Machine Learning 2:
Multi-Layer Perceptrons (MLPs), Backpropagation

유환조, POSTECH

http://di.postech.ac.kr/hwanjoyu

Roadmap

Features

Multi-Layer Perceptron (MLP)

Backpropagation

Cross entropy loss

Two components in linear predictor

Linear predictor:

$$\mathbf{w} \cdot \phi(\mathbf{x})$$

- Assume learning choose the optimal $\mathbf{w}$.
- How does feature extraction affect quality of $f_{\mathbf{w}}$?

Hypothesis class: example

Regression: $x \in \mathbb{R}, y \in \mathbb{R}$

Linear functions:

$$\phi(x) = x$$
$$\mathcal{F}_1 = \{x \longmapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 = 0\}$$

Quadratic functions:

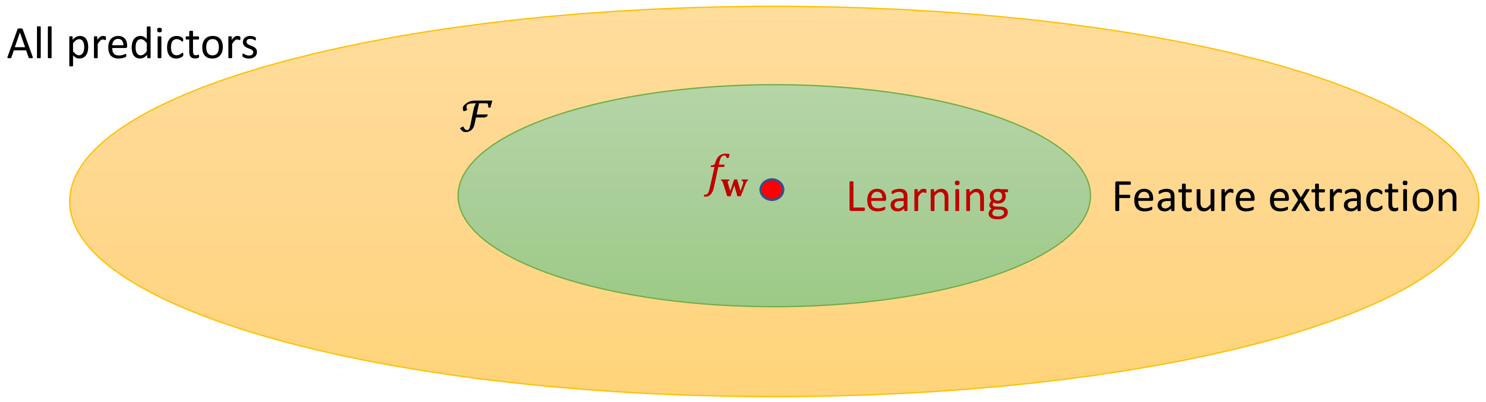
$$\phi(x) = [x, x^2]$$
$$\mathcal{F}_2 = \{x \longmapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

Hypothesis class

Definition: hypothesis class (or function class)

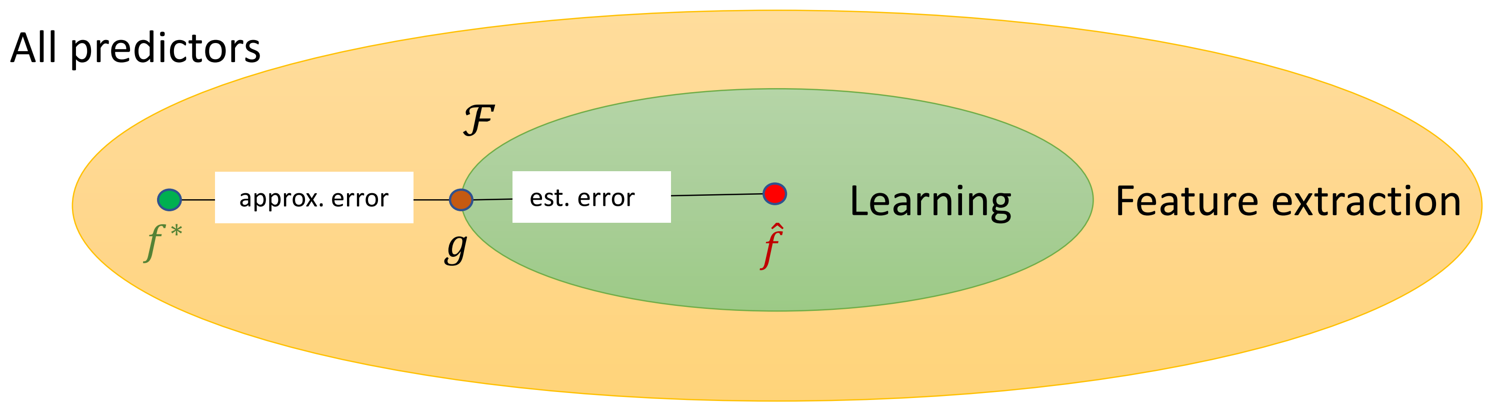
- A **hypothesis class** is the set of possible predictors with a fixed $\phi(x)$ and varying $\mathbf{w}$.

$$\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^d\}$$



Question: does $\mathcal{F}$ contain a good predictor?

Approximation error and estimation error



- Approximation error: how good is the hypothesis class?
- Estimation error: how good is the learned predictor **relative to** the hypothesis class?

$$\underbrace{Err\left(\hat{f}\right) - Err\left(g\right)}_{\text{estimation}} + \underbrace{Err\left(g\right) - Err\left(f^*\right)}_{\text{approximation}}$$

Effect of hypothesis class size

As the hypothesis class size increases …

Approximation error decreases because…

Estimation error increases because..

Features in linear model

Three issues (**non-linearity** in original measurements):

- Non-monotonicity
- Saturation
- Interactions between features

Example: predicting health (extract any features that might be relevant)

- Input: patient information $\mathbf{x}$
- Output: health $y \in \mathbb{R}$ (positive is good)

Features for medical diagnosis: height, weight, body temperature, blood pressure, etc.

