9/1/15

1

DEEP LEARNING : BACKPROP

1 Basic Setup / Notation

→ (Raw) Input

$\vec{x} \in \mathbb{R}^d$ (continuous input) } sometimes both

$\in \{0,1\}$ (discreete input)

→ Scalars : a, b, c, i, j, k, x, y (small chars)

Vectors : $\vec{x}, \vec{y}, \vec{w}, \vec{h}$ (vec notation)

Matrices : A, B, W, X, H (capital chars)

(or sets or Random Variables) [d : # dimension, N : # inputs]

→ Output / Target / Labels

y or $\vec{y} \in \mathbb{R}^K$ (K-dim regression)

$\in \{0,1\}$ or $\{+1,-1\}$ (Binary classification)

$\in \{1, 2, \ldots, K\}$ (Multi-class classification)

or $\in \{0,1\}^K$ s.t. $\sum_{c=1}^{K} y_c = 1$ } 1-hot encoding for multiclass classification

→ $\boxed{\text{Unknown}}$ Target Function

$f^{gt} : X \rightarrow Y$

input space, Output space

$(\vec{x}_i, f^{gt}(\vec{x}_i))$

→ Given: Dataset $D = \{(\vec{x}_1, y_1), \ldots (\vec{x}_N, y_N)\}$
or "Training Data"

$\underbrace{(\vec{x}_1, y_1)}_{\text{sample or data-point}}$

→ Goal: Given D (input, outputs) from unknown f, predict $f^{gt}(\vec{x})$ on NEW $\vec{x}$

→ Approach: [All of Supervised Learning]

① Pick a model class / Hypothesis Space

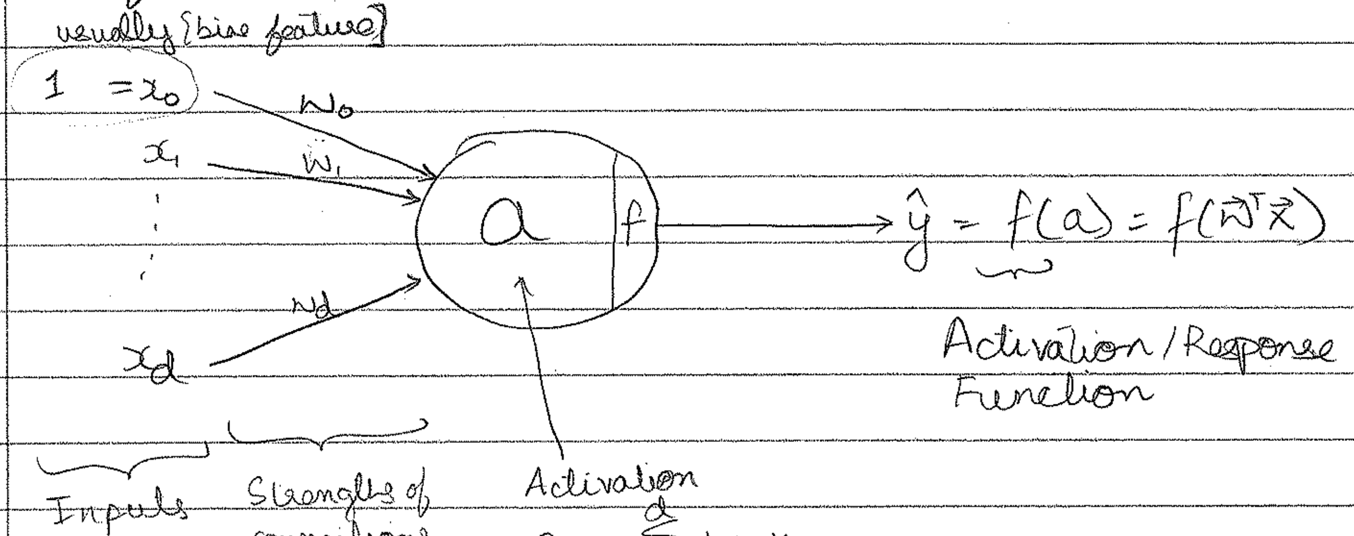
$H = \{g : X \rightarrow Y\}$
a set of mappings from $X \rightarrow Y$

[A lot of ML is about what kind of H "makes sense" for our task]

