Learning of Artificial Neural Networks

Sudong Lee

sudonglee@ulsan.ac.kr

https://dais.ulsan.ac.kr/

Goal

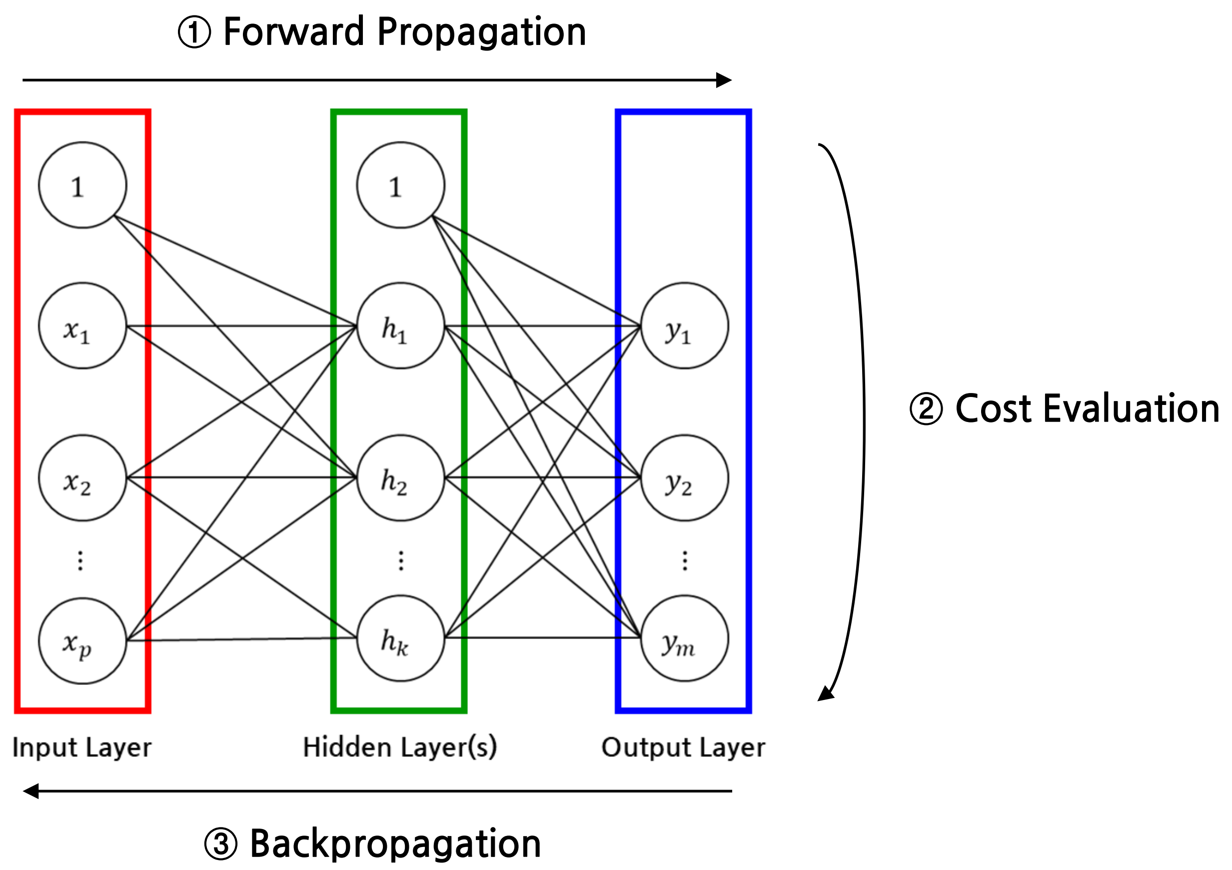
"Understand how to learn an artificial neural network using training data."

Contents

- Forward propagation
- Cost evaluation
- Backpropagation

Operation of Multilayer Perceptron

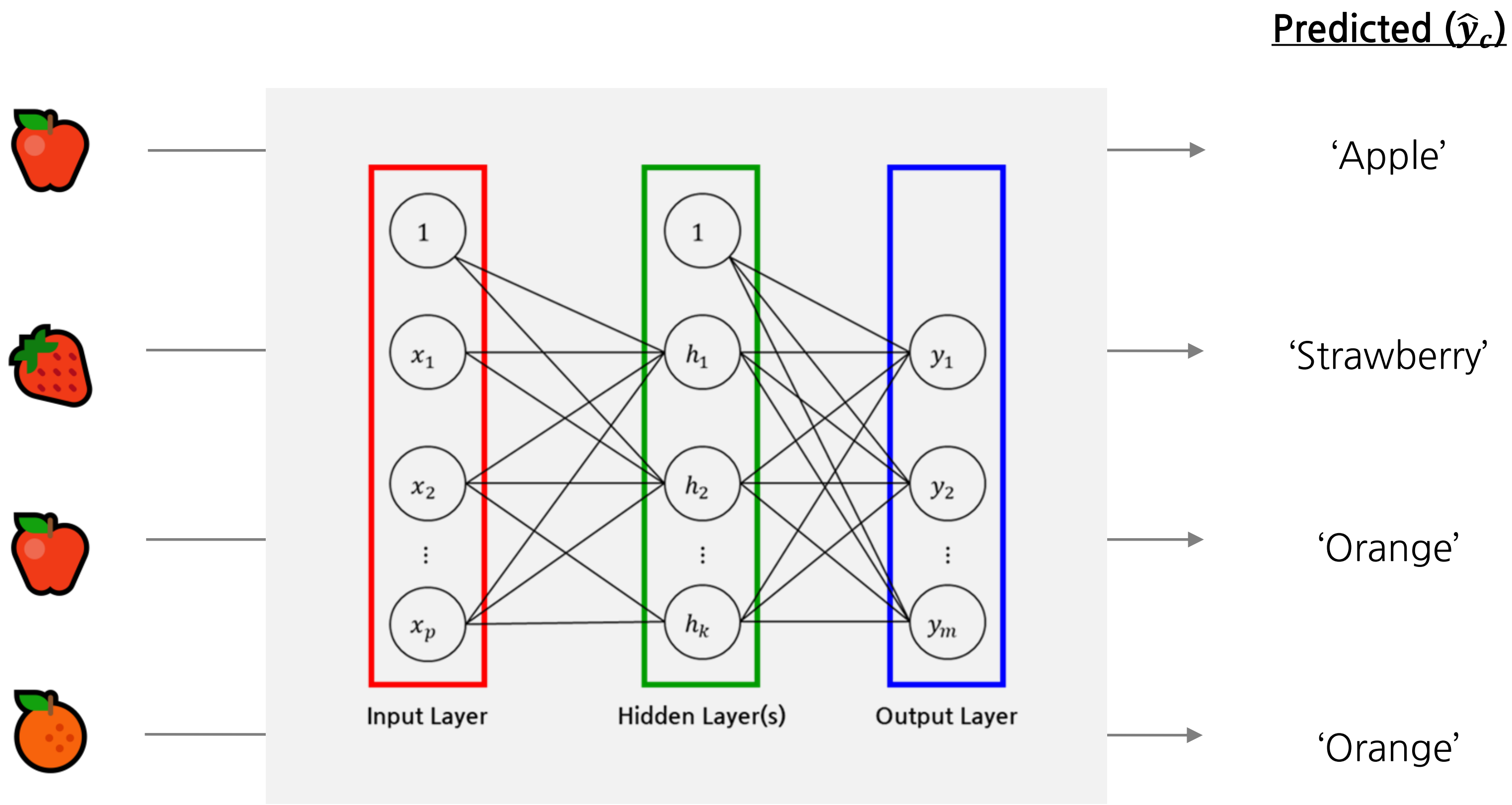
"The MLP operation consists of ① Forward Propagation, ② Cost Evaluation, and ③ Backpropagation."



