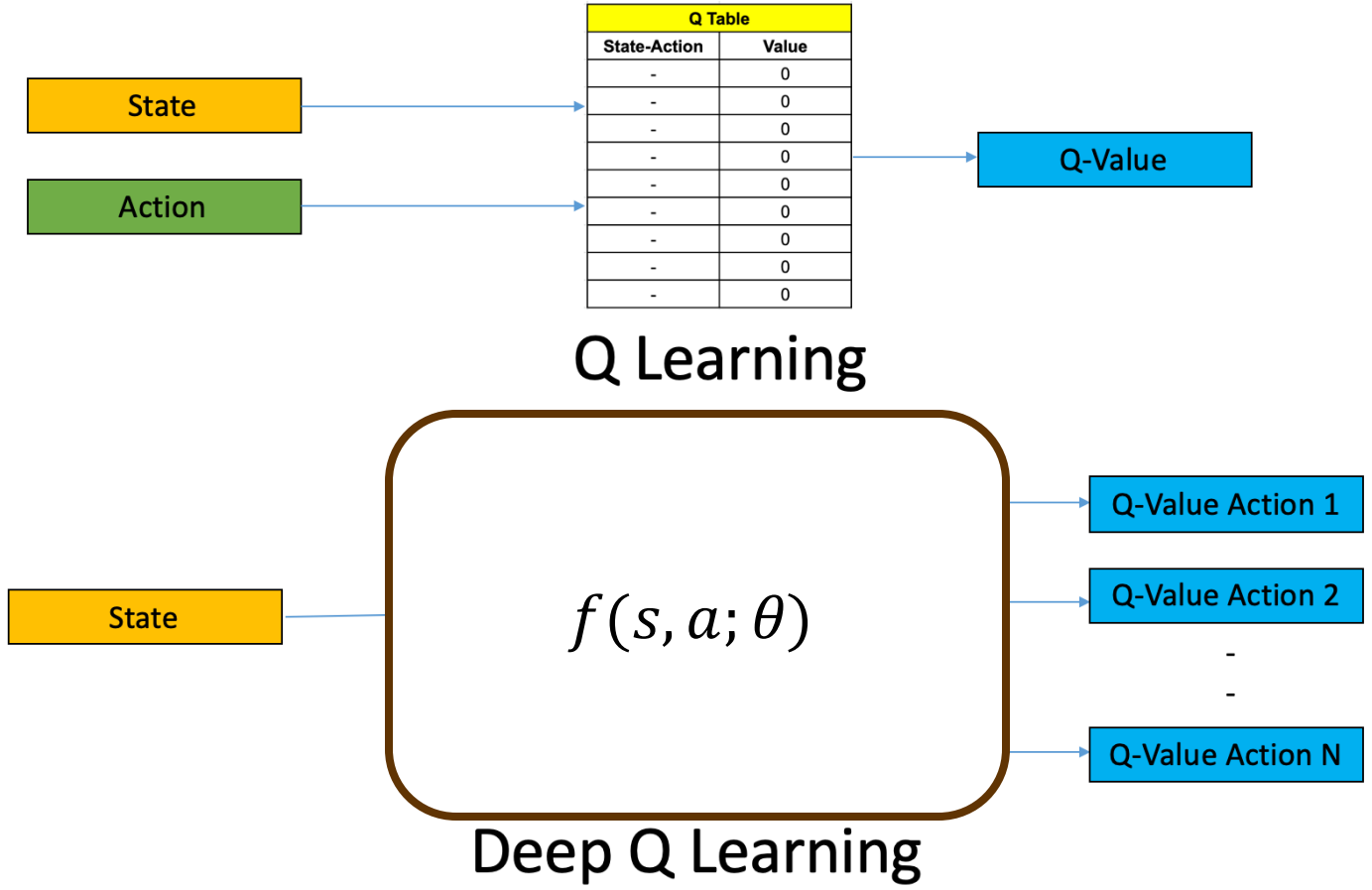
지능 시스템
Intelligent Systems

Lecture 6 – Learning of Neural Networks

From Last Lecture



Revision – Perceptron

The circles containing data is called "**node**".

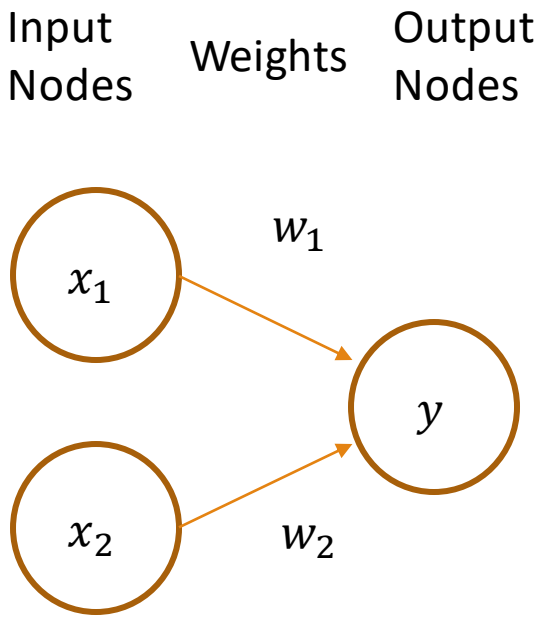
The Output node, containing y, may be expressed in 2 forms

1: Linear function of the input and weights:

$$y = w_1 x_1 + w_2 x_2$$

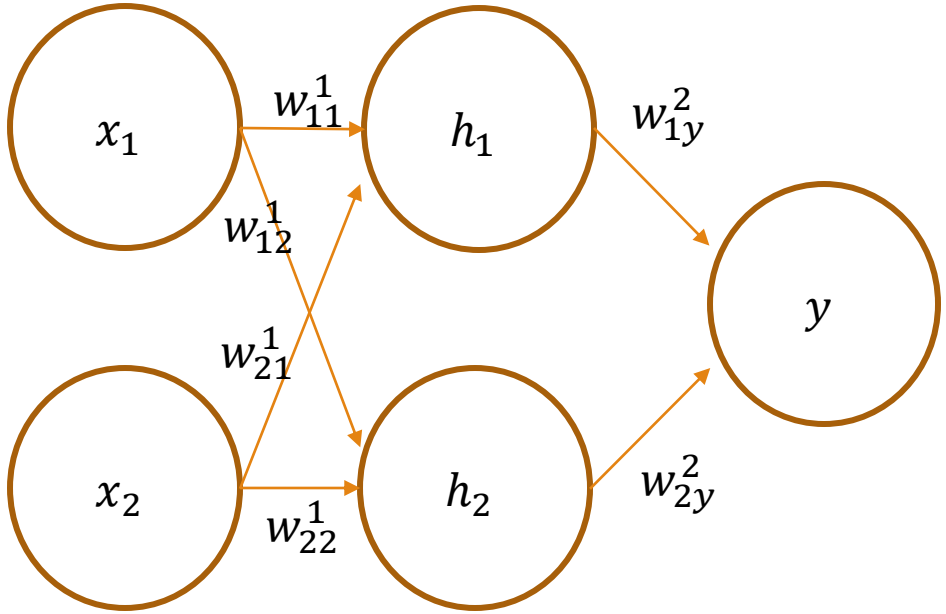
2: Binary Classification via a threshold value, θ, via activation function:

$$y = \begin{cases} 0, & w_1 x_1 + w_2 x_2 \leq \theta \\ 1, & w_1 x_1 + w_2 x_2 > \theta \end{cases}$$



