UNIT - 4

NEURAL NETWORKS

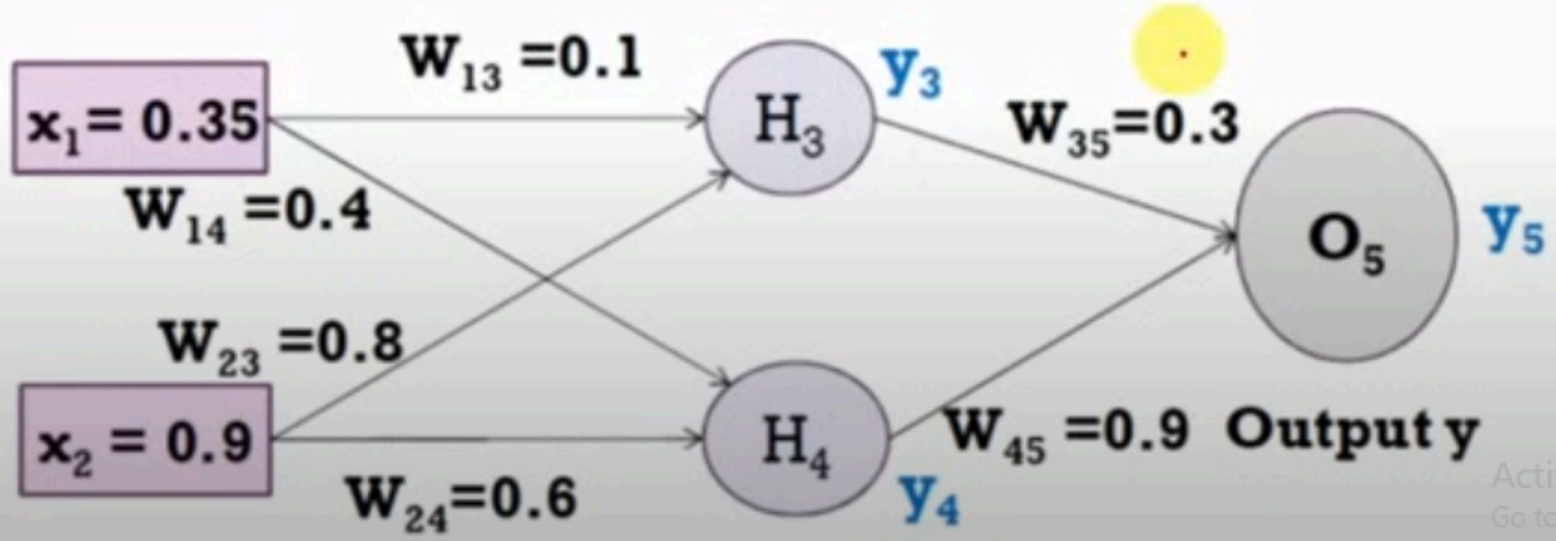
UNIT IV NEURAL NETWORKS 9

Multilayer perceptron, activation functions, network training – gradient descent optimization – Stochastic gradient descent, error backpropagation, from shallow networks to deep networks – Unit saturation (aka the vanishing gradient problem) – ReLU, hyperparameter tuning, batch normalization, regularization, dropout