Non-monotonicity

Features: $\phi(\mathbf{x}) = [\text{temperature}(\mathbf{x}), 1]$

Output: health $y \in \mathbb{R}$

Linear model: $y = w_1 \text{t}(\mathbf{x}) + w_0$

Problem: favor extremes; true relationship is non-monotonic

Non-monotonicity: attempt

Attempt 1: Add quadratic features

$$\phi(\mathbf{x}) = [(\text{temperature}(\mathbf{x}) - 37)^2, 1],$$

Linear model: $y = w_1(\text{t}(\mathbf{x}) - 37)^2 + w_0$

Disadvantage: requires manually-specified domain knowledge

Attempt 2: *Design features to be simple building blocks to be pieced together!*

$$\phi(\mathbf{x}) = [\text{temperature}(\mathbf{x})^2, \text{temperature}(\mathbf{x}), 1]$$

Linear model: $y = w_2\text{t}(\mathbf{x})^2 + w_1\text{t}(\mathbf{x}) + w_0$

Saturation

Example: product recommendation

- Input: product information $\mathbf{x}$
- Output: relevance $y \in \mathbb{R}$

Let $N(\mathbf{x})$ be number of people who bought $\mathbf{x}$

- Identity: $\phi(\mathbf{x}) = [N(\mathbf{x}), 1]$

Linear model: $y = w_1 N(\mathbf{x}) + w_0$

Problem: is 1000 people really 10 times more relevant than 100 people? Not quite…

Saturation: attempt

Attempt 1: Add logarithmic features

$$\phi(\mathbf{x}) = [\log N(\mathbf{x}), 1]$$

Linear model: $y = w_1 \log N(\mathbf{x}) + w_0$

Disadvantage: requires manually-specified domain knowledge

Attempt 2: Approximate with piece-wise linear models

$$\phi(\mathbf{x}) = [\mathbf{1}[0 < N(\mathbf{x}) < 10], \mathbf{1}[10 < N(\mathbf{x}) < 20], \ldots, 1]$$

Linear model: $y = w_1[0 < N(\mathbf{x}) < 10] + w_2[0 < N(\mathbf{x}) < 10] + \cdots + w_0$

Interaction between features

Example: health prediction

- Input: patient information $\mathbf{x}$
- Output: health $y \in \mathbb{R}$ (positive is good)

$$\phi(\mathbf{x}) = [\text{height}(\mathbf{x}), \text{weight}(\mathbf{x})]$$

Problem: can’t capture relationship between height and weight.

Interaction between features: attempt

Attempt 1: add complex features

$$\phi(\mathbf{x}) = [(52 + 1.9(\text{height}(\mathbf{x}) - 60) - \text{weight}(\mathbf{x}))^2, 1]$$

Disadvantage: requires manually-specified domain knowledge

Attempt 2: add features involving multiple measurements

$$\phi(\mathbf{x}) = [1, \text{height}(\mathbf{x}), \text{weight}(\mathbf{x}), \text{height}(\mathbf{x})^2, \text{weight}(\mathbf{x})^2, \underbrace{\text{height}(\mathbf{x})\text{weight}(\mathbf{x})}_{\text{cross term}}]$$

Linear in what?

Prediction driven by score:

$$\mathbf{w} \cdot \phi(\mathbf{x})$$

- Linear in $\mathbf{w}$? Yes
- Linear in $\phi(\mathbf{x})$? Yes
- Linear in $\mathbf{x}$? No! (not necessarily even a vector)

Key idea: non-linearity

- Predictors $f_{\mathbf{w}}(\mathbf{x})$ can be expressive **non-linear** functions and decision boundaries of $\mathbf{x}$.
- Score $\mathbf{w} \cdot \phi(\mathbf{x})$ is **linear** function of $\mathbf{w}$, which permits efficient learning.

Roadmap

Features

Multi-Layer Perceptron (MLP)

Backpropagation

Cross entropy loss

Motivating example

Example: predicting car collision

- Input: position of two oncoming cars $\mathbf{x} = [x_1, x_2]$
- Output: whether safe $(y = +1)$ or collide $(y = -1)$

True function: safe if cars sufficiently far

$$y = \text{sign}(|x_1 - x_2| - 1)$$

Examples:

x	y
[1,3]	+1
[3,1]	+1
[1,0.5]	-1

Decomposing the problem

Test if car 1 is far right of car 2:

$$h_1 = \mathbf{1}[x_1 - x_2 \geq 1]$$

Test if car 2 is far right of car 1:

$$h_2 = \mathbf{1}[x_2 - x_1 \geq 1]$$

Safe if at least one is true:

$$y = \text{sign}(h_1 + h_2)$$

x	h_1	h_2	y
[1,3]	0	1	+1
[3,1]	1	0	+1
[1,0.5]	0	0	-1

Decomposing the problem

Define: $\phi(\mathbf{x}) = [1, x_1, x_2]$:

Intermediate hidden subproblems:

$$h_1 = \mathbf{1}[\mathbf{v}_1 \cdot \phi(\mathbf{x}) \geq 0] \qquad \mathbf{v}_1 = [-1, +1, -1]$$

$$h_2 = \mathbf{1}[\mathbf{v}_2 \cdot \phi(\mathbf{x}) \geq 0] \qquad \mathbf{v}_2 = [-1, -1, +1]$$

Final prediction:

$$f_{\mathbf{V},\mathbf{w}}(\mathbf{x}) = \text{sign}(w_1 h_1 + w_2 h_2) \qquad \mathbf{w} = [1,1]$$

Key idea: joint learning

- Goal: learn both hidden subproblems $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2)$ and combination weights $\mathbf{w} = [w_1, w_2]$

Sigmoid activation

Problem: gradient of h_1 with respect to $\mathbf{v}_1$ is 0.

$$h_1 = \mathbf{1}[\mathbf{v}_1 \cdot \phi(\mathbf{x}) \geq 0]$$

Definition: logistic function (or sigmoid function)

- The logistic function maps $(-\infty, \infty)$ to $[0,1]$:

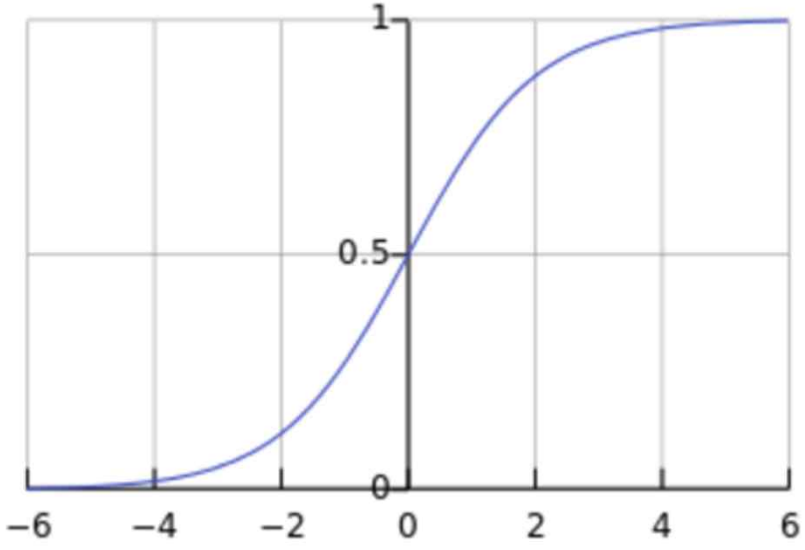
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Derivative of sigmoid:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

