Lecture 12

Thursday, October 12, 2023   10:04 AM

Linear regression

$$x_1\ x_2 \dots x_d \longrightarrow y$$

real-valued

Logistic regression

$$x_1\ x_2 \dots x_d \longrightarrow y$$

categorical or symbolic

$$\{c_1, c_2, \dots, c_k\}$$

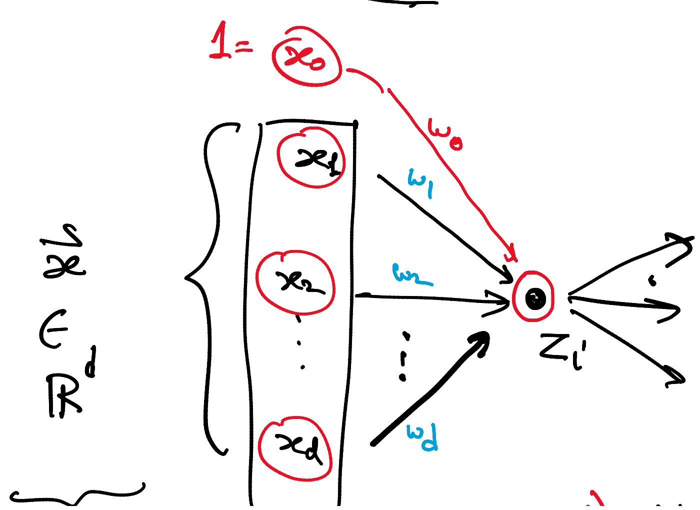
linear/identity

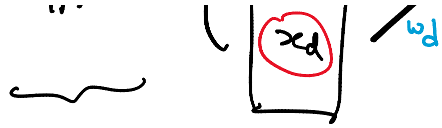
$$f(x_1, x_2, x_d) = w_1 x_1 + w_2 x_2 + \dots + w_d x_d + w_0 = \tilde{w}^T \tilde{x}$$

$$f(x_1, x_2, x_d) = \underbrace{\boxed{\text{softmax}}}_{\text{non-linear}}\Big(\underbrace{\tilde{w}^T \tilde{x}}_{\text{linear}}\Big) \quad \text{or} \quad \underbrace{\boxed{\text{sigmoid}}\big(\tilde{w}^T \tilde{x}\big)}_{\text{binary case}}$$

MLP: Multilayer Perceptrons (Feed Forward Neural Nets)

<u>Multi-layer</u> <u>logistic regression</u>   (artificial NN)





input
data points

1) $Net_{z_i} = \vec{w}^T \vec{x} = w_0 + w_1 x_1 + \dots + w_d x_d$

2) activation function

$$z_i = f(Net_{z_i})$$

0) Step function

$$z_i = f(net_i) = \begin{cases} 1 & \text{if } \boxed{net_i > 0} \\ 0 & \text{otherwise} \end{cases}$$

$\underbrace{w_1 x_1 + w_2 x_2 + \dots + w_d x_d > -w_0}_{}$

threshold

Common activation functions

1) Linear / Identity

$$f(net_i) = net_i$$

2) Sigmoid

$$f(net_i) = \frac{1}{1 + e^{-net_i}}$$

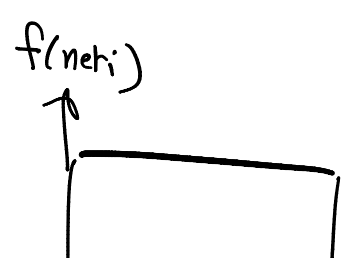
output $[0, 1]$

3) tanh

$$f(net_i) = \frac{e^{net_i} - e^{-net_i}}{e^{net_i} + e^{-net_i}}$$

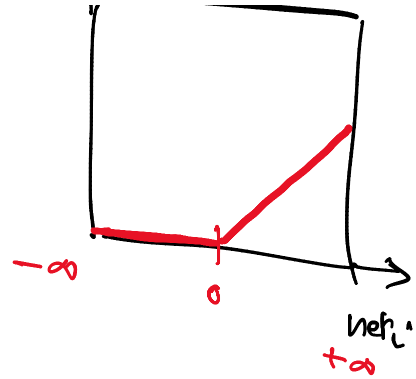
output $[-1, 1]$

4) ReLU (rectified linear)



$f(net_i) = \begin{cases} net_i & \text{if } net_i > 0 \end{cases}$

$$f(net_i) = \begin{cases} net_i & \text{if } net_i > 0 \\ 0 & \text{otherwise} \end{cases}$$