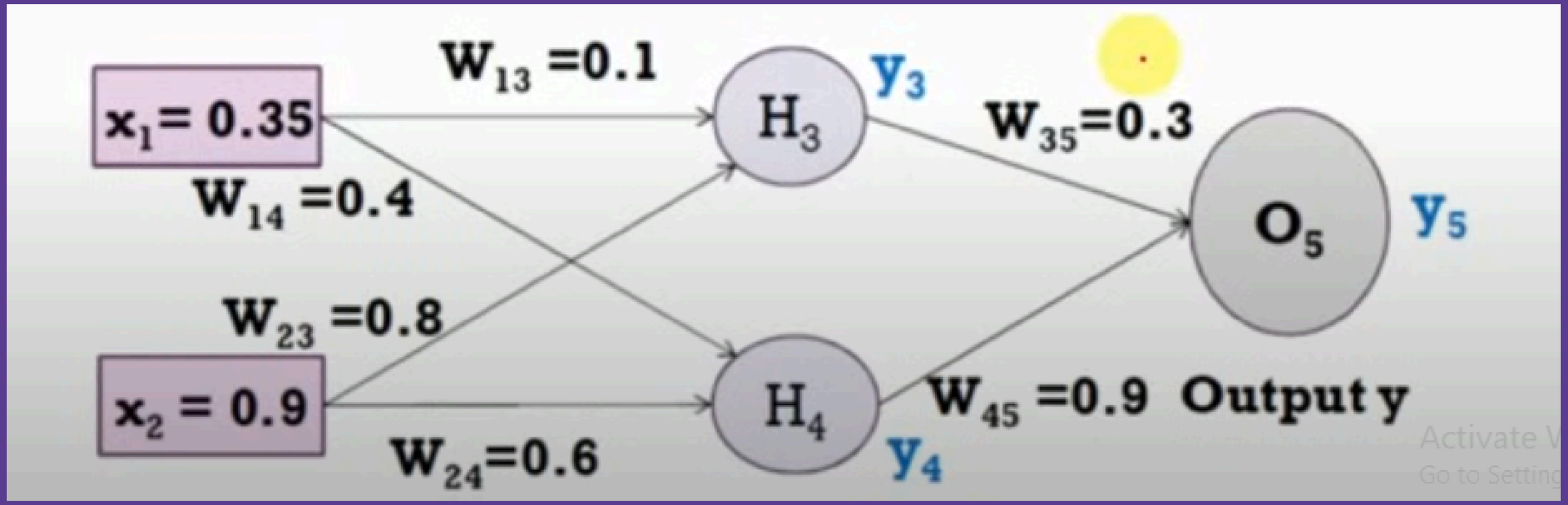
HYPERPARAMETER TUNING

BACKPROPAGATION PROBLEMS

- Assume that the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.



BACKPROPAGATION PROBLEMS



Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_i (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$ ✓

$= (0.1 * 0.35) + (0.8 * 0.9) = 0.755$

$y_3 = f(a_1) = 1/(1 + e^{-0.755}) = 0.68$

Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_i (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$ ✓

$= (0.1 * 0.35) + (0.8 * 0.9) = 0.755$

$y_3 = f(a_1) = 1/ (1 + e^{-0.755}) = 0.68$

$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$ ✓

$= (0.4 * 0.35) + (0.6 * 0.9) = 0.68$

$y_4 = f(a_2) = 1/ (1 + e^{-0.68}) = 0.6637$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$

$= (0.3 * 0.68) + (0.9 * 0.6637) = 0.801$

$y_5 = f(a_3) = 1/ (1 + e^{-0.801}) = \mathbf{0.69}$ (Network Output)

$$\text{Error} = y_{\text{target}} - y_5 = -0.19$$

$0.5 - 0.69$

- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- where η is a constant called the learning rate
- tj is the correct teacher output for unit j
- δj is the error measure for unit j

Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

$$\delta_5 = y(1-y)\,(y_{target} - y)$$

$$= 0.69*(1-0.69)*(0.5-0.69) = -0.0406$$

For hidden unit:

$$\delta_3 = y_3(1-y_3)\,w_{35} * \delta_5$$

$$= 0.68*(1 - 0.68)*(0.3 * -0.0406) = -0.00265$$

nit $\delta_4 = y_4(1-y_4)w_{45} * \delta_5$

$$= 0.6637*(1 - 0.6637)*(0.9 * -0.0406) = -0.0082$$

Compute new weights

$$\Delta w_{ji} = \eta \delta_j o_i$$

$\Delta w_{45} = \eta\, \delta_5\, y_4 = 1 * -0.0406 * 0.6637 = -0.0269$

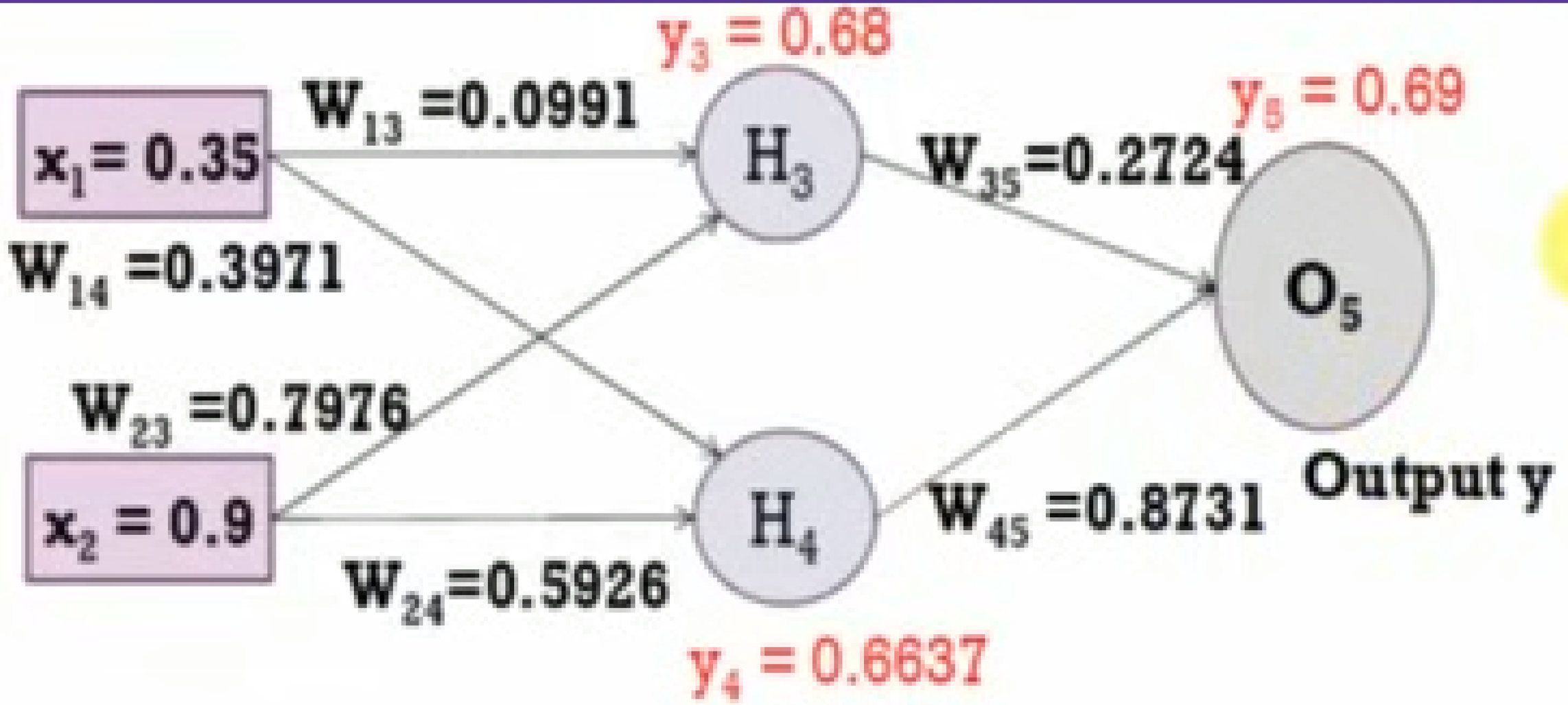
w_{45} (new) $= \Delta w_{45} + w_{45}$(old) $= -0.0269 + (0.9) = 0.8731$

$\Delta w_{14} = \eta\, \delta_4\, x_1 = 1 * -0.0082 * 0.35 = -0.00287$

w_{14} (new) $= \Delta w_{14} + w_{14}$(old) $= -0.00287 + 0.4 = 0.3971$

- Similarly, update all other weights

i	j	w_{ij}	δ_j	x_i	η	Updated w_{ij}
1	3	0.1	-0.00265	0.35	1	0.0991
2	3	0.8	-0.00265	0.9	1	0.7976
1	4	0.4	-0.0082	0.35	1	0.3971
2	4	0.6	-0.0082	0.9	1	0.5926
3	5	0.3	-0.0406	0.68	1	0.2724
4	5	0.9	-0.0406	0.6637	1	0.8731


$y_3 = 0.68$
$W_{13} = 0.0991$
$x_1 = 0.35$
H_3
$y_5 = 0.69$
$W_{35} = 0.2724$
$W_{14} = 0.3971$
O_5
$W_{23} = 0.7976$
$x_2 = 0.9$
H_4
$W_{45} = 0.8731$
Output y
$W_{24} = 0.5926$
$y_4 = 0.6637$

- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_i (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$

