딥러닝을 활용한 **디지털 영상 처리**
Digital Image Processing via Deep Learning

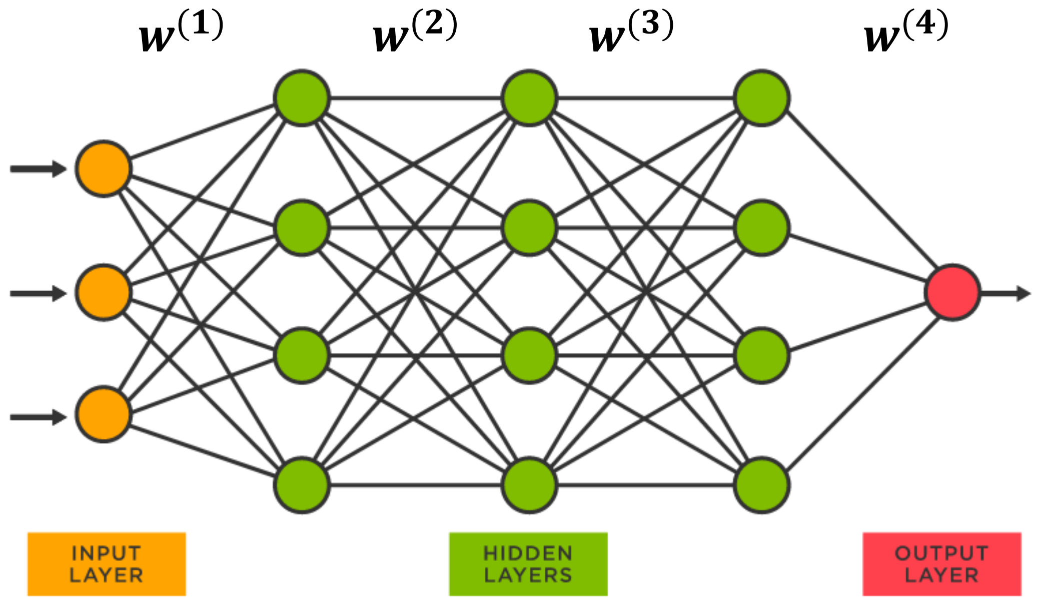
Lecture 4 – Loss functions and Gradient Descent method

From Last Lecture

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

$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^{1} & w_{11}^{1} & w_{21}^{1} & w_{31}^{1} \\ w_{b2}^{1} & w_{12}^{1} & w_{22}^{1} & w_{32}^{1} \\ w_{b3}^{1} & w_{13}^{1} & w_{23}^{1} & w_{33}^{1} \\ w_{b4}^{1} & w_{14}^{1} & w_{24}^{1} & w_{34}^{1} \end{pmatrix}$$

We usually have weight parameters in a scale of millions. 60 in the example on left.



From Last Lecture
Activation Functions

Step Function:

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

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

Logistic Sigmoid:

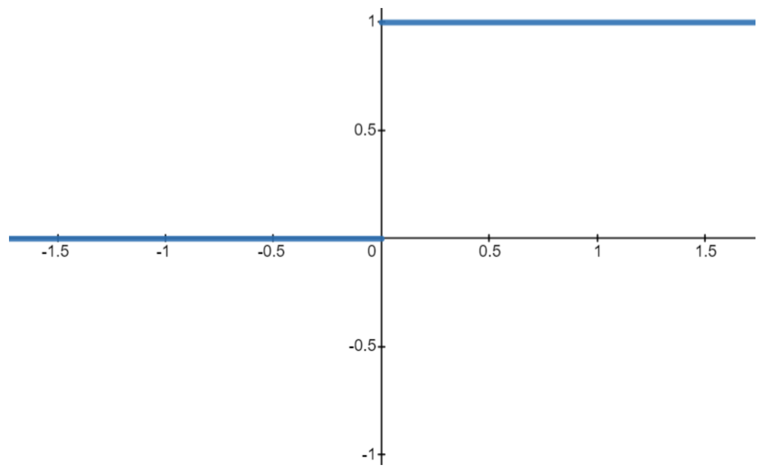
$$a(x) = \frac{1}{1 + e^{-\beta x}}$$

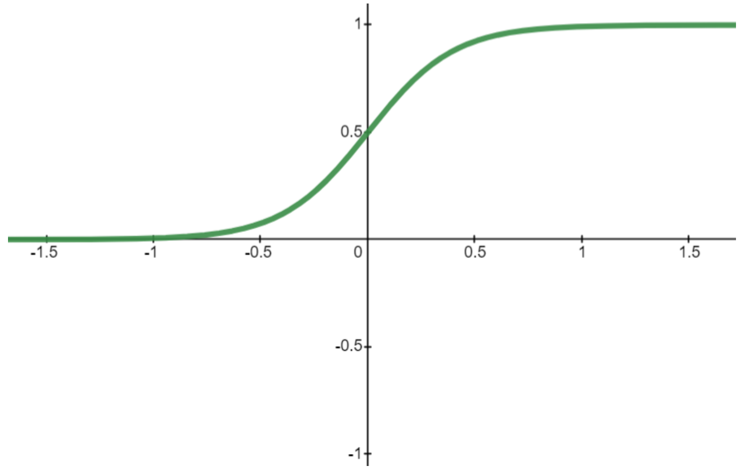
Where $\beta > 0$

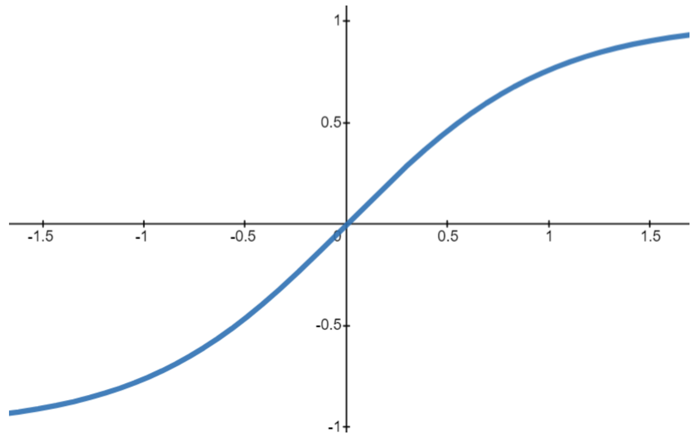
Hyperbolic Tangent:

$$a(x) = \tanh \beta x$$

Where $\beta > 0$



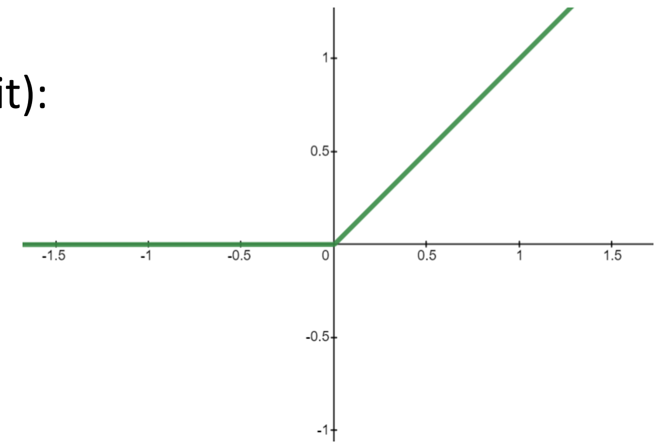




From Last Lecture
Activation Functions

ReLU(Rectified Linear Unit):

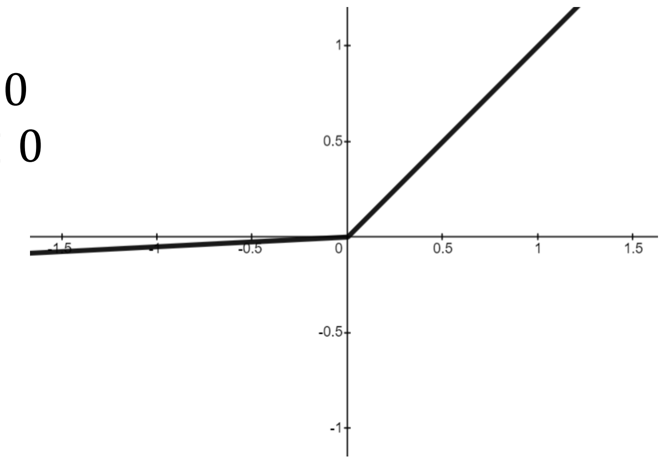
$$a(x) = \max(0, x)$$



Leaky ReLU:

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

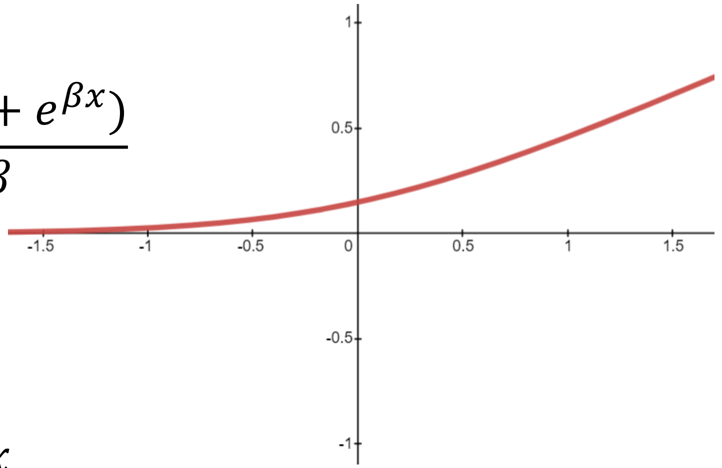
Where $0 < \beta \ll 1$



Softplus:

$$a(x) = \frac{\log(1 + e^{\beta x})}{\beta}$$

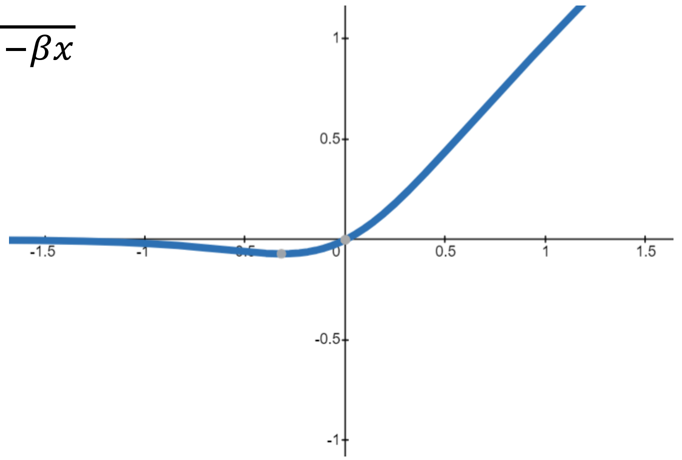
Where $\beta > 0$



Swish:

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

Where $\beta > 0$



From Last Lecture
Output Layer Activation - Softmax

The neural network mostly outputs a number indicating a “score” of the input data to a certain label.
In other words, how likely is this input to have a certain label.
This is typically expressed in probability format.

$$\begin{pmatrix} 1.058 \\ 0.013 \\ 0.568 \\ 1.345 \end{pmatrix} \Rightarrow \begin{pmatrix} 0.303 \\ 0.107 \\ 0.186 \\ 0.404 \end{pmatrix}$$

We use a softmax function for this purpose. It takes a portion of the exponential of a given indexed output out of the total → equivalent to probability.