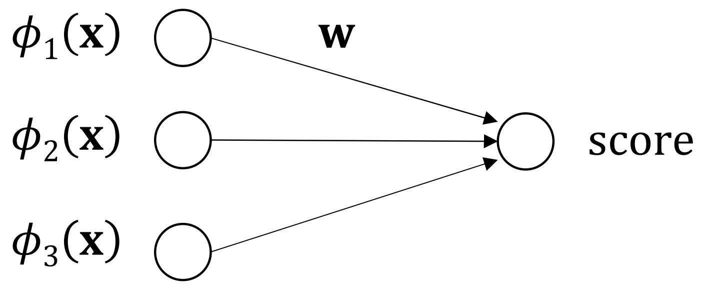
Solution:

$$h_1 = \sigma(\mathbf{v}_1 \cdot \phi(\mathbf{x}))$$



Linear predictors

Linear predictor:

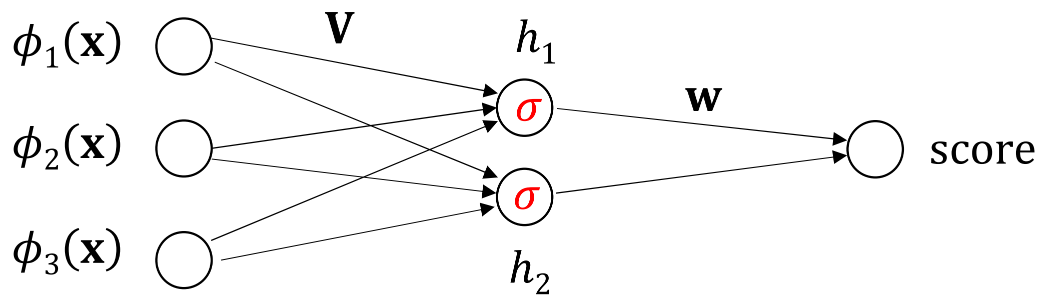


Output:

$$\text{score} = \mathbf{w} \cdot \phi(\mathbf{x})$$

Neural networks

Neural network:



Intermediate hidden units:

$$h_j = \sigma(\mathbf{v}_j \cdot \phi(\mathbf{x})) \qquad \sigma(z) = (1 + e^{-z})^{-1}$$

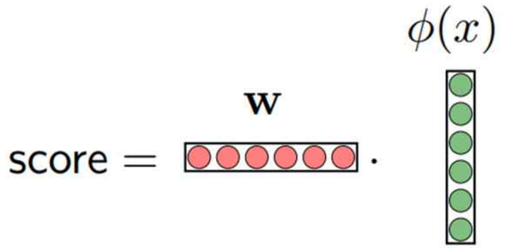
Output:

$$\text{score} = \mathbf{w} \cdot \mathbf{h}$$

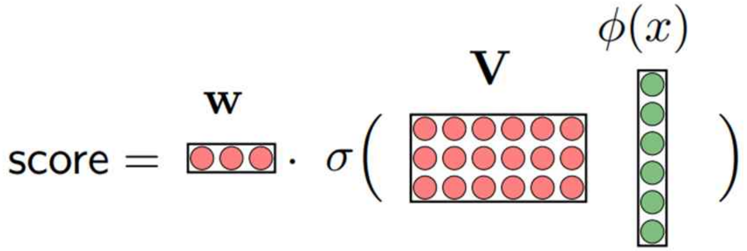
Note: In neural network, σ is called activation function. Traditionally the sigmoid function was used, but the **rectified linear function σ(z) = max{z,0}** is now popularly used.

Deep neural networks

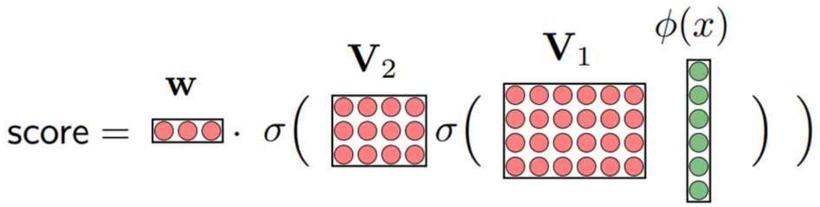
1-layer neural network:



2-layer neural network:

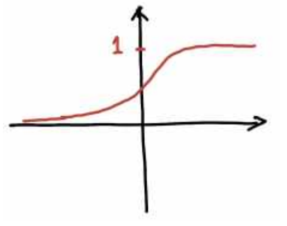


3-layer neural network:

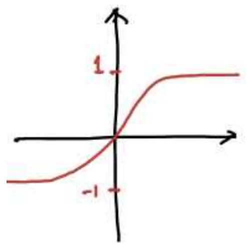


Activation functions

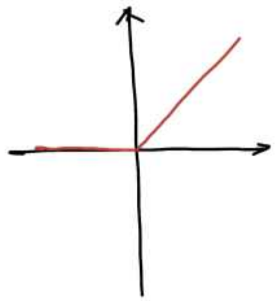
- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



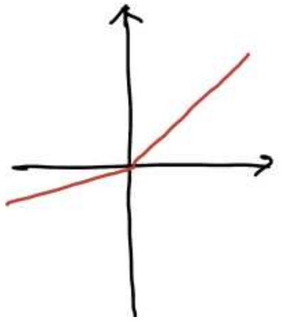
- Tanh: $\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