Forward Propagation

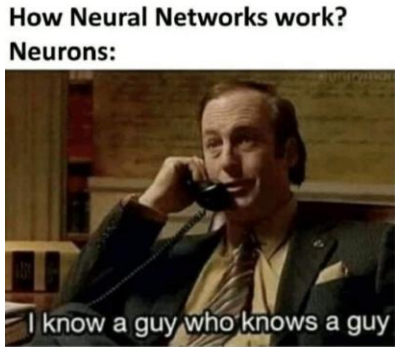
Forward Propagation

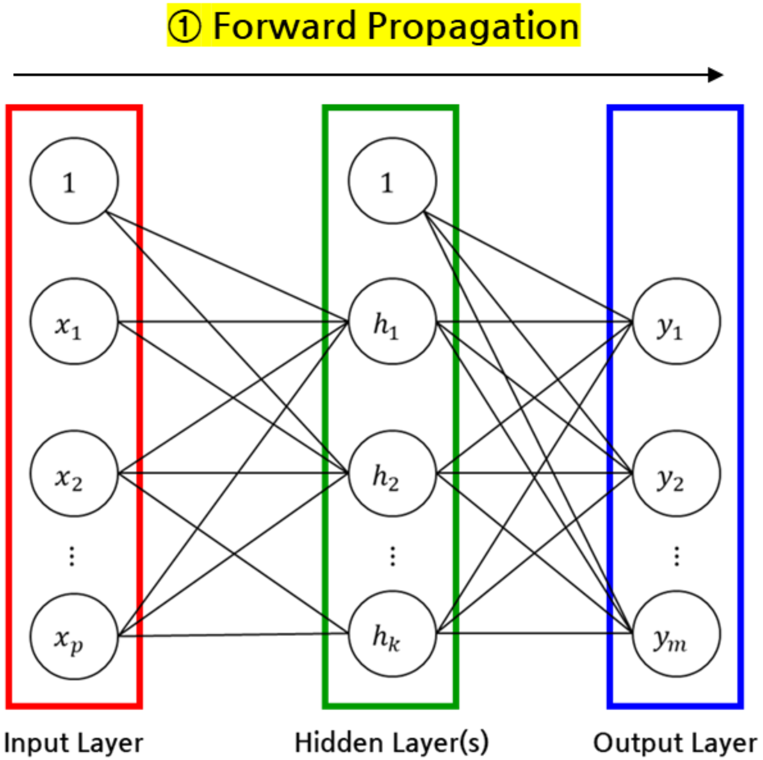
"Input data propagate to the output layer through forward propagation. This is called *inference*."



Feedforward Network

"Input data propagate forward with recursive *aggregation* and *activation*."





- Input layer

$$\boldsymbol{x}^T = [x_1, x_2, \dots, x_p].$$

- Input layer → Hidden layer

$$h_j^{(1)} = f\left(\sum_{i=1}^{n} w_{i,j}^{(1)} x_i + b_j^{(1)}\right) \text{ where } f(\cdot) \text{ is an activation function.}$$

- Hidden layer → Hidden layer

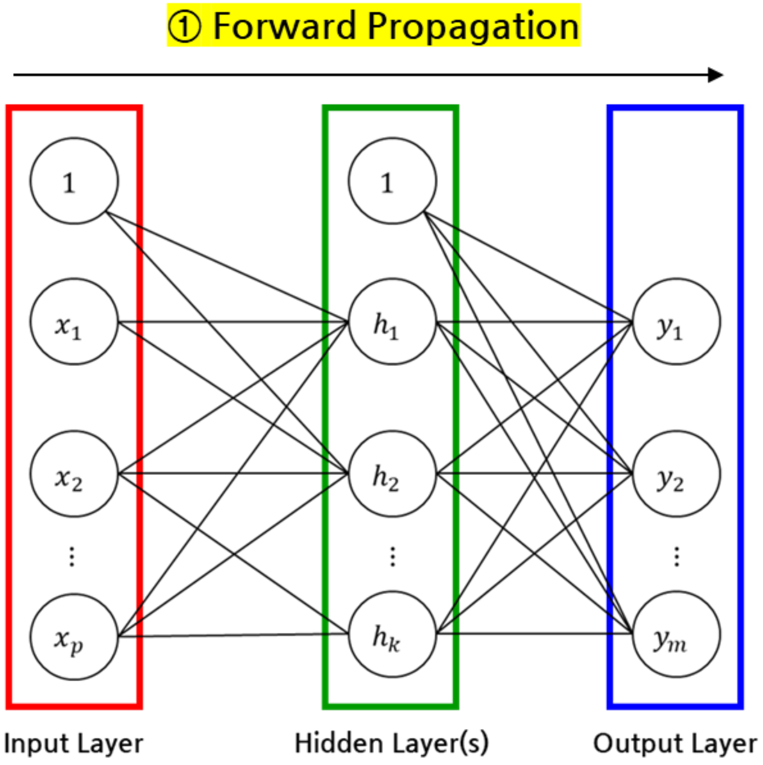
$$h_j^{l+1} = f\left(\sum_{i=1}^{k_l} w_{i,j}^{(l+1)} h_i^{(l)} + b_j^{(l+1)}\right) \text{ for } l = 1, \dots, L.$$

- Hidden layer → Output layer

$$y_j = f\left(\sum_{i=1}^{k_L} w_{i,j}^{(L+1)} h_i^{(L)} + b_j^{(L+1)}\right).$$

Matrix Formulation of Forward Propagation

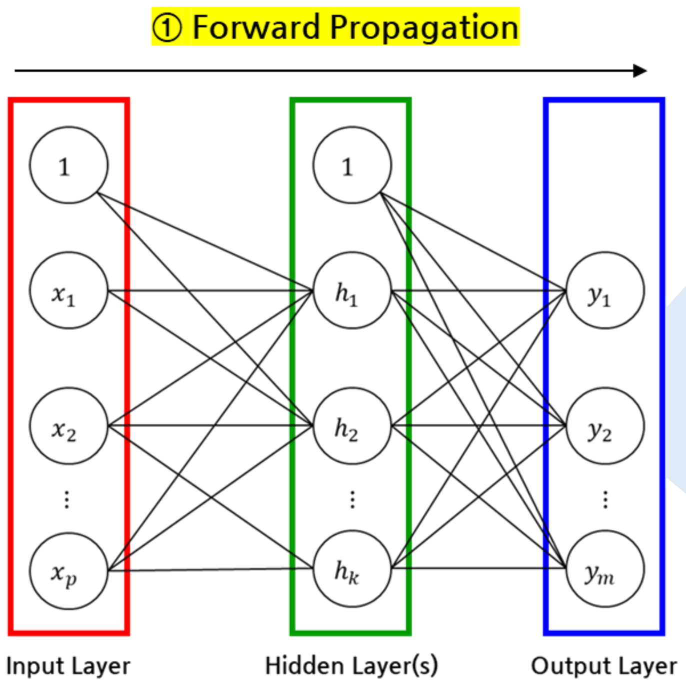
"Matrix formulation simply represents the operations of forward propagation."



- $\mathbf{x} = \left[x_1, x_2, \dots, x_p\right]^T$. $\quad (1 \times p)$
- $\boldsymbol{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$. $\quad (p \times n)$
- $\boldsymbol{X}^T = \left[x_{i,j}\right]$ for $i = 1, \dots, n; j = 1, \dots, p$. $\quad (n \times p)$
- $\boldsymbol{H}^T = \left[h_{i,j}\right]$ for $i = 1, \dots, n; j = 1, \dots, k$. $\quad (n \times k)$
- $\boldsymbol{W}^{(l+1)} = [w_{i,j}^{(l+1)}]$ for $i = 1, \dots, k_l; j = 1, \dots, k_{l+1}$. $\quad (k_l \times k_{l+1})$
- ${\boldsymbol{H}^{(l+1)}}^T = f({\boldsymbol{H}^{(l)}}^T \boldsymbol{W})$. $\quad (n \times k_{l+1})$

Determination of Layers

"The input layer is given, the hidden layers are user-defined, and the output layer depends on the task."



