Basic Linear Algebra for AI and Computer Vision

Dongbo Min
Department of Computer Science and Engineering
Ewha Womans University, Korea
E-mail: dbmin@ewha.ac.kr

Contents

1. Basics for linear algebra
   - Eigenvalue/Eigenvector and Linear regression
   - Applications for classical computer vision tasks (Homography, camera calibration, epipolar geometry)

2. Partial derivatives and chain rules
   - Feed-forward/backpropagation of multi-layer perceptron (MLP)

이화여자대학교
EWHA WOMANS UNIVERSITY

Eigenvalue and Eigenvector

- Heterogeneous linear system

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$$

  - with a non-zero vector $\boldsymbol{b} \neq \mathbf{0}$
  - If an inversion of $\mathbf{A}$ or $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ exists, an unique solution for $\boldsymbol{x}$ can be obtained simply.

- Homogeneous linear system

$$\mathbf{A}\boldsymbol{x} = \mathbf{0}$$

  - Trivial solution: $\boldsymbol{x} = \mathbf{0}$
  - **Q**: Can we obtain any meaningful solution for the homogeneous linear system?

이화여자대학교
EWHA WOMANS UNIVERSITY

Eigenvalue and Eigenvector

- Eigenvalue and eigenvector of $n \times n$ matrix $\mathbf{A}$
  - A set of σ and x satisfying $\mathbf{A}\boldsymbol{x} = \sigma\boldsymbol{x}$
  - Eigenvalue: $\{\sigma_i | i = 1,2,\dots,n\}$
  - Eigenvector: $\{\boldsymbol{x}_i | i = 1,2,\dots,n\}$
  - Eigenvector is orthonormal as below.

$$\boldsymbol{x}_i{}^T \boldsymbol{x}_j = \begin{cases} 1 & if\ i = j \\ 0 & otherwise \end{cases}$$

- When $n \times n$ matrix A is full rank, n non-zero eigenvalues exist

  $rank(A) =$ the number of non-zero σ_i $(i = 1,2,\dots,n)$

이화여자대학교
EWHA WOMANS UNIVERSITY

Eigenvalue and Eigenvector

- For a full-rank $n \times n$ matrix $\mathbf{A}$, i.e., $rank(\mathbf{A}) = n$

$$\sum_{i=1}^{n} \sigma_i x_i {x_i}^T = \sigma_1 x_1 {x_1}^T + \sigma_2 x_2 {x_2}^T + \cdots + \sigma_n x_n {x_n}^T$$

Independent space

Generalizing this form for a non-rectangular matrix $\mathbf{A}$ $(m \times n)$
→ Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD)

- Any $m \times n$ matrix $\mathbf{A}$ can be written as the product of three matrices

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$$

- $\mathbf{U}$: $m \times m$ orthonormal matrix
  (columns are mutually orthogonal unit vectors)
- $\mathbf{V}$: $n \times n$ orthonormal matrix
  (columns are mutually orthogonal unit vectors)
- $\mathbf{D}$: $m \times n$ diagonal matrix (its diagonal elements σ_i: singular values, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$)

- Note) both $\mathbf{U}$ and $\mathbf{V}$ are not unique, but $\mathbf{D}$ is fully determined by $\mathbf{A}$

이화여자대학교
EWHA WOMANS UNIVERSITY

Properties of the SVD

- Property 1
  - The singular values provide the info on the singularities of a square matrix $\mathbf{A}$.
  - Square matrix $\mathbf{A}$ is nonsingular iff all singular values are different from zero
  - $\frac{\sigma_1}{\sigma_n}$: condition number (measuring the degree of singularity of $\mathbf{A}$)

- Property 2
  - For a rectangular matrix $\mathbf{A}$,
    $rank(\mathbf{A}) =$ the number of non-zero σ_i ($i = 1, \ldots, n$)
  - With a fixed tolerance ϵ (typically of the order of 10^{-6}),
    the effective $rank(\mathbf{A}) =$ the number of nonzero σ_i ($i = 1, \ldots, n$) which is greater than ϵ

이화여자대학교
EWHA WOMANS UNIVERSITY

Properties of the SVD

- Property 3
  - For a square, nonsingular matrix $\mathbf{A} = \mathbf{UDV}^{\mathrm{T}}$,
$$\mathbf{A}^{-1} = \mathbf{VD}^{-1}\mathbf{U}^{\mathrm{T}}$$
  - For a square matrix $\mathbf{A} = \mathbf{UDV}^{\mathrm{T}}$ (i.e., singular or nonsingular)
    the pseudo-inverse matrix $\mathbf{A}^{+} = \mathbf{V}{\mathbf{D}_0}^{-1}\mathbf{U}^{\mathrm{T}}$
    ${\mathbf{D}_0}^{-1}$ is equal to $\mathbf{D}^{-1}$ for all non-zero singular values and zero otherwise.

- Property 4
  - The columns of U corresponding to non-zero singular values = A's range
  - The columns of V corresponding to zero singular values = A's null space

이화여자대학교 EWHA WOMANS UNIVERSITY

Properties of the SVD

- Property 5
  - $n \times n$ matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$

    non-zero eigenvalues = the squares of non-zero singular values σ_i

    eigenvectors = columns of $\mathbf{V}$

  - $m \times m$ matrix $\mathbf{A}\mathbf{A}^{\mathrm{T}}$

    non-zero eigenvalues = the squares of non-zero singular values σ_i

    eigenvectors = columns of $\mathbf{U}$

  - For $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ (columns of $\mathbf{U}$ and $\mathbf{V}$ corresponding to σ_k)

$$\mathbf{A}\boldsymbol{u}_k = \sigma_k \boldsymbol{v}_k$$
$$\mathbf{A}^T\boldsymbol{v}_k = \sigma_k \boldsymbol{u}_k$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Properties of the SVD

- Property 6
  - Frobenius norm $\|\mathbf{A}\|_F$ of matrix $\mathbf{A}$
  - $\|\mathbf{A}\|_F = \sum_{i,j} a_{ij}$
  - $\|\mathbf{A}\|_F = \sum_k \sigma_k$

Solving non-homogeneous and homogeneous linear system

- $\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \rightarrow \boldsymbol{x} = (\mathbf{A}^{\mathbf{T}}\mathbf{A})^{-\mathbf{1}}\mathbf{A}^{\mathbf{T}}\mathbf{b}$
  - This solution is known to be optimal in the least square sense.
  - Namely, it is equivalent to minimizing $||\mathbf{A}\boldsymbol{x} - \boldsymbol{b}||^2$
- $\mathbf{A}\boldsymbol{x} = \mathbf{0}$
  - $\mathbf{A}$: $m \times n$ matrix, $m \geq n-1, rank(\mathbf{A}) = n-1$
  - Its trivial solution is $\mathbf{0}$
  - To find a non-trivial solution, we can find the solution up to a scale factor through Singular Value Decomposition (SVD).
  - As the norm of the solution is arbitrary, we impose a *unit norm constraint* on the solution

$$\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

Introducing the Lagrange multiplier λ ⇨ $$\min_{\boldsymbol{x}} (||\mathbf{A}\boldsymbol{x}||^2 - \lambda(||\boldsymbol{x}||^2 - 1))$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Solving non-homogeneous and homogeneous linear system

$$\min_{\mathbf{f}}(\|\mathbf{A}\boldsymbol{x}\|^2 - \lambda(\|\boldsymbol{x}\|^2 - 1))$$

- Equating to zero the derivative with respect to f gives

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\boldsymbol{x} - \lambda\boldsymbol{x} = 0$$

- This equation tells
  $\lambda =$ eigenvalue of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ and $\boldsymbol{x} = \mathbf{e}_{\lambda}$ corresponding eigenvector.

- Then, with this solution the objective becomes

$$\|\mathbf{A}\boldsymbol{x}\|^2 - \lambda(\|\boldsymbol{x}\|^2 - 1) = \lambda$$

- In short,
  the solution = the column of $\mathbf{V}$ corresponding to the null (non-zero) singular value of $\boldsymbol{A}$

Solving non-homogeneous and homogeneous linear system - Rayleigh quotient

- For a given complex Hermitian matrix $\mathbf{M}$ and nonzero vector $\boldsymbol{x}$, the Rayleigh quotient $R(M, x)$ is defined as follows.

$$R(\mathbf{M}, \boldsymbol{x}) = \frac{\boldsymbol{x}^* \mathbf{M} \boldsymbol{x}}{\boldsymbol{x}^* \boldsymbol{x}}$$

- For covariance matrix $\mathbf{M} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, let us denote λ_i and v_i as eigenvalue and eigenvector of $\mathbf{M}$

$$\mathbf{M}\boldsymbol{v}_i = \mathbf{A}^{\mathrm{T}}\mathbf{A}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$$

$$\Rightarrow \quad \boldsymbol{v}_i^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\boldsymbol{v}_i = \boldsymbol{v}_i^{\mathrm{T}}\lambda_i \boldsymbol{v}_i \qquad \textit{subject to } |v_i| = 1$$

$$\Rightarrow \quad \|\mathbf{A}\boldsymbol{v}_i\|^2 = \lambda_i \|\boldsymbol{v}_i\|^2$$

$$\Rightarrow \quad \frac{\|\mathbf{A}\boldsymbol{v}_i\|^2}{\|\boldsymbol{v}_i\|^2} = \lambda_i$$

이화여자대학교
EWHA WOMANS UNIVERSITY

Solving non-homogeneous and homogeneous linear system

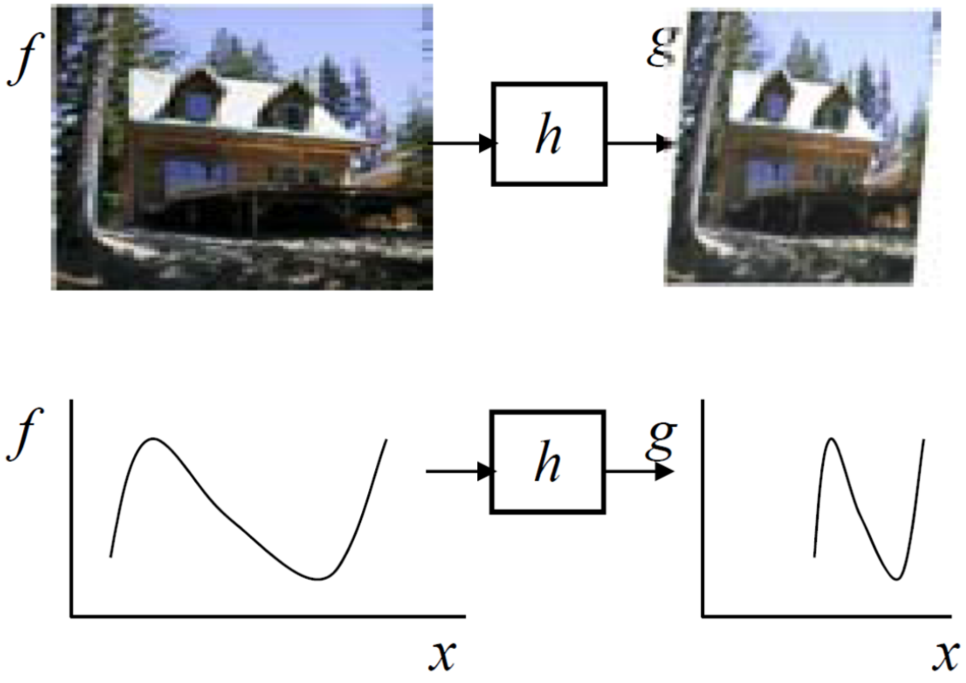
Problem statement	Solution
Minimize $\|\|\mathbf{A}\boldsymbol{x}-\boldsymbol{b}\|\|^2$	$\boldsymbol{x} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$
Least square solution to $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$	$\boldsymbol{x} = \mathbf{A}\backslash\boldsymbol{b}$ (in matlab)