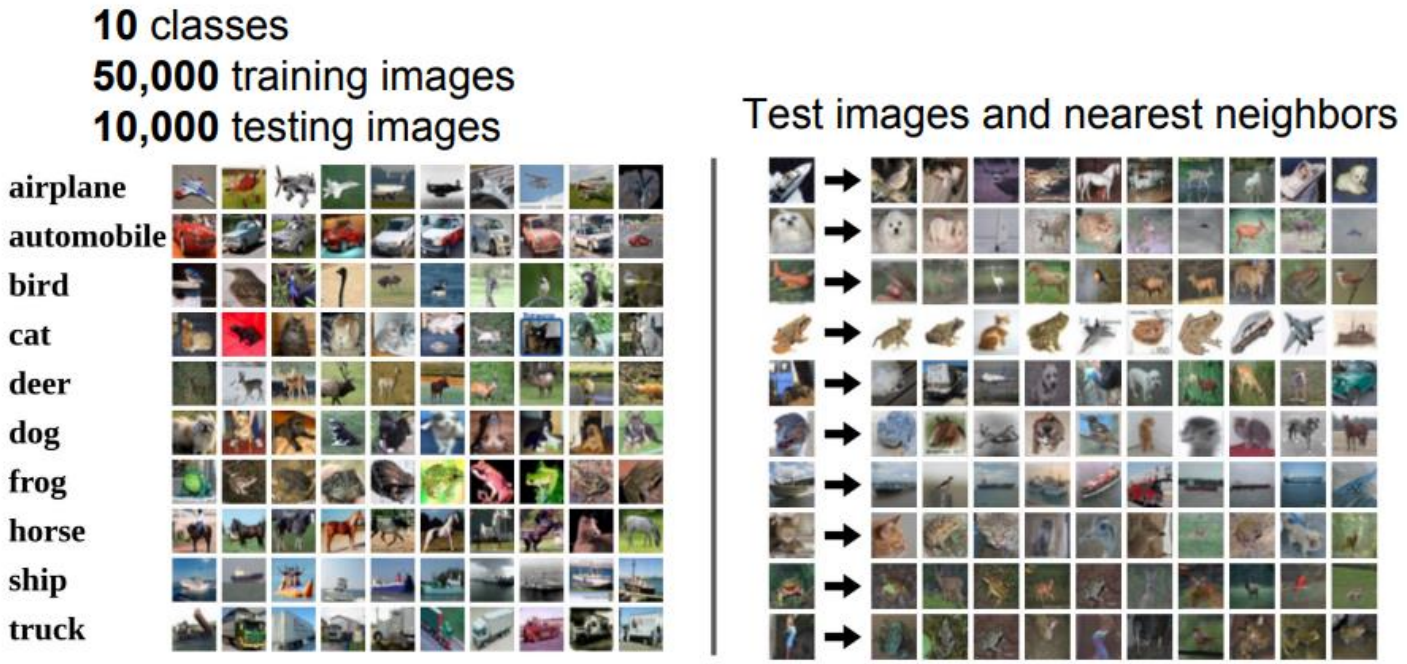
$$softmax(i, \boldsymbol{x}) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

Learning from data

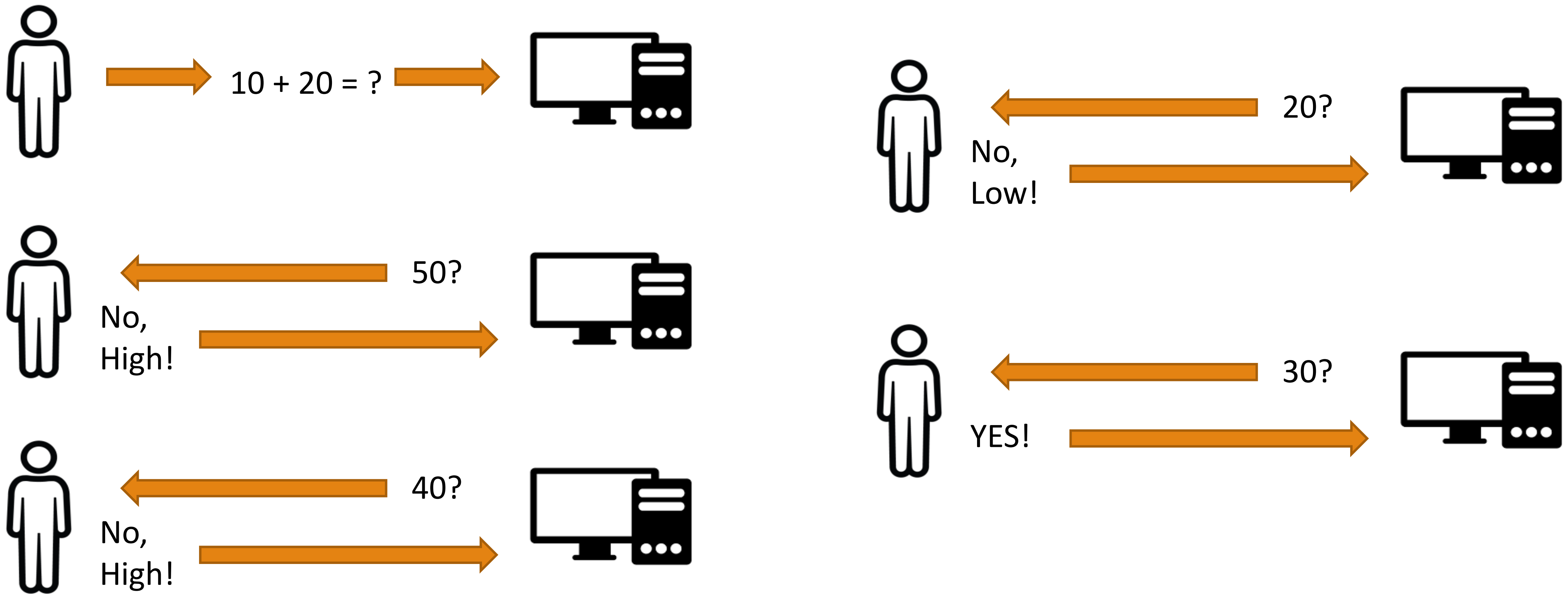
We are living in a data era.