Given

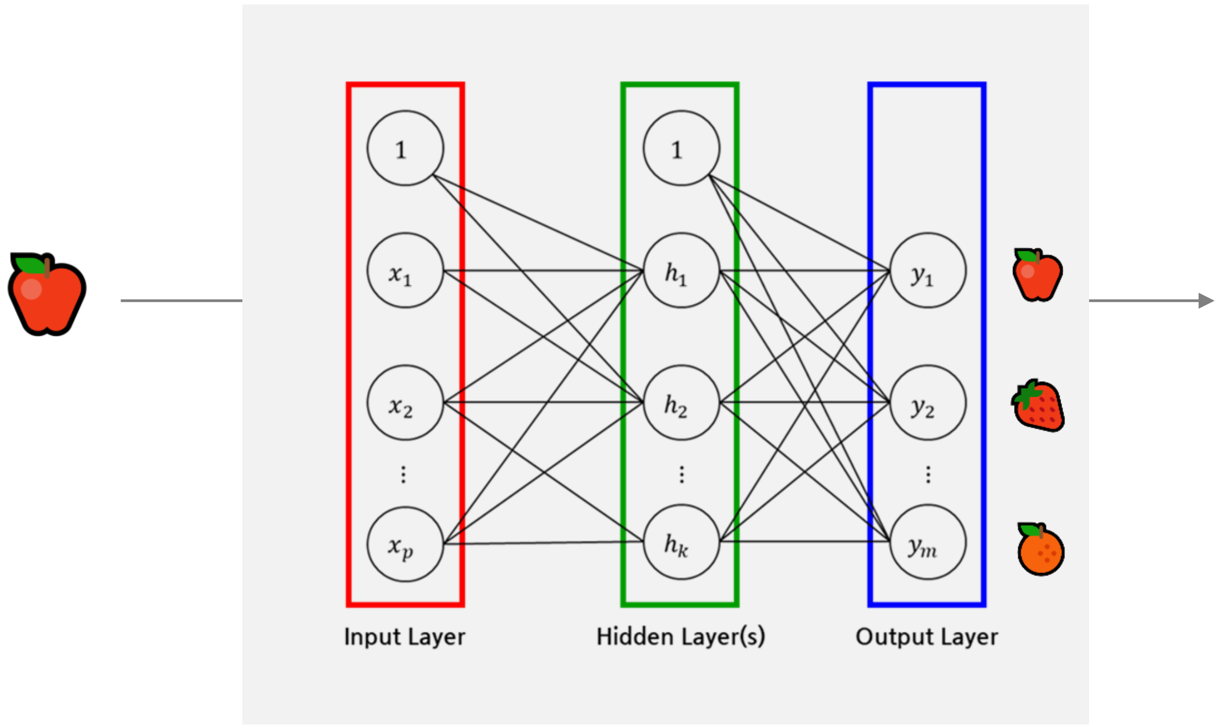
User-defined

(# of layers and neurons, activation function type, ⋯)

Task	# of Output Neurons	Activation Function of Output Neurons
Regression	Equal to # of y's.	Linear
Classification	Equal to # of classes (one if binary)	Softmax

One-hot Encoding and Softmax for Multi-class Classification

"Input data propagate to the output layer through forward propagation. This is called *inference*."



Predicted ($\hat{y}$)	Actual (y)	Actual Class (y_c)
$\begin{bmatrix} 0.72 \\ 0.08 \\ 0.20 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} \text{'Apple'} \\ \text{'Strawberry'} \\ \text{'Orange'} \end{bmatrix}$
Softmax	*One-hot encoding*	

Softmax Activation Function

"Softmax transforms a vector of real numbers into a probability distribution, where each output is between 0 and 1, and the sum of all output values equals 1."

- Definition. *Softmax* function

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{x_j \in \boldsymbol{x}} e^{x_j}}$$

(ex) $\boldsymbol{x} = [x_1, x_2, x_3] = [2.0, 1.0, 0.1]$

$\sigma(x_1) = \frac{e^{2.0}}{e^{2.0}+e^{1.0}+e^{0.1}} \approx 0.66$, $\sigma(x_2) = \frac{e^{1.0}}{e^{2.0}+e^{1.0}+e^{0.1}} \approx 0.24$, $\sigma(x_3) = \frac{e^{0.1}}{e^{2.0}+e^{1.0}+e^{0.1}} \approx 0.10$

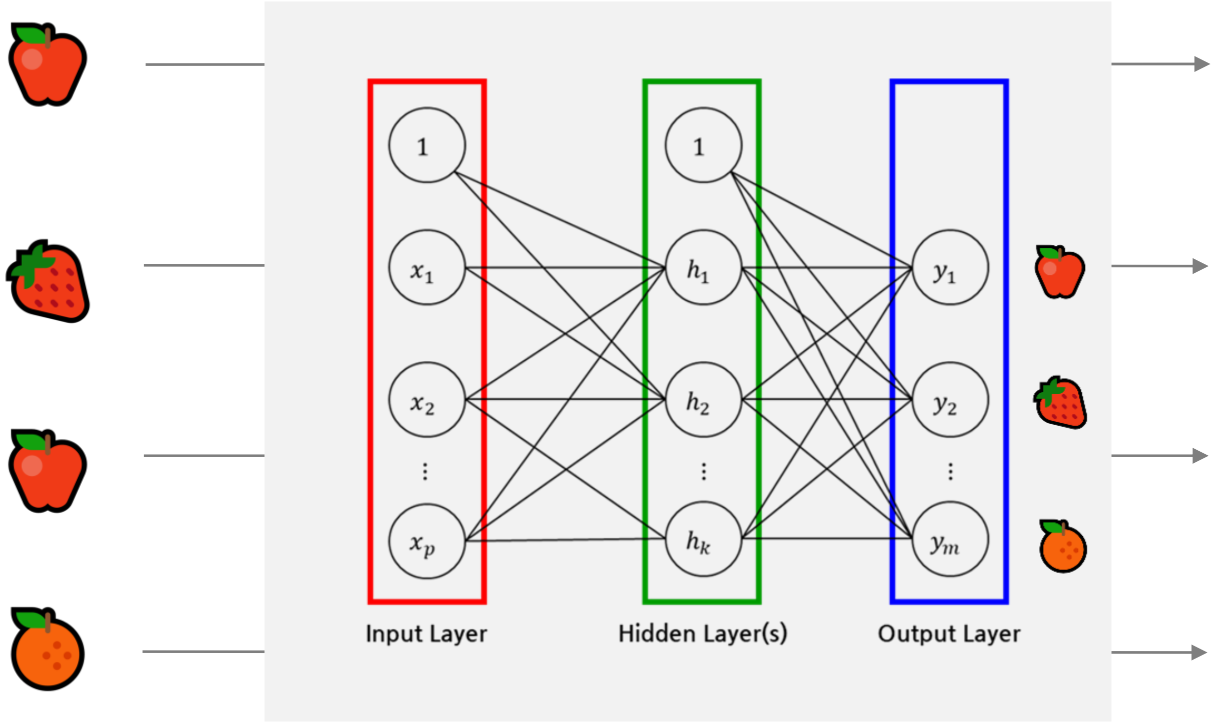
$\sigma(\boldsymbol{x}) = [\sigma(x_1), \sigma(x_2), \sigma(x_3)]$

$y_c = \arg\max \sigma(\boldsymbol{x}) = \arg\max[0.66, 0.24, 0.10] = 1$ ('Apple')

Cost Evaluation

Cost Evaluation

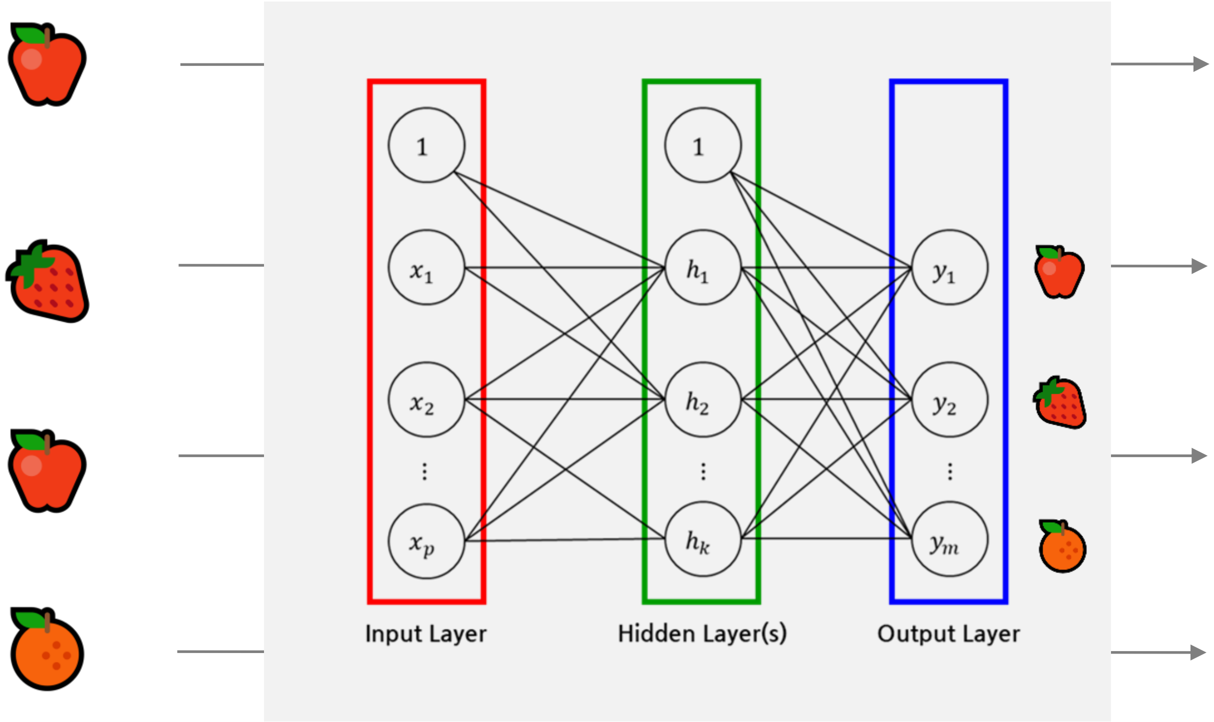
“Errors are computed by comparing the actual and predicted values.”