$= (0.0991 * 0.35) + (0.7976 * 0.9) = 0.7525$

$y_3 = f(a_1) = 1/ (1 + e^{-0.7525}) = 0.6797$

$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$

$= (0.3971 * 0.35) + (0.5926 * 0.9) = 0.6723$

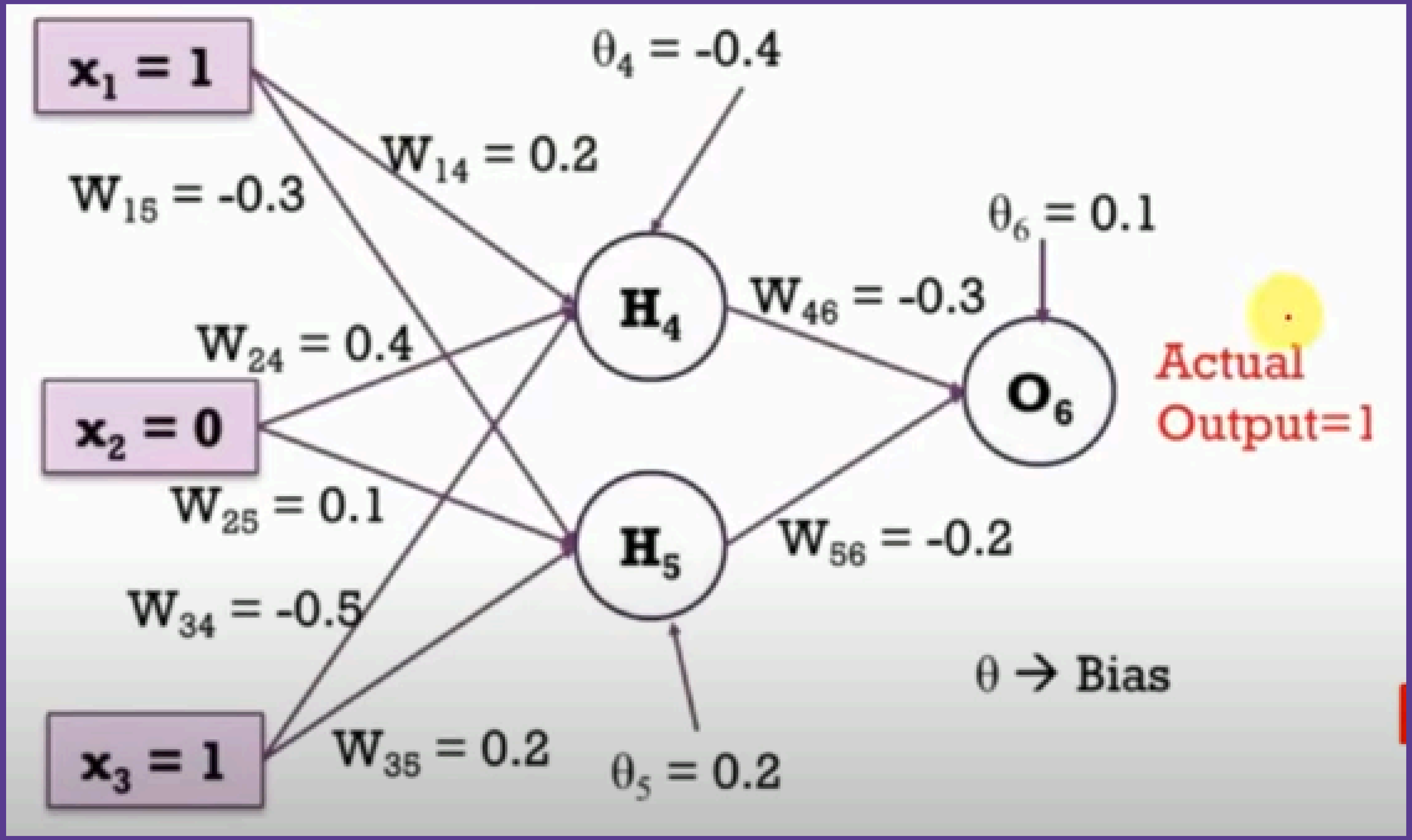
$y_4 = f(a_2) = 1/ (1 + e^{-0.6723}) = 0.6620$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$