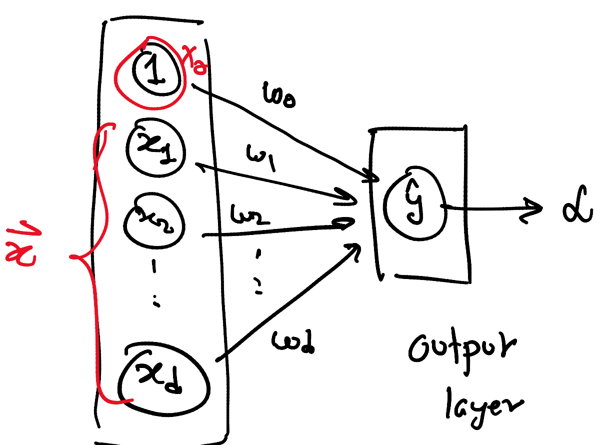
$$= \max\{0, net_i\}$$

5) $$\text{softmax}\Big(net_i \,\Big|\, \underbrace{net_1, net_2, \ldots, net_m}_{\text{m different neurons}}\Big)$$

Output prob distribution over the m neurons !

$$= \frac{e^{net_i}}{\sum_{j=1}^{m} e^{net_j}}$$

Linear Network (Linear regression)



Input layer

Output layer

f = linear

$$net_{\hat{y}} = \tilde{w}^T \tilde{x}$$

$$= w_0 + w_1 x_1 + \ldots + w_d x_d$$

$$\hat{y} = \text{linear}(net_i) = net_i$$

$\tilde{w}$ is unknown !

$f \equiv$ linear

learn: how?

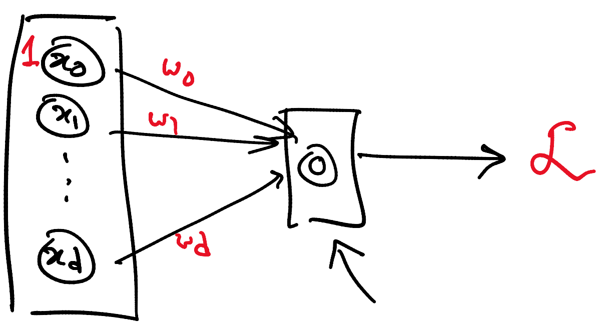
$\mathcal{L} =$ squared error $= (y - \hat{y})^2$

true   output of NN

$$\nabla_{\tilde{\omega}} = \frac{\partial \mathcal{L}}{\partial \tilde{\omega}}$$

train over a mini-batch of data via gradient descent.

Logistic regression network



Input layer

Single output neuron o (for output)

$f \equiv$ linear

$f^o \equiv$ sigmoid

$$net_o = w_0 x_0 + w_1 x_1 + \dots + w_d x_d$$
$$= \tilde{\omega}^T \tilde{x}$$

$$o = f^o(net_o) = \text{sigmoid}(net_o)$$

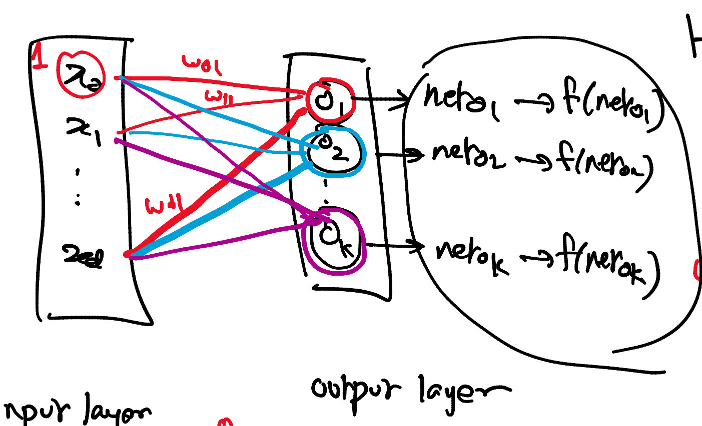
$\mathcal{L} \equiv$ binary cross entropy loss

$$= -\left(y \log(o) + (1-y)\log(1-o)\right)$$

$y \in \{0,1\}$ true class

$$\nabla_{\tilde{\omega}} = \frac{\partial \mathcal{L}}{\partial \tilde{\omega}} \qquad o \in [0,1]$$