② Search / Optimize a loss function over H to find "best" $g \in H$

$$\Rightarrow \text{Learning} = \underset{g \in H}{\text{argmin}} \; \underbrace{Loss(g; D)}_{\text{usually } \sum_{i=1}^{N} L_i(y_i^{gt}, \overbrace{g(\vec{x}_i)}^{\hat{y}_i})}$$

2 Artificial "Neuron"



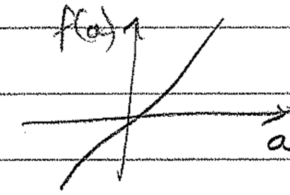
$$a = \sum_{j=0}^{d} w_j x_j$$

$$= \vec{w}^T \vec{x}$$

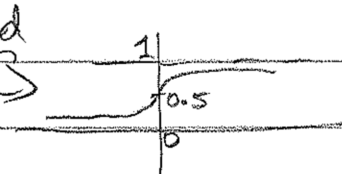
Many different activation functions

→ Linear: $f(a) = a$

$\Rightarrow \hat{y} = \vec{w}^T\vec{x}$ [Linear Regression]



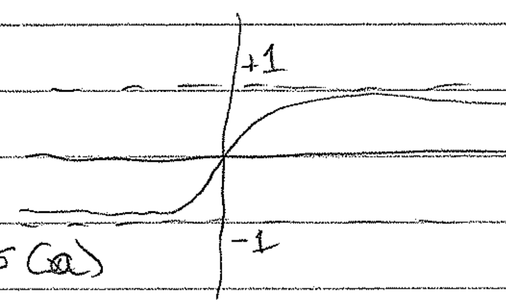
→ Logistic $f(a) = \frac{1}{1+e^{-a}} = \underbrace{\sigma(a)}_{\text{sigmoid}}$



$\hat{y} = \frac{1}{1+e^{\vec{w}^T\vec{x}}}$ $\boxed{\text{Logistic Regression } P(Y=1 \mid \vec{x}, \vec{w})}$

→ Tanh

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

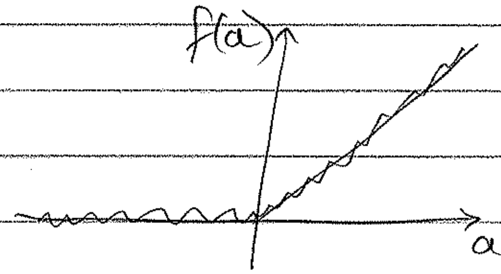


For hidden units in NN we always prefer tanh over $\sigma(a)$

Why?

→ ReLU [Rectified Linear Unit]

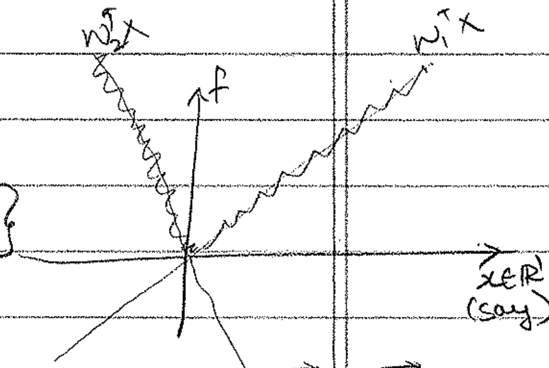
$$f(a) = \max\{0, a\}$$



In hidden layer of deep NN, this is always prefered over $\sigma(a)$ or $\tanh(a)$. Why??

→ Maxout

$$f(a) = \max\{\underbrace{\vec{w}_1^T x}_{a_1}, \underbrace{\vec{w}_2^T \vec{x}}_{a_2}\}$$



Each neuron has (say) 2 weights $\vec{w}_1, \vec{w}_2$

Take max activation.

ReLU is a special case of this. How?

3 Loss Functions

→ functions of both parameters $\vec{w}$ & training data

1 Log-Loss / Cross-Entropy / Maximum-Likelihood / KL-Divergence

$$L(\vec{w}; D) = \sum_{i=1}^{N} L_i(\vec{w}) \quad \Big\}\ \text{Decomposable Loss}$$

where $L_i(\vec{w}) = -\log \underbrace{p(y_i^{gt} \mid \vec{x}_i, \vec{w})}$

How much prob. does your model assign to GT labels?

≡ negative log-likelihood for this sample

→ Why is this called Cross-Entropy? And where is the KL divergence coming in?

