}}_{b} > 0 \tag{2}$$

Linear (Binary) Classifier

$f(x) = w^\top x + b$. $f(x) > 0 \to$ class $1(y=1)$, $f(x) < 0 \to$ class $2(y=-1)$

Logistic Regression (Binary) Classifier

$f(x) = w^\top x + b, \sigma(z) = \frac{1}{1+e^{-z}}, p(y|x) = \sigma(y \cdot f(x))$:

$$p(y=+1|x) = \sigma(f(x)), \quad p(y=-1|x) = 1 - \sigma(f(x)) = \sigma(-f(x)) \tag{3}$$

MLE: $(w^*, b^*) = \underset{w,b}{\arg\max}\sum_{i=1}^{N}\log p(y_i|x_i) = \underset{w,b}{\arg\min}\sum_{i=1}^{N}\log(1 + e^{-y_i f(x_i)})$
$z_i \triangleq y_i f(x_i) > 0 \Rightarrow$ classified correctly, $z_i < 0 \Rightarrow$ wrongly, $z_i = 0 \Rightarrow$ boundary
Logistic regression only forms a linear decision surface.

Logistic loss function: $L(z_i) = \log(1 + \exp(-z_i))$. convex $\to$ one optimum

Regularization: $(w^*, b^*) = \underset{w,b}{\arg\max}\left[\log p(w) + \sum_{i=1}^{N}\log p(y_i|x_i)\right]$.

① Gaussian: $p(w) = \mathcal{N}(0, \frac{C}{2}I) \Rightarrow \log p(w) = -\frac{1}{C}w^T w + \text{constant}$

$\Rightarrow (w^*, b^*) = \underset{w,b}{\arg\max}\frac{1}{C}w^T w + \sum_{i=1}^{N}\log(1 + \exp(-y_i f(x_i))), : w^T w = \sum_{j=1}^{d} w_j^2$.
Large $C \to$ big $w \Leftrightarrow$ Small $C \to$ small w. Equal to L_1 regularization.

Multiclass logistic regression

$$p(y=c|x) = \text{softmax}(f(x)) = \frac{e^{f_c(x)}}{e^{f_1(x)} + \cdots + e^{f_K(x)}}, f_c(x) = w_c^T x. \tag{4}$$

MLE: ① likelihood: $p(y|x) = \prod_{j=1}^{K} p(y=j|x)^{y_j}$. ② log-likelihood: $\log p(y|x)$.
③ (min) cross-entropy loss: $-\log p(y|x) = -\sum_{j=1}^{K} y_j \log p(y=j|x)$

$$\max_{\{w_j\}}\sum_{i=1}^{N}\log p(y_i|x_i) = \max_{\{w_j\}}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\log p(y=j|x_i) \tag{5}$$

where $y = [y_1, \cdots, y_K]$ is one-hot vector.

Support Vector Machine (SVM)

Margin distance: $\gamma = \min_i d_i = \min_i \frac{|f(x_i)|}{\|w\|} = \min_i \frac{|w^\top x_i + b|}{\|w\|}$
Support vector: points on $y = w^\top x + b$.
Normalization: $\frac{|aw^T x_i + ab|}{||aw||} = \frac{|w^T x_i + b|}{||w||} \Rightarrow$ just let $f(x_i) = w^T x_i + b = 1$.

Maximize margin: $(\hat{w}, b) = \underset{w,b}{\arg\max}\frac{1}{\|w\|}$ s.t. $\min_i |f(x_i)| = 1$

$\Leftrightarrow (\hat{w}, b) = \underset{w,b}{\arg\min}\frac{1}{2}\|w\|^2 = \frac{1}{2}w^\top w$ s.t. $y_i f(x_i) \geqslant 1, \forall\, i$. (for binary class)

Lagrangian: $L(x,\lambda) = f(x) - \lambda g(x)$
$\Rightarrow$ *SVM dual*: $\min L(w,\alpha) = \frac{1}{2}w^T w - \sum_i \alpha_i[y_i(w^T x_i + b) - 1], w = \sum_{i=1}^{N}\alpha_i y_i x_i$

$$\underset{\alpha}{\arg\max}\sum_i \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j \text{ s. t. } \sum_{i=1}^{N}\alpha_i y_i = 0, \quad \alpha_i \geqslant 0 \tag{1}$$

non-separable data: $(\hat{w}, b) = \underset{w,b}{\arg\min}\ \frac{1}{2}w^\top w$, s.t. $y_i f(x_i) \geqslant 1 - \xi_i, \xi_i \geqslant 0, \forall\, i$

soft-SVM: $(\hat{w}, b) = \underset{w,b}{\arg\min}\ \frac{1}{2}w^\top w + \sum_{i=1}^{N} C\xi_i$, s.t. $y_i f(x_i) \geqslant 1 - \xi_i, \xi_i \geqslant 0, \forall\, i$

objective function: $\underset{w,b}{\arg\min}\frac{1}{C}w^T w + \underbrace{\sum_{i=1}^{N}\max(0, 1 - y_i f(x_i))}_{\text{hinge loss}}$

prediction: need training data to calculate similarity $f(x) = \sum_i \alpha_i y_i k(x_i, x) + b$.