K-class logistic regression



$w_{ij} \equiv$ weight between input x_i and output o_j

$$W = \begin{array}{cccc} o_1 & o_2 & & o_k \\ \left[\begin{array}{c} w_{01} \\ w_{11} \\ \vdots \\ w_{d1} \end{array}\right. & \begin{array}{c} w_{02} \\ w_{12} \\ \vdots \\ w_{d2} \end{array} & \cdots & \left.\begin{array}{c} w_{0k} \\ w_{1k} \\ \vdots \\ w_{dk} \end{array}\right] \end{array}$$

$d \times k$ matrix

$\overset{o}{f} \equiv$ softmax

Output of the network

these are the probabilities for each class

$$\begin{cases} o_1 = \underset{\text{softmax}}{\overset{o}{f}}(net_1 \mid net_1 \; net_2 \ldots net_k) \\ o_2 = \text{softmax}(net_2) \\ \vdots \\ o_k = \text{softmax}(net_k) \end{cases}$$

$$\mathcal{L} = f(o_1, o_2 \ldots o_k, \vec{y})$$

one-hot class vector

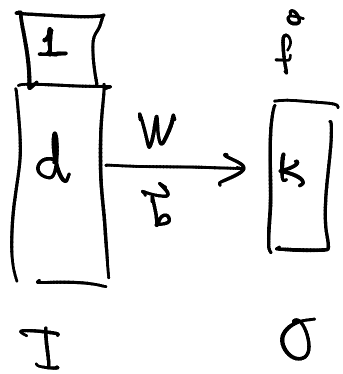
$$\mathcal{L} \equiv CE \equiv \text{Cross entropy}$$

$$= -\sum^{k} y_i \log(o_i)$$

$$= -\sum_{i=1} y_i \log(o_i)$$

$$\nabla_W = \frac{\partial \mathcal{L}}{\partial W}$$

SGD ⟸ Stochastic "mini-batch" GD



$$\vec{b} = \begin{pmatrix} w_{01} \\ w_{02} \\ \vdots \\ w_{0d} \end{pmatrix}$$

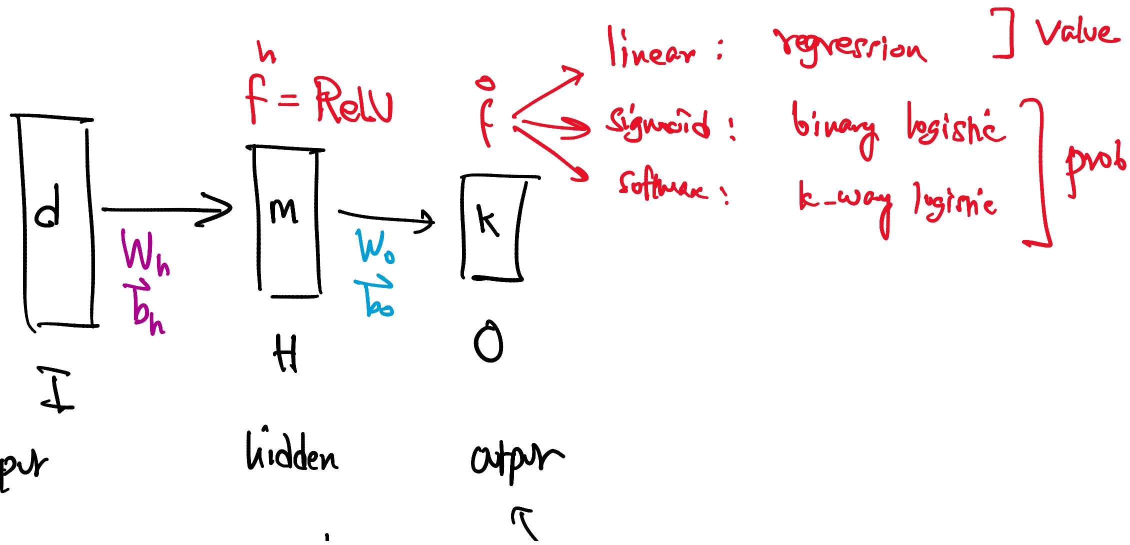
$$\underset{d \times k}{W} = \{w_{ij}\} \quad \begin{matrix} i = 1 \dots d \\ j = 1 \dots k \end{matrix}$$

Two parameters $\Theta = \{W, \vec{b}\}$

learn via backprop (SGD)