We do not set weights by ourselves.

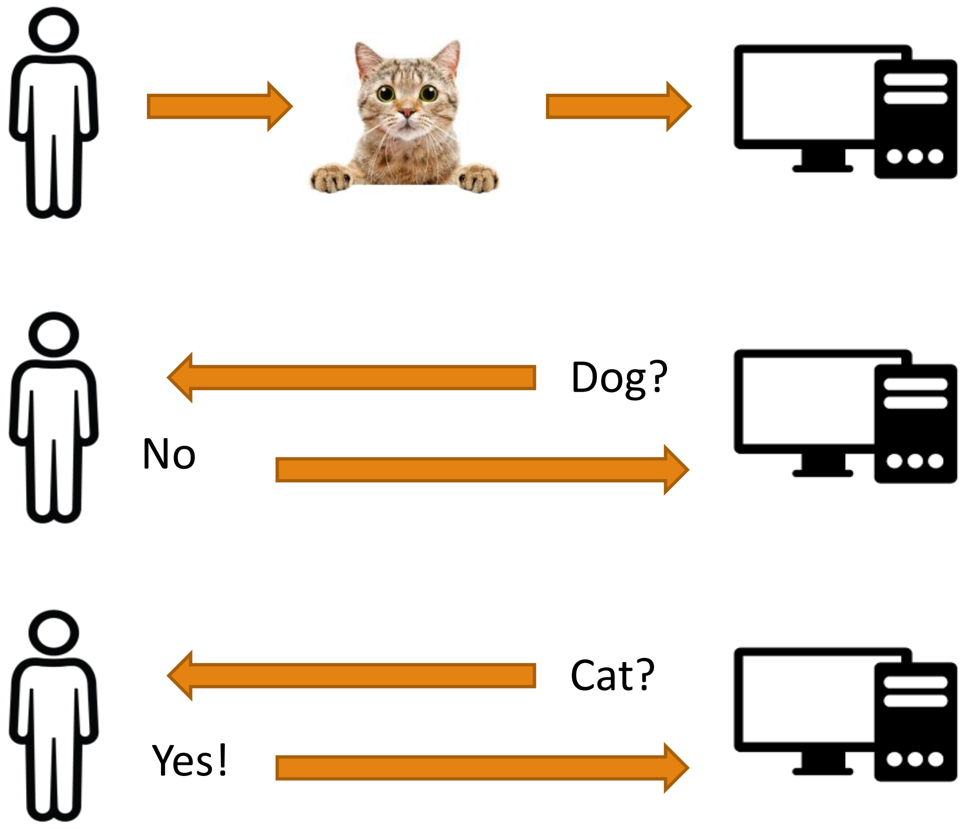
The Neural Network Learn from data.



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;

$$y = [0.1, \quad 0.005, \quad 0, \quad 0.6, \quad 0.1, \quad 0.015]$$

$$t = [0, \quad 0, \quad 0, \quad 0, \quad 1, \quad 0]$$

$$L =$$

Cross Entropy Error, CEE

Cross Entropy Error, CEE

$$L = -\sum_k t_k \log(y_k)$$

y denotes the network output
t denotes answer labels.

Eg;

$$y = [0.1, \quad 0.005, \quad 0, \quad 0.6, \quad 0.1, \quad 0.015]$$
$$t = [0, \quad 0, \quad 0, \quad 0, \quad 1, \quad 0]$$

$$L =$$

Used for classification tasks, where t is 1 for correct labels and others are all 0.

From Last Lecture
Finding optimal weights – Loss function

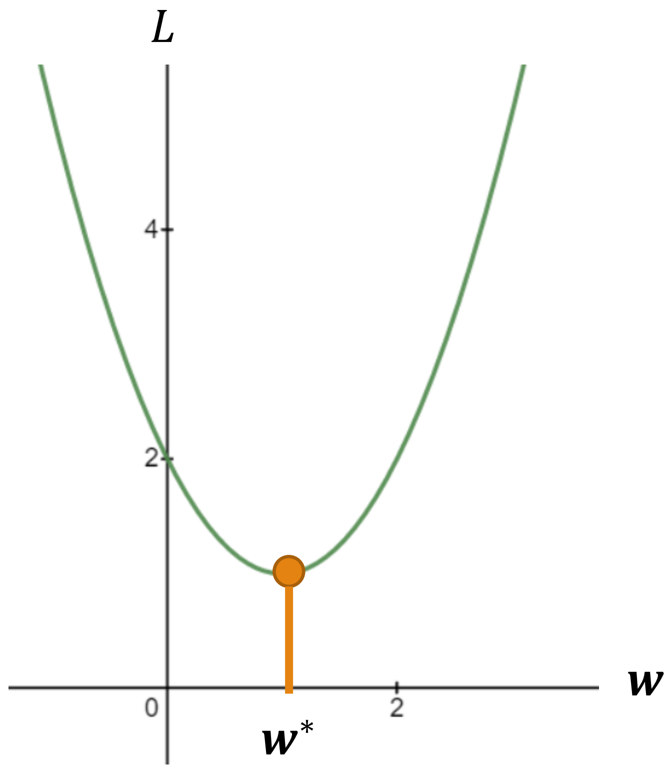
Consider: $L(\boldsymbol{x}, \boldsymbol{w}) = (y\ - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$

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.