Revision – Multi-layer Perceptron

Multi-layer Perceptron enables non-linear classifications due to a stacking of non-linear activation functions.
However, we inevitably have more weight parameters to determine.
We call this structure a **Fully-Connected Neural Network**
Notice that the notations have changed.



$$h_1 = \begin{cases} 0, & w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2 \leq 0 \\ 1, & w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2 > 0 \end{cases}$$

$$h_2 = \begin{cases} 0, & w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2 \leq 0 \\ 1, & w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2 > 0 \end{cases}$$

$$y = \begin{cases} 0, & w_{by}^2 + w_{1y}^2 h_1 + w_{2y}^2 h_2 \leq 0 \\ 1, & w_{by}^2 + w_{1y}^2 h_1 + w_{2y}^2 h_2 > 0 \end{cases}$$

Revision – Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$h_1 = a\left(w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2\right)$$
$$h_2 = a\left(w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2\right)$$

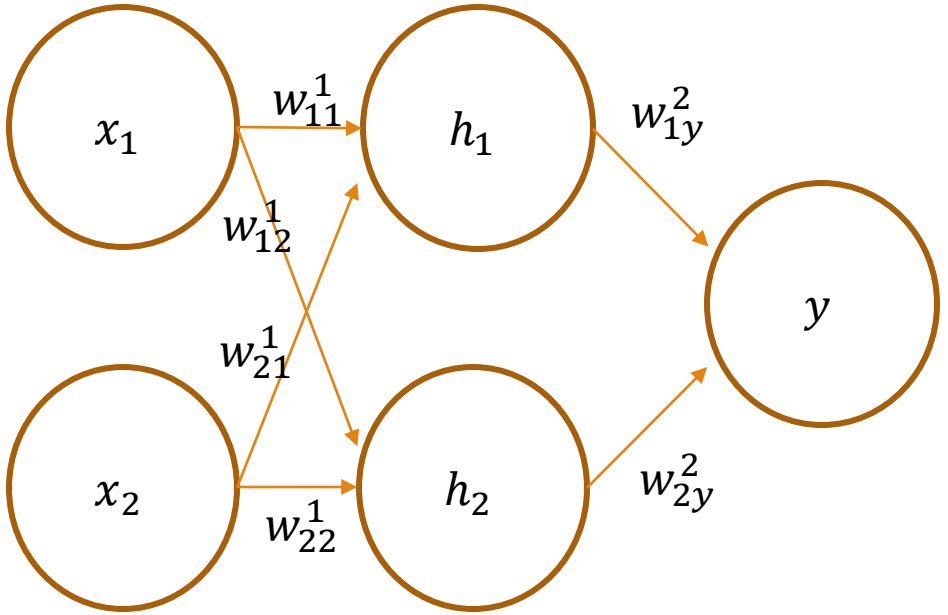
And we can combine these expressions,

$$\boldsymbol{h} = a(\boldsymbol{w}^{(1)}\boldsymbol{x})$$

Where

$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Where (1) denotes that the weights are between input and the first hidden layer. (Not power)



We denote the activation function as $a()$

Revision – Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$\boldsymbol{h} = a(\boldsymbol{w}^{(1)}\boldsymbol{x})$$

Where

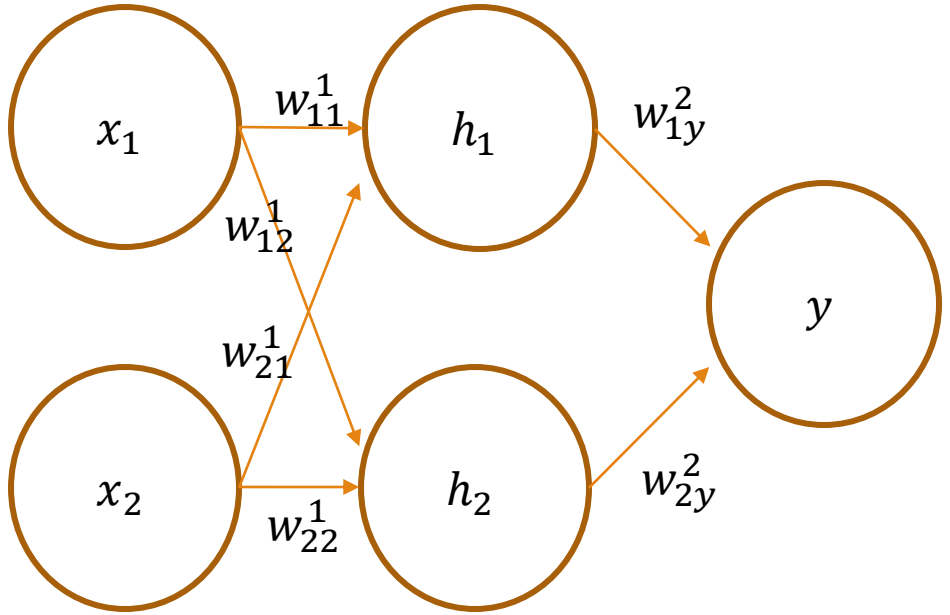
$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Then,

$$y = a(\boldsymbol{w}^{(2)}\boldsymbol{h})$$

Where

$$\boldsymbol{w}^{(2)} = \begin{pmatrix} w_{by}^2 & w_{1y}^2 & w_{2y}^2 \end{pmatrix}$$



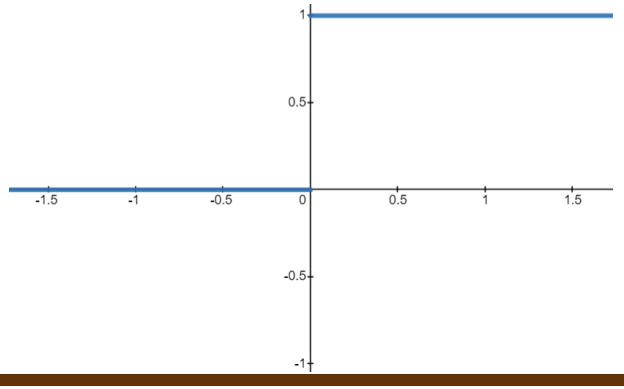
We denote the activation function as $a()$