Simplest extension:

add one hidden layer



Input | hidden | output

d : # of independent vars $(x_1, x_2 \ldots x_d)$

m : we have to choose

k : determined by nature of y

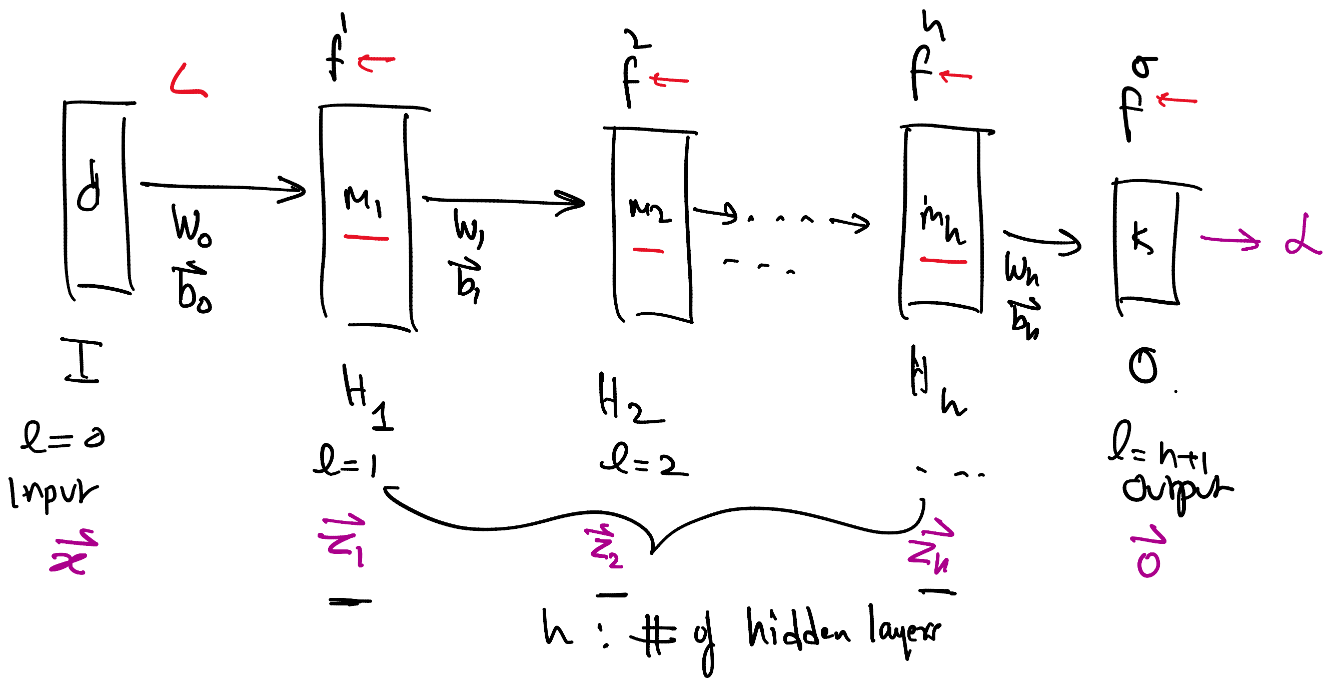
$W_h : d \times m$ matrix of synaptic weights (between I and H)

$\vec{b}_h : m$ bias vector for H

$W_o : m \times k$ (between H & O)

$\vec{b}_o : k$ bias for O

MLP



W_i is the weight matrix between layer i and $i+1$

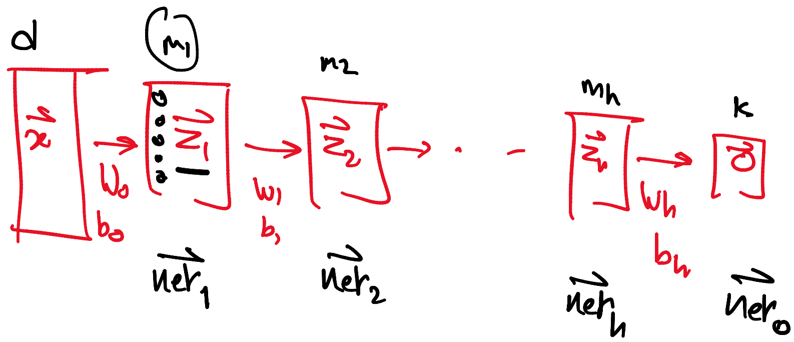
$\vec{b}_i$ is the bias for $i+1$

$$\Theta = \{ W_i, \vec{b}_i \mid i = 0 \ldots h \}$$

use back propagation

① Feed-forward step

$\vec{x} \longrightarrow$ how do I compute $\vec{o}$



h : # of hidden layers.

