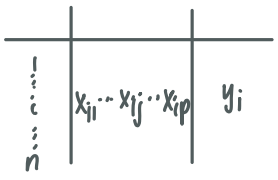
Deep Learning

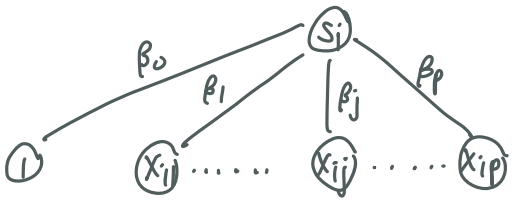
neural network

Recall regression



$$s_i = \beta_0 + x_{i1}\beta_1 + \dots + x_{ij}\beta_j + \dots + x_{ip}\beta_p = \beta_0 + x_i^T\beta$$

$$= \langle w, x \rangle + b$$



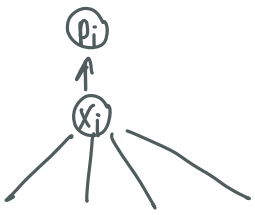
$$y_i \sim N(s_i, \sigma^2)$$

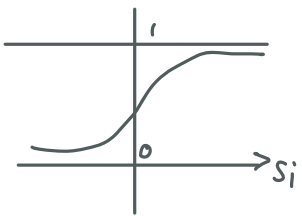
$$\text{Loss} = \frac{1}{2}\sum_{i=1}^{n}(y_i - s_i)^2 \qquad \frac{\partial \text{Loss}}{\partial s_i} = -(y_i - s_i) = -e_i$$

Logistic regression

$y_i \in \{0, 1\}$

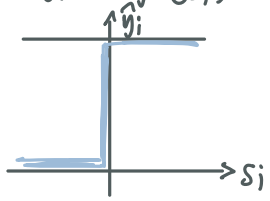
$$p_i = \text{sigmoid}(s_i) = \frac{e^{s_i}}{1 + e^{s_i}}$$





perceptron (hard decision)

$$\hat{y}_i = \text{sign}(s_i)$$



SVM
max margin

nonlinearity, elementwise/coordinatewise, given
non-linear rectification

logistic regression (soft decision)

$$P(y_i = 1 \mid s_i) = p_i$$

$$P(y_i = 0 \mid s_i) = 1 - p_i$$

$$\boxed{y_i \sim \text{Bernoulli}(p_i)}$$

cross-entropy loss (maximum likelihood)

$$\text{likelihood} = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

$$= \prod_{i=1}^{n} \left(\frac{e^{s_i}}{1+e^{s_i}}\right)^{y_i}\left(\frac{1}{1+e^{s_i}}\right)^{1-y_i}$$

$$= \prod_{i=1}^{n} \left(\frac{e^{s_i y_i}}{1+e^{s_i}}\right)$$

$$\text{log-lik} = \sum_{i=1}^{n}\left[s_i y_i - \log(1+e^{s_i})\right]$$

$$\text{cross-entropy loss} = -\text{log-lik} = -\sum_{i=1}^{n}\left[s_i y_i - \log(1+e^{s_i})\right]$$

$$\frac{d\,\text{loss}}{ds_i} = -\left(y_i - \frac{e^{s_i}}{1+e^{s_i}}\right) = -(y_i - p_i) = -e_i$$

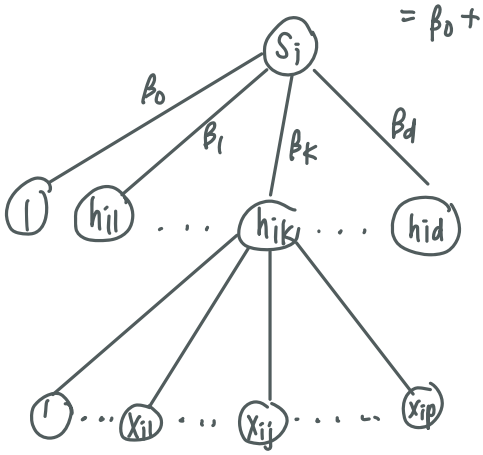
result of derivative of loss func. w.r.t. s_i is consistent for both linear & logistic regression

non-linear model

hidden variable (node)
feature
nonlinear function of x_i

$$s_i = f(x_i) = \beta_0 + h_{i1}\beta_1 + \ldots + h_{ik}\beta_k + \ldots + h_{id}\beta_d$$