Problem statement	Solution
Minimize $\boldsymbol{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\boldsymbol{x}$ s.t. $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}=1$	$[\boldsymbol{v},\lambda] = \mathrm{eig}(\mathbf{A}^{\mathrm{T}}\mathbf{A})$
Minimize $\dfrac{\boldsymbol{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\boldsymbol{x}}{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}$	$\boldsymbol{x} = \boldsymbol{v}_1 \colon \lambda_1 < \lambda_{2,\ldots,n}$
Non-trivial solution to $\mathbf{A}\boldsymbol{x}=\mathbf{0}$	

이화여자대학교
EWHA WOMANS UNIVERSITY

Applications: Estimating Geometric Transformation

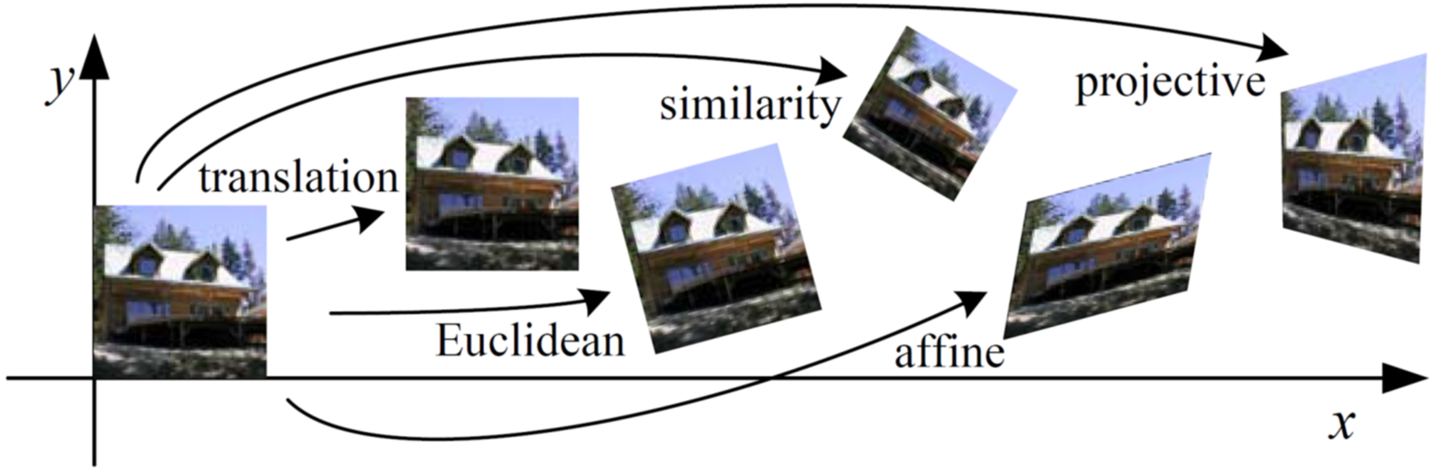
- General form of geometric transformation
  - Including translation, rotation, scale, skew, and so on.



p. 35-38 of Computer Vision: Algorithms and Applications (Richard Szeliski)
http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf

Applications: Estimating Geometric Transformation

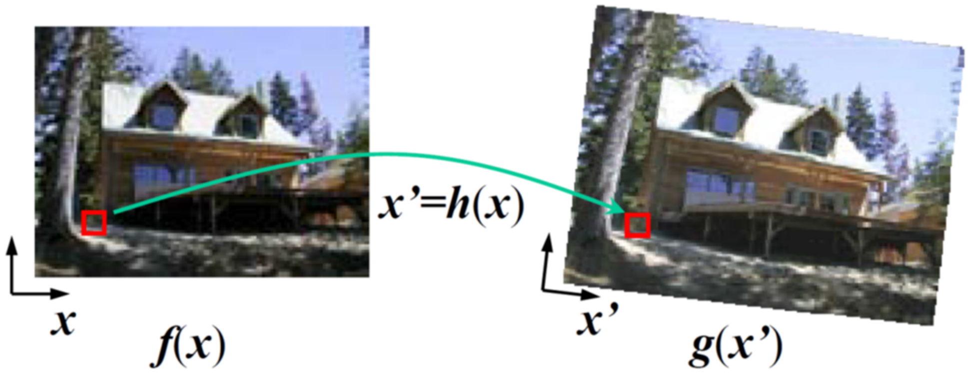
- **2D parametric transformation**
  - Translation
  - Rigid (Euclidean) transformation
  - Similarity transformation
  - Affine transformation
  - Projective transformation



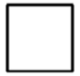
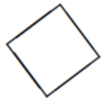
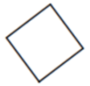
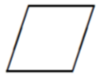
p. 35-38 of Computer Vision: Algorithms and Applications (Richard Szeliski)
http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf

이화여자대학교
EWHA WOMANS UNIVERSITY

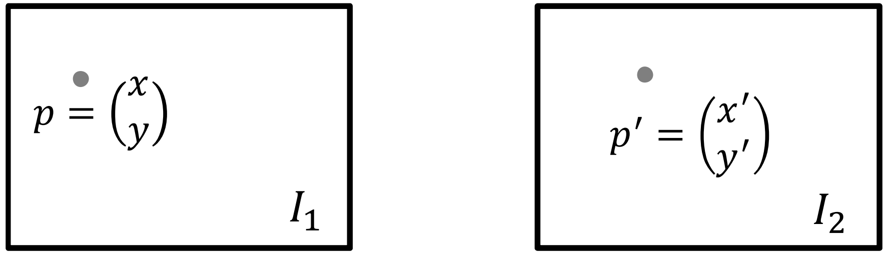
Applications: Estimating Geometric Transformation



$$\boldsymbol{x}' = h(\boldsymbol{x}) = \mathbf{M}\tilde{\boldsymbol{x}} \qquad \text{where } \tilde{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$$

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

Estimating Affine Transformation



For a pair of corresponding pixels

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \longrightarrow \begin{pmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

For $N \geq 3$ pairs of corresponding pixels, affine transform for $I_1 \rightarrow I_2$ can be computed as follows.

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$$

$$\rightarrow \boldsymbol{x} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\boldsymbol{b}$$

$$\underbrace{\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ & & \vdots & & & \\ x_N & y_N & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_N & y_N & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ \vdots \\ x_N' \\ y_N' \end{pmatrix}}_{\boldsymbol{b}}$$

이화여자대학교
EWHA WOMANS UNIVERSITY

Homography

Question
Given a set of point correspondences between two views, can we match an arbitrary point in a view to another view?

Note: All the points should be on the same planar surface.

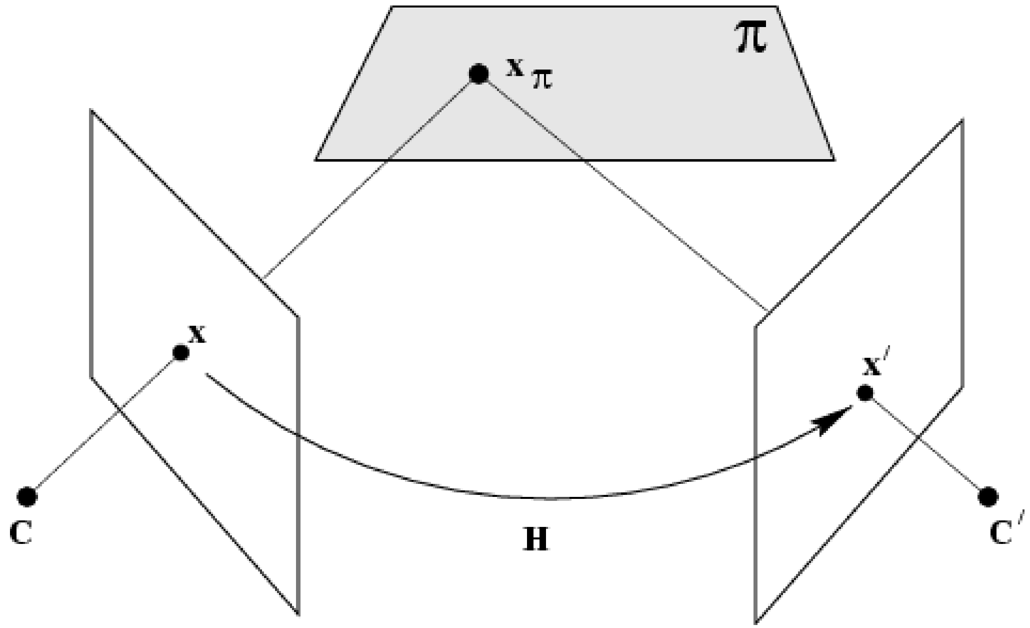
이화여자대학교 EWHA WOMANS UNIVERSITY

Homography

- Relationship between two views

$$x' \cong Hx$$

  - They have same directions.
  - Hx are collinear: $x' \times Hx = 0$



이화여자대학교 EWHA WOMANS UNIVERSITY

Estimating Homography

- How to compute homography matrix

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \cong \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$\Rightarrow$

For $N \geq 4$ pairs of corresponding pixels

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & & \vdots & & & \\ x_N & y_N & 1 & 0 & 0 & 0 & -x_N'x_N & -x_N'y_N & -x_N' \\ 0 & 0 & 0 & x_N & y_N & 1 & -y_N'x_N & -y_N'y_N & -y_N' \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

Solving $\mathbf{Ah} = \mathbf{0}$ requires using SVD.

Image Stitching using Homography

Stitched image using
the estimated homography

Neural Networks

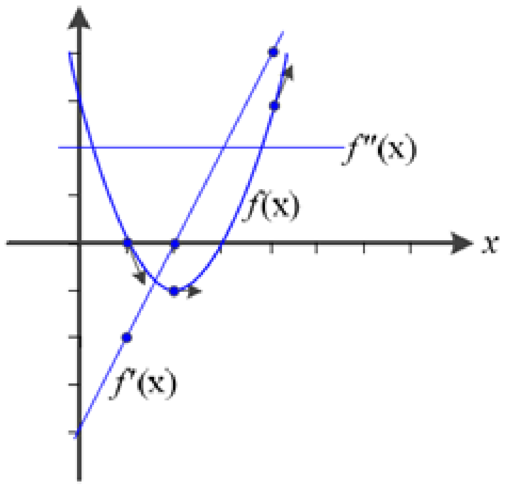
Simple Example: Multi-Layer Perceptron (MLP)

Derivative

- Optimization using derivative
  - 1^{st} order derivative $$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$
  - $f'(x)$: The slope of the function, indicating the direction in which the value increases

  → The minima of the objective function may exist in the direction of $-f'(x)$.