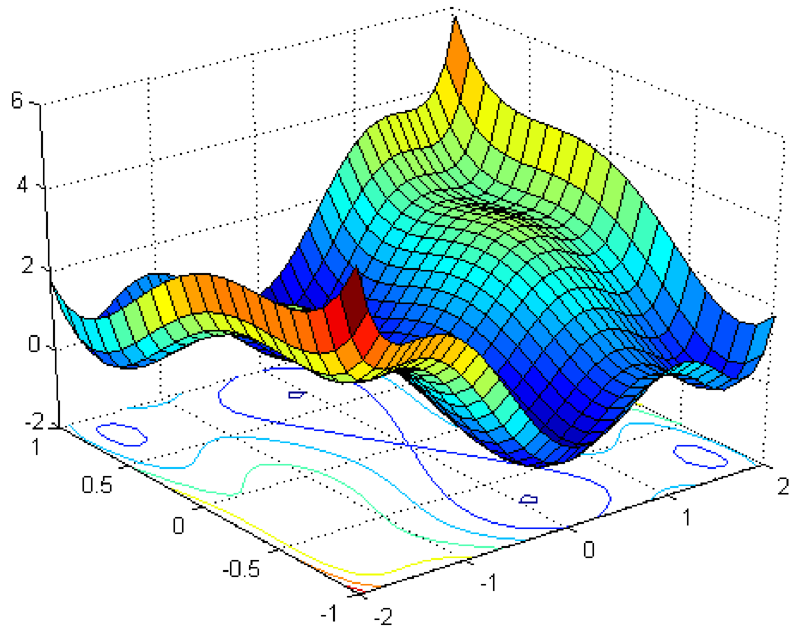
$$\boldsymbol{w}^* = \boldsymbol{argmin}_{\boldsymbol{w}}(L(\boldsymbol{x}, \boldsymbol{w}))$$

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



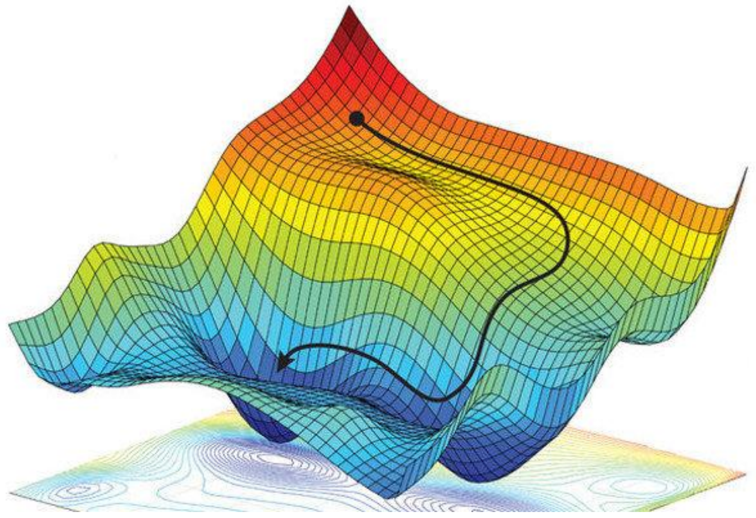
From Last Lecture
Finding optimal weights – Loss function