Predicted ($\hat{y}_c$)		Actual (y_c)
‘Apple’	=	‘Apple’
‘Strawberry’	=	‘Strawberry’
‘Orange’	≠	‘Apple’
‘Orange’	=	‘Orange’

Cost Evaluation

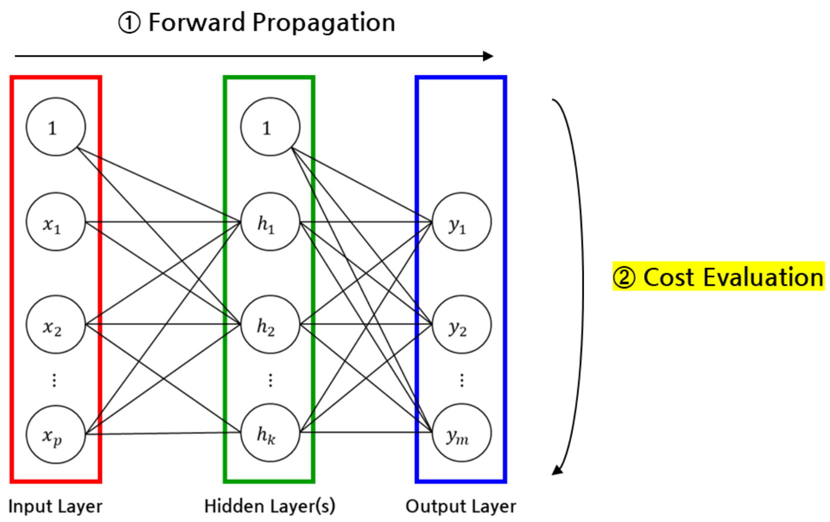
"How can I give the best feedback to the network for further improvement?"



Predicted ($\hat{y}$)	Actual (y)
$[0.82, 0.13, 0.05]$	$[1, 0, 0]$
$[0.21, 0.64, 0.15]$	$[0, 1, 0]$
$[0.35, 0.07, 0.58]$	$[1, 0, 0]$
$[0.05, 0.01, 0.94]$	$[0, 0, 1]$

Cost Evaluation

"The actual and predicted target values are compared by a pre-defined *loss function*."



- 'Loss function' vs 'Cost function' *
  - A loss function, ℓ, computes the distance between the actual y and $\hat{y}$.
  - A cost function, $\mathcal{L}$, averages the losses over an entire training data.
- Different types of loss functions
  - Numerical (for regression)
    - Squared error loss: $\ell(y, \hat{y}) = (y - \hat{y})^2$
    - Mean squared error (MSE): $\mathcal{L}(\boldsymbol{y}, \boldsymbol{\hat{y}}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - Categorical (classification):
    - Binary Cross entropy loss: $\ell(y, \hat{y}) = -[y \log \hat{y} + (1 - y)\log(1 - \hat{y})]$
    - Binary cross entropy error: $\mathcal{L}(\boldsymbol{y}, \boldsymbol{\hat{y}}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$

* 'Loss function' and 'cost function' are often used interchangeably.

Cross Entropy

"Cross entropy measures the difference between the predicted and actual probability distribution for the output."

- Entropy

  - **Definition.** ***Entropy***

    For a random variable X with possible values in the set $\mathcal{X}$ and a probability distribution p,

    the entropy $H(X)$ is defined as:

    'information', 'surprise', or 'uncertainty'

    $$H(X) = E\left[\log\frac{1}{x}\right]$$

    For a discrete random variable X,

    $$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log(x)$$

  - Entropy measures the *unpredictability* or *information content* associated with a random variable's possible values.

Cross-Entropy

"Cross-entropy measures the difference between the predicted and actual probability distribution for the output."

- **Cross-Entropy**
  - **Definition.** *Cross-Entropy*

    The cross-entropy of the distribution q relative to a distribution p over a given set is defined as follows:

    $$H(p, q) = -E_p[\log q]$$

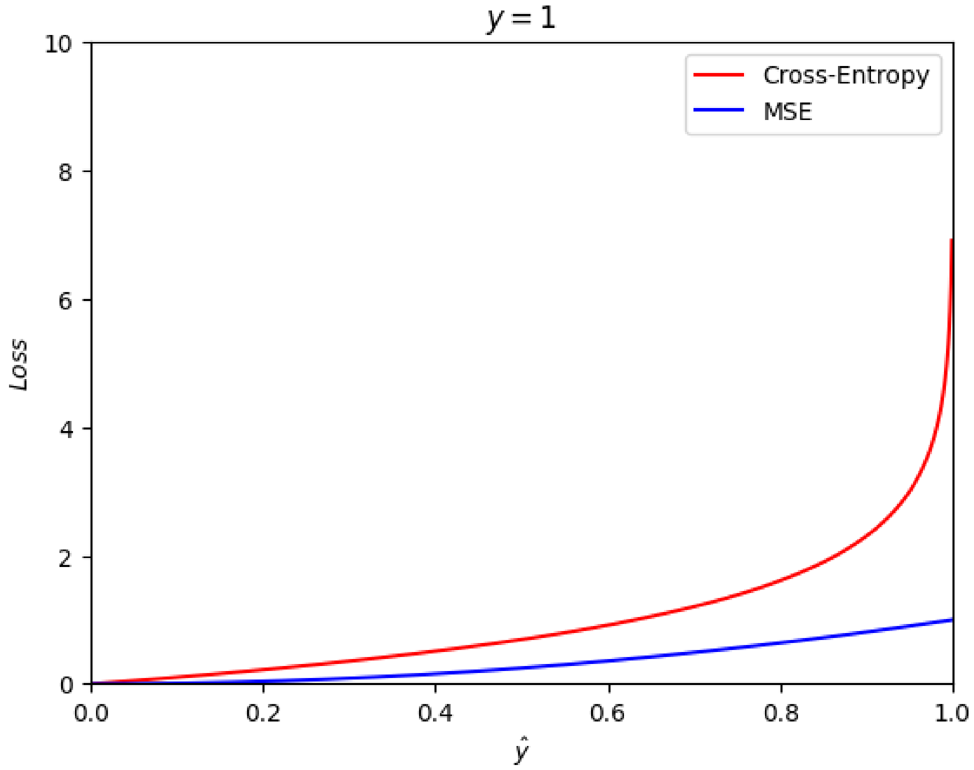
    For a discrete random variable X,

    $$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log\big(q(x)\big)$$

  - Binary cross-entropy loss: $\ell(y, \hat{y}) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$
  - Categorical cross-entropy loss: $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{j=1}^{C} y_j \log \hat{y}_j$ where C is the number of classes.