Tricks

① Normalization for Logistic regression and SVM, due to their sensitivity to absolute value. ② Apply weights to loss of unbiased data. ③ Bigger weights to loss of more important class. ④ Change threshold to one class.

Kernel SVM

idea: learn linear classifier in high-dim space $\phi(x) \in \mathbb{R}^D$ rather than $x \in \mathbb{R}^d$
kernel function: $k(x_i, x_j) = \phi(x_i)^\top\phi(x_j)$, which is $x_i^\top x_j$ in *dual SVM*
example: ① polynomial kernel: $k(x,x') = (x^T x')^p = (\sum_{i=1}^{d} x_i x_i')^p$
② RBF(radial basis function): $k(x,x') = e^{-\gamma\|x-x'\|^2}$. $\gamma\uparrow$: smooth function.
③ Laplacian kernel: $k(x,x') = \exp(-\alpha||x - x'||)$

kernel SVM: $\hat{y} = \text{sign}(\sum_{i=1}^{N}\alpha_i y_i k(x_i, x_*) + b)$, where

$$\alpha = \underset{\alpha}{\arg\max}\sum_i \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j k(x_i, x_j) \text{ s. t. } \sum_{i=1}^{N}\alpha_i y_i = 0, \quad \alpha_i \geqslant 0 \tag{2}$$

Kernel matrix: K $= \left[k(x_i, x_j)\right]_{i,j}$, where $k(x_i, x_j) = \phi(x_i)^\top\phi(x_j)$.

K is a positive semi-definite matrix, i.e. $z^T K z \geq 0, \forall z$.
Kernel computation for high dimension is of high cost. Memory usage is based on the number of support vectors kept.

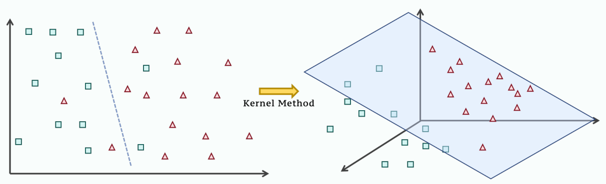

Figure 1: Kernel trick

Regression

Linear Regression. $\hat{y} = w_0 + w_1 x_1 + \ldots + w_d x_d = \boldsymbol{w}^\top \boldsymbol{x} = f(\boldsymbol{x}_i)$, where $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^\top$, $\boldsymbol{x} = [1, x_1, \ldots, x_d]^\top$
Ordinary Least Squares (OLS). $\min_{\boldsymbol{w}} \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2 = \min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}^\top \boldsymbol{w}\|^2$, where $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_N]$ is the data matrix, $\boldsymbol{y} = [y_1, \ldots, y_N]$.
$\Rightarrow \min_{\boldsymbol{w}} \boldsymbol{y}^T \boldsymbol{y} - 2\boldsymbol{y}^T \boldsymbol{X}^T \boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{X}\boldsymbol{X}^T \boldsymbol{w} \Rightarrow \boldsymbol{w}^* = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}$, i.e. *pseudo-inverse*

Ridge Regression. $\min_{\boldsymbol{w}} \alpha ||\boldsymbol{w}||_2^2 + \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2$, where $||\boldsymbol{w}||_2^2 = \sum_{j=0}^d w_j^2$
$\Rightarrow \boldsymbol{w}^* = (\boldsymbol{X}\boldsymbol{X}^T + \alpha \boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$. (In OLS, $\boldsymbol{X}\boldsymbol{X}^\top$ may be non-invertible)

LASSO *(Least absolute shrinkage and selection operator).* $\min_{\boldsymbol{w}} \alpha ||\boldsymbol{w}||_1 + \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2$, where $||\boldsymbol{w}||_1 = \sum_{j=0}^d |w_j|$. *LASSO* encourage more sparsity

Sparsity Constraints. $\min_{\boldsymbol{w}} \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2$, s.t. $||\boldsymbol{w}||_0 \leqslant K$, where $||\boldsymbol{w}||_0 =$ # non-zero entries. nonconvex and NP-hard v.s. *Ridge* and *LASSO* are convex

OMP *(Orthogonal Matching Pursuit).* Iteratively and greedily selects the most related feature to current residual error.