  → Gradient descent algorithm: $d\theta \leftarrow -f'(x)$



$$y = f(x) = x^2 - 4x + 3$$

$$y' = f'(x) = 2x - 4$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative

- Partial derivative
  - Derivatives of functions with multiple variables
  - Gradient: the vector of the partial derivative

Ex) $\nabla f, \frac{\partial f}{\partial \mathbf{x}}, \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)^{\mathrm{T}}$

$$f(\mathbf{x}) = f(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

$$\nabla f = f'(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)^{\mathrm{T}} = (2x_1^5 - 8.4x_1^3 + 8x_1 + x_2, 16x_2^3 - 8x_2 + x_1)^{\mathrm{T}}$$

이화여자대학교
EWHA WOMANS UNIVERSITY

Chain Rule

- Chain rule

$$f(x) = g(h(x))$$
$$f(x) = g(h(i(x)))$$

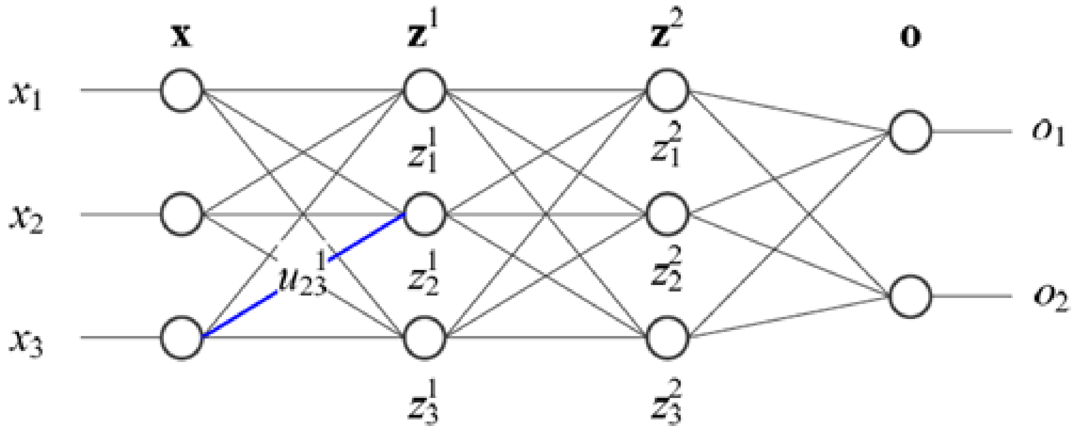
$$f'(x) = g'\big(h(x)\big)h'(x)$$
$$f'(x) = g'\big(h(i(x))\big)h'(i(x))i'(x)$$

Ex) $f(x) = 3(2x^2 - 1)^2 - 2(2x^2 - 1) + 5$ $\qquad h(x) = 2x^2 - 1$

$$f'(x) = \underbrace{(3 * 2(2x^2 - 1) - 2)}_{g'(h(x))}\underbrace{(2 * 2x)}_{h'(x)} = 48x^3 - 32x$$

- Multi-layer perceptron (MLP)
  - Example of composite function
  - Error back propagation:

    use the chain rule to compute $\frac{\partial o_i}{\partial u_{23}^1}$



이화여자대학교
EWHA WOMANS UNIVERSITY

Jacobian Matrix and Hessian Matrix

- Jacobian matrix
  - 1st order partial derivative matrix for $\mathbf{f}: \mathbb{R}^d \mapsto \mathbb{R}^m$

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_d} \end{pmatrix}$$

Ex) $\mathbf{f}: \mathbb{R}^2 \mapsto \mathbb{R}^3$ $\quad \mathbf{f}(\mathbf{x}) = (2x_1 + x_2^2, -x_1^2 + 3x_2, 4x_1x_2)^{\mathrm{T}}$

$$\mathbf{J} = \begin{pmatrix} 2 & 2x_2 \\ -2x_1 & 3 \\ 4x_2 & 4x_1 \end{pmatrix} \qquad \mathbf{J}|_{(2,1)^{\mathrm{T}}} = \begin{pmatrix} 2 & 2 \\ -4 & 3 \\ 4 & 8 \end{pmatrix}$$

2×3 **or** 3×2 **matrix can be used.**
Here, we define it as 2×3 **matrix.**

- Hessian matrix
  - 2nd order partial derivative matrix

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 x_1} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2 x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n x_n} \end{pmatrix}$$

Ex) $f(\mathbf{x}) = f(x_1, x_2)$

$$= \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

$$\mathbf{H} = \begin{pmatrix} 10x_1^4 - 25.2x_1^2 + 8 & 1 \\ 1 & 48x_2^2 - 8 \end{pmatrix}$$

$$\mathbf{H}|_{(0,1)^{\mathrm{T}}} = \begin{pmatrix} 8 & 1 \\ 1 & 40 \end{pmatrix}$$

Applications: Neural Networks

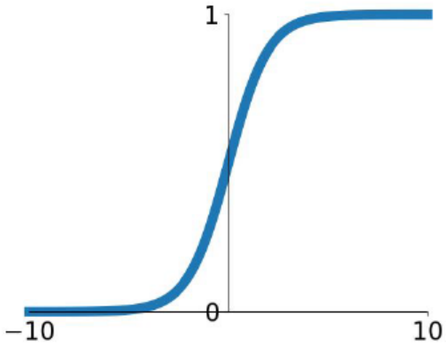
- Activation function
  - Softmax, Sigmoid, ReLU, Leaky ReLU

- Loss function
  - Regression loss, Hinge loss, Cross-entropy loss, Log likelihood loss

Activation Functions

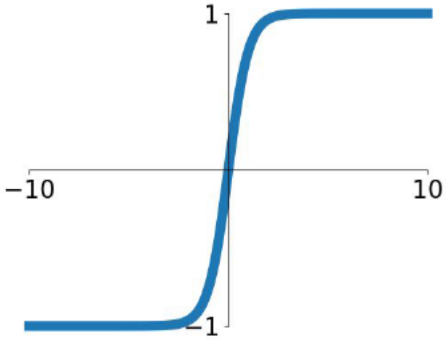
Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



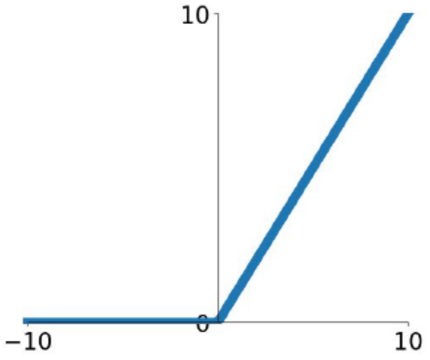
tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



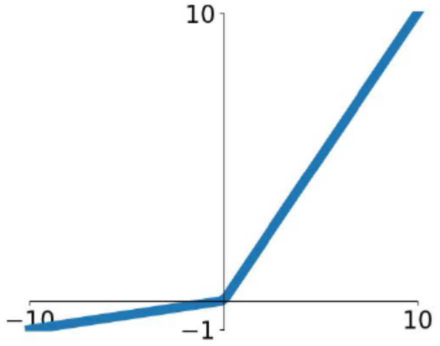
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

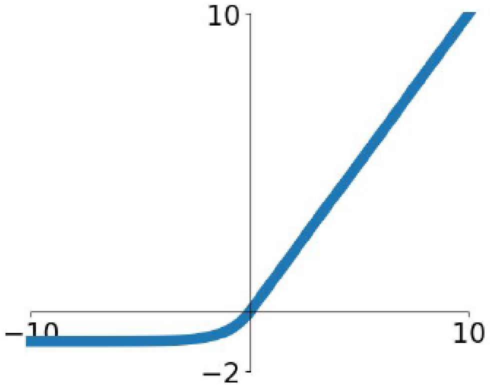


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^{x} - 1) & x < 0 \end{cases}$$



이화여자대학교
EWHA WOMANS UNIVERSITY

Softmax Activation Function

- Softmax activation function

scores = unnormalized log probabilities of the classes.

→ **Probability can be computed using scores as below.**

Probability of class label being k for an image $\boldsymbol{x}_i$

$$P(Y = k | X = \boldsymbol{x}_i) = p_k = \boxed{\frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}}}$$

Softmax activation function

$\boldsymbol{x}_i$: image

y_i: class label (integer, $1 \leq y_i \leq C$)

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x}_i + \boldsymbol{b}$$

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_C^T \end{pmatrix}$$

	unnormalized log probabilities	exp →	unnormalized probabilities	normalize →	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Loss function

- Loss function
  - quantifies our unhappiness with the scores across the training data.

- Type of loss function
  - Regression loss
  - Hinge loss
  - Cross-entropy loss
  - Log likelihood loss

이화여자대학교
EWHA WOMANS UNIVERSITY

Loss Function: Log Likelihood Loss

- Log likelihood loss

$$L_i = -\log p_j \text{ where } j \text{ satisfies } z_{ij} = 1$$

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_C \end{pmatrix}$$ probability for i^{th} image
(It is assumed to be *normalized*, i.e. $|\boldsymbol{p}| = 1$.)

$\boldsymbol{z_i}$: class label for i^{th} image
($C \times 1$ vector, $z_{ij} = 1$ when $j = y_i$ and 0 otherwise)

y_i: class label (integer, $1 \leq y_i \leq C$)

Example
Suppose i^{th} image belongs to class 2 and $C = 10$.

$$\boldsymbol{z_i} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \boldsymbol{p} = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix} \Rightarrow L_i = -\log 0.7$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Softmax + Log Likelihood Loss

- Log likelihood loss

$$L_i = -\log p_j \text{ where } j \text{ satisfies } z_{ij} = 1$$

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_C \end{pmatrix}$$ probability for i^{th} image
(It is assumed to be *normalized*, i.e. $|\boldsymbol{p}| = 1$.)

$\boldsymbol{z_i}$: class label for i^{th} image
($C \times 1$ vector, $z_{ij} = 1$ when $j = y_i$ and 0 otherwise)

y_i: class label (integer, $1 \le y_i \le C$)

⇨ $$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}}\right)$$

This can be interpreted as minimizing the negative log likelihood of the correct class.
→ *Maximum Likelihood Estimation* (MLE)

이화여자대학교
EWHA WOMANS UNIVERSITY

Softmax + Log Likelihood Loss

Softmax + Log likelihood loss: is often called ‘softmax classifier’

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp →	unnormalized probabilities	normalize →	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ L_i = -log(0.13)
= **0.89**

이화여자대학교 EWHA WOMANS UNIVERSITY

Loss Function: Regression Loss

- Regression loss
  - Using L1 or L2 norms
  - Widely used in pixel-level prediction (e.g. image denoising)

$$L_i = |\boldsymbol{y}_i - \boldsymbol{s}_i|$$

$$L_i = (\boldsymbol{y}_i - \boldsymbol{s}_i)^2$$

$$\boldsymbol{y}_i = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \boldsymbol{s}_i = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix} \Rightarrow L_i = |\boldsymbol{y}_i - \boldsymbol{s}_i| = |0 - 0.1| + |1 - 0.7| + |0 - 0.2|$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative (Jacobian Matrix) of Linear Equation

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \quad \longleftrightarrow \quad \begin{aligned} s_1 &= \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 \\ s_2 &= \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} + b_2 \\ &\vdots \\ s_n &= \boldsymbol{w}_n^{\mathrm{T}}\boldsymbol{x} + b_n \end{aligned}$$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\frac{\partial s_1}{\partial \boldsymbol{w}_1} = \boldsymbol{x}$$

$$\frac{\partial s_2}{\partial \boldsymbol{w}_1} = \mathbf{0}$$

$$\vdots$$

$$\frac{\partial s_n}{\partial \boldsymbol{w}_1} = \mathbf{0}$$

$$\Rightarrow \quad \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_1} = [\boldsymbol{x}\ \mathbf{0}\ \mathbf{0}\ \cdots \mathbf{0}] \in \Re^{d\times n}$$

j[th] column

$$\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j} = [\mathbf{0}\ \mathbf{0}\ \boldsymbol{x}\ \cdots \mathbf{0}] \in \Re^{d\times n}$$

$$\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} = \mathbf{I} \in \Re^{n\times n}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative (Jacobian Matrix) of Linear Equation