MSE vs. Cross-Entropy

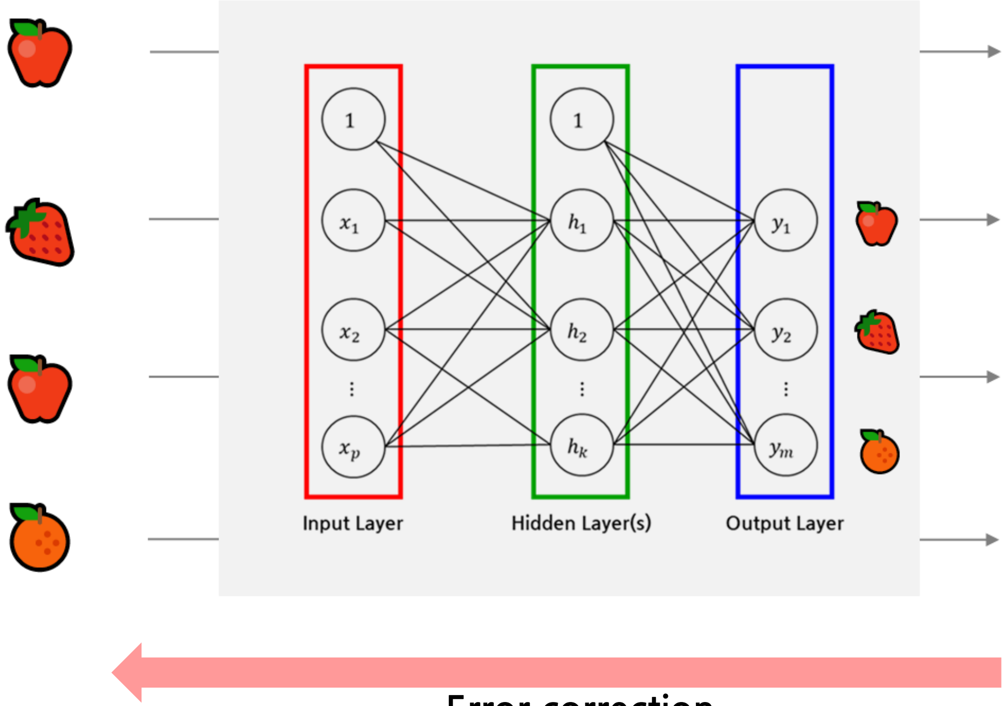
"Cross entropy yields more exponential gradient difference than MSE does."



Backpropagation

Cost Evaluation

"How can I give the best feedback to the network for further improvement?"

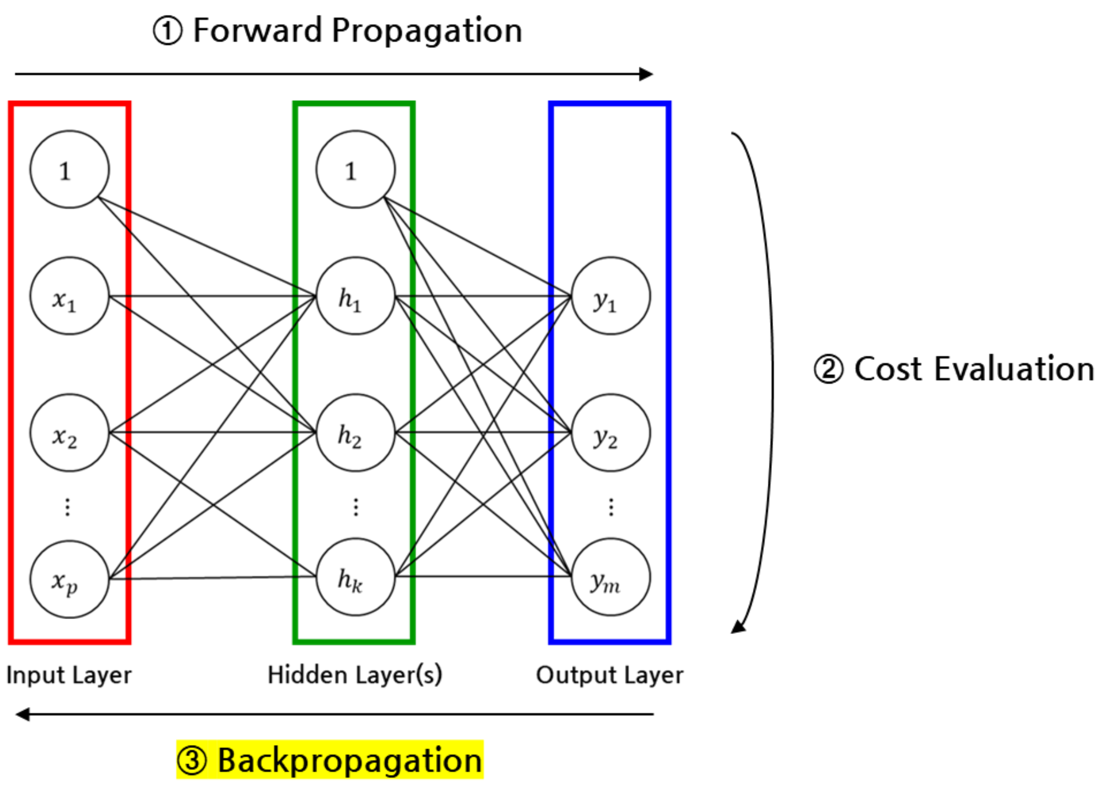


Predicted ($\hat{y}$)	Actual (y)
$[0.82, 0.13, 0.05]$	$[1, 0, 0]$
$[0.21, 0.64, 0.15]$	$[0, 1, 0]$
$[0.35, 0.07, 0.58]$	$[1, 0, 0]$
$[0.05, 0.01, 0.94]$	$[0, 0, 1]$

Error correction

Backpropagation

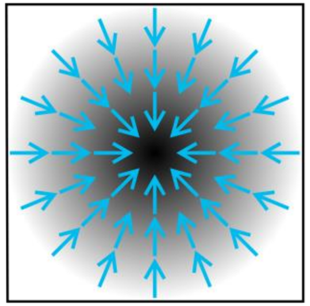
"Computes the *gradient* of the loss function w.r.t. the parameters and updates them to minimize the cost."

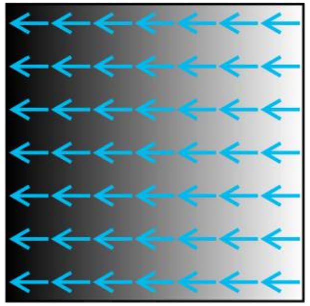


Error Correction Using Gradients

"Gradients can provide the direction of error correction at each training iteration."

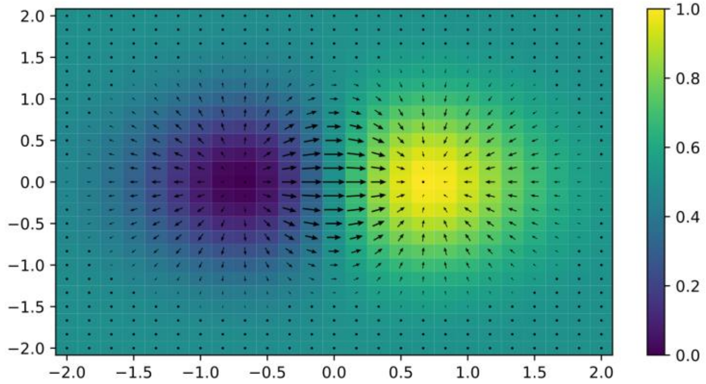
- **Error correction**
  - Error-correction learning, used with supervised learning, compares the predicted output to the desired output value and uses that error to direct the training.
  - In the training of ANNs, the cost function represents the current error.





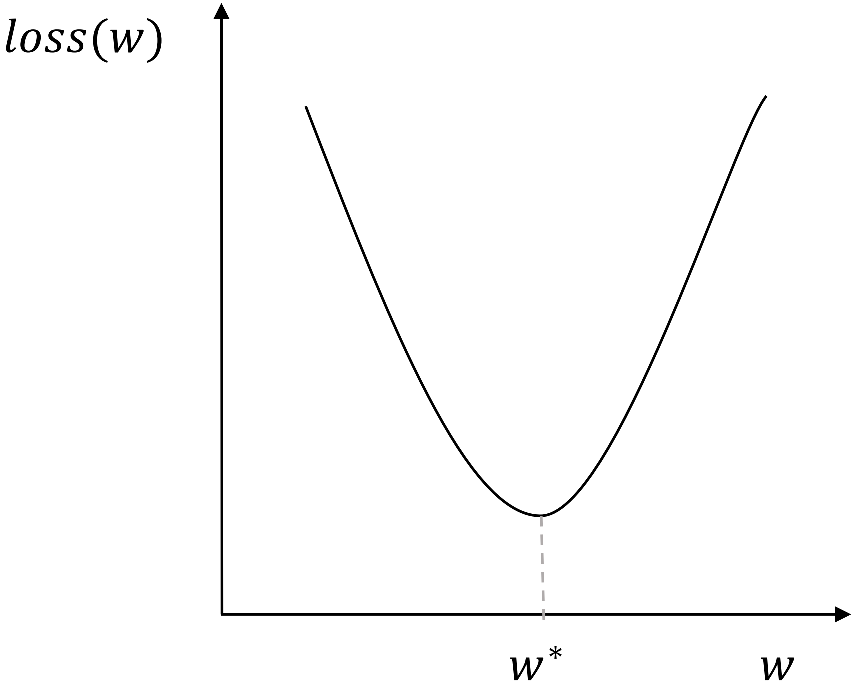
- **Gradient**
  - **Definition.** ***Gradient***

    In vector calculus, the gradient of a scalar-valued differentiable function f of multiple variables is the **vector field** (or vector-valued function) ∇f whose value at a point p gives the direction and rate of the **fastest increase**.