$$= \beta_0 + h_i^T \beta$$

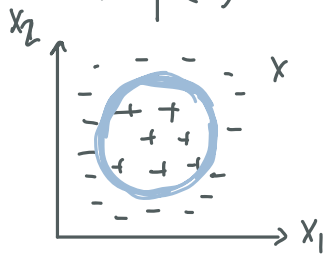


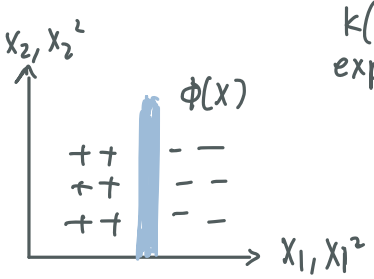
$$h_i = \begin{pmatrix} h_{i1} \\ \vdots \\ h_{ik} \\ \vdots \\ h_{id} \end{pmatrix}$$

SVM/adaboost/tree can all be considered within this framework

In kernel SVM

$$h_i = \phi(x_i)$$



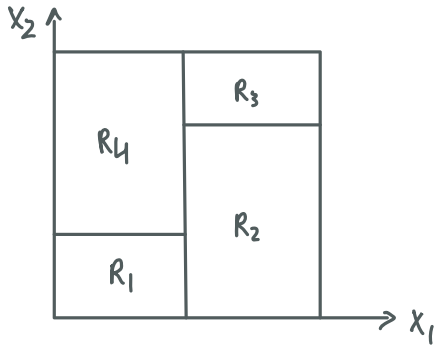


implicit

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

explicit

Trees



$$s_i = \sum_{k=1}^{d} c_k \mathbf{1}(x_i \in R_k)$$

$c_k \rightarrow \beta_k$

$\mathbf{1}(x_i \in R_k) \rightarrow h_{ik}$ 0/1 binary

Adaboost

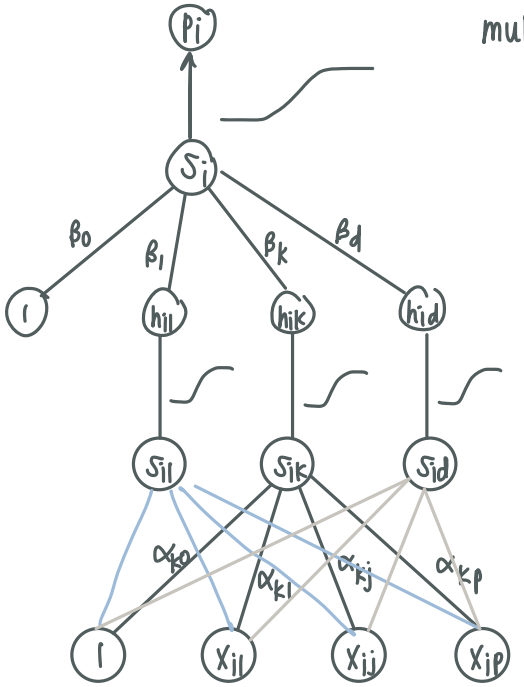
$$s_i = \sum_{k=1}^{d} h_{ik} \beta_k$$

$\downarrow$

Decision tree
weak classifier
0/1 binary

---

perceptron on top of perceptrons



multilevel perceptron (MLP)

another layer of logistic regression

$$\frac{\partial Loss}{\partial \beta_k} = \sum_{i=1}^{n} \frac{\partial Loss}{\partial s_i} \frac{\partial s_i}{\partial \beta_k}$$

$$= -\sum_{i=1}^{n} e_i h_{ik}$$

$$\frac{\partial Loss}{\partial \alpha_{kj}} = \sum_{i=1}^{n} \frac{\partial Loss}{\partial s_i} \frac{\partial s_i}{\partial h_{ik}} \frac{\partial h_{ik}}{\partial s_{ik}} \frac{\partial s_{ik}}{\partial \alpha_{kj}}$$

$$= -\sum_{i=1}^{n} e_i \beta_k \cdot \text{sigmoid}'(s_{ik}) \cdot x_{ij}$$

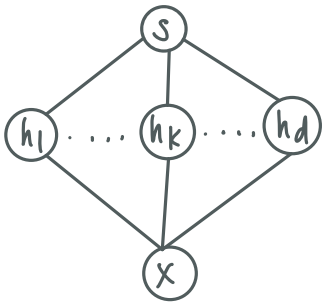
chain rule

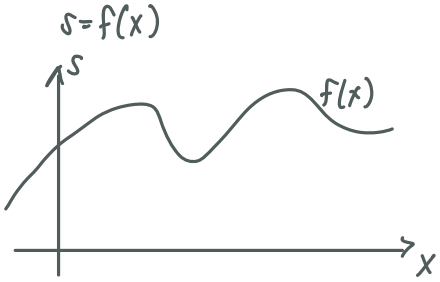
error back-propagation

main algorithm for training neural network.