input: $\vec{x}$

$W_0 : d \times m_1$

$\vec{b}_0 : m_1$

$$H_1 \begin{cases} \vec{net}_1 = W_0^T \vec{x} + \vec{b}_0 \\ \vec{z}_1 = f^1(\vec{net}_1) = f\Big(\underbrace{\underbrace{\underbrace{W_0^T}_{m_1 \times d} \underbrace{\vec{x}}_{d \times 1}}_{m_1 \times 1} + \underbrace{\vec{b}_0}_{m_1 \times 1}}_{m_1 \times 1} \Big) \end{cases}$$

$$H_2 \begin{cases} \vec{net}_2 = W_1^T \vec{z}_1 + \vec{b}_1 \\ \vec{z}_2 = f^2(\vec{net}_2) \end{cases}$$

$\vdots$

$$H_h \begin{cases} \vec{net}_h = W_{h-1}^T \vec{z}_{h-1} + \vec{b}_{h-1} \\ \vec{z}_h = f^h(\vec{net}_h) = f^h\left(W_{h-1}^T \vec{z}_{h-1} + \vec{b}_{h-1} \right) \end{cases}$$

$\vec{z}_h = f(\vec{net}_h) = f(W_{h-1}^T \vec{z}_{h-1} + \vec{b}_{h-1})$

$$0 \begin{cases} \vec{net}_o = W_h^T \vec{z}_h + \vec{b}_h \\ \vec{o} = f^o(\vec{net}_o) = f^o(W_h^T \vec{z}_h + \vec{b}_h) \end{cases}$$

$\underset{\text{error}}{\mathcal{E}} \equiv \underset{\text{loss}}{\mathcal{L}}$ → squared error $= \frac{1}{2} \|\vec{y} - \vec{o}\|^2 = \mathcal{L}$

↘ Cross entropy → binary

↘ k-way

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log(o_i)$$

$\vec{o} = (o_1, o_2, \ldots o_k)^T$

$\vec{y} = \underbrace{(y_1, y_2, \ldots, y_k)^T}_{\text{One-hot vector}}$

Initialization: randomly initialize all

W_i and $\vec{b}_i$ $i = 0 \ldots h$

(small, random normal)

---

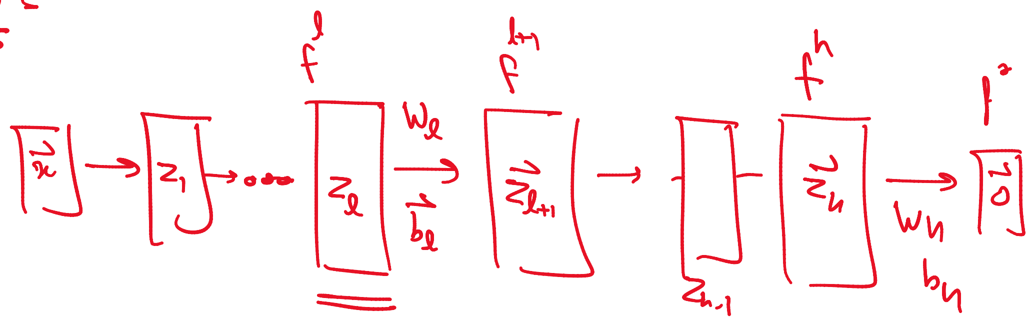
2) Backpropagation step

(update the W_i, $\vec{b}_i$ based on $\mathcal{L}$)

what: $\nabla_{W_i} = \frac{\partial \mathcal{L}}{\partial W_i}$ $\quad \nabla_{\vec{b}_i} = \frac{\partial \mathcal{L}}{\partial \vec{b}_i}$ $\quad \forall i = 0 \ldots h$

$W_i^{(t+1)} = W_i^{(t)} - \eta \cdot \nabla_{W_i}$ $\qquad \vec{b}_i^{(t+1)} = \vec{b}_i^{(t)} - \eta \cdot \nabla_{b_i}$

How ?



$$z_\ell = f^{\ell}\left(W_{\ell-1}^{T}\vec{z}_{\ell-1} + \vec{b}_{\ell-1}\right)$$

$$\vdots$$

$$\vec{o} = f^{o}\left(W_h^{T}\vec{z}_h + \vec{b}_h\right)$$

$$= f^{o}\left(W_h^{T}\left(f^{h}\left(W_{h-1}\vec{z}_{h-1} + \vec{b}_{h-1}\right)\right) + \vec{b}_h\right)$$

$$\mathcal{L} = \frac{1}{2}\|\vec{y} - \vec{o}\|^2$$

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = \frac{1}{2}\left\|\vec{y} - \underbrace{(((\quad)\quad))}_{\text{chain-rule}}\right\|^2$$