Recall: Gradient Descent Algorithm

"The algorithm finds the values for the model parameters that minimize the cost function."



Gradient Descent Algorithm

1. Calculate the gradient $\nabla_t = \frac{\partial \mathcal{L}}{\partial w}\Big|_{w=w_t}$ where w_t is the value for w at iteration t.

2. Update w_t with ∇_t and a learning rate η:

$$w_{t+1} = w_t - \eta \nabla_{\mathrm{t}}$$

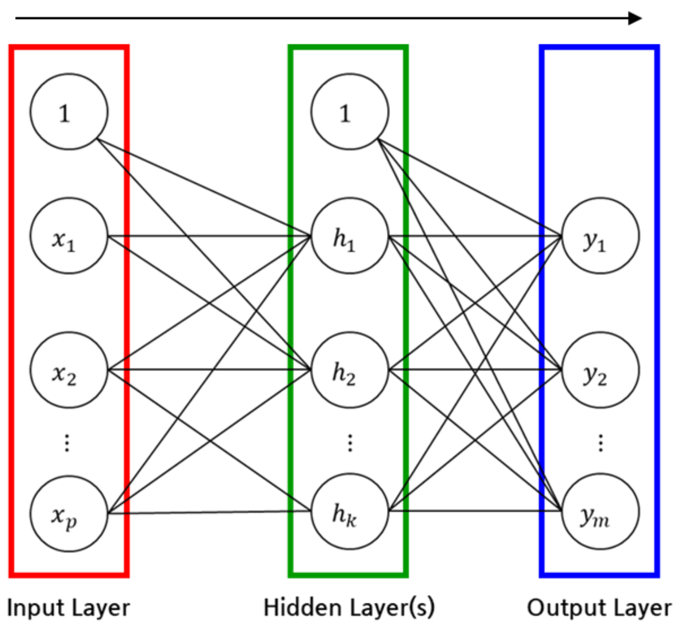
3. Calculate ∇_{t+1}.

   Stop if the stopping criteria are met. (i.e., $\nabla_{t+1} < \delta$)

   Otherwise, repeat step 1.

Derivation of the Gradients Using Chain Rule

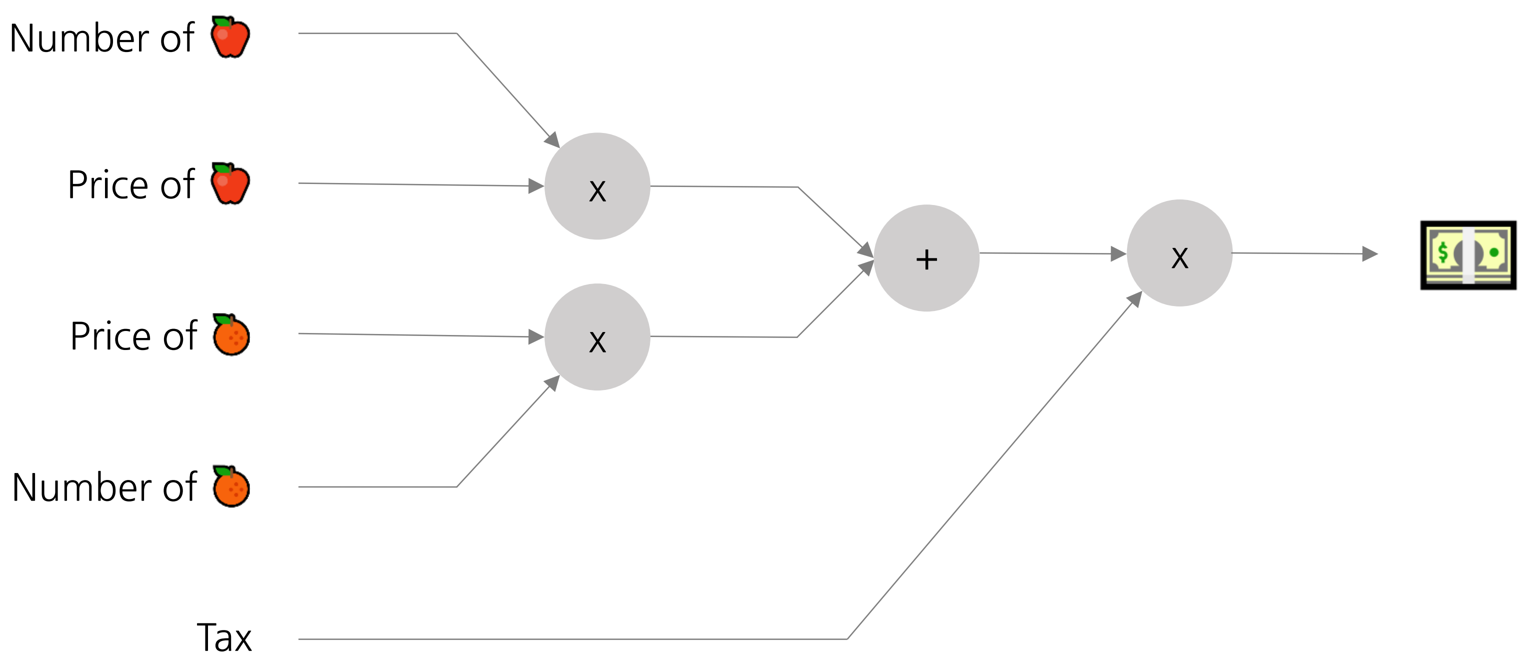
"How can we compute the gradients of loss w.r.t. the parameters, $\boldsymbol{w}$ and $\boldsymbol{b}$?"