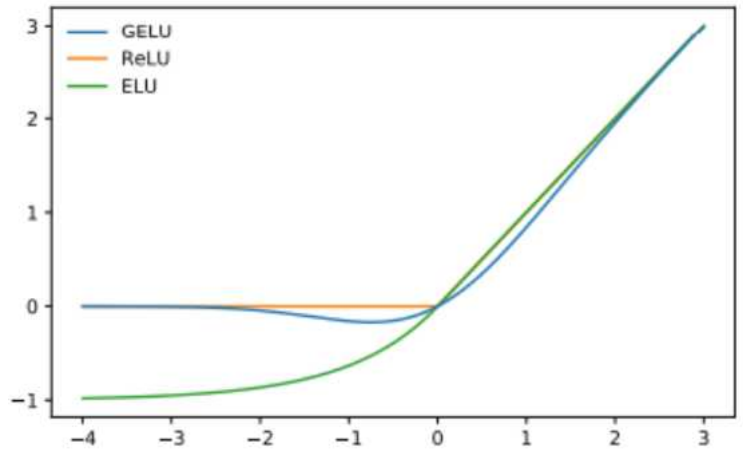
- ReLU: $\sigma(z) = \max(z, 0)$
  - Cheap to compute
  - Alleviate vanishing gradient problem
  - Sparsely activated
  - Dying ReLU neuron problem



- Leaky ReLU: $\sigma(z) = \max(z, 0.01z)$



- ELU, SELU, GELU, …



Neural networks

Think of intermediate hidden units as learned features of a linear predictor.

Key idea: feature learning

- Before: apply linear predictor on manually specified features

$$\phi(\mathbf{x})$$

- Now: apply linear predictor on automatically learned features

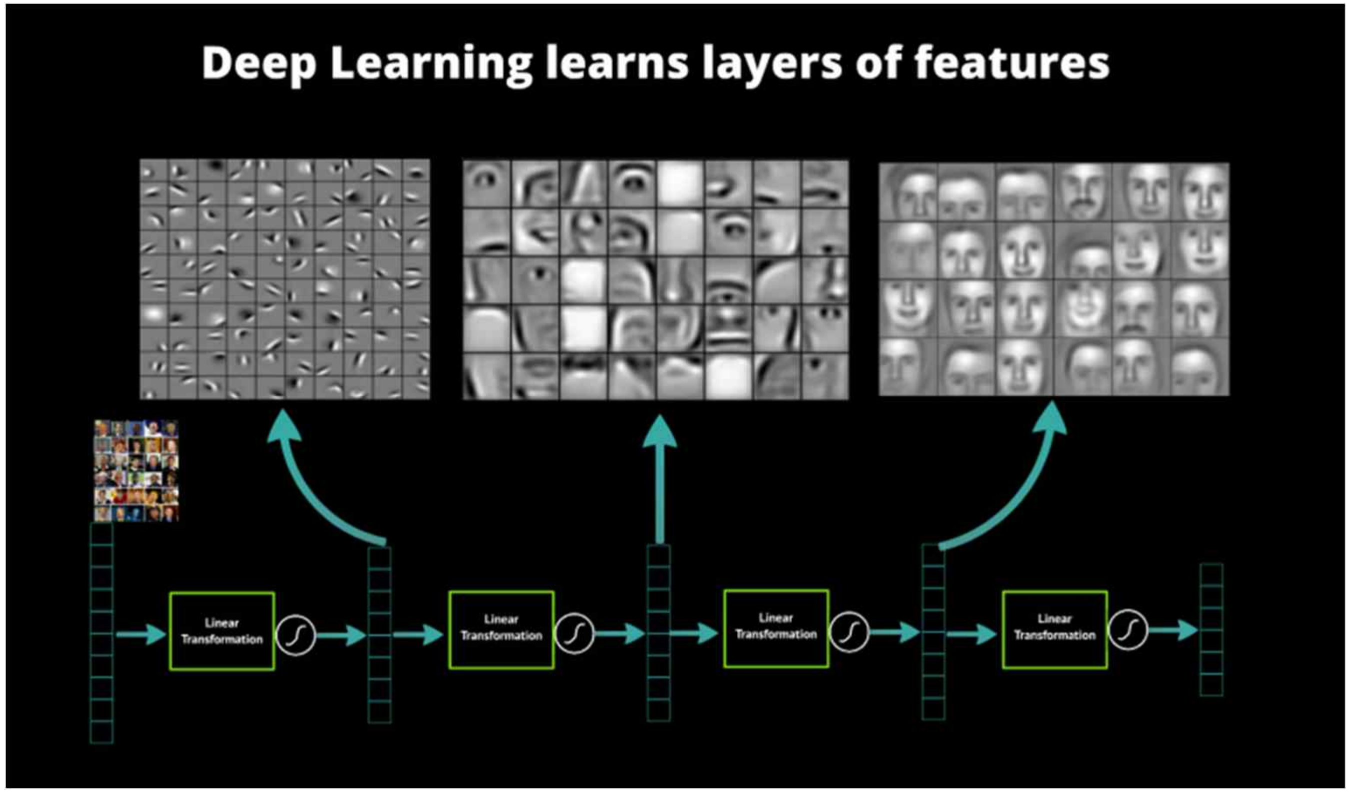
$$h(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_k(\mathbf{x})]$$

Question: can the functions $h_j = \sigma(\mathbf{v}_j \cdot \phi(\mathbf{x}))$ supply good features for a linear predictor?

Universal approximation:

- A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
- In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error

Deep Learning



Roadmap

Features

Multi-Layer Perceptron (MLP)

Backpropagation

Cross entropy loss

Motivation: loss minimization

Optimization problem:

$$\mathbf{V}^*, \mathbf{w}^* = \underset{\mathbf{V},\mathbf{w}}{arg\,\min}\, \mathcal{J}(\mathbf{V}, \mathbf{w})$$

$$\mathcal{J}(\mathbf{V}, \mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \left(f_{\mathbf{V},\mathbf{w}}(\mathbf{x}) - y\right)^2$$

$$f_{\mathbf{V},\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{d} w_i \sigma(\mathbf{v}_i \cdot \phi(\mathbf{x}))$$

Goal: compute gradient

$$\nabla_{\mathbf{V},\mathbf{w}} \mathcal{J}(\mathbf{V}, \mathbf{w})$$

=> Doable but tedious

Approach

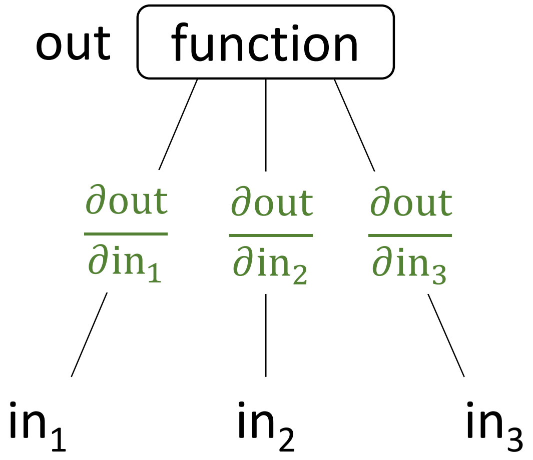
Mathematically: just grind through the chain rule