Consider Multiclass-classification w/ 1-HOT encoding

$$\underbrace{\overset{y^{gt}}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}_{1\times K}}}_{p^{gt}(y)} \qquad \underbrace{\overset{\text{model prediction}}{\begin{bmatrix} p(y=1 \mid \vec{x}, \vec{w}) \\ \vdots \\ p(y=K \mid \vec{x}, \vec{w}) \end{bmatrix}}}_{\hat{P}(y)}$$

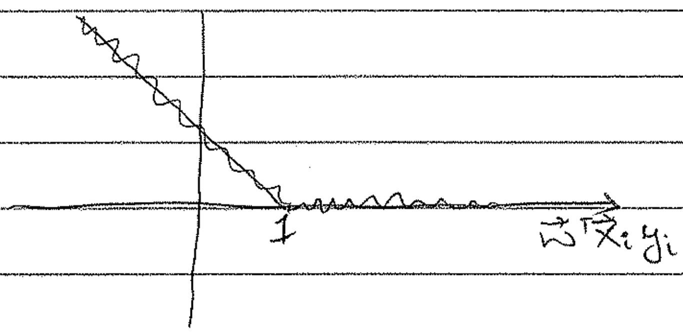
[delta distribution] [Model distribution]

$$KL(p^{gt} \| \hat{p}) = -\sum_{y=1}^{K} p^{gt}(y) \log \hat{p}(y)$$

$$= -\log p(y = y_i^{gt} \mid \vec{x}_i, \vec{w})$$

(2) Hinge-Loss [for binary-classification]

$L_i(\vec{w}) = \max\{0, 1 - \vec{w}^T \vec{x}_i y_i\}$ where $y_i \in \{+1, -1\}$



4 Detour: Matrix/ Vector differentiation

	S	V	M
S	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \vec{x}}$	$\frac{\partial y}{\partial X}$
V	$\frac{\partial \vec{y}}{\partial x}$	$\frac{\partial \vec{y}}{\partial \vec{x}}$	
M	$\frac{\partial Y}{\partial x}$	Tensors	

$x, y \in \mathbb{R}^1$

$\vec{x} \in \mathbb{R}^d$

$\vec{y} \in \mathbb{R}^K$

Convention: $\frac{\partial \vec{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \vdots \\ \frac{\partial y_K}{\partial x} \end{bmatrix}$ numerator = dim 1 = col-vector

[Gradient] $\frac{\partial y}{\partial \vec{x}} = \left[\frac{\partial y}{\partial x_1} \cdots \frac{\partial y}{\partial x_d} \right]$ denominator = dim 2 = row-vector

[Jacobian Matrix] $\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{array}{c c} & j \\ i & \begin{bmatrix} & \vdots & \\ \cdots & \frac{\partial y_i}{\partial x_j} & \\ & & \end{bmatrix}_{k \times d} \end{array}$

Easy to prove: $\rightarrow \frac{\partial(\vec{w}^T \vec{x})}{\partial \vec{w}} = \left[\frac{\partial(\vec{w}^T\vec{x})}{\partial w_1} \cdots \frac{\partial(\vec{w}^T\vec{x})}{\partial w_d} \right]$

$= X^T$

$\rightarrow \frac{\partial(\vec{w}^T A \vec{w})}{\partial \vec{w}} = 2W^T A$

$\rightarrow \quad \vec{y} = A\vec{x} \qquad \frac{\partial \vec{y}}{\partial \vec{x}} = A$

⑤ Chain Rule

some books/people use opposite order

→ Function Composition: $L(x) = (f \circ g)(x)$
$$= f(g(x))$$

Chain Rule:

→ [Most General Notation]
$$\underbrace{D_x}_{\text{total derivative}} (f \circ g) = D_{g(x)} f \circ D_x g$$

→ [More concrete notation for scalars]
$$L'(x) = f'(g(x))\, g'(x)$$

→ [With intermediate variables]
$$y = g(x)$$
$$z = f(y)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

Example: $L_i(w) = -\log\left(\frac{1}{1+e^{-\vec{w}^T x_i}}\right)$ [For $y_i = +1$]

$$p \longleftarrow \quad a \longleftarrow$$

$$= \Big(\underbrace{-\log(\cdot)}_{\frac{\partial L}{\partial p}} \circ \underbrace{\frac{1}{1+e^{-(\cdot)}}}_{\frac{\partial p}{\partial a}} \circ \underbrace{x^T(\cdot)}_{\frac{\partial a}{\partial \vec{w}}}\Big)(w)$$

$$\frac{\partial L_i}{\partial \vec{w}} = \left[\frac{1}{p}\right] \cdot \left[\underbrace{\frac{-1}{(1+e^{-a})^2} \cdot -e^{-a}}_{p \cdot (1-p)}\right] \cdot x^T = (1-p)x^T$$