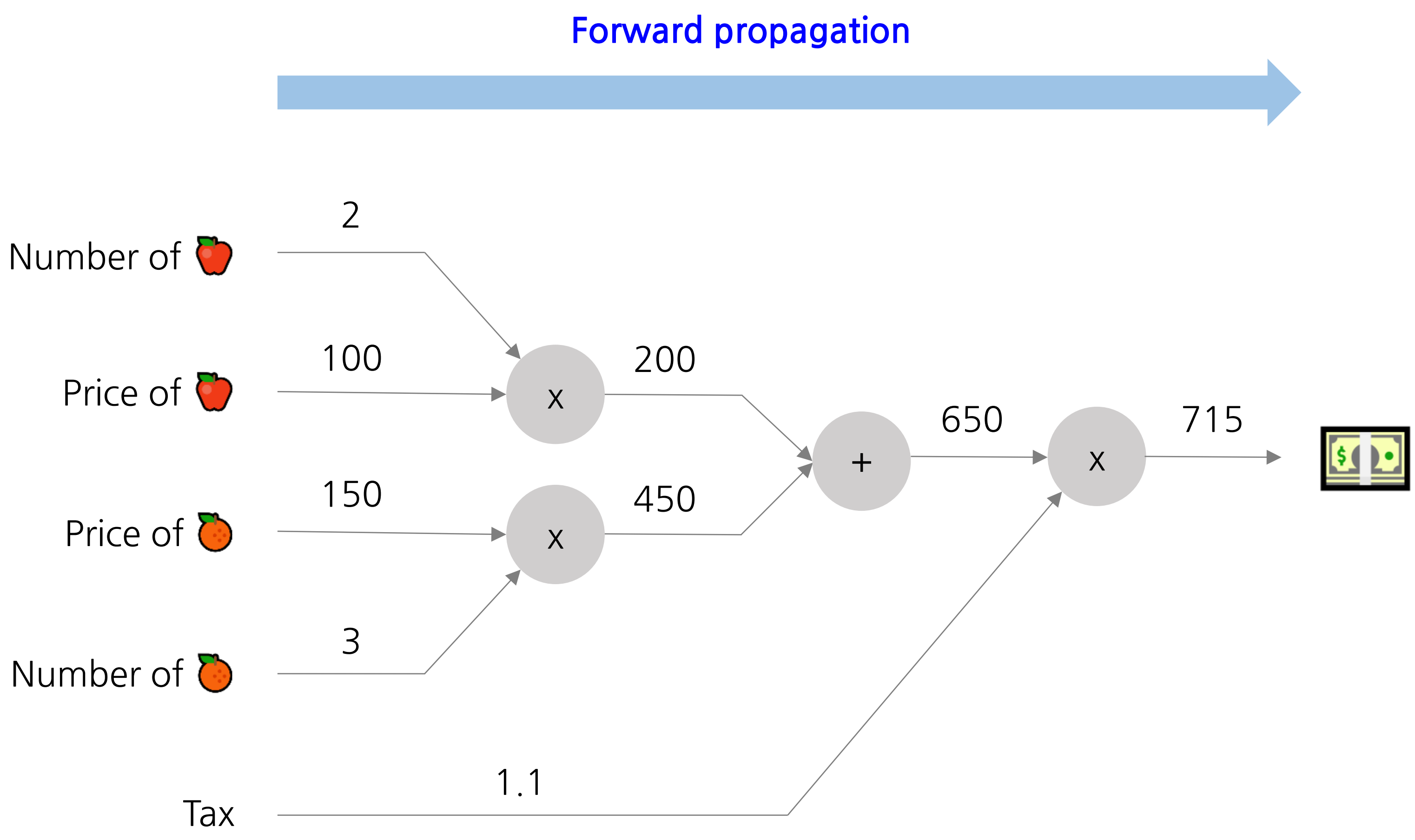
- "Loss as a function of $\boldsymbol{w}$ and $\boldsymbol{b}$."
  - $\ell(y, \hat{y}) = \ell(y, NN(x; \boldsymbol{w}, \boldsymbol{b}))$: must be very complex. 😫
  - But if we think of it simply, $y = f\left(g\left(f\left(g\left(\dots f(g(x; \boldsymbol{w}, \boldsymbol{b}))\right)\right)\right)\right)$. 😁
- Chain rule
  - The chain rule is a formula to compute the derivative of a composite function.
  - $[f(g(x))]' = f'(g(x))g'(x) \Rightarrow \frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$

Understanding of Backpropagation

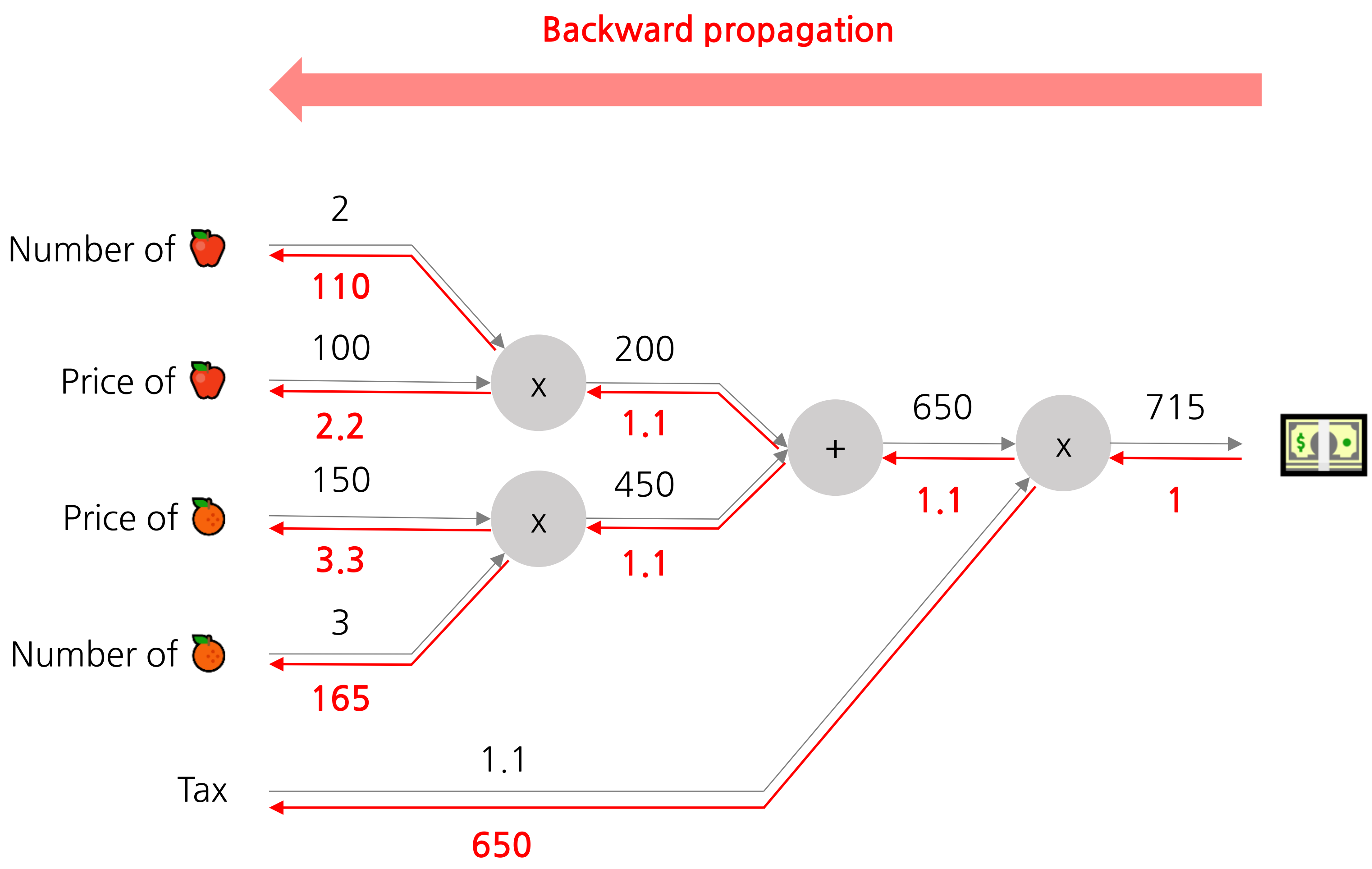
Computational graph: grocery shopping



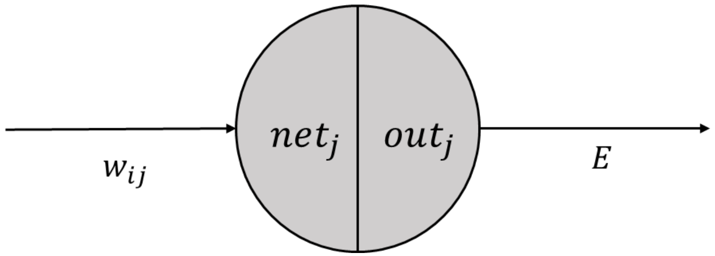
Understanding of Backpropagation



Understanding of Backpropagation

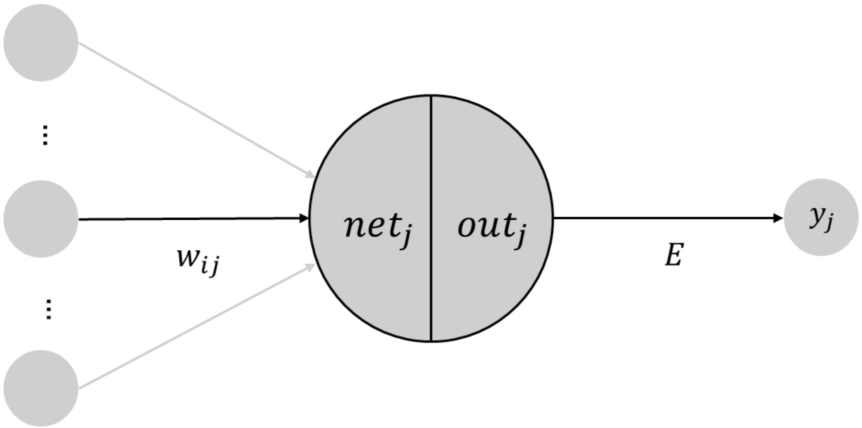


Computational Graph of MLP



- $net_j = \sum_i (w_{ij} out_i + b_j)$
- $out_j = f(net_j)$
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$