Next: visualize the computation using a computation graph

Advantage:

- Avoid long equations
- Reveal structure of computations (modularity, efficiency, dependencies)

Function as boxes

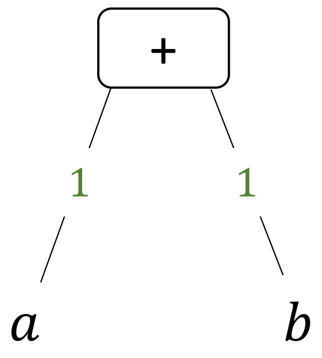


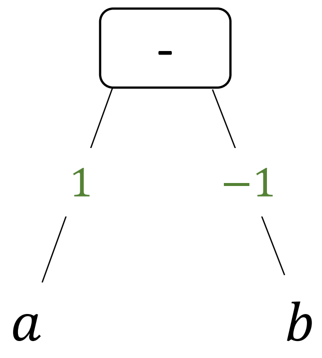
out
function
$\frac{\partial \text{out}}{\partial \text{in}_1}$
$\frac{\partial \text{out}}{\partial \text{in}_2}$
$\frac{\partial \text{out}}{\partial \text{in}_3}$
in_1
in_2
in_3

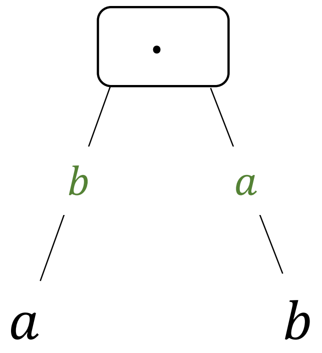
Partial derivatives (gradients): how much does the output change if an input changes?

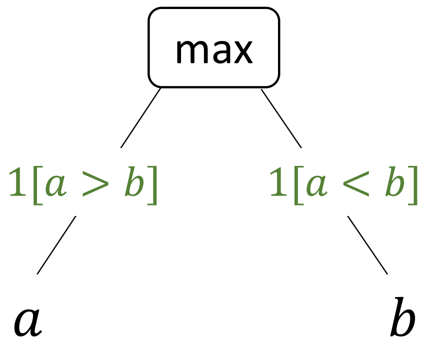
Example: $\text{out} = 2\text{in}_1 + \text{in}_2\text{in}_3 \Longrightarrow 2\text{in}_1 + (\text{in}_2 + \epsilon)\text{in}_3 = \text{out} + \text{in}_3\epsilon$