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \quad \longleftrightarrow \quad \begin{aligned} s_1 &= \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 \\ s_2 &= \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} + b_2 \\ &\vdots \\ s_n &= \boldsymbol{w}_n^{\mathrm{T}}\boldsymbol{x} + b_n \end{aligned}$$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\frac{\partial s_1}{\partial \boldsymbol{x}} = \boldsymbol{w}_1$$

$$\frac{\partial s_2}{\partial \boldsymbol{x}} = \boldsymbol{w}_2$$

$$\vdots$$

$$\frac{\partial s_n}{\partial \boldsymbol{x}} = \boldsymbol{w}_n$$

$$\Rightarrow \quad \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \cdots \boldsymbol{w}_n] = \mathbf{W}^{\mathrm{T}} \in \Re^{d \times n}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative (Jacobian Matrix) of Sigmoid Function

Sigmoid function

For a scalar x

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \rightarrow \quad \frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{1+e^{-x}} \frac{1}{1+e^{-x}} = (1-\sigma(x))\sigma(x)$$

Similarly, for a vector $\boldsymbol{s} \in \Re^{n\times 1}$

$$\boldsymbol{p} = \sigma(\boldsymbol{s}) = \frac{1}{1+e^{-\boldsymbol{s}}} \quad \rightarrow \quad \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = diag\big((1-\sigma(s_j))\sigma(s_j)\big) = \begin{bmatrix} (1-\sigma(s_1))\sigma(s_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (1-\sigma(s_n))\sigma(s_n) \end{bmatrix}$$

for $j = 1, \ldots, n$

Partial Derivative (Jacobian Matrix) of Softmax Activation Function

- Softmax function

$$p_k = \frac{e^{s_k}}{\sum_{j=1}^{n} e^{s_j}} \Rightarrow \boldsymbol{p} = \frac{e^{\boldsymbol{s}}}{\sum_{j=1}^{n} e^{s_j}} \quad \text{in vector form}$$

score function $\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$ probability $\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}$

- 1st order derivative of softmax function

$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \frac{diag(e^{\boldsymbol{s}}) \cdot \sum e^{s_j} - e^{\boldsymbol{s}}(e^{\boldsymbol{s}})^{\mathrm{T}}}{(\sum e^{s_j})^2} = \frac{1}{(\sum e^{s_j})^2} \left\{ \begin{pmatrix} e^{s_1}\sum e^{s_j} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{s_n}\sum e^{s_j} \end{pmatrix} - \begin{pmatrix} e^{s_1}e^{s_1} & \cdots & e^{s_1}e^{s_n} \\ \vdots & \ddots & \vdots \\ e^{s_n}e^{s_1} & \cdots & e^{s_n}e^{s_n} \end{pmatrix} \right\}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative (Jacobian Matrix) of Softmax Activation Function

$$\mathbf{D} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \frac{diag(e^{\boldsymbol{s}}) \cdot \sum e^{s_j} - e^{\boldsymbol{s}}(e^{\boldsymbol{s}})^{\mathrm{T}}}{(\sum e^{s_j})^2} = \frac{1}{(\sum e^{s_j})^2}\left\{\begin{pmatrix} e^{s_1}\sum e^{s_j} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{s_n}\sum e^{s_j} \end{pmatrix} - \begin{pmatrix} e^{s_1}e^{s_1} & \cdots & e^{s_1}e^{s_n} \\ \vdots & \ddots & \vdots \\ e^{s_n}e^{s_1} & \cdots & e^{s_n}e^{s_n} \end{pmatrix}\right\}$$

For $a = b$

$$\frac{e^{s_a}(\sum e^{s_j} - e^{s_a})}{(\sum e^{s_j})^2} = p_a(1 - p_a)$$

For $a \neq b$

$$-\frac{e^{s_a}e^{s_b}}{(\sum e^{s_j})^2} = -p_a p_b$$

⇨

$$D_{ab} = p_a(\delta_{ab} - p_b)$$

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative (Jacobian Matrix) of Regression Loss

For simplicity of notation, i is omitted here

$$L = (\boldsymbol{y} - \boldsymbol{s})^2 = (\boldsymbol{y} - \mathbf{W}\boldsymbol{x} - \boldsymbol{b})^2$$
$$= \sum_{j=1}^{C} (y_j - \boldsymbol{w}_j^T \boldsymbol{x} - b_j)^2$$

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\Rightarrow s_j = \boldsymbol{w}_j^T \boldsymbol{x} + b_j$$

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_C^T \end{pmatrix} \quad \boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_C \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_C \end{pmatrix}$$

- 1st order derivative

$$\frac{\partial L}{\partial \boldsymbol{w}_j} = -2(y_j - \boldsymbol{w}_j^T \boldsymbol{x} - b_j)\boldsymbol{x} \quad \longleftrightarrow \quad \frac{\partial L}{\partial \mathbf{W}} = -2(\boldsymbol{y} - \mathbf{W}\boldsymbol{x} - \boldsymbol{b})\boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = -2(\boldsymbol{y} - \mathbf{W}\boldsymbol{x} - \boldsymbol{b})$$

이화여자대학교 EWHA WOMANS UNIVERSITY

Partial Derivative (Jacobian Matrix) of Regression Loss

For simplicity of notation, i is omitted here

$$L = (\boldsymbol{y} - \boldsymbol{s})^2 = (\boldsymbol{y} - \mathbf{W}\boldsymbol{x})^2$$
$$= \sum_{j=1}^{C} (y_j - \boldsymbol{w}_j^T \boldsymbol{x})^2$$

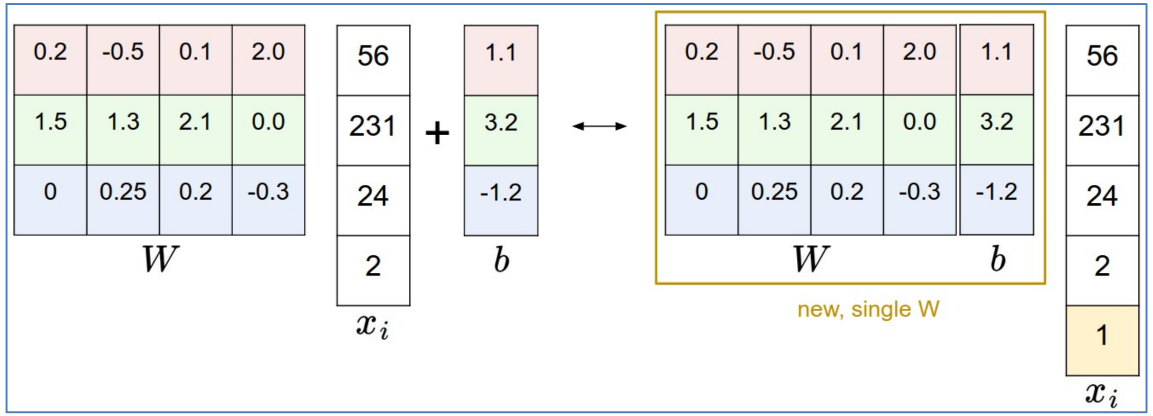
$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x}$$

$$\Rightarrow s_j = \boldsymbol{w}_j^T \boldsymbol{x}$$

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_C^T \end{pmatrix} \quad \boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_C \end{pmatrix}$$

- 1st order derivative

$$\frac{\partial L}{\partial \boldsymbol{w}_j} = -2(y_j - \boldsymbol{w}_j^T \boldsymbol{x})\boldsymbol{x} \quad \longleftrightarrow \quad \frac{\partial L}{\partial \mathbf{W}} = -2(\boldsymbol{y} - \mathbf{W}\boldsymbol{x})\boldsymbol{x}^{\mathrm{T}}$$

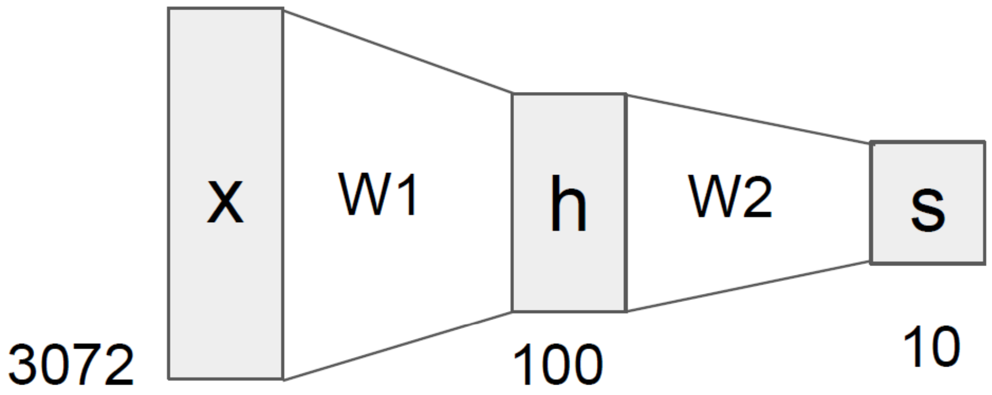


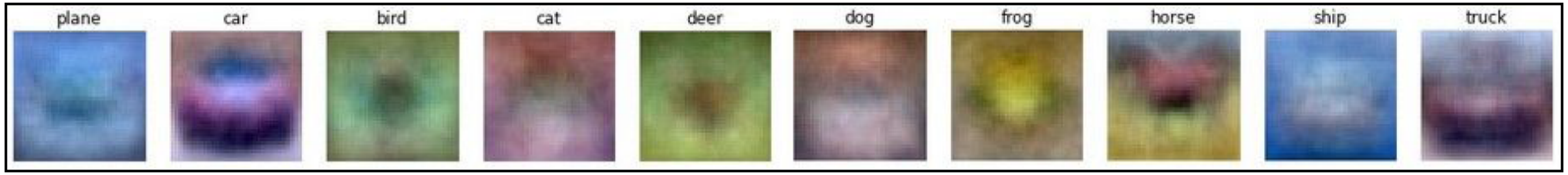
Neural Networks: Architectures

(**Before**) Linear score function: $\boldsymbol{f} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$

(**Now**) 2-layer Neural Network: $\boldsymbol{f} = \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$

3-layer Neural Network: $\boldsymbol{f} = \mathbf{W}_3 \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3$