Algorithm 1 Orthogonal Matching Pursuit (OMP)

Initialize the residual: $\boldsymbol{r} = \boldsymbol{y}$
for $t = 1$ to K **do**
  Find the most correlated feature: $j = \arg\max_j |\boldsymbol{r}^T \boldsymbol{x}_j|$, where $\boldsymbol{x}_j$ is the j-th row of $\boldsymbol{X}$ (the j-th feature).
  Compute the weight: $\boldsymbol{w}_j = \arg\min_{\boldsymbol{w}_j} \|\boldsymbol{r} - \boldsymbol{x}_j \boldsymbol{w}_j\|^2$
  Update the residual: $\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{x}_j \boldsymbol{w}_j$
end for

RANSAC *(**RAN**dom **SA**mple **C**onsensus).* Split the data into inliers (good data) and outliers (bad data) and learn the model only from the inliers.

Algorithm 2 RANSAC (Random Sample Consensus)

Given: training set $D = \{\boldsymbol{x}_i, y_i\}$, threshold ϵ, loss function $L = \sum l(x, y)$
Classify all data as inlier or outlier by calculating the prediction errors l and comparing to the threshold ϵ. (typically set as the median absolute deviation (MAD) of y, i.e. $\text{median}(|y_i - \text{median}(y)|)$)
The set of inliers is called the *consensus set*.
Save the model with the highest number of inliers.
Use the largest consensus set to learn the final model.

Nonlinear Regression. $\hat{y} = \boldsymbol{w}^\top \phi(x)$, where $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^\top$.
Kernel Ridge Regression. $\min_{\boldsymbol{w}} \|\boldsymbol{y} - \Phi \boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2$, where $\Phi = [\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)]^\top \in \mathbb{R}^{N \times d}, \boldsymbol{w} \in \mathbb{R}^d \Rightarrow \boldsymbol{w} = (\Phi^\top \Phi + \lambda \boldsymbol{I})^{-1} \Phi^\top \boldsymbol{y} \Rightarrow \hat{y} = \hat{\boldsymbol{k}}^\top (K + \lambda I)^{-1} \boldsymbol{y}$, where $K = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)] \in \mathbb{R}^{N \times N}$, $\hat{\boldsymbol{k}} = [k(\boldsymbol{x}_1, \hat{\boldsymbol{x}}), \ldots, k(\boldsymbol{x}_N, \hat{\boldsymbol{x}})]^\top \in \mathbb{R}^N$.
Gaussian Process Regression (GPR). *Gaussian process:* an sequential random variable set whose any finite subset is joint Gaussian distributed. $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$ where f, m, k are value, mean and covariance function separately. $f_1, \ldots, f_N | x_1, \ldots, x_N \sim \mathcal{N}(0, K)$, where K is the kernel matrix.
GPR: ① observation noise: $y = f + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$, $p(y|f) = \mathcal{N}(y|f, \sigma^2 I)$ ② function prior: $f \sim \mathcal{GP}(0, k(x, x'))$ ③ train: $p(f|X, y) = \frac{p(y|X,f)p(f|X)}{p(y|X)} = \frac{p(y|f)p(f)}{p(y|X)}$ ④ inference: $p(\hat{f}|\hat{x}, X, y) = \int p(\hat{f}|\hat{x}, f) p(f|X, y) df$. ⑤ prediction: $p(f_*|x_*, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(f_*|\mu_*, \sigma_*^2)$, $\mu_* = \boldsymbol{k}_*^T (\boldsymbol{K} + \sigma^2 I)^{-1} \boldsymbol{y}$, $\sigma_*^2 = k_{**} - \boldsymbol{k}_*^T (\boldsymbol{K} + \sigma^2 I)^{-1} \boldsymbol{k}_*$, where $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$.
Support Vector Regression (SVR). ① objective: $\min_{\boldsymbol{w}, b} \sum_{i=1}^N \left|y_i - (\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right|_\epsilon + \frac{1}{C} ||\boldsymbol{w}||^2$, where $|z|_\epsilon = \begin{cases} 0, & |z| \leqslant \epsilon \\ |z| - \epsilon, & |z| > \epsilon \end{cases}$

Regression – Model Ensemble

Idea: combine multiple regression model together to form a better algorithm. ① ***bagging***: train multiple models from random selection of training data. ② ***boosting***: train multiple models which focus on errors made by previous one.