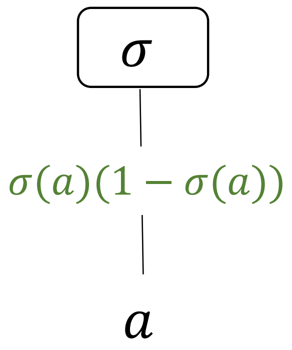
Basic building blocks



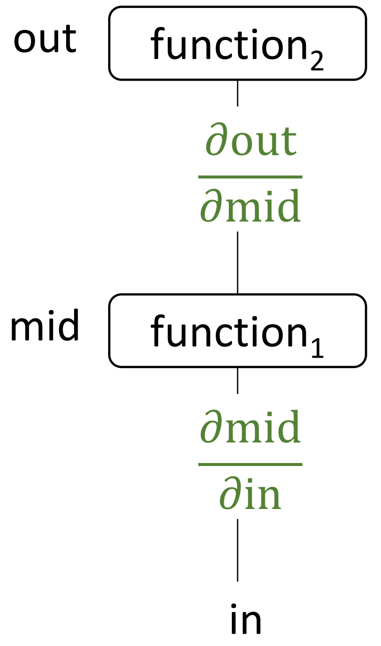








Composing functions

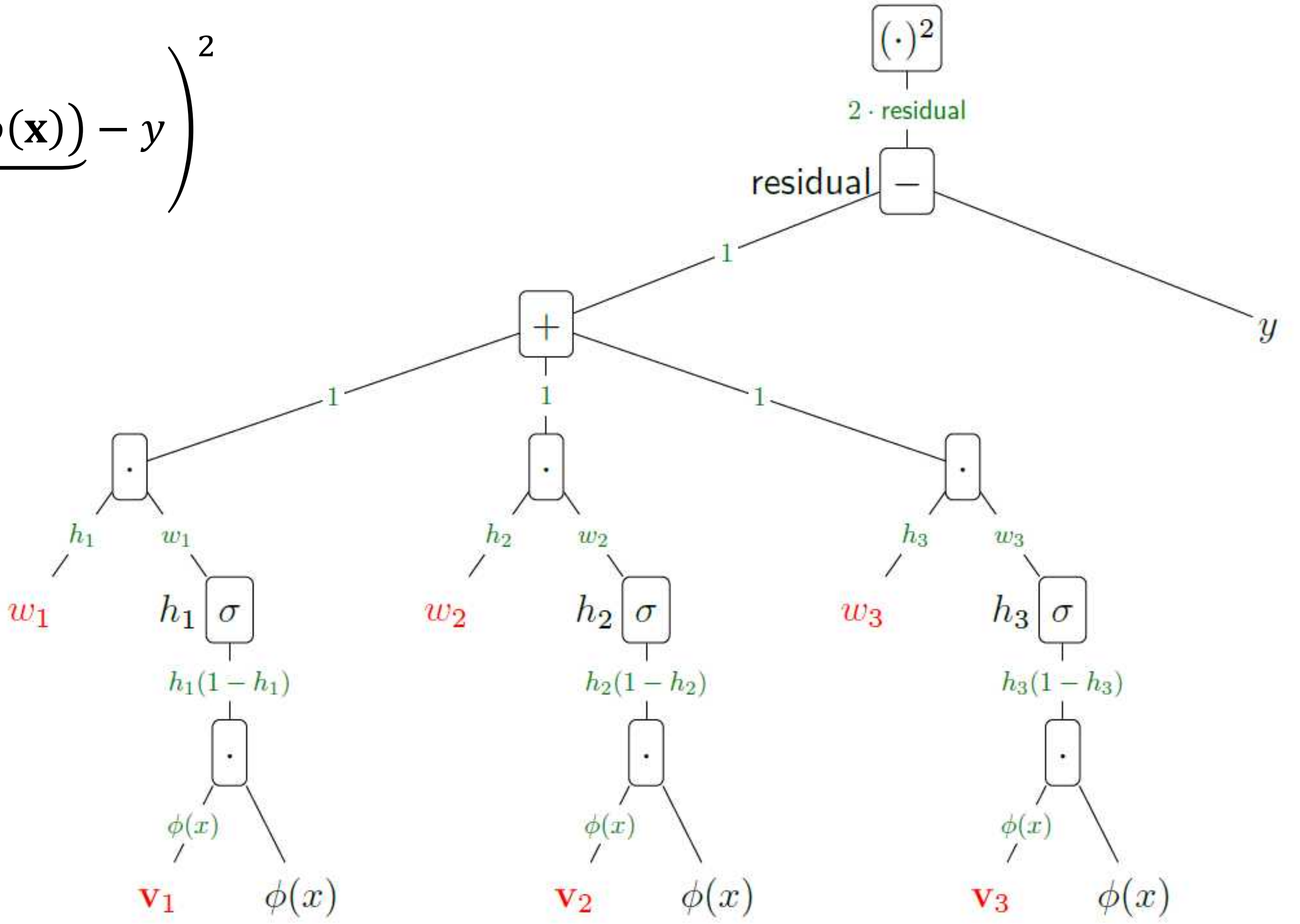


out
function_2
$\frac{\partial \text{out}}{\partial \text{mid}}$
mid
function_1
$\frac{\partial \text{mid}}{\partial \text{in}}$
in

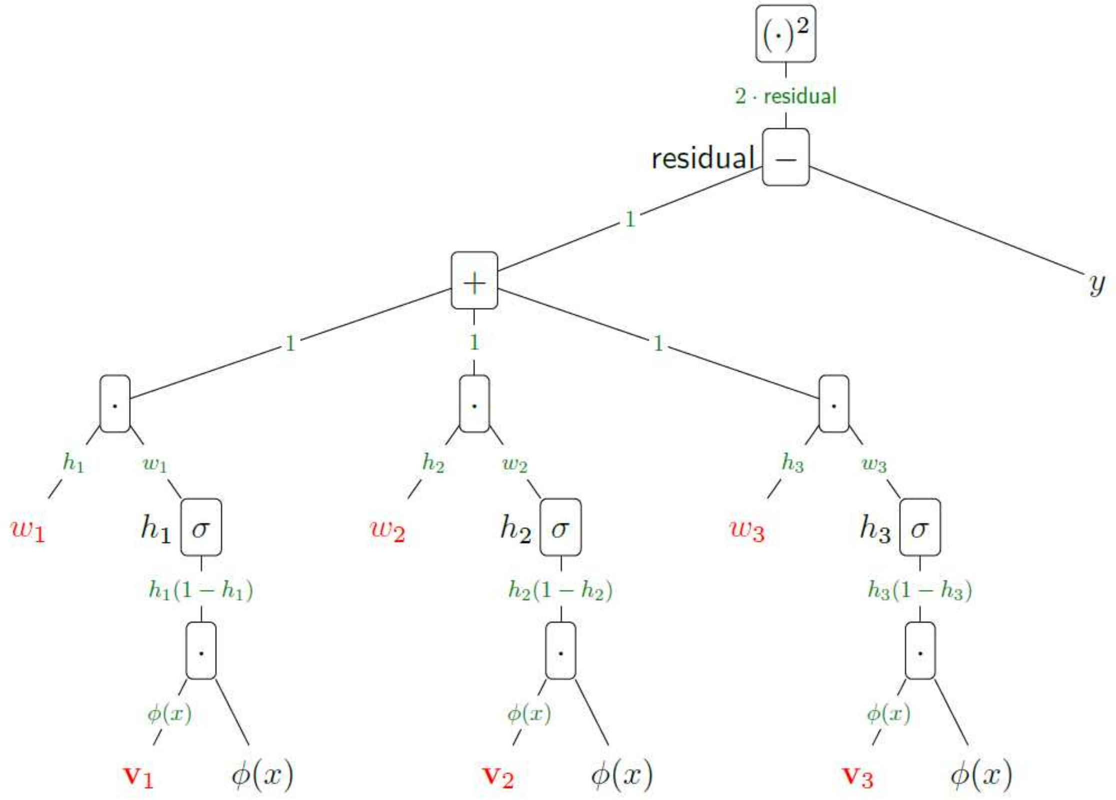
Chain rule: $\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$

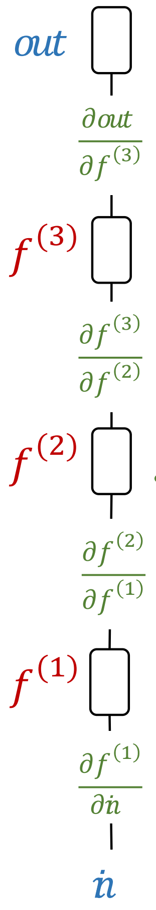
Neural network

$$\text{Loss}(\mathbf{x}, y, \mathbf{V}, \mathbf{w}) = \left(\sum_{i=1}^{d} w_i \underbrace{\sigma\big(\mathbf{v}_i \cdot \phi(\mathbf{x})\big)}_{h_i} - y\right)^2$$



Backpropagation





out
$\frac{\partial out}{\partial f^{(3)}}$
$f^{(3)}$
$\frac{\partial f^{(3)}}{\partial f^{(2)}}$
$f^{(2)}$
$\frac{\partial f^{(2)}}{\partial f^{(1)}}$
$f^{(1)}$
$\frac{\partial f^{(1)}}{\partial in}$
in

$$g^{(3)} = \frac{\partial out}{\partial f^{(3)}}$$

$$g^{(2)} = g^{(3)} \frac{\partial f^{(3)}}{\partial f^{(2)}} = \frac{\partial out}{\partial f^{(2)}}$$

$$g^{(1)} = g^{(2)} \frac{\partial f^{(2)}}{\partial f^{(1)}} = \frac{\partial out}{\partial f^{(1)}}$$

$$g^{(0)} = g^{(1)} \frac{\partial f^{(1)}}{\partial in} = \frac{\partial out}{\partial in}$$

Algorithm: backpropagation

- Forward pass: compute each f (from leaves to root)
- Backward pass: compute each g (from root to leaves)

Forward: $f^{(k)}$ is value for subexpression rooted at k.