$= (0.2724 * 0.6797) + (0.8731 * 0.6620) = 0.7631$

$y_5 = f(a_3) = 1/ (1 + e^{-0.7631}) = 0.6820$ (Network Output)

$$\text{Error} = y_{target} - y_5 = -0.182$$


$x_1 = 1$
$x_2 = 0$
$x_3 = 1$
$\theta_4 = -0.4$
$W_{14} = 0.2$
$W_{15} = -0.3$
$W_{24} = 0.4$
$W_{25} = 0.1$
$W_{34} = -0.5$
$W_{35} = 0.2$
H_4
H_5
$\theta_6 = 0.1$
$W_{46} = -0.3$
$W_{56} = -0.2$
O_6
$\theta_5 = 0.2$
Actual Output=1
$\theta \rightarrow$ Bias

Assume that the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 1 and learning rate is 0.9. Perform another forward pass.

weights, bias

parameters

Learning rate, number of layers, neurons in each layer, epoch

Hyper-parameters

Parameters	Something that is learnt during machine learning process
Hyper-parameters	Manually specified

HYPERPARAMETER

- Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning.
- The prefix 'hyper_' suggests that they are 'top-level' parameters that control the learning process and the model parameters that result from it.

PARAMETER VS HYPERPARAMETER

- A parameter is a variable that is learned from the data during the training process.
- It is used to represent the underlying relationships in the data and is used to make predictions on new data.
- A hyperparameter, on the other hand, is a variable that is set before the training process begins.

HYPERPARAMETER TUNING

- A Hyperparameter tuning is the process of selecting the optimal values for a machine learning model's hyperparameters.
- Hyperparameters are settings that control the learning process of the model, such as the learning rate, the number of neurons in a neural network, or the kernel size in a support vector machine.
- The goal of hyperparameter tuning is to find the values that lead to the best performance on a given task.

HYPERPARAMETERS IN NEURAL NETWORK

1. **Learning rate:** This hyperparameter controls the step size taken by the optimizer during each iteration of training. Too small a learning rate can result in slow convergence, while too large a learning rate can lead to instability and divergence.
2. **Epochs:** This hyperparameter represents the number of times the entire training dataset is passed through the model during training. Increasing the number of epochs can improve the model's performance but may lead to overfitting if not done carefully.

HYPERPARAMETERS IN NEURAL NETWORK

3. Number of layers: This hyperparameter determines the depth of the model, which can have a significant impact on its complexity and learning ability.

4. Number of nodes per layer: This hyperparameter determines the width of the model, influencing its capacity to represent complex relationships in the data.

HYPERPARAMETERS IN NEURAL NETWORK

5. Architecture: This hyperparameter determines the overall structure of the neural network, including the number of layers, the number of neurons per layer, and the connections between layers. The optimal architecture depends on the complexity of the task and the size of the dataset

6. **Activation function:** This hyperparameter introduces non-linearity into the model, allowing it to learn complex decision boundaries. Common activation functions include sigmoid, tanh, and Rectified Linear Unit (ReLU).

HYPERPARAMETER TUNING TECHNIQUES

1. **Grid Search**
2. **Random Search**
3. **Bayesian Optimization**

●●●

GRID SEARCH

- Grid search can be considered as a “brute force” approach to hyperparameter optimization. We fit the model using all possible combinations after creating a grid of potential discrete hyperparameter values. We log each set’s model performance and then choose the combination that produces the best results. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.
- An exhaustive approach that can identify the ideal hyperparameter combination is grid search. But the slowness is a disadvantage. It often takes a lot of processing power and time to fit the model with every potential combination, which might not be available.

GRID SEARCH

- For example:
    - If we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.
- As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of C=0.3 and Alpha=0.2, the performance score comes out to be 0.726(Highest), therefore it is selected.

C	0.1	0.2	0.3	0.4
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

Alpha

RANDOM SEARCH

- The random search method selects values at random as opposed to the grid search method's use of a predetermined set of numbers. Every iteration, random search attempts a different set of hyperparameters and logs the model's performance. It returns the combination that provided the best outcome after several iterations. This approach reduces unnecessary computation.
- RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. The advantage is that, in most cases, a random search will produce a comparable result faster than a grid search.