Revision – Activation Functions

Step Function:

$$a(x) = \begin{cases} 1, & x \geq 0 \\ \beta, & x < 0 \end{cases}$$

Where $\beta = 0$ or $\beta = -1$

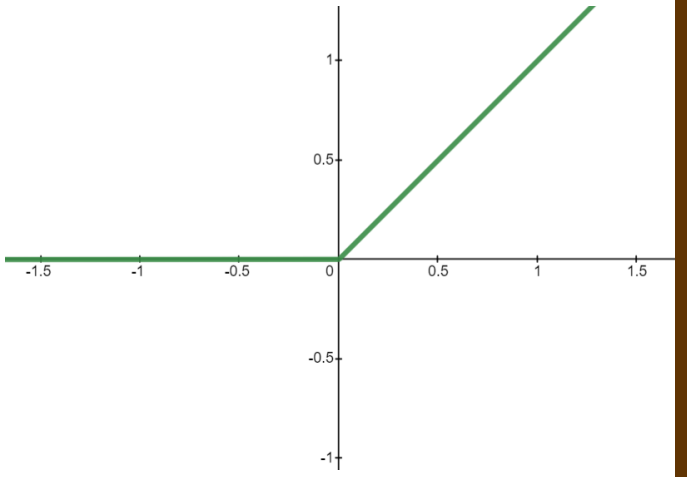


Step function:

- Enables Classification.
- Gradient is always 0 for any input.

ReLU(Rectified Linear Unit):

$$a(x) = \max(0, x)$$



ReLU:

- Enables Classification.
- Gradient is always 0 for negative input.
- Gradient is always 1 for positive input.

Finding optimal weights – Loss function

As discussed so far, the key of deep learning and using neural networks lies on being able to find optimal/near-optimal weight parameters, $\boldsymbol{w}$.

Consider a function expression of neural network:

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w})$$

If wrong weights are used, the outputs will be far from our expectations and correct answers.
Therefore, we need a measure of how good the weights are!
We call this a **Loss function**.

Loss Function: a function that represents differences between network outputs and given answers.

Eg; $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$, where t is the given answer label

Finding optimal weights – Loss function

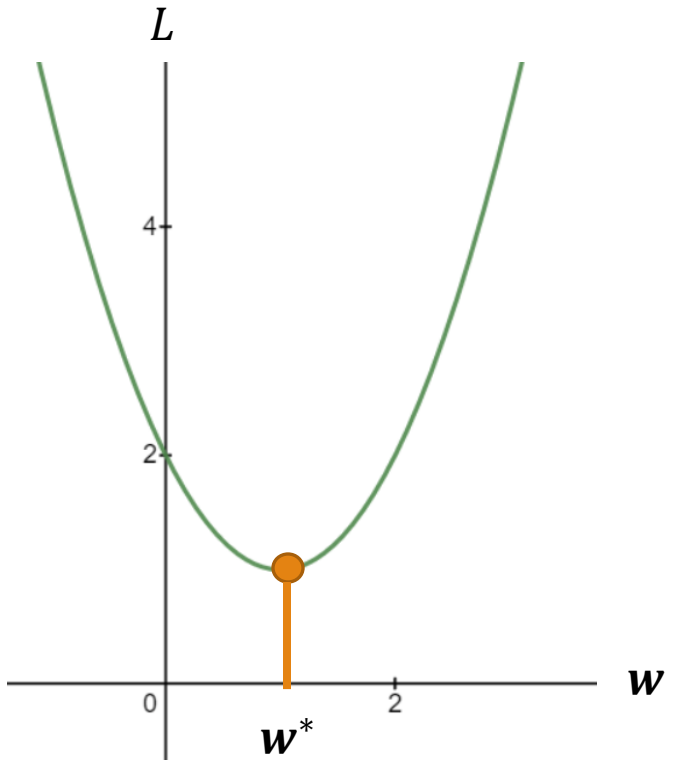
Consider: $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$

Our task is finding weights, $\boldsymbol{w}$, that minimizes the loss, L.

From intuition, we can simply find the optimal weights is finding the turning points of the loss function by differentiating it.

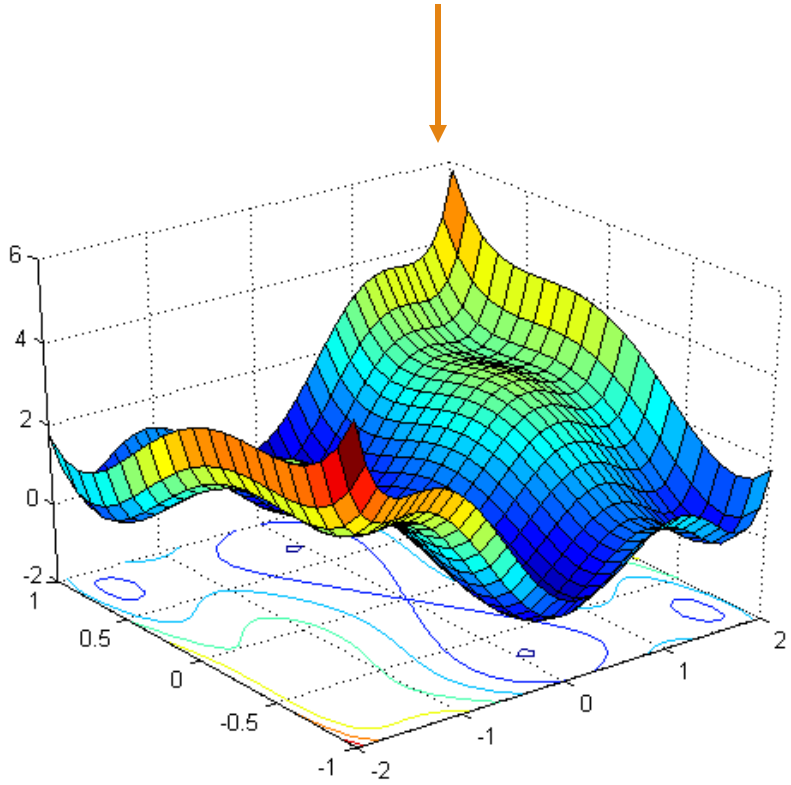
$$\boldsymbol{w}^* = \boldsymbol{argmin}_{\boldsymbol{w}}(L(\boldsymbol{x}, \boldsymbol{w}))$$

$$0 = \frac{\partial L}{\partial \boldsymbol{w}}|_{\boldsymbol{w}=\boldsymbol{w}^*}$$



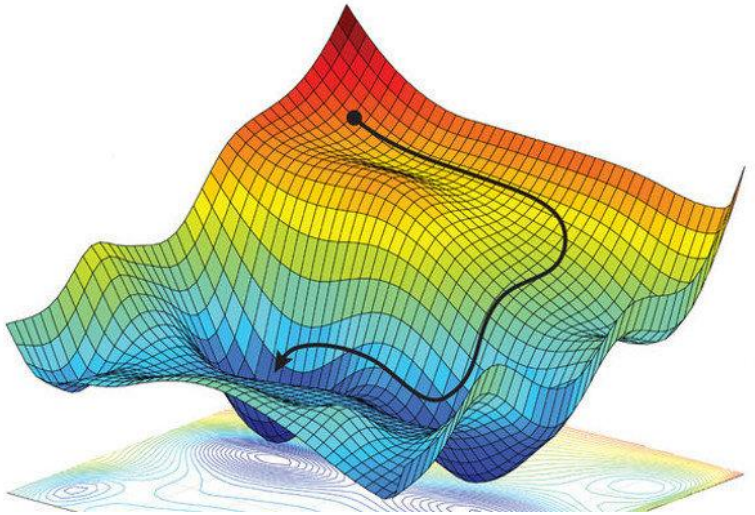
Finding optimal weights – Loss function

Example loss function with two weights



Actual loss functions with billions of weight parameters are computationally impossible to compute the full derivative.