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, $L = 5$

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

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

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

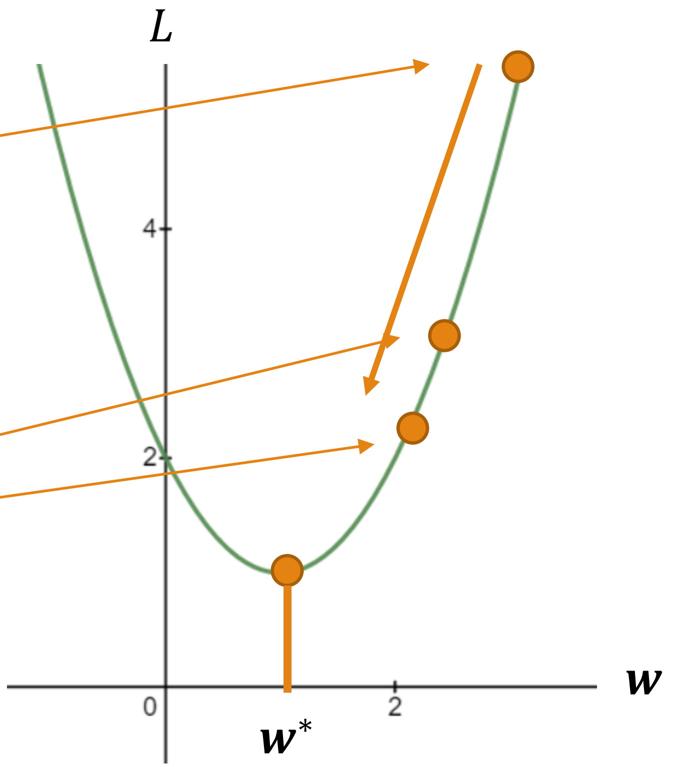
$$L = w^2 - 2w + 2$$

Repeat with $w_1 = 2.6$

Then, $L = 3.56$

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

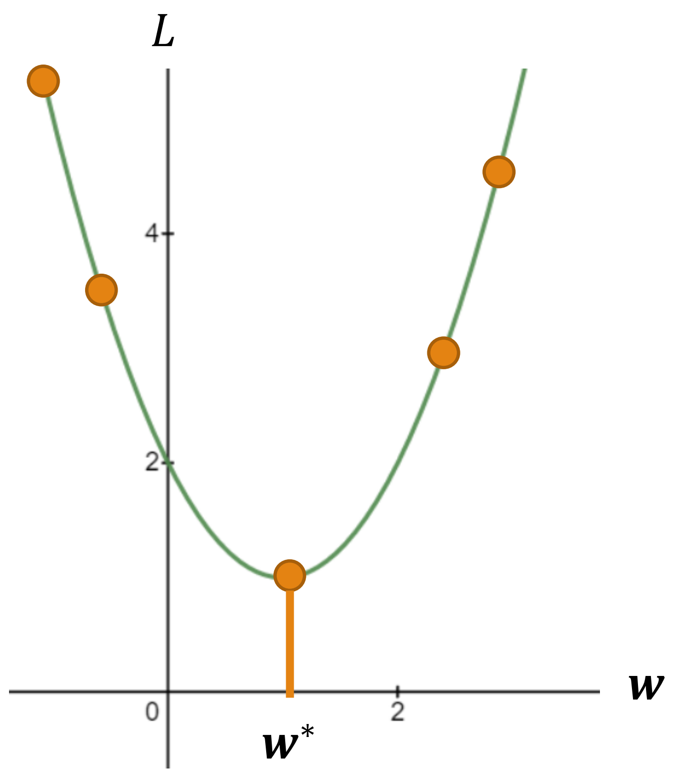
$$w_2 = 2.6 - 0.1 * 3.2 = 2.28$$



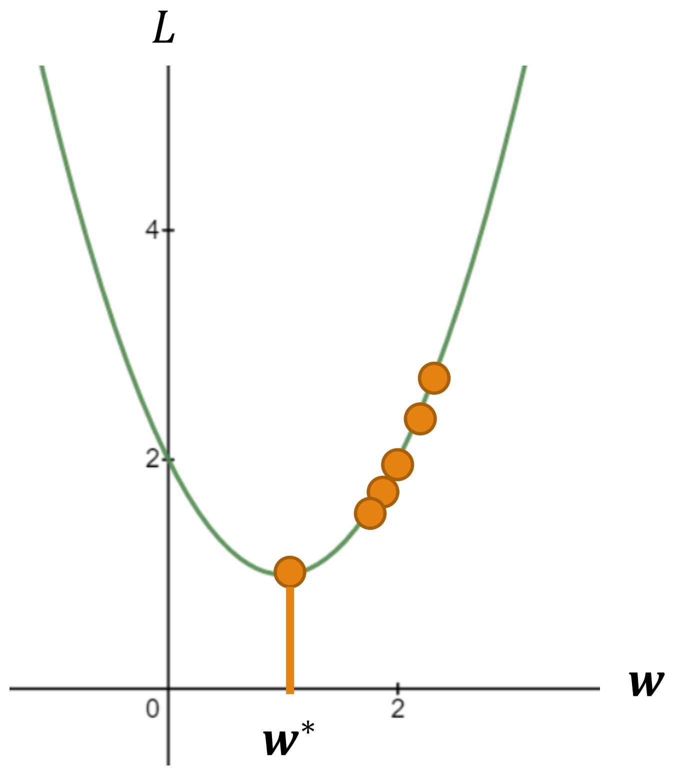
Here, η is called the learning rate. It determines how fast the learning will take place. In the example above, it is set to 0.1

Learning rate

When learning rate is too big, say, 10.



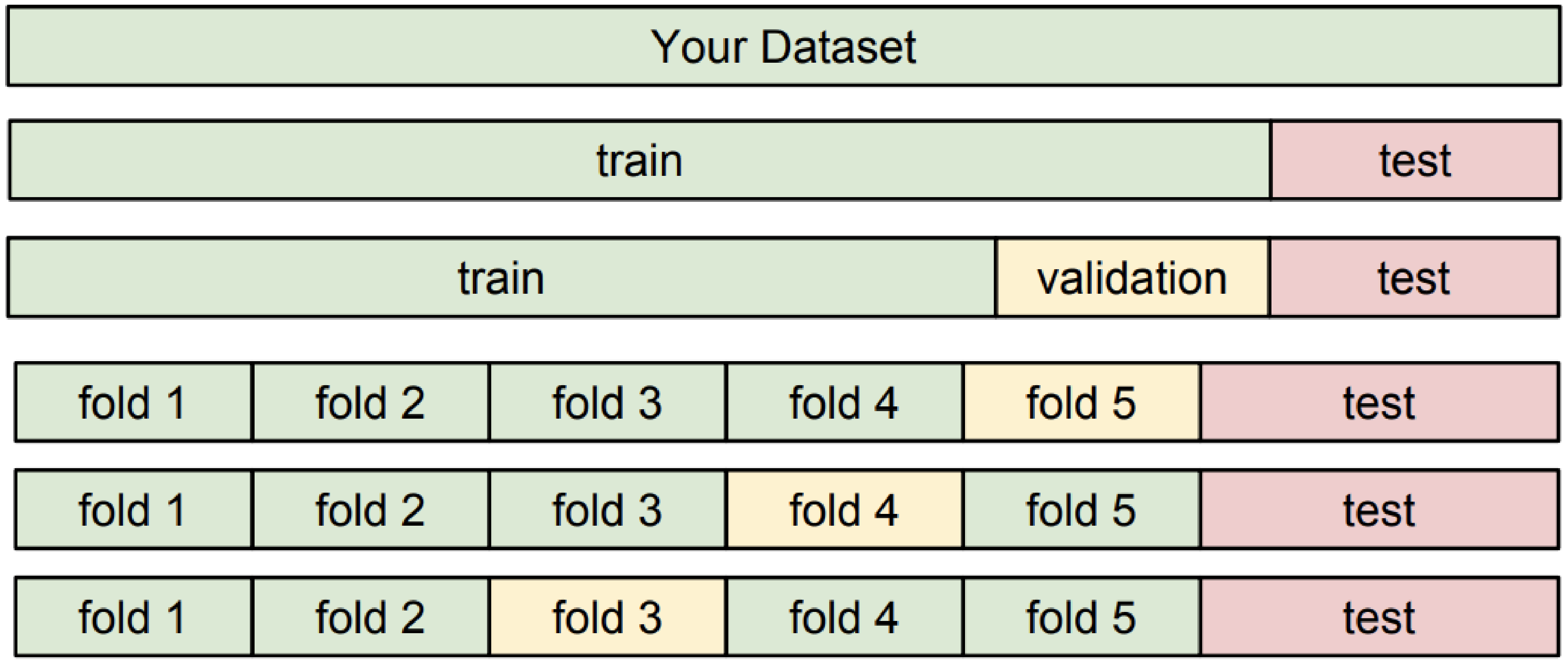
When learning rate is too small, say 0.000001.