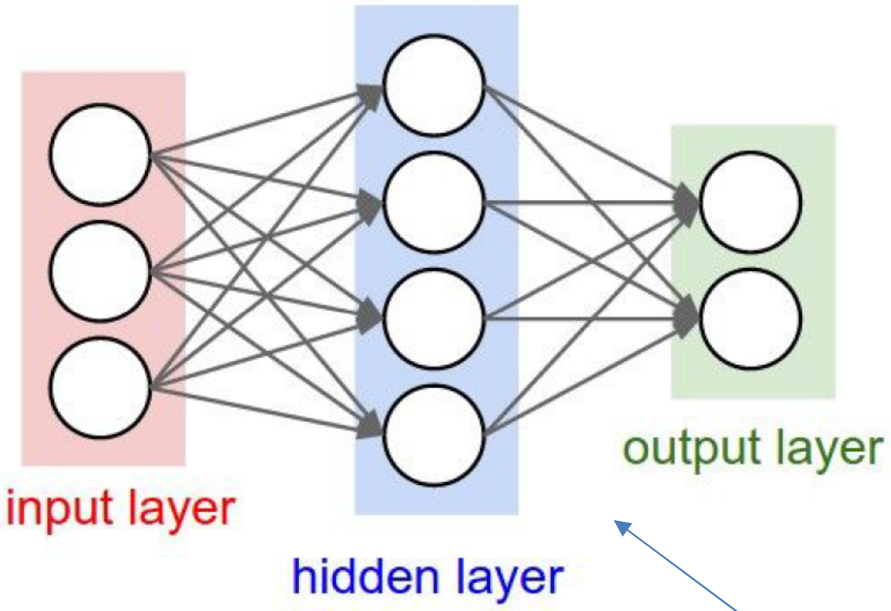




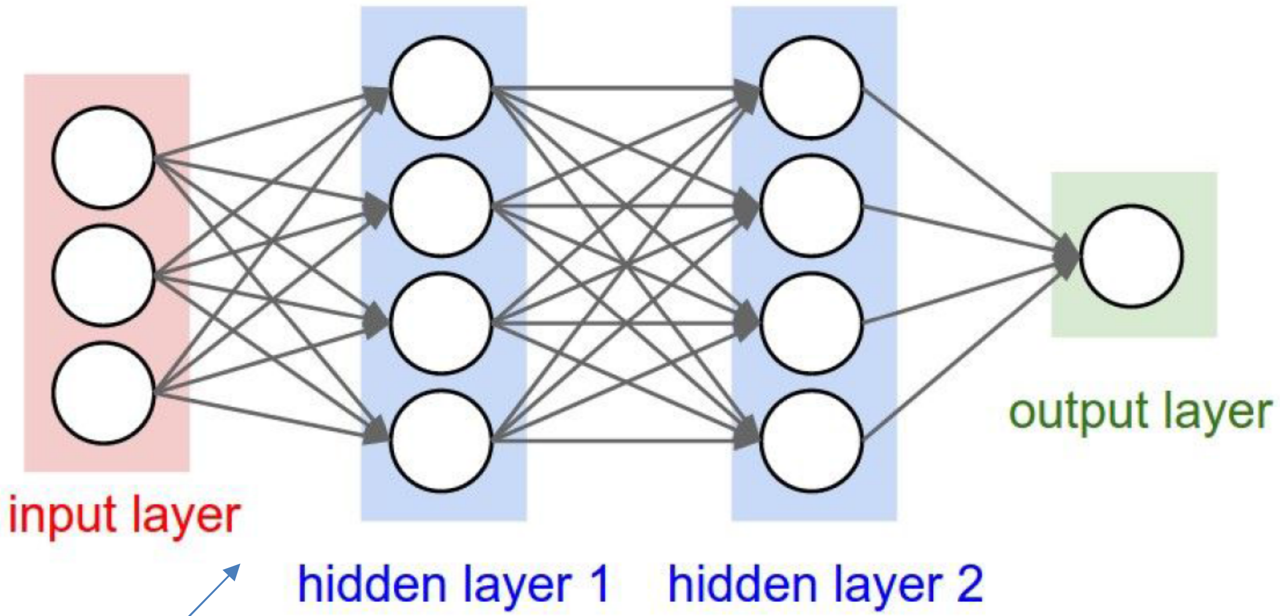
이화여자대학교
EWHA WOMANS UNIVERSITY

Neural Networks: Architectures

"2-layer Neural Net", or
"1-hidden-layer Neural Net"



"3-layer Neural Net", or
"2-hidden-layer Neural Net"



"Fully-connected" layers

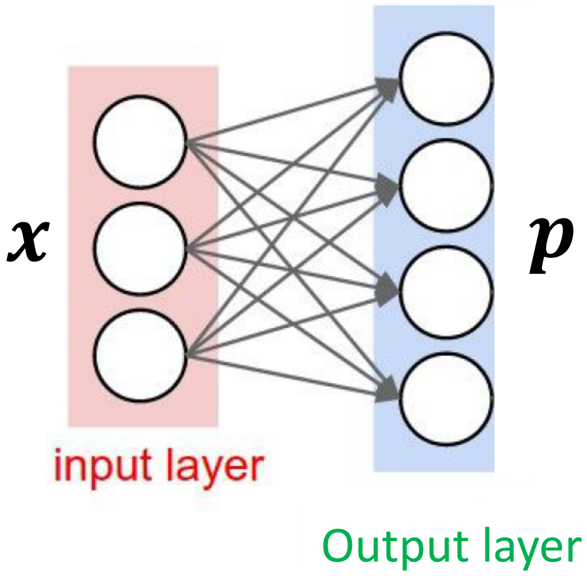
이화여자대학교
EWHA WOMANS UNIVERSITY

Derivative of Neural Net using Chain Rules

- **Example**
  1. 1-layer Neural Net (L2 regression loss)
  2. 2-layer Neural Net (L2 regression loss)

  3. 1-layer Neural Net (Softmax classifier)
  4. 2-layer Neural Net (Softmax classifier)

1. 1-layer Neural Net (L2 regression loss)

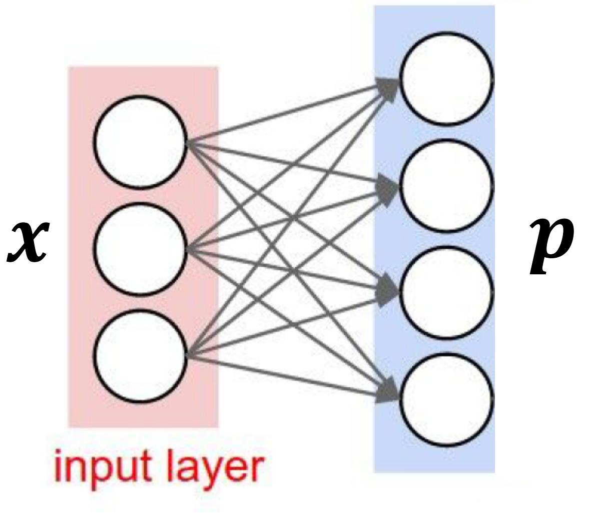


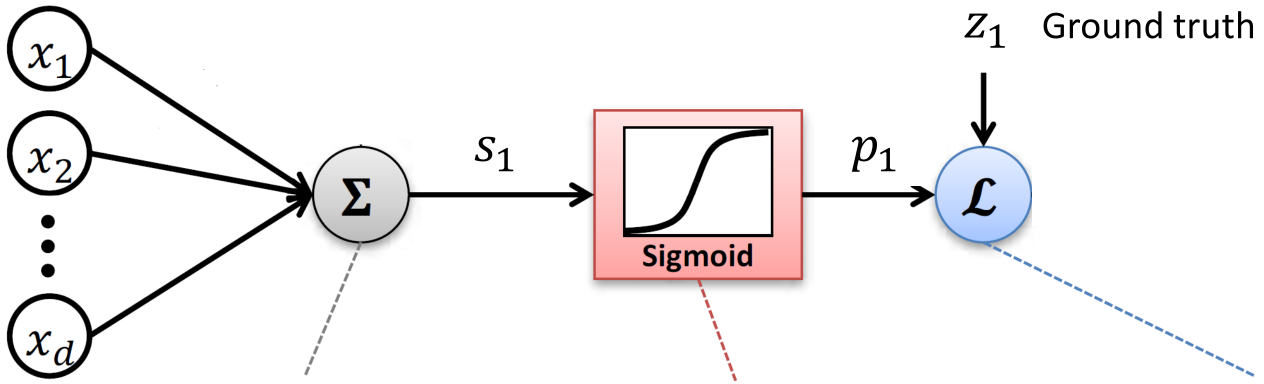
1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T \boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1 + e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

이화여자대학교
EWHA WOMANS UNIVERSITY

1. 1-layer Neural Net (L2 regression loss)





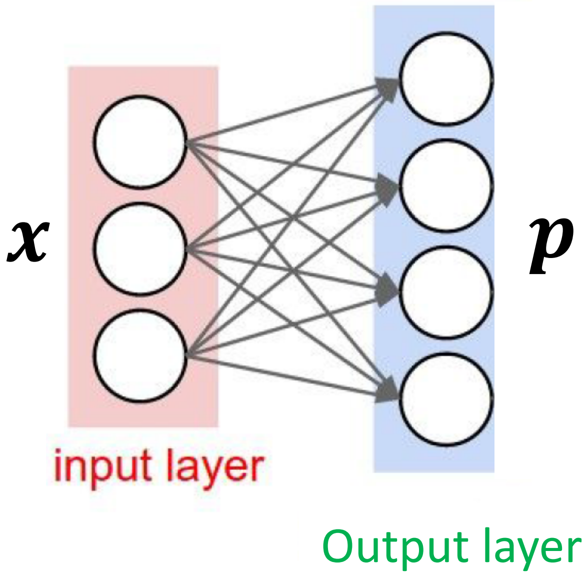
$$s_1 = \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 = \sum_{k=1}^{d} w_{1k}x_k + b_1 \qquad p_1 = \frac{1}{1+e^{-s_1}} \qquad (z_1 - p_1)^2$$

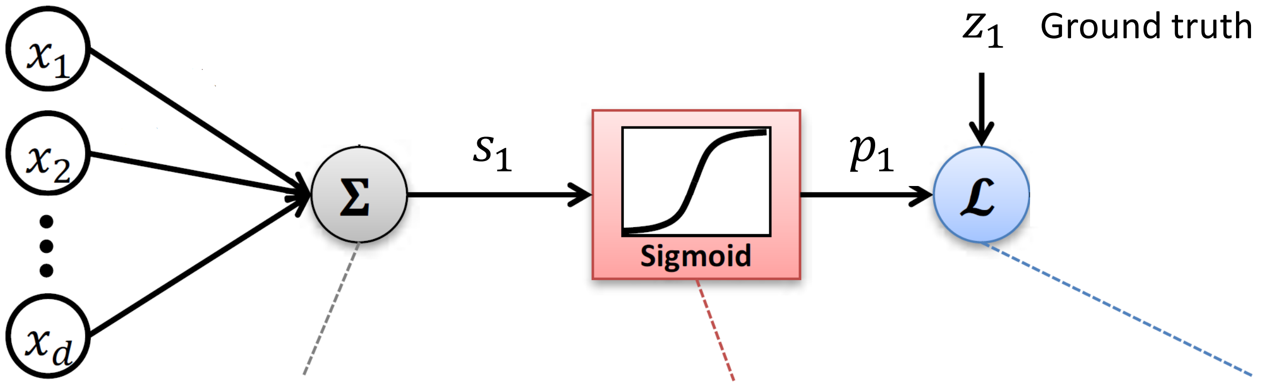
1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T\boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1+e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