So, without the derivative, how can we find the optimal weights?

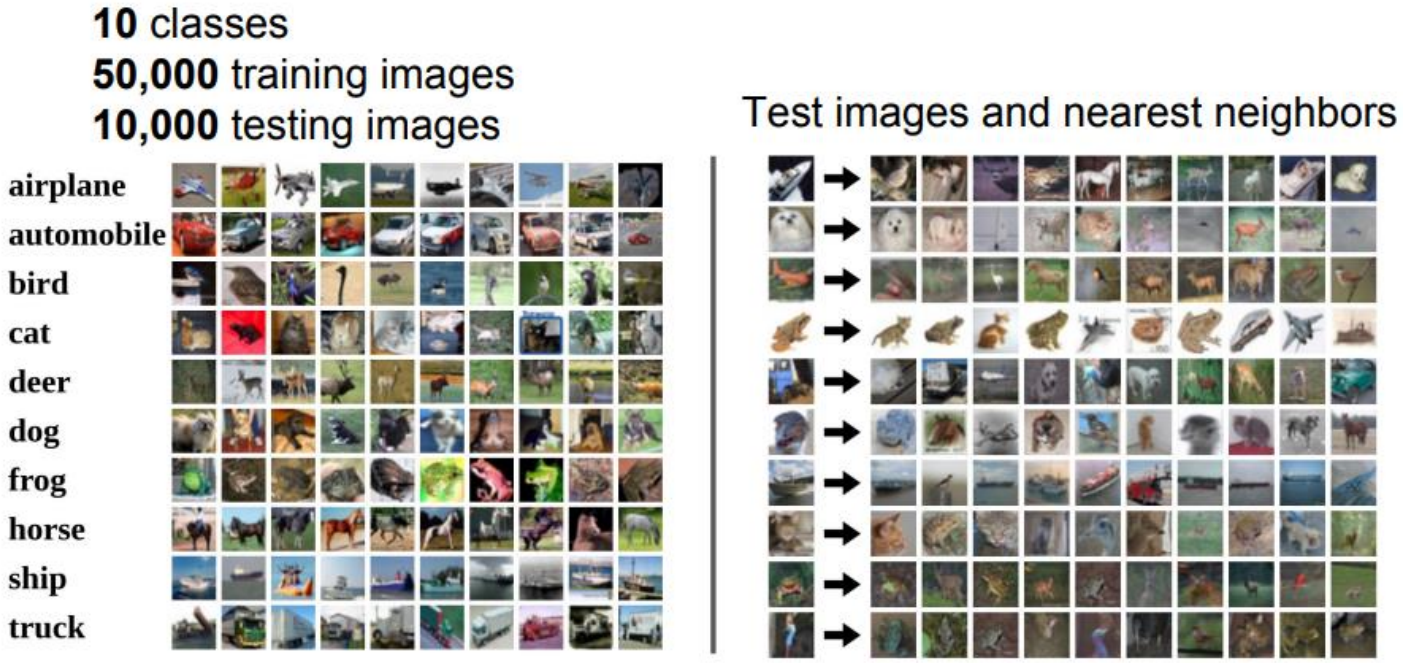


Learning from data

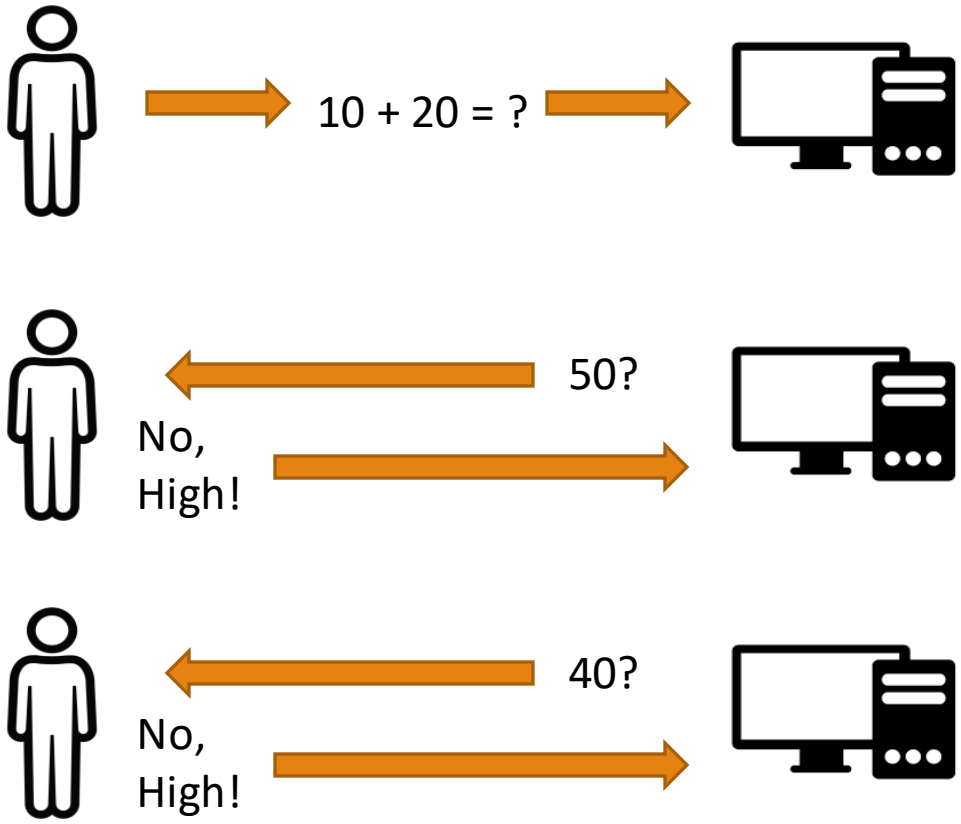
We are living in a data era.