Learning rate

The learning rate, η, is known to be one of the hyperparameters to be determined prior to the training.

We usually run multiple training procedures to verify which learning rate is good enough to use.



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.

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

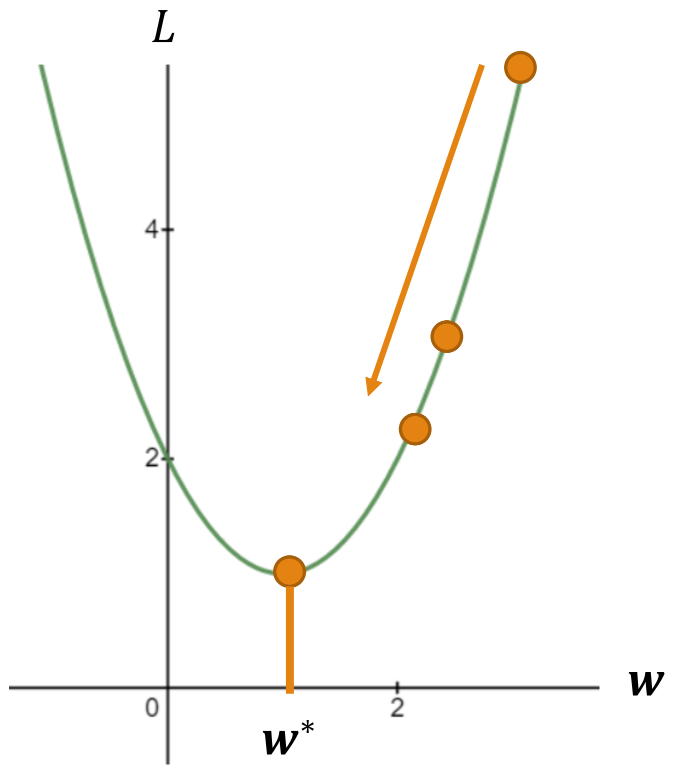
We call this approach a “Minibatch”, due to a small batch size being more efficient in training.

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,

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

Where

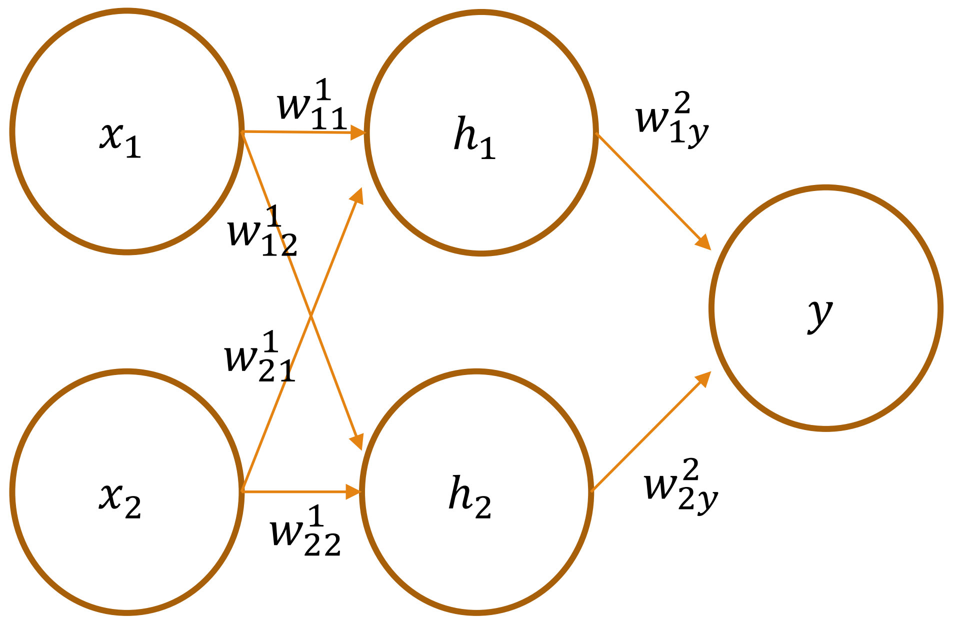
$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Then,