$\Uparrow$6 $out = \left(f^{(5)}\right)^2$

$\Uparrow$5 $f^{(5)} = f^{(4)} - y$

$\Uparrow$4 $f^{(4)} = \sum_{i=1}^{d} f_i^{(3)}$

$\Uparrow$3 $f_1^{(3)} = w_1 f_1^{(2)}$

$\Uparrow$2 $f_1^{(2)} = \sigma(f_1^{(1)}) = h_1$

$\Uparrow$1 $f_1^{(1)} = \mathrm{v}_1 \phi(x)$

Backward: $g^{(k)} = \frac{\partial out}{\partial f^{(k)}}$ is how $f^{(k)}$ influences output.

$\Downarrow$1 $g^{(5)} = \frac{\partial out}{\partial f^{(5)}} = 2f^{(5)}$

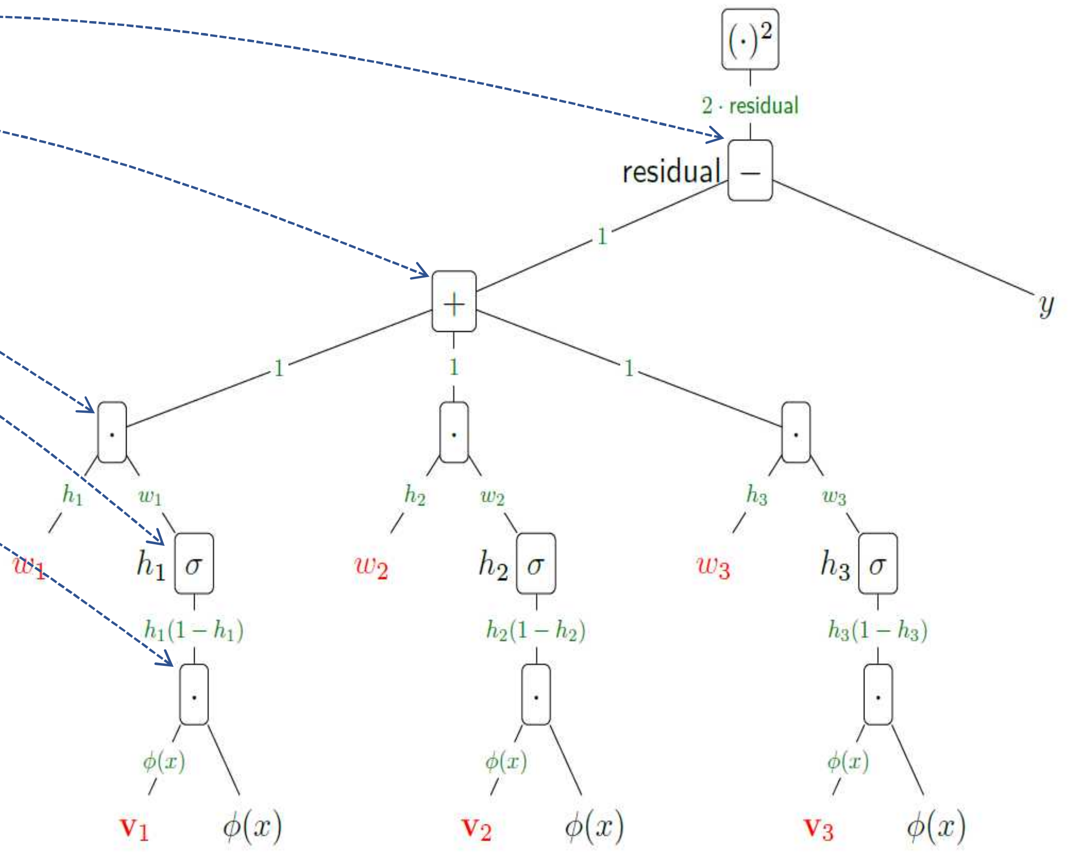
$\Downarrow$2 $g^{(4)} = \frac{\partial out}{\partial f^{(4)}} = \frac{\partial f^{(5)}}{\partial f^{(4)}} \cdot g^{(5)} = 2f^{(5)}$

$\Downarrow$3 $g_1^{(3)} = \frac{\partial out}{\partial f_1^{(3)}} = \frac{\partial f^{(4)}}{\partial f_1^{(3)}} \cdot g^{(4)} = 2f^{(5)}$

$\Downarrow$4 $g_1^{(2)} = \frac{\partial out}{\partial f_1^{(2)}} = \frac{\partial f_1^{(3)}}{\partial f_1^{(2)}} \cdot g_1^{(3)} = w_1 2f^{(5)}$

$\Downarrow$5 $g_1^{(1)} = \frac{\partial out}{\partial f_1^{(1)}} = \frac{\partial f_1^{(2)}}{\partial f_1^{(1)}} \cdot g_1^{(2)} = h_1(1-h_1) w_1 2f^{(5)}$

$\Downarrow$6 $\frac{\partial out}{\partial \mathrm{v}_1} = \frac{\partial f_1^{(1)}}{\partial \mathrm{v}_1} \cdot g_1^{(1)} = \phi(x) h_1(1-h_1) w_1 2f^{(5)}$



Optimization with backpropagation

$$\text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = \left(\sum_{i=1}^{d} w_i \underbrace{\sigma(\mathbf{v}_i \cdot \phi(x))}_{h_i} - y \right)^2 = (\mathbf{w} \cdot \mathbf{h} - y)^2$$

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2 \cdot (f(x) - y) \cdot \mathbf{h}$$

$$\nabla_{\mathbf{v}_1} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2 \cdot (f(x) - y) \cdot w_1 h_1 (1 - h_1) \cdot \phi(x)$$

$$\nabla_{\mathbf{v}_2} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2 \cdot (f(x) - y) \cdot w_2 h_2 (1 - h_2) \cdot \phi(x)$$

…

Stochastic Gradient Descent (SGD):

- Initialize $\mathbf{w}$ randomly.
- Repeat for each $\mathcal{D}$ (**epoch**):
  - Iterate for each batch $\boldsymbol{\mathcal{B}}$ ($\subset \mathcal{D}$) (**iteration**):

$$\mathbf{V}, \mathbf{w}_{(k+1)} = \mathbf{V}, \mathbf{w}_{(k)} - \alpha \nabla_{\mathbf{V},\mathbf{w}} \text{BatchLoss}(x, y, \mathbf{V}, \mathbf{w})$$

- $\nabla_{\mathbf{V},\mathbf{w}} \text{BatchLoss}(x, y, \mathbf{V}, \mathbf{w})$: gradient of Loss on batch $\boldsymbol{\mathcal{B}}$.