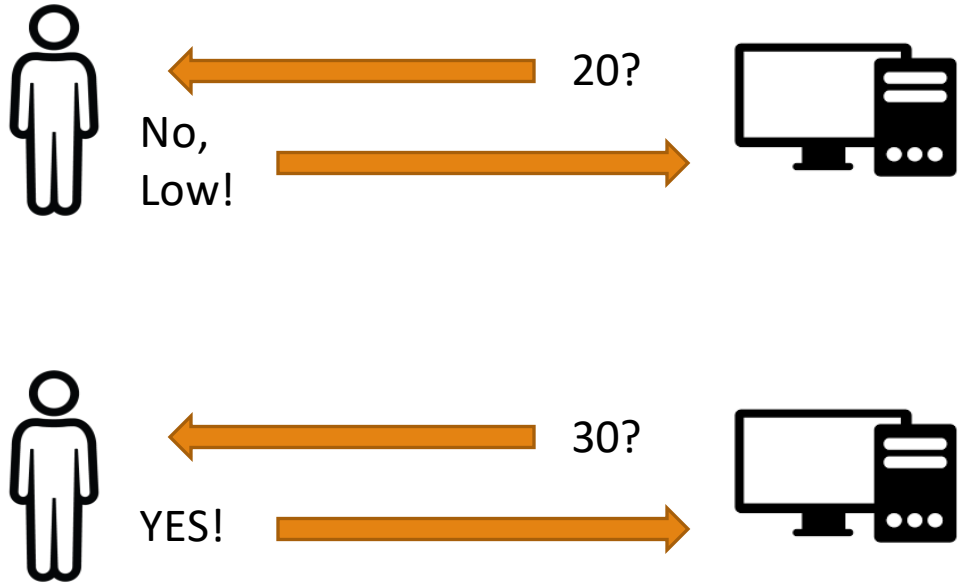
We do not set weights by ourselves.

The Neural Network Learn from data.

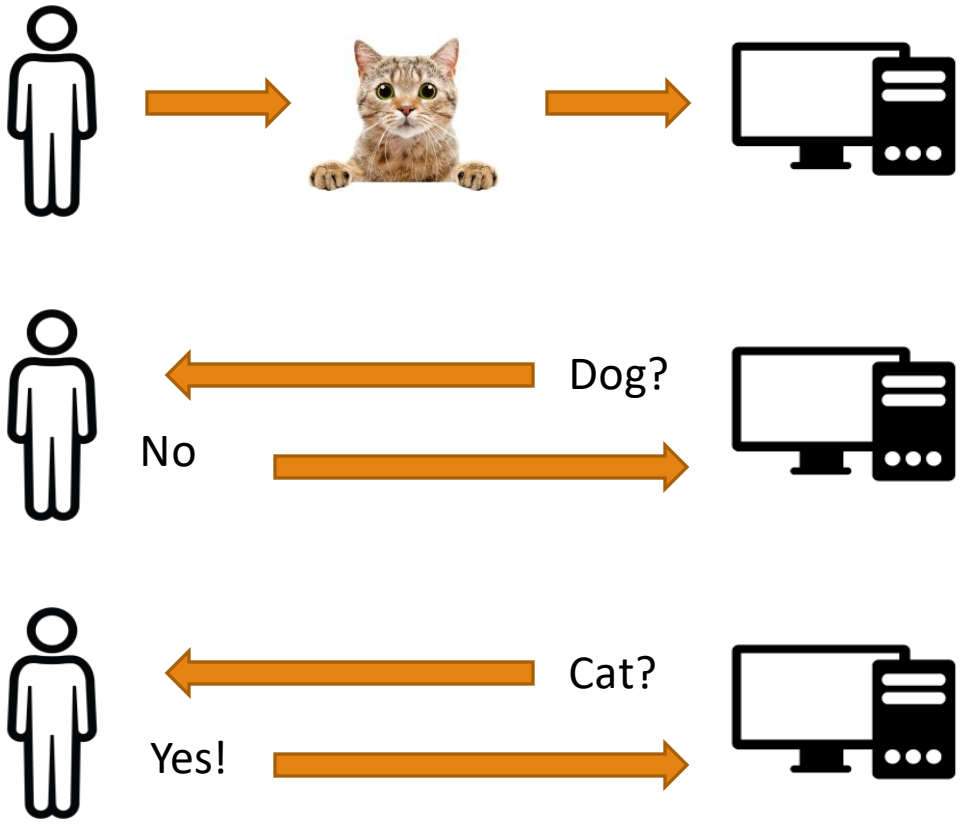


Learning from data





Learning from data



We, the humans tell the machine whether the machine's answer is wrong or correct.

Human feedback can be enumerated, denoting how wrong (off from the correct answer) the machine's answer is.

The machine's answer refers to the output of a neural network.

From Last Lecture
Finding optimal weights – Loss function

As discussed so far, the key of deep learning and using neural networks lies on being able to find optimal/near-optimal weight parameters, $\boldsymbol{w}$.

Consider a function expression of neural network:

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w})$$

If wrong weights are used, the outputs will be far from our expectations and correct answers.
Therefore, we need a measure of how good the weights are!
We call this a **Loss function**.

Loss Function: a function that represents differences between network outputs and given answers.

Eg; $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$, where t is the given answer label

Sum of Squares for Error, SSE

Sum of Squares for Error, SSE $$L = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

y denotes the network output
t denotes answer labels.

Eg;

$$y = [1, \quad 0.5, \quad 0, \quad 0.6, \quad 3, \quad 2.5]$$
$$t = [0, \quad 0.8, \quad 0.1, \quad 0, \quad 1, \quad 2]$$

$$L =$$

Error for Q

Optimal Q values

Q	Dealer showing 6 & No usable ace									
	12	13	14	15	16	17	18	19	20	21
Hit	0.9	0.8	0.7	0.7	0.6	0	0	0	0	0
Stick	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1

Network output for current policy:

$$Q^{\pi}([16, 6, No]) = [0.5, 1.0]$$

Hit

What is the SSE value?

From Last Lecture
Finding optimal weights – Loss function

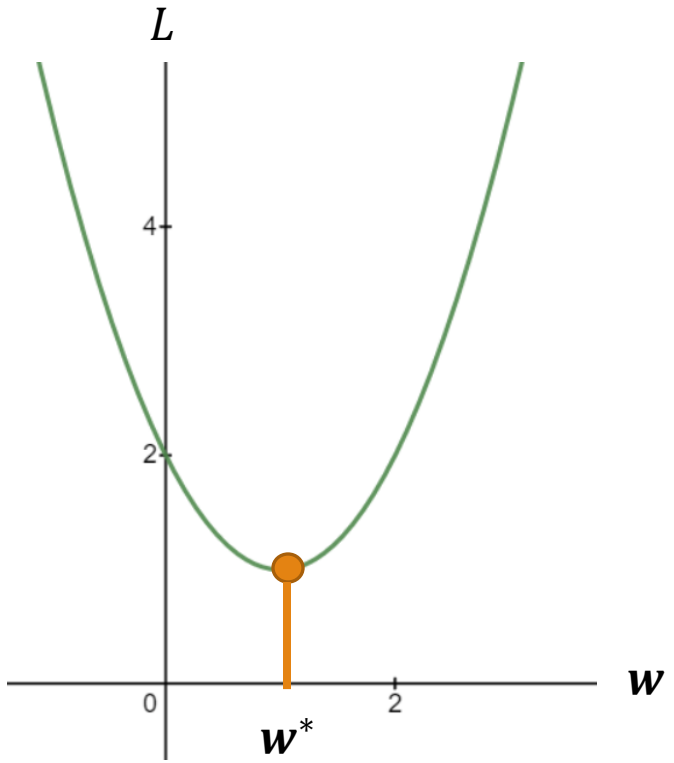
Consider: $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$

Our task is finding weights, $\boldsymbol{w}$, that minimizes the loss, L.