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

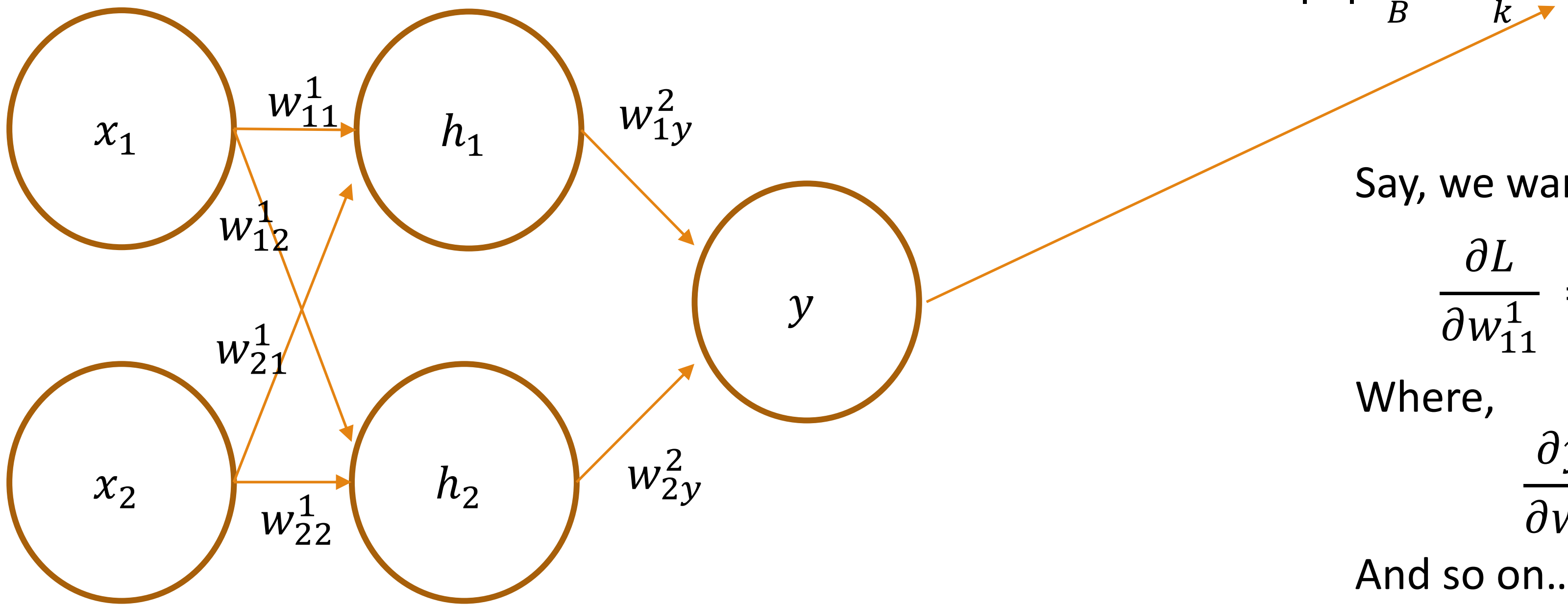
Where

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



We denote the activation function as $a()$

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 $\frac{\partial L}{\partial w^1_{11}}$, then,

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

Where,

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

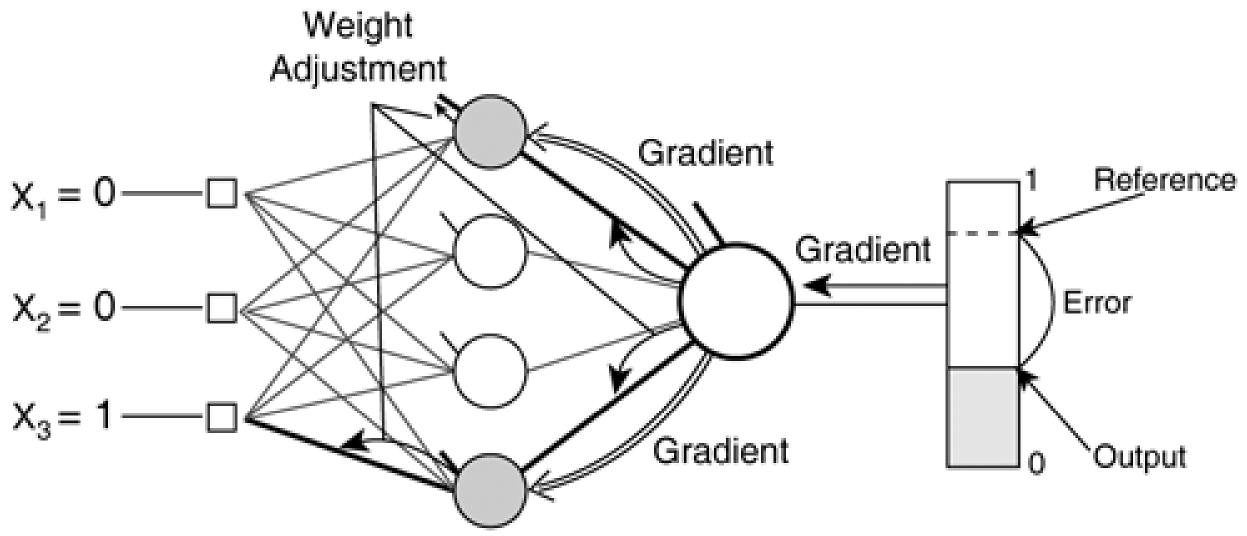
And so on…….

Algorithm

```
Algorithm: Stochastic Gradient Descent
Input: Training data $D$ ={X,Y}, epoch e, learning rate $\eta$, stop threshold $\tau$
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 $\tau$ 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 $\boldsymbol{w_e}$
```

Next Lecture

Backward Propagation


Weight Adjustment
Gradient
Gradient
Gradient
$X_1 = 0$
$X_2 = 0$
$X_3 = 1$
1
Reference
Error
Output
0