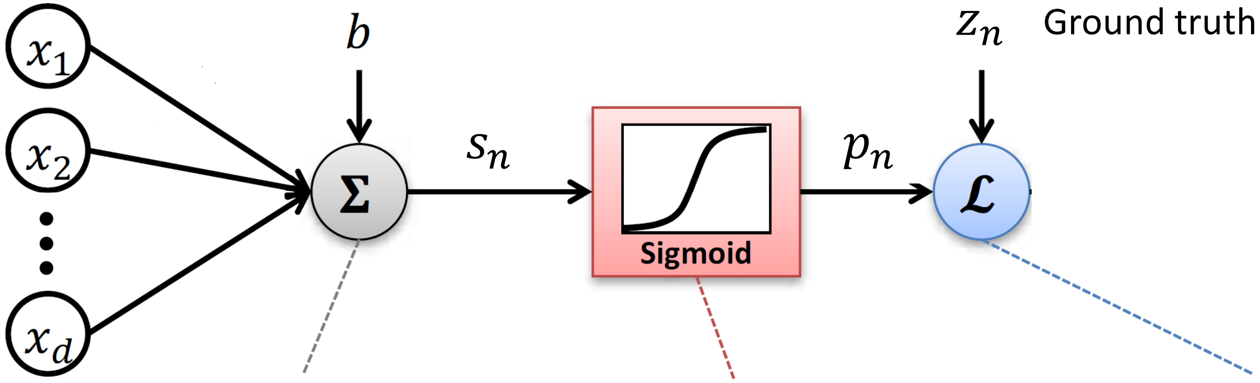
1. 1-layer Neural Net (L2 regression loss)





$$s_1 = \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 = \sum_{k=1}^{d} w_{1k}x_k + b_1 \qquad p_1 = \frac{1}{1+e^{-s_1}} \qquad (z_1 - p_1)^2$$

⋮

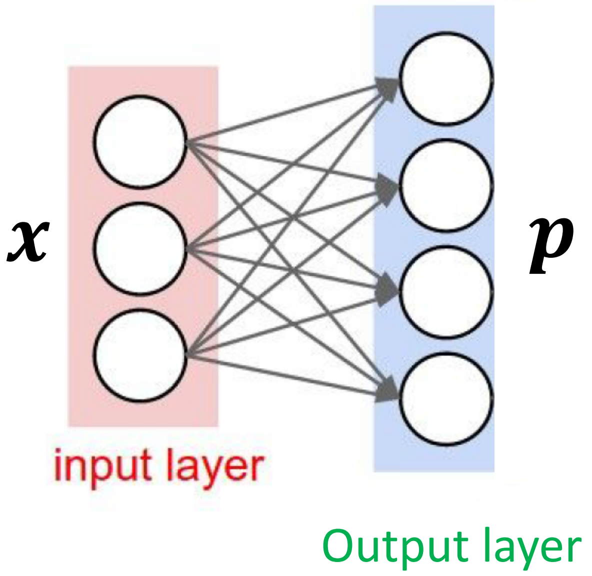


$$s_n = \boldsymbol{w}_n^{\mathrm{T}}\boldsymbol{x} + b_n = \sum_{k=1}^{d} w_{nk}x_k + b_n \qquad p_n = \frac{1}{1+e^{-s_n}} \qquad (z_n - p_n)^2$$

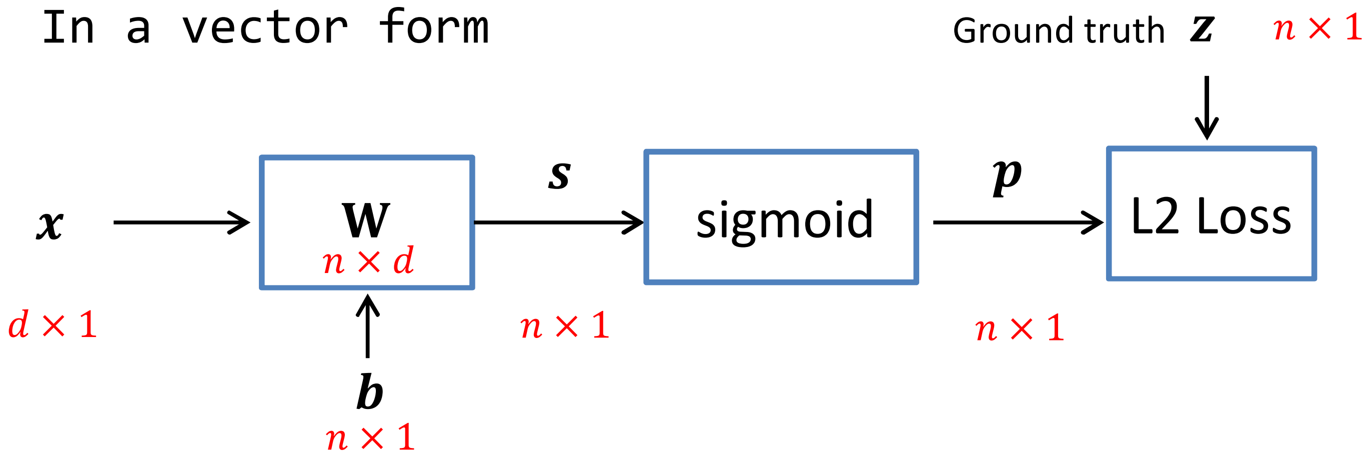
1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T\boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1+e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

1. 1-layer Neural Net (L2 regression loss)



In a vector form



1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T \boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1 + e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

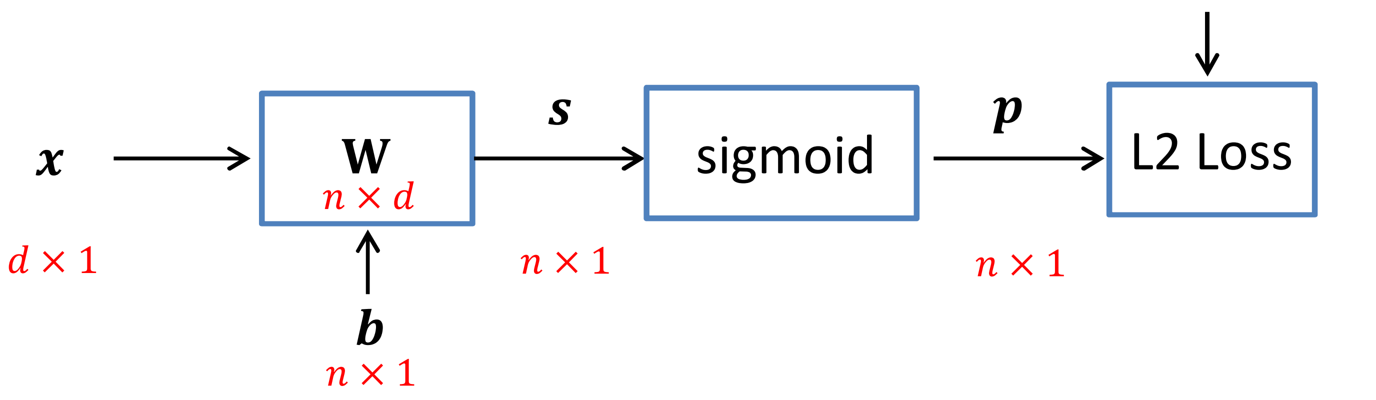
We need to compute gradients of $\mathbf{W}, \boldsymbol{b}, \boldsymbol{s}, \boldsymbol{p}$ with respect to the loss function L.

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

1. 1-layer Neural Net (L2 regression loss)

In a vector form



$$\frac{\partial L}{\partial \boldsymbol{p}} = -2(\boldsymbol{z} - \boldsymbol{p})$$

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = diag\Big((1-\sigma(s_j))\sigma(s_j)\Big) \frac{\partial L}{\partial \boldsymbol{p}} = -2 \begin{bmatrix} (1-\sigma(s_1))\sigma(s_1)(z_1 - p_1) \\ (1-\sigma(s_2))\sigma(s_2)(z_2 - p_2) \\ \vdots \\ (1-\sigma(s_n))\sigma(s_n)(z_n - p_n) \end{bmatrix} = (1-\sigma(\boldsymbol{s})) \otimes \sigma(\boldsymbol{s}) \otimes \frac{\partial L}{\partial \boldsymbol{p}}$$

$\otimes$: element-wise multiplication

j[th] column

$$\frac{\partial L}{\partial \boldsymbol{w}_j} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{X}_j \frac{\partial L}{\partial \boldsymbol{s}} = [\mathbf{0}\ \mathbf{0}\ \boldsymbol{x}\ \cdots \mathbf{0}] \frac{\partial L}{\partial \boldsymbol{s}} = \left(\frac{\partial L}{\partial \boldsymbol{s}}\right)_j \boldsymbol{x}$$

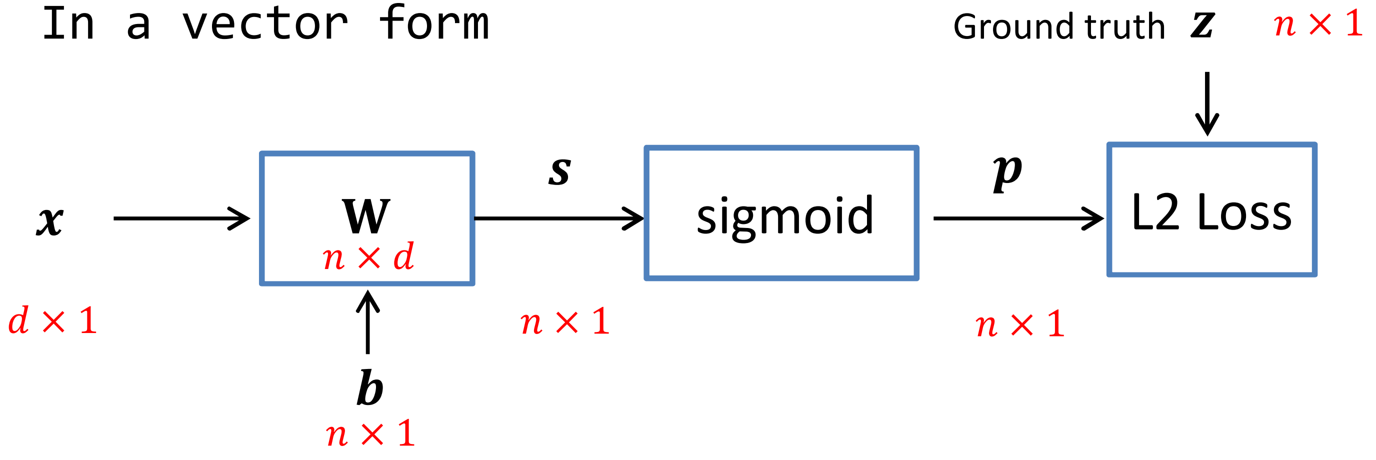
$(\boldsymbol{a})_j$: j[th] element at vector $\boldsymbol{a}$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \boldsymbol{w}_1} \quad \frac{\partial L}{\partial \boldsymbol{w}_2} \quad \cdots \quad \frac{\partial L}{\partial \boldsymbol{w}_n}\right)^{\mathrm{T}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} \frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

1. 1-layer Neural Net (L2 regression loss)

In a vector form



Summary

$$\frac{\partial L}{\partial \boldsymbol{p}} = -2(\boldsymbol{z} - \boldsymbol{p})$$

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = (1 - \sigma(\boldsymbol{s})) \otimes \sigma(\boldsymbol{s}) \otimes \frac{\partial L}{\partial \boldsymbol{p}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}}$$

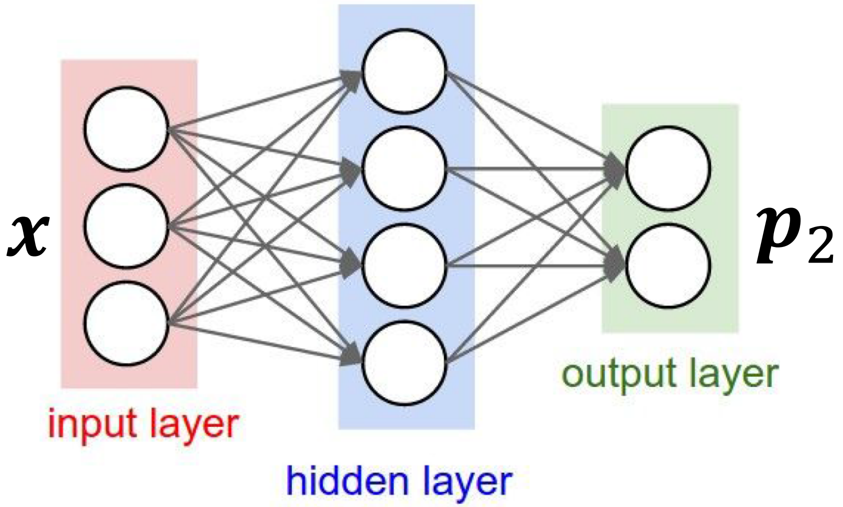
$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

Note that the following derivative can also be computed, but here $\boldsymbol{x}$ is an input data that is fixed during training. Thus, it is not necessary to compute its derivative.