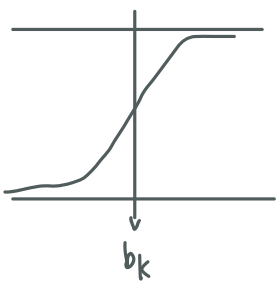
Drop subindex i

Consider 1-dimensional x





$$s = \beta_0 + \sum_{k=1}^{d} \text{sigmoid}\left(\alpha_k (x - b_k)\right) \cdot \beta_k \qquad \text{s.t.} \quad \alpha_{k0} = -\alpha_k \cdot b_k$$



α_k = sharpness of jump

b_k = location of jump

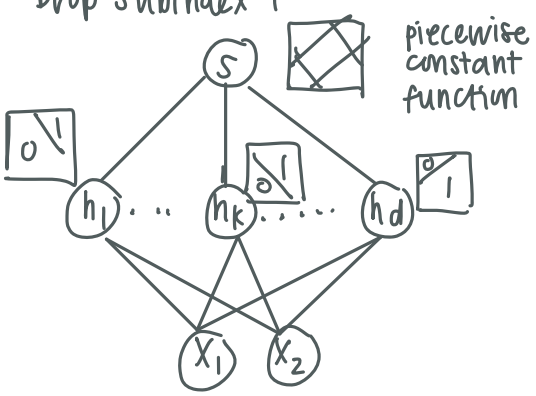
β_k = magnitude of jump

smoothed staircase

NN in 1-dimensional situation

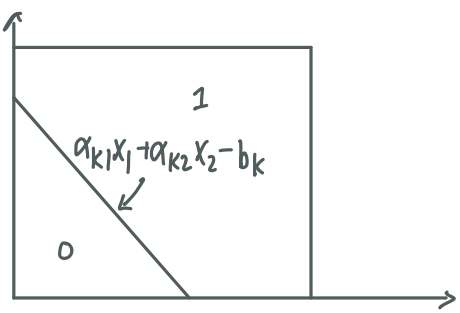
2D laminar surface

Drop subindex i

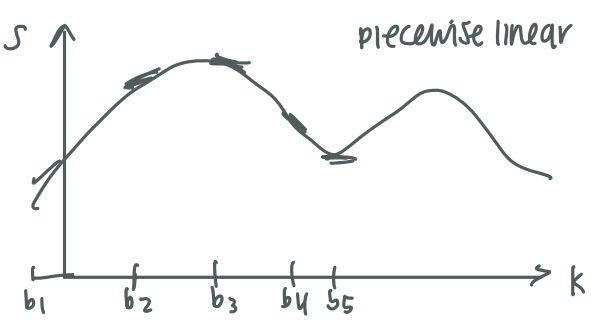


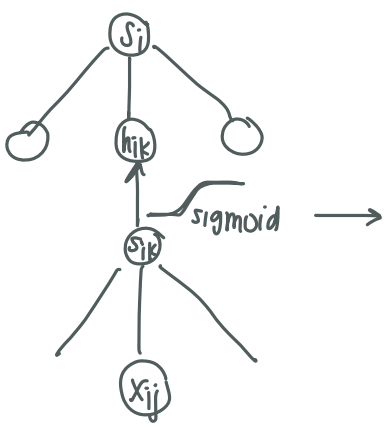
$b_k = -\alpha_{k0}$

$$s = \beta_0 + \sum_{k=1}^{d} \text{sigmoid}\left(\alpha_{k1} x_1 + \alpha_{k2} x_2 - b_k\right) \beta_k$$

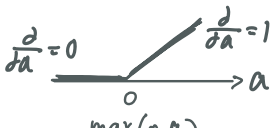


better approximation





ReLU rectified linear unit



leaky ReLU