From intuition, we can simply find the optimal weights is finding the turning points of the loss function by differentiating it.

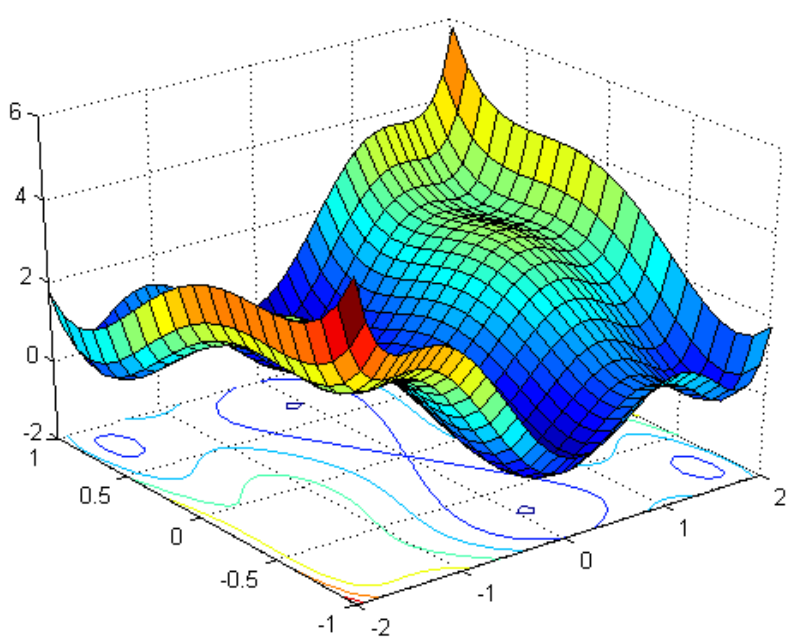
$$\boldsymbol{w}^* = \boldsymbol{argmin}_{\boldsymbol{w}}(L(\boldsymbol{x}, \boldsymbol{w}))$$

$$0 = \frac{\partial L}{\partial \boldsymbol{w}} |_{\boldsymbol{w}=\boldsymbol{w}^*}$$



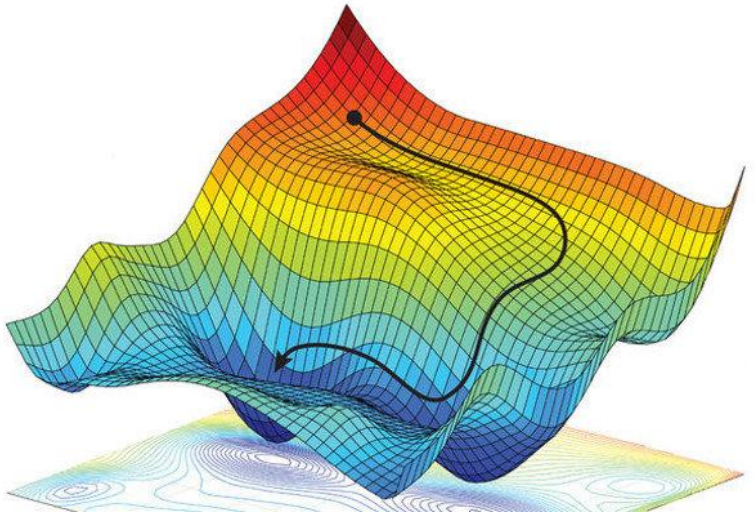
From Last Lecture
Finding optimal weights – Loss function

Example loss function with two weights



Actual loss functions with billions of weight parameters are computationally impossible to compute the full derivative.

So, without the derivative, how can we find the optimal weights?



Gradient Descent

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta \frac{\partial L}{\partial \boldsymbol{w}}$$