① ***Random Forest Regression.*** Use **bagging** to make an ensemble of *Decision Tree Regressor*. Random subset of data is used to train different tree, whose nodes use different features. The prediction is the average value of each tree. **RFR is sensitive to outliers.**
② ***XGBoost Regression.*** *(eXtreme Gradient Boosting)* Use **boosting** to combine a set of less accurate models to create a accurate model. Weak learner fits the gradient of the loss: $h_t(x) \approx \frac{dL}{df}$, $f_t(x) = f_{t-1}(x) - \alpha_t h_t(x) \approx f_{t-1}(x) - \alpha_t \frac{dL}{df_{t-1}}$.
XGBoost / AdaBoost can run in parallel.

Tricks

Improve accuracy: more features + complex features + complex model

Neural Network

Neural Network

Perceptron. Model a single neuron $y = f(w^T x) = f(\sum_{j=0}^d w_j x_j) = \begin{cases} 1 & \text{if } w^\top x \geq 0 \\ -1 & \text{otherwise} \end{cases}$, *loss*: $E(w) = \sum_{i=1}^N L(z_i) = \sum_{i=1}^N \max(0, -z_i), z_i = y_i w^\top x_i$.

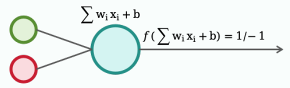

Figure 1: One perceptron

Perceptron Alg. / SGD. $w \leftarrow w + \eta y_i x_i = w - \eta g_w(x_i)$, where x_i is misclassified
Result (1st in ML): 1. only converge on linearly separable data; 2. # iteration $\sim \frac{1}{m}$, where m is the separation (margin) between classes.
Multi-layer Perceptron now called *neural network*.
Multi-class Logistic regression. class labels $y \in \{1, \cdots, C\}$, class vector $\boldsymbol{y} \in \mathbb{R}^C$ is one-hot vector. Output $p(y = j|\boldsymbol{x}) = f_j(\boldsymbol{x}) = s_j(\boldsymbol{g}(\boldsymbol{x})) = \frac{\exp(g_j(\boldsymbol{x}))}{\sum_{k=1}^C \exp(g_k(\boldsymbol{x}))}$, $g_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x}$, for $j \in \{1, \cdots, C\}$.
MLE: $\log p(\boldsymbol{y}|\boldsymbol{x}) = \log \prod_{j=1}^C f_j(\boldsymbol{x})^{y_j} = \sum_{j=1}^C y_j \log f_j(\boldsymbol{x}) = \boldsymbol{y}^T \log \boldsymbol{f}(\boldsymbol{x})$
$\boldsymbol{W}^* = \arg\max_{\boldsymbol{W}} \sum_{i=1}^N \log p(\boldsymbol{y}_i|\boldsymbol{x}_i) = \arg\max_{\boldsymbol{W}} \sum_{i=1}^N \boldsymbol{y}_i^T \log \boldsymbol{f}(\boldsymbol{x}_i) = \arg\min_{\boldsymbol{W}} \sum_{i=1}^N \{-\sum_{j=1}^C y_{ij} \log f_j(\boldsymbol{x}_i)\}$.
cross-entropy loss: $L(\boldsymbol{y}, \boldsymbol{f}) = -\sum_{j=1}^C y_j \log f_j(\boldsymbol{x})$.
chain rule: $\frac{dL}{d\boldsymbol{w}_j} = \frac{dL}{dg} \frac{dg^T}{d\boldsymbol{w}_j} = \frac{dL}{df} \frac{df}{dg} \frac{dg^T}{d\boldsymbol{w}_j} = \boldsymbol{x}(f_j(\boldsymbol{x}) - y_j)$.
Multi-layer Perceptron(MLP). Add hidden layers between inputs and outputs. $\boldsymbol{h} = f(\boldsymbol{W}^T \boldsymbol{x})$, where $f(\cdot)$ is the *activation function*.
Activation function: ① Sigmoid: $\mathbb{R} \mapsto [0, 1], f(x) = \frac{1}{1+e^{-x}}$ ② Tanh: $\mathbb{R} \mapsto [-1, 1], f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ ③ Rectifier Linear Unit (ReLU): $f(x) = \max(0, x)$.
Overfitting: the training loss decreases, but the validation loss increases.
Early stopping: stop training when the validation loss is stable (change below a threshold) for a number of iterations to prevent overfitting.
Universal Approximation Theorem: A *MLP* with one hidden layer and finite nodes can approximate any continuous function up to a desired error.
Vanishing Gradient problem: Backprop recursively multiplies gradients causing numerical values get smaller.