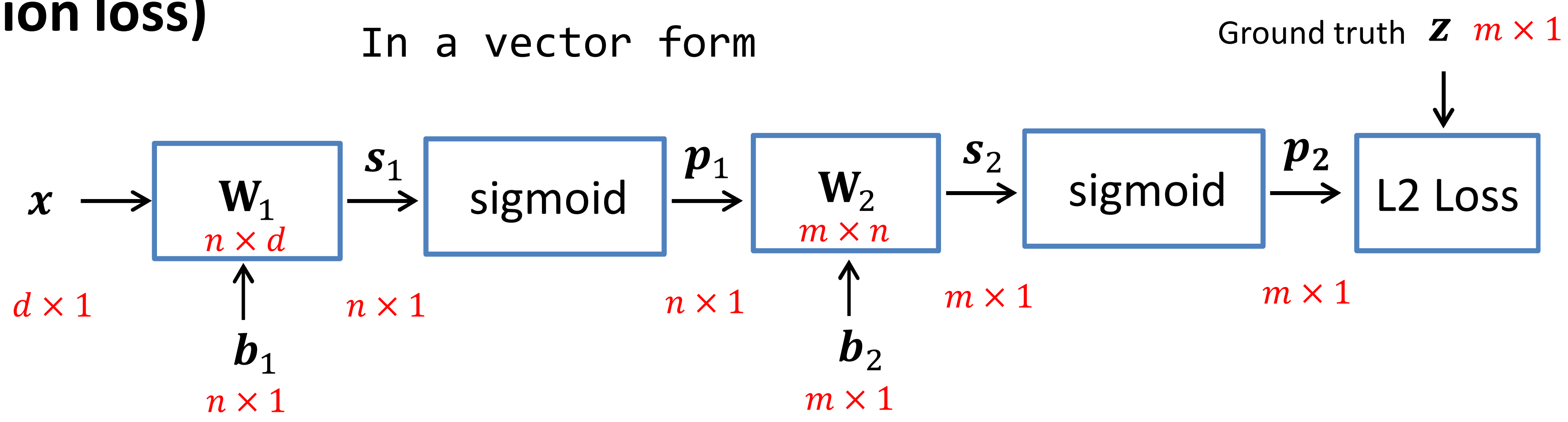
$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

2. 2-layer Neural Net (L2 regression loss)



In a vector form



$$\frac{\partial L}{\partial \boldsymbol{p}_2} = -2(\boldsymbol{z} - \boldsymbol{p}_2)$$

$$\frac{\partial L}{\partial \boldsymbol{s}_2} = \frac{\partial \boldsymbol{p}_2}{\partial \boldsymbol{s}_2}\frac{\partial L}{\partial \boldsymbol{p}_2} = diag\big((1-\sigma(s_{2,j}))\sigma(s_{2,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_2}$$

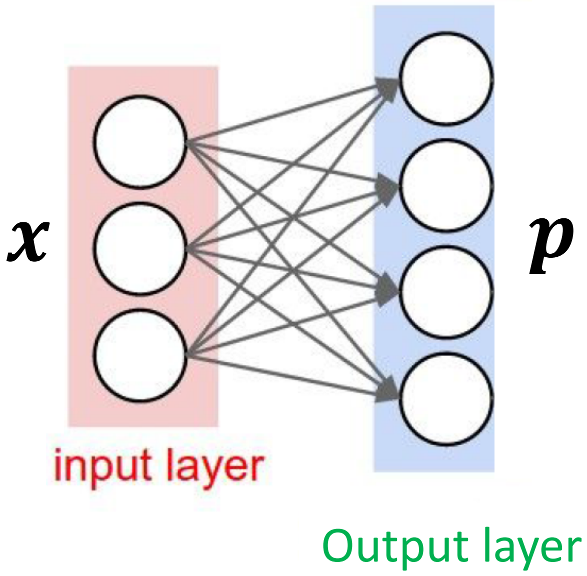
$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}\boldsymbol{p}_1^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}$$

$$\frac{\partial L}{\partial \boldsymbol{p}_1} = \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{p}_1}\frac{\partial L}{\partial \boldsymbol{s}_2} = \mathbf{W}_2^{\mathrm{T}}\frac{\partial L}{\partial \boldsymbol{s}_2}$$

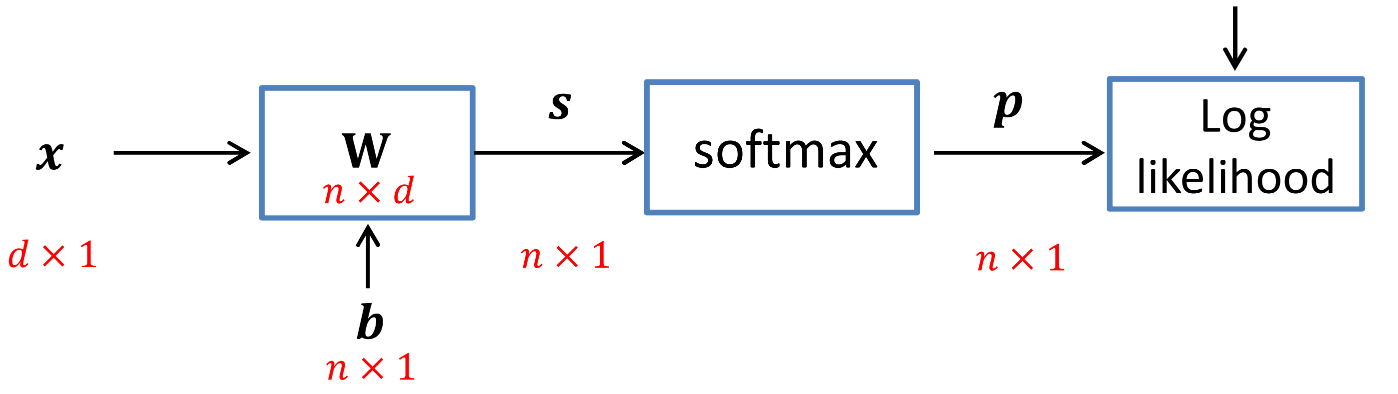
$$\frac{\partial L}{\partial \boldsymbol{s}_1} = \frac{\partial \boldsymbol{p}_1}{\partial \boldsymbol{s}_1}\frac{\partial L}{\partial \boldsymbol{p}_1} = diag\big((1-\sigma(s_{1,j}))\sigma(s_{1,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_1}$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}\boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}$$

3. 1-layer Neural Net (Softmax classifier)



In a vector form



1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T\boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \dfrac{e^{\boldsymbol{s}}}{\sum_{j=1}^{n} e^{s_j}}$
3. Loss $L = -\log p_y$ where y satisfies $z_y = 1$

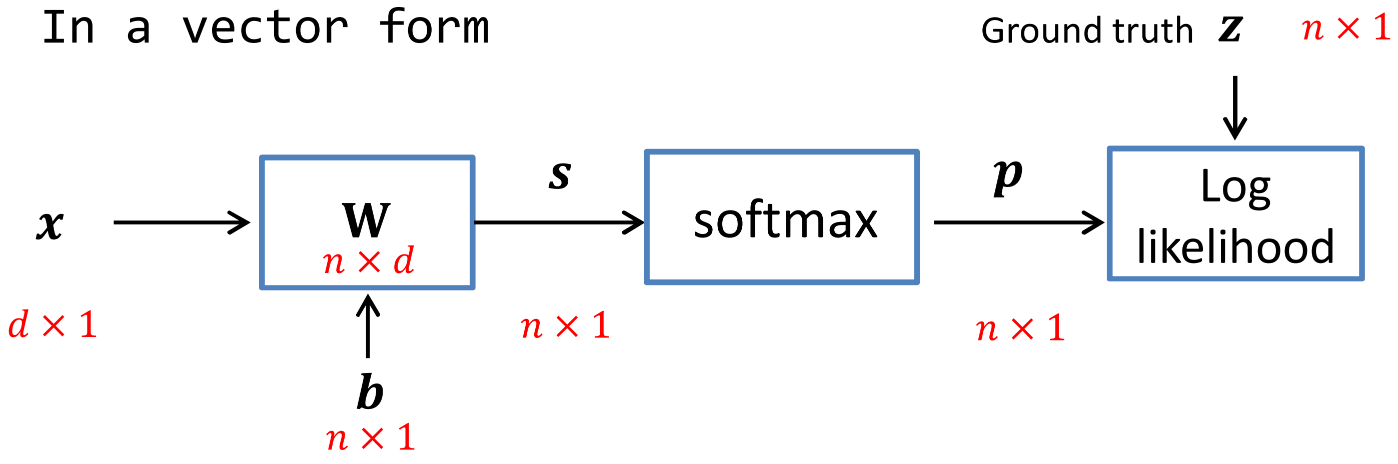
   For $\boldsymbol{z} = (z_1\ z_2 \ldots z_n)^T$, $z_y = 1$ and $z_{k \neq y} = 0$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

We need to compute gradients of $\mathbf{W}, \boldsymbol{b}, \boldsymbol{s}, \boldsymbol{p}$ with respect to the loss function L.