$L = w^2 - 2w + 2$

Let's start with $w_0 = 3$

Then, $L = 5$

$$\frac{\partial L}{\partial w} = 2w - 2$$

$$\frac{\partial L}{\partial \mathrm{w}}|_{w=3} = 4$$

$$w_1 = 3 - 0.1 * 4 = 2.6$$

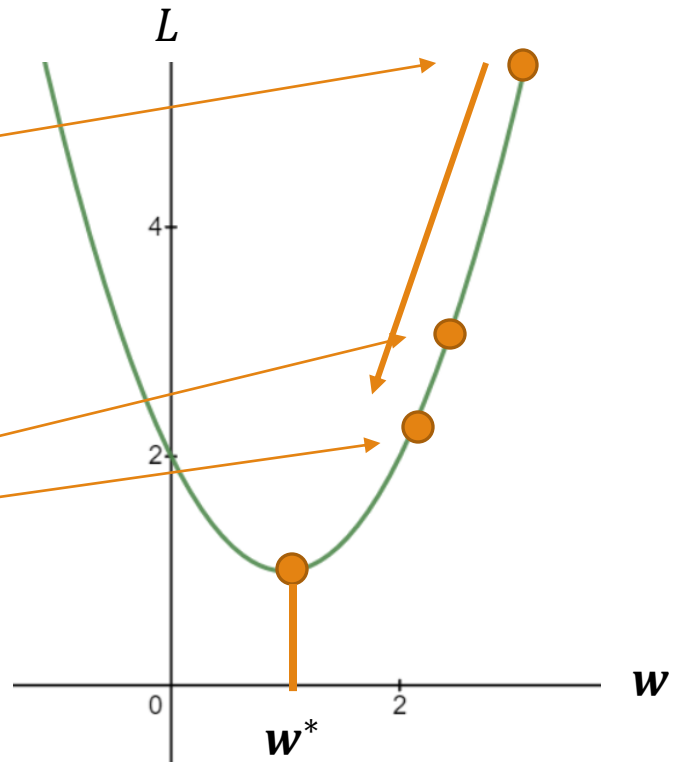
$L = w^2 - 2w + 2$

Repeat with $w_1 = 2.6$

Then, $L = 3.56$

$$\frac{\partial L}{\partial \mathrm{w}}|_{w=2.6} = 3.2$$

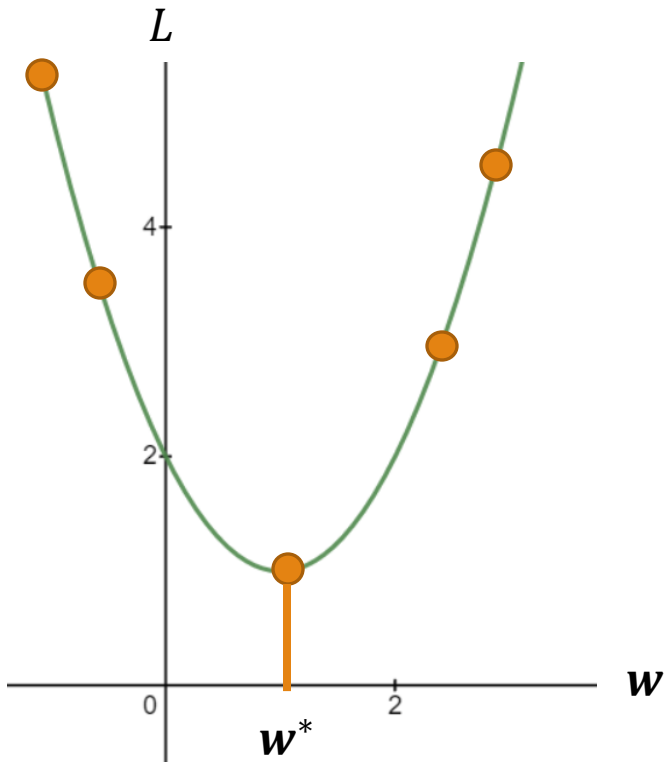
$$w_2 = 2.6 - 0.1 * 3.2 = 2.28$$



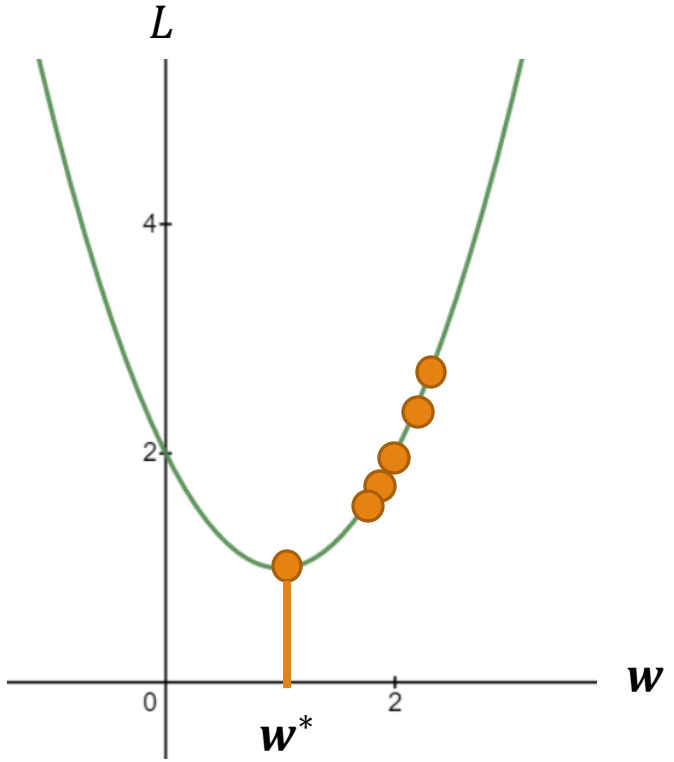
Here, η is called the learning rate. It determines how fast the learning will take place. In the example above, it is set to 0.1

Learning rate

When learning rate is too big, say, 10.



When learning rate is too small, say 0.000001.



Minibatch

Computing Loss with only a single data is very inefficient.

We usually take a batch of data and compute the mean loss.

Take Sum of Squares for Error as an example, where B denotes the batch size.

$$L = \frac{1}{|B|}\sum_{B}\frac{1}{2}\sum_{k}(y_k - t_k)^2$$

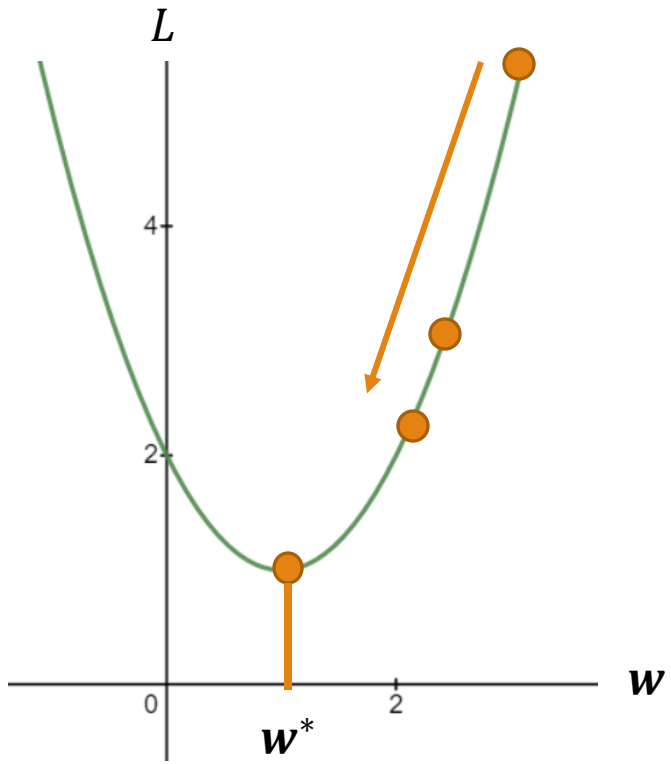
We call this approach a “Minibatch”, due to a small batch size being more efficient in training.

Stochastic Gradient Descent

Stochastic Gradient Descent: Applying mean loss of randomly sampled minibatch data and performing gradient descent algorithm.

$$L = \frac{1}{|B|}\sum_{B}\frac{1}{2}\sum_{k}(y_k - t_k)^2$$

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta\frac{\partial L}{\partial \boldsymbol{w}}$$



Gradient Computation

$$\boldsymbol{w}^{(z)} = \begin{pmatrix} w_{b1}^{1} & w_{11}^{1} & w_{21}^{1} & \cdots & w_{n1}^{1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{bp}^{1} & w_{1p}^{1} & w_{2p}^{1} & \cdots & w_{np}^{1} \end{pmatrix}$$

$$\frac{\partial L}{\partial \boldsymbol{w}}^{(z)} = \begin{pmatrix} \frac{\partial L}{\partial w_{b1}^{1}} & \frac{\partial L}{\partial w_{11}^{1}} & \frac{\partial L}{\partial w_{21}^{1}} & \cdots & \frac{\partial L}{\partial w_{n1}^{1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{bp}^{1}} & \frac{\partial L}{\partial w_{1p}^{1}} & \frac{\partial L}{\partial w_{2p}^{1}} & \cdots & \frac{\partial L}{\partial w_{np}^{1}} \end{pmatrix}$$

The Weight matrix and the Gradient Matrix have the same dimension.