Roadmap

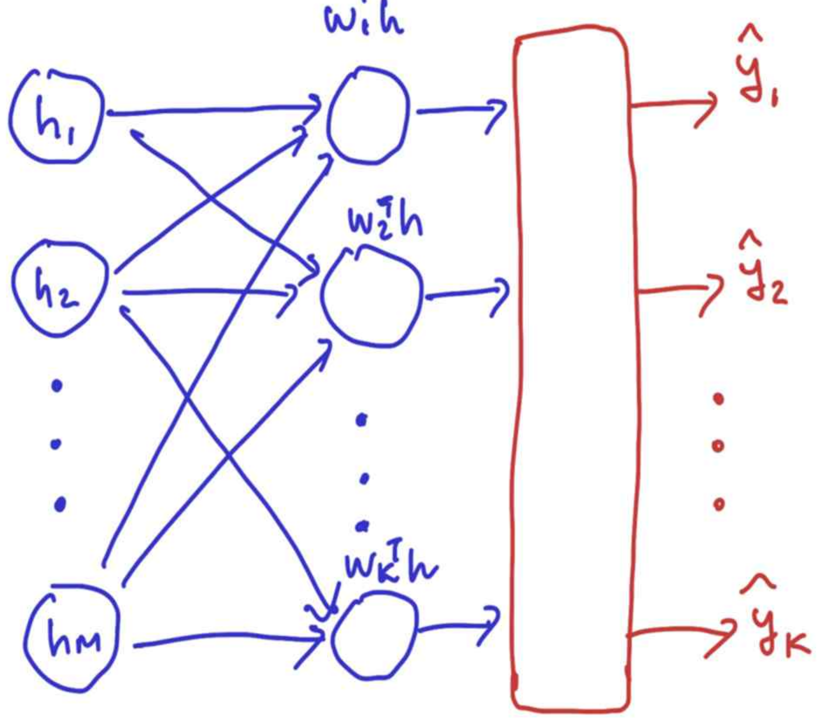
Features

Multi-Layer Perceptron (MLP)

Backpropagation

Cross entropy loss

Softmax layer for multiclass classification



$$\hat{y}_k = \frac{\exp(\mathbf{w}_k^{\mathrm{T}}\mathbf{h})}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^{\mathrm{T}}\mathbf{h})}$$

Softmax layer for multiclass classification

- Model input-output by a softmax transformation of logits $\theta_k = \mathbf{w}_k^{\mathrm{T}}\mathbf{x}_n$:

$$p(y_n = k|\mathbf{x}_n) = \mathrm{softmax}\left(\theta_k = \mathbf{w}_k^{\mathrm{T}}\mathbf{x}_n\right) = \frac{\exp\left(\mathbf{w}_k^{\mathrm{T}}\mathbf{x}_n\right)}{\sum_{j=1}^{K}\exp\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x}_n\right)}$$

- Given $\mathbf{Y} \in \mathbb{R}^{K\times N}$ (each column $\mathbf{y}_n \in \mathbb{R}^K$ follows the 1-of-K coding) and $\mathbf{X} \in \mathbb{R}^{D\times N}$, the likelihood is given by

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(y_n = k|\mathbf{x}_n)^{Y_{k,n}} \ .$$

- The log-likelihood is given by

$$\mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{K} Y_{k,n}\log[p(y_n = k|\mathbf{x}_n)]\,.$$

Cross entropy loss

- For binary classification where $p \in \{y, 1-y\}, q \in \{\hat{y}, 1-\hat{y}\}$, **cross entropy loss** is:

$$\mathcal{J} = \sum_{n=1}^{N} [-y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)].$$

- For multiclass classification where $p \in \{y_1, \dots y_K\}, q \in \{\hat{y}_1, \dots, \hat{y}_K\}$, **cross entropy loss** is:

$$\mathcal{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} \left[-y_{k,n} \log \hat{y}_{k,n}\right].$$

- Note, the cross entropy loss $\mathcal{J}$ is equal to the negative log-likelihood $-\mathcal{L}(\mathbf{w})$.

Cross entropy loss

- $y = \begin{bmatrix} \text{tiger} \\ \text{lion} \\ \text{cat} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \hat{y} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix},\ \mathcal{J} = -\log 0.7 = 0.36$
- $y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \hat{y} = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix},\ \mathcal{J} = -\log 0.5 = 0.69$

Multi-label learning

- $y = \begin{bmatrix} \text{dog} \\ \text{cat} \\ \text{sky} \\ \text{sand} \\ \text{lake} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

Labels: dog, sand, sky

- The error function is given by

$$\mathcal{J} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \underbrace{\ell(y_{k,n}, \hat{y}_{k,n})}_{\text{logistic loss}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \left[-y_{k,n} \log \hat{y}_{k,n} - (1 - y_{k,n}) \log(1 - \hat{y}_{k,n}) \right] ,$$

- where $\hat{y}_{k,n}$ is the output of the kth logistic regression model,

$$\hat{y}_{k,n} = \sigma\left(\mathbf{w}_k^{\mathrm{T}} \mathbf{x}_n\right)$$

Multi-label learning

- For single-label learning, $y = \begin{bmatrix} \text{tiger} \\ \text{lion} \\ \text{cat} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \hat{y} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}, \mathcal{E} = -\log 0.7 = 0.36$
- For multi-label learning, $y = \begin{bmatrix} \text{tiger} \\ \text{lion} \\ \text{cat} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \hat{y} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}, \mathcal{E} = -\log 0.7 - \log 0.8 - \log 0.9 = 0.69$
- When having '?' labels such as $[\mathbf{Y} \in \mathbb{R}^{K \times N}] = \begin{bmatrix} 1 & 0 & ? & \dots \\ 0 & 0 & 1 & \dots \\ 1 & ? & 0 & \dots \\ 1 & 1 & ? & \dots \\ 0 & 1 & ? & \dots \end{bmatrix},$
- k runs only for indices associated with 0 or 1 in $\mathcal{J} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \underbrace{\ell(y_{k,n}, \hat{y}_{k,n})}_{\text{logistic loss}}$.

Deep learning topics after this

- Linear models
- Multi-Layer Perceptrons (MLP), Backpropagation
- What next?

Study CNN, RNN, Transformer (with Pytorch)

(not covered in this course)

- You are ready to start or participate in research.
- After that?

Take courses like machine Learning, deep learning, computer vision, NLP