3. 1-layer Neural Net (Softmax classifier)

In a vector form



$$\frac{\partial L}{\partial \boldsymbol{p}} = \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix}$$

y^{th} row

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = \mathbf{D} \frac{\partial L}{\partial \boldsymbol{p}} = -\frac{1}{p_y} \begin{bmatrix} D_{1y} \\ D_{2y} \\ \vdots \\ D_{ny} \end{bmatrix} = \boldsymbol{p} - \boldsymbol{z}$$

$$D_{ab} = p_a(\delta_{ab} - p_b)$$

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

j^{th} column

$$\frac{\partial L}{\partial \boldsymbol{w}_j} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{X}_j \frac{\partial L}{\partial \boldsymbol{s}} = [\mathbf{0}\ \mathbf{0}\ \boldsymbol{x}\ \cdots\ \mathbf{0}] \frac{\partial L}{\partial \boldsymbol{s}} = \left(\frac{\partial L}{\partial \boldsymbol{s}}\right)_j \boldsymbol{x}$$

$(\boldsymbol{a})_j$: j^{th} element at vector $\boldsymbol{a}$

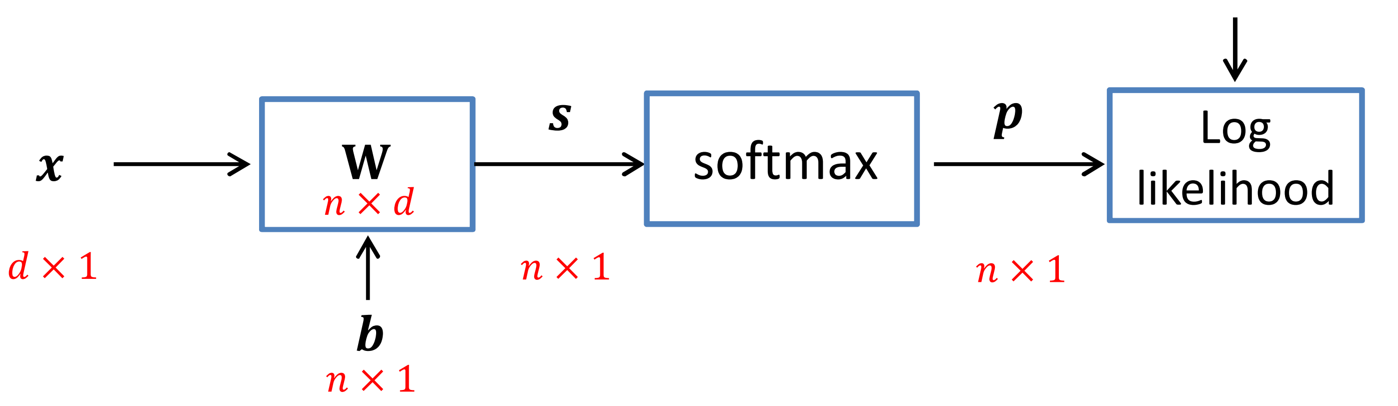
$$\Rightarrow \frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \boldsymbol{w}_1} \quad \frac{\partial L}{\partial \boldsymbol{w}_2} \quad \cdots \quad \frac{\partial L}{\partial \boldsymbol{w}_n}\right)^{\mathrm{T}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} \frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

3. 1-layer Neural Net (Softmax classifier)

In a vector form



Summary

$$\frac{\partial L}{\partial \boldsymbol{p}} = \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix}$$

y^{th} row

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = \mathbf{D} \frac{\partial L}{\partial \boldsymbol{p}} = -\frac{1}{p_y} \begin{bmatrix} D_{1y} \\ D_{2y} \\ \vdots \\ D_{ny} \end{bmatrix} = \boldsymbol{p} - \boldsymbol{z}$$

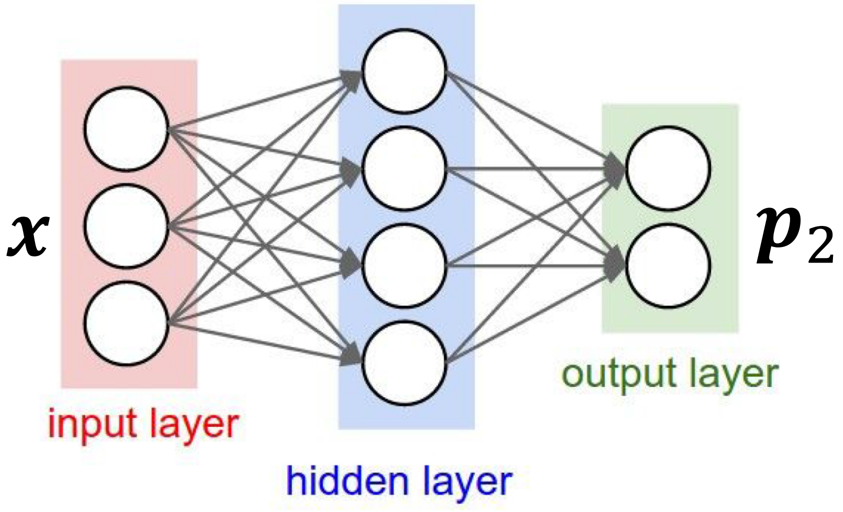
$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

Note that the following derivative can also be computed, but here $\boldsymbol{x}$ is an input data that is fixed during training. Thus, it is not necessary to compute its derivative.

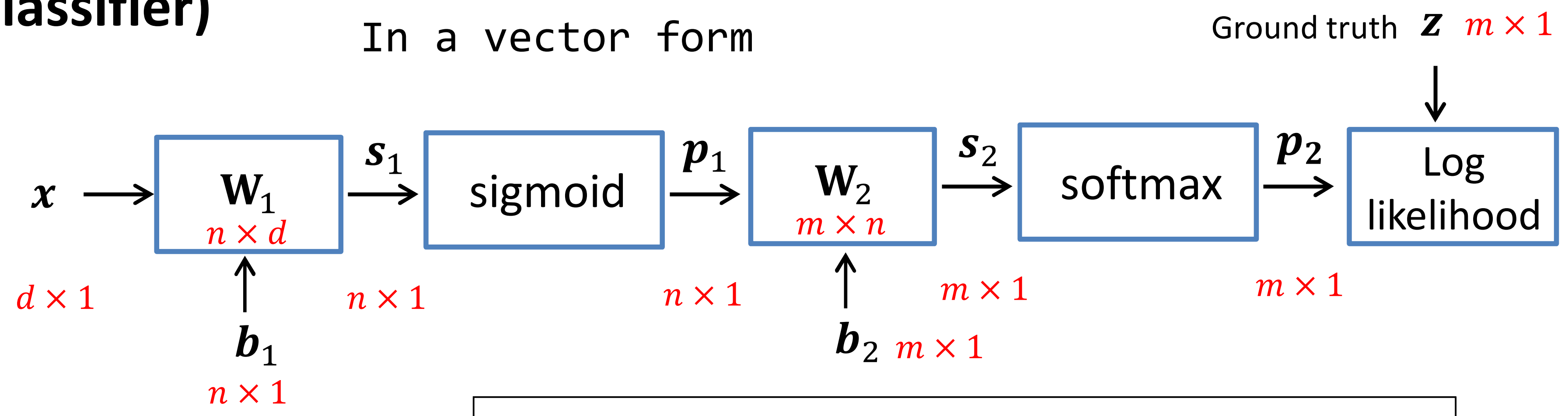
$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

이화여자대학교 EWHA WOMANS UNIVERSITY

4. 2-layer Neural Net (Softmax classifier)



In a vector form



$$\frac{\partial L}{\partial \boldsymbol{p}_2} = \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix}$$

y^th row

$$\frac{\partial L}{\partial \boldsymbol{s}_2} = \frac{\partial \boldsymbol{p}_2}{\partial \boldsymbol{s}_2}\frac{\partial L}{\partial \boldsymbol{p}_2} = \mathbf{D}\frac{\partial L}{\partial \boldsymbol{p}_2}$$

$$D_{ab} = p_a(\delta_{ab} - p_b)$$

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial L}{\partial \mathbf{W_2}} = \frac{\partial L}{\partial \boldsymbol{s}_2}\boldsymbol{p}_1^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{p}_1} = \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{p}_1}\frac{\partial L}{\partial \boldsymbol{s}_2} = \mathbf{W}_2^{\mathrm{T}}\frac{\partial L}{\partial \boldsymbol{s}_2} \qquad \frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}$$

$$\frac{\partial L}{\partial \boldsymbol{s}_1} = \frac{\partial \boldsymbol{p}_1}{\partial \boldsymbol{s}_1}\frac{\partial L}{\partial \boldsymbol{p}_1} = diag\big((1-\sigma(s_{1,j}))\sigma(s_{1,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_1}$$

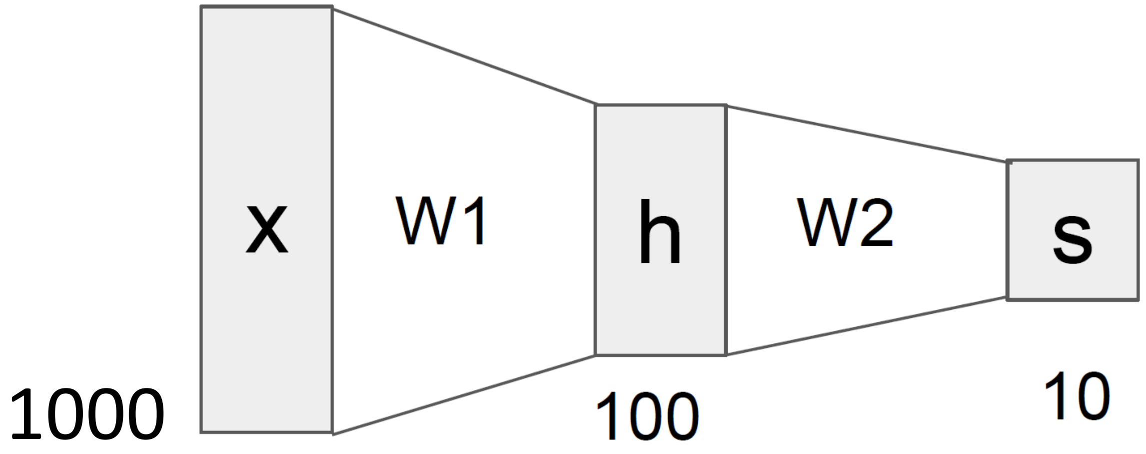
$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}\boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}$$

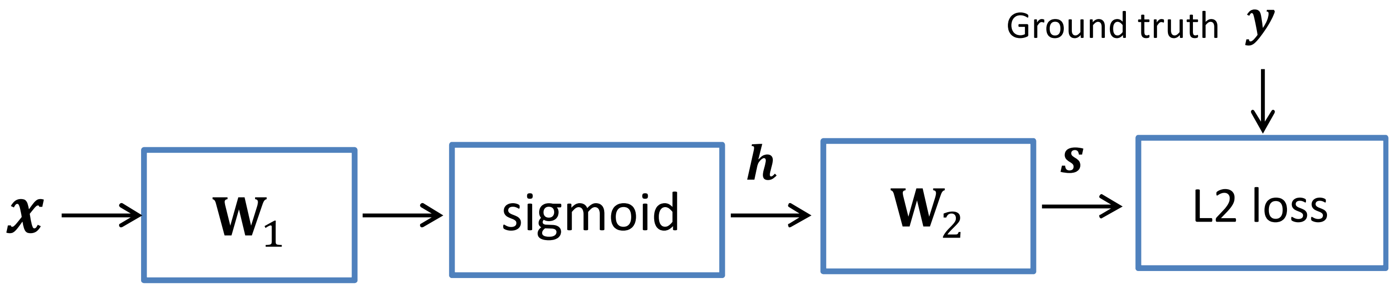
EWHA WOMANS UNIVERSITY

Full implementation of training a 2-layer Neural Network

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

N: batch size
D_in: input feature size
H: input feature size of the second layer
D_out: output feature size





이화여자대학교
EWHA WOMANS UNIVERSITY