Multivariate Chain Rule

$g: \mathbb{R}^d \to \mathbb{R}^m$

$f: \mathbb{R}^m \to \mathbb{R}^k$

$$L(\vec{x}) = (f \circ g)(\vec{x})$$

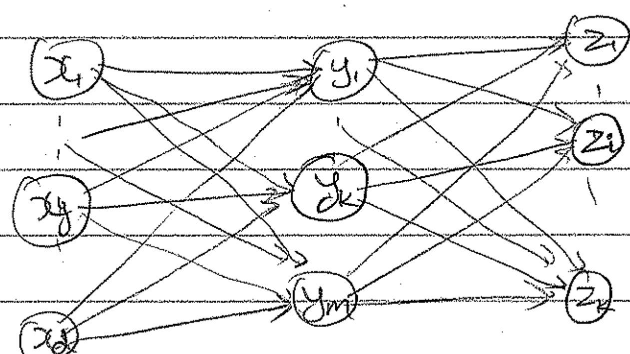
$$\vec{y} = g(\vec{x}) \qquad \vec{z} = f(\vec{y})$$

→ Chain Rule: $D_{\vec{x}}(f \circ g) = D_{g(\vec{x})} f \circ D_{\vec{x}}(g)$

[Abstract form holds]

But what does this mean??

Visualize:



"outcome"

$$\frac{\partial z_i}{\partial x_j} = \sum_k \frac{\partial z_i}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_j}$$

"knob"

all intermediate variables

how those intermediate vars affect outcome!

how my "knob" affects intermediate variable

Formally,

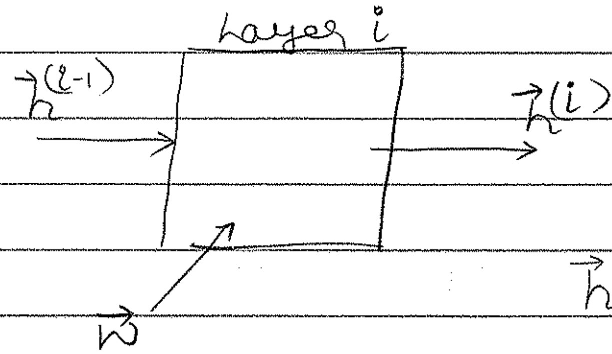
Jacobian relationship

$$J_{f\circ g} = (J_f \circ g)\, J_g$$

$$\begin{array}{cc} & j \\ i & \begin{bmatrix} \cdots \dfrac{\partial z_i}{\partial x_j} \end{bmatrix}_{k\times d} \end{array} = \begin{array}{cc} & k \\ i & \begin{bmatrix} \cdots \dfrac{\partial z_i}{\partial y_k} \end{bmatrix}_{k\times m} \end{array} \begin{array}{cc} & j \\ k & \begin{bmatrix} \cdots \dfrac{\partial y_k}{\partial x_j} \end{bmatrix}_{m\times d} \end{array}$$

→ what if my $\vec{x}, \vec{y}, \vec{z}$ are tensors?
- → string up into vectors & proceed
- → Matlab notation x-vec = X(:);
- → Trust me, this is the cleanest way

→ In Neural Nets $\vec{z} \in \mathbb{R}^1$ (Loss) $L(\vec{w})$



$$\vec{h}^{(i)} = g(\vec{h}^{(i-1)}, \vec{w})$$

$$L(\vec{w}) = f(h^{(i)})$$

$$\frac{\partial L}{\partial \vec{w}} = \frac{\partial L}{\partial \vec{h}^{(i)}} \frac{\partial \vec{h}^{(i)}}{\partial \vec{w}} \qquad \frac{\partial L}{\partial \vec{w}} = \left\langle \frac{\partial L}{\partial out}, \frac{\partial out}{\partial \vec{w}} \right\rangle$$

$$[\quad]_{1\times d} \quad [\quad]_{1\times |h|} \quad \begin{bmatrix} \quad \end{bmatrix}_{|h^i|\times d}$$