From Last Lecture

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$\boldsymbol{h} = a(\boldsymbol{w^{(1)}x})$$

Where

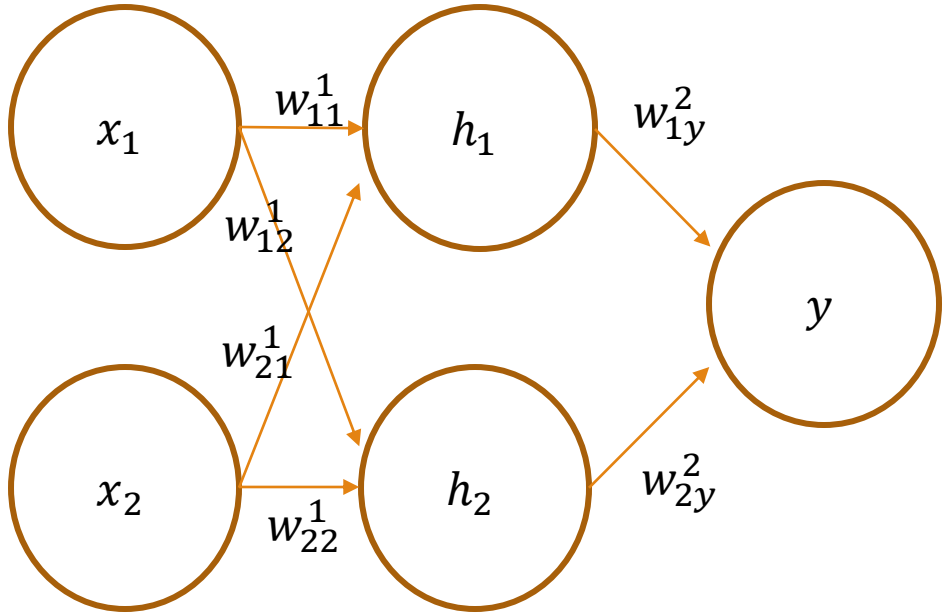
$$\boldsymbol{w^{(1)}} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Then,

$$y = a(\boldsymbol{w^{(2)}h})$$

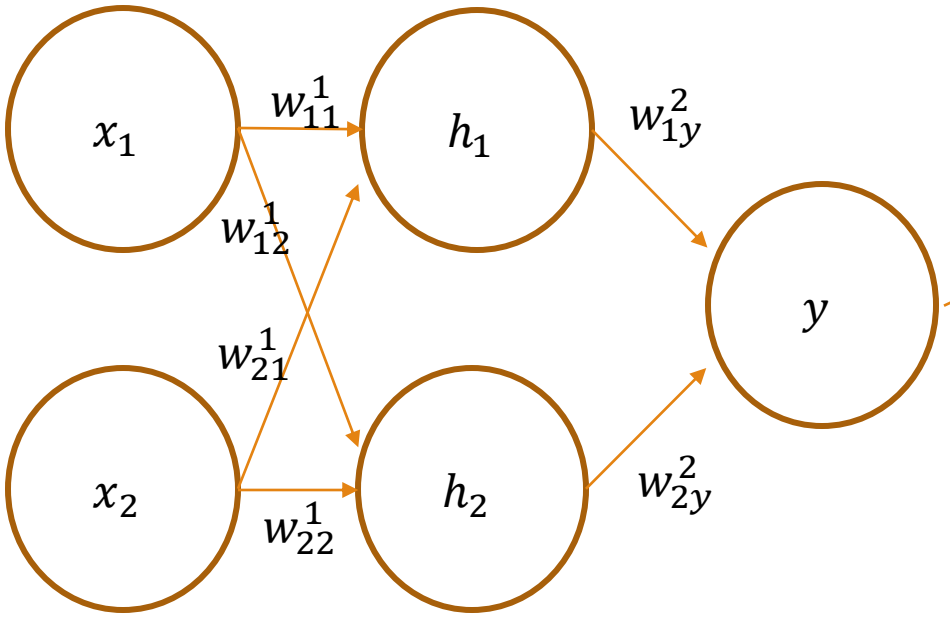
Where

$$\boldsymbol{w^{(2)}} = \begin{pmatrix} w_{by}^2 & w_{1y}^2 & w_{2y}^2 \end{pmatrix}$$



We denote the activation function as $a()$

Gradient Computation



$$L = \frac{1}{|B|}\sum_B \frac{1}{2}\sum_k (y_k - t_k)^2$$

Label, t

Say, we want to compute $\frac{\partial L}{\partial w_{11}^1}$, then,

$$\frac{\partial L}{\partial w_{11}^1} = \frac{1}{|B|}\sum_B \sum_k (y_k - t_k)\frac{\partial y_k}{\partial w_{11}^1}$$

Where,

$$\frac{\partial y_k}{\partial w_{11}^1} = \frac{\partial}{\partial \boldsymbol{h}} a(\boldsymbol{w}^{(2)}\boldsymbol{h})\frac{\partial \boldsymbol{h}}{\partial w_{11}^1}$$

And so on…….

Algorithm

Algorithm: Stochastic Gradient Descent
Input: Training data D ={X,Y}, epoch e, learning rate η, stop threshold τ
Output: Optimum weight matric, $\boldsymbol{w}^*$

1. Initialize $\boldsymbol{w_0}$ with random numbers
2. For i=1,2,3,…,e repeat
    3. Sample minibatch data D_M from D
    4. Compute y via forward propagation
    5. Compute loss, L
        6. If L is below τ break
    7. Compute $\frac{\partial L}{\partial \boldsymbol{w}}$
    8. $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \frac{\partial L}{\partial \boldsymbol{w}}$
9. Return $\boldsymbol{w_e}$

Loss function in Q-Learning

Unlike standard Supervised Learning(SL), in Reinforcement Learning(RL), the correct answer, which we call **a label, does not exist.**

So how can we compute a loss?

Consider the Q update equation of Q-Learning:

$$\mathrm{Q}(s_t, a_t) \leftarrow \mathrm{Q}(s_t, a_t) + \alpha\big(r_t + \gamma \mathrm{Q}(s_{t+1}, a^*) - \mathrm{Q}(s_t, a_t)\big)$$

When we reach the Optimal Q values, the error term becomes 0. → Our Objective!
Similar to the goal is SL to make a loss 0 (or near zero to avoid overfitting).

We can compute $\mathrm{Q}(s_t, a_t)$ and $r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$ by forward propagation of our current Neural Network.

Loss function in Q-Learning

$$\mathrm{Q}(s_t, a_t) \leftarrow \mathrm{Q}(s_t, a_t) + \alpha\big(r_t + \gamma \mathrm{Q}(s_{t+1}, a^*) - \mathrm{Q}(s_t, a_t)\big)$$

When we reach the Optimal Q values, the error term becomes 0. → Our Objective!
Similar to the goal is SL to make a loss 0 (or near zero to avoid overfitting).

We can compute $\mathrm{Q}(s_t, a_t)$ and $r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$ by forward propagation of our current Neural Network.

We assume that $r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$ is a more accurate Q value than $\mathrm{Q}(s_t, a_t)$, hence we call it a target that $\mathrm{Q}(s_t, a_t)$ must follow.

$$\mathrm{y} = \mathrm{Q}(s_t, a_t)$$

$$t = r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$$

$$L = \frac{1}{2}\sum_k (y_k - t_k)^2$$

Next Lecture

Deep Q-Learning