Computational Graph of MLP: Output Layer



- $net_j = \sum_i w_{ij} out_i + b_j$
- $out_j = f(net_j)$
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$

① ② ③

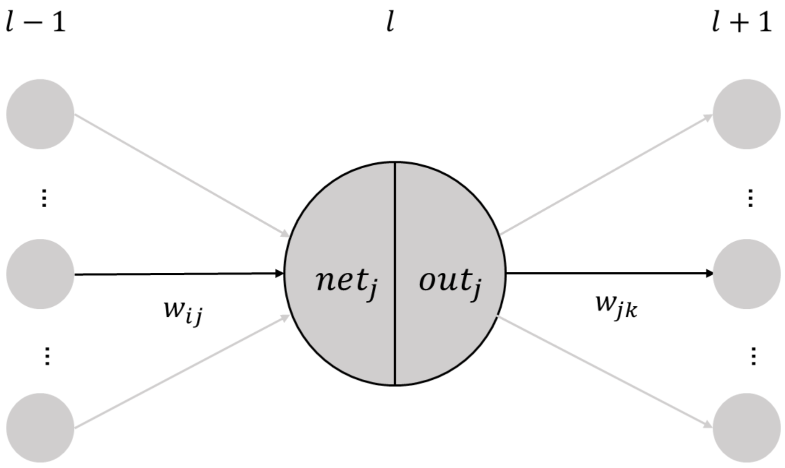
① $\frac{\partial E}{\partial out_j} = \frac{\partial}{\partial out_j} \sum_k \frac{1}{2} (target_k - out_k)^2 = out_j - target_j$

② $\frac{\partial out_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = f'(net_j)$

③ $\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_k w_{kj}\, out_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} out_i = out_i$

$\Rightarrow \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \left(out_j - target_j \right) \times f'\left(net_j \right) \times out_i$

Computational Graph of MLP: Hidden Layer



- $net_j = \sum_i w_{ij} out_i + b_j$
- $out_j = f(net_j)$
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$

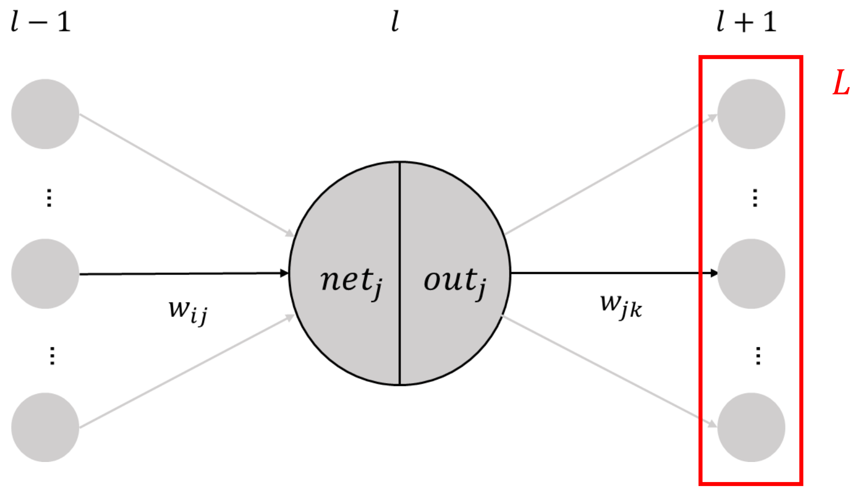
① ② ③

① $\frac{\partial E}{\partial out_j} = ?$

② $\frac{\partial out_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = f'(net_j)$

③ $\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_k w_{kj}\, out_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} out_i = out_i$

Computational Graph of MLP: Hidden Layer



- $net_j = \sum_i w_{ij} out_i + b_j$
- $out_j = f(net_j)$
- $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$

  ① ② ③

① $\frac{\partial E}{\partial out_j} = \sum_{k \in L} \left(\frac{\partial E}{\partial out_k} \frac{\partial out_k}{\partial net_k} \frac{\partial net_k}{\partial out_j} \right) = \sum_{k \in L} \left(\frac{\partial E}{\partial out_k} \frac{\partial out_k}{\partial net_k} w_{jk} \right)$

$$\delta_k = \frac{\partial E}{\partial out_k} \frac{\partial out_k}{\partial net_k}$$

② $\frac{\partial out_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = f'(net_j)$

③ $\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_k w_{kj} \, out_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} out_i = out_i$

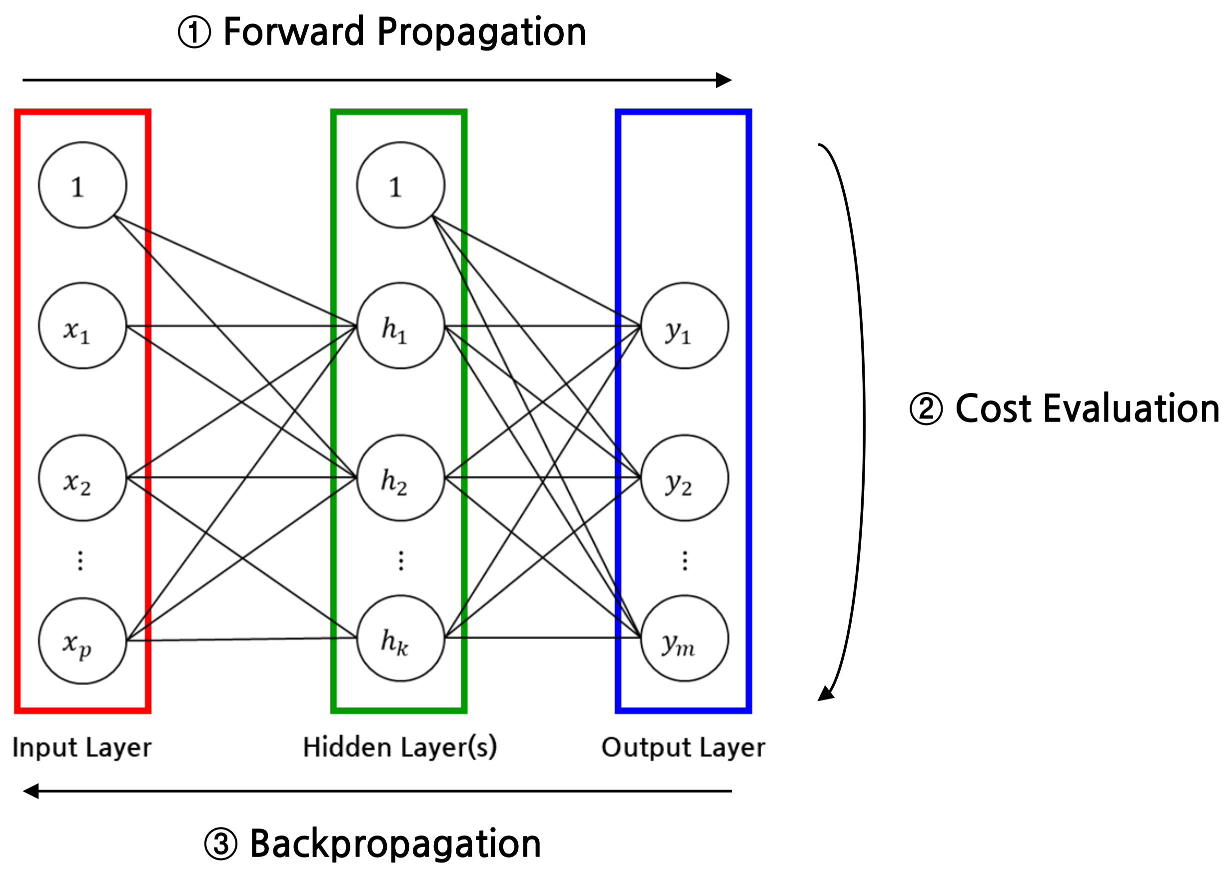
$\Rightarrow \frac{\partial E}{\partial w_{ij}} = \delta_j out_j$

where $\delta_j = \begin{cases} (out_j - t_j) f'(net_j), & \text{for an output unit } j \\ \sum_k w_{jk} \, \delta_k, & \text{for a hidden unit } j \end{cases}$

Takeaways

Operation of Multilayer Perceptron

"The MLP operation consists of ① Forward Propagation, ② Cost Evaluation, and ③ Backpropagation."



Thank you! 😀