BAYESIAN OPTIMIZATION

- Grid search and random search are often inefficient because they evaluate many unsuitable hyperparameter combinations without considering the previous iterations' results. Bayesian optimization, on the other hand, treats the search for optimal hyperparameters as an optimization problem. It considers the previous evaluation results when selecting the next hyperparameter combination and applies a probabilistic function to choose the combination that will likely yield the best results. This method discovers a good hyperparameter combination in relatively few iterations.
- Data scientists use a probabilistic model when the objective function is unknown. The probabilistic model estimates the probability of a hyperparameter combination's objective function result based on past evaluation results. P(score(y)|hyperparameters(x))

BAYESIAN OPTIMIZATION

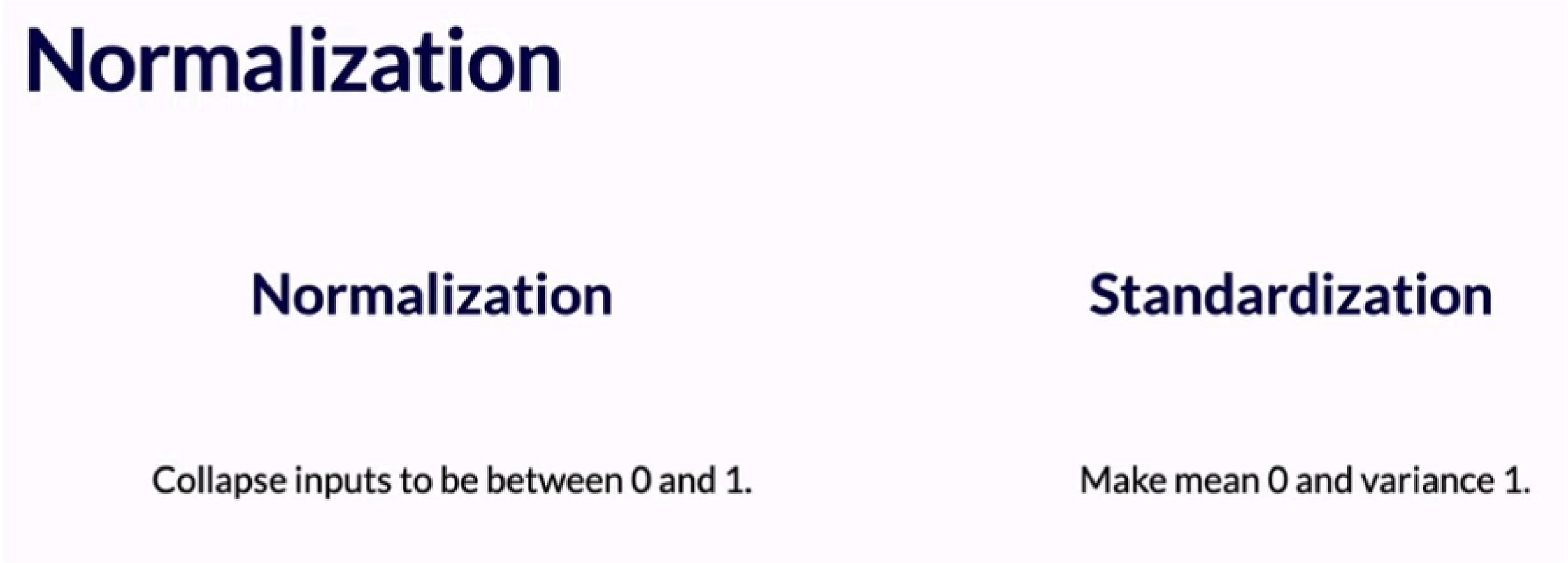
- It is a “surrogate” of the objective function, which can be the root-mean-square error (RMSE), for example. The objective function is calculated using the training data with the hyperparameter combination, and we try to optimize it (maximize or minimize, depending on the objective function selected).
- Applying the probabilistic model to the hyperparameters is computationally inexpensive compared to the objective function. Therefore, this method typically updates and improves the surrogate probability model every time the objective function runs. Better hyperparameter predictions decrease the number of objective function evaluations needed to achieve a good result. Gaussian processes, random forest regression, and tree-structured Parzen estimators (TPE) are examples of surrogate models.

BAYESIAN OPTIMIZATION

- The Bayesian optimization model is complex to implement, but off-the-shelf libraries like Ray Tune can simplify the process.
- It’s worth using this type of model because it finds an adequate hyperparameter combination in relatively few iterations.
- However, compared to grid search or random search, we must compute Bayesian optimization sequentially, so it doesn’t allow distributed processing.
- Therefore, Bayesian optimization takes longer yet uses fewer computational resources.

BATCH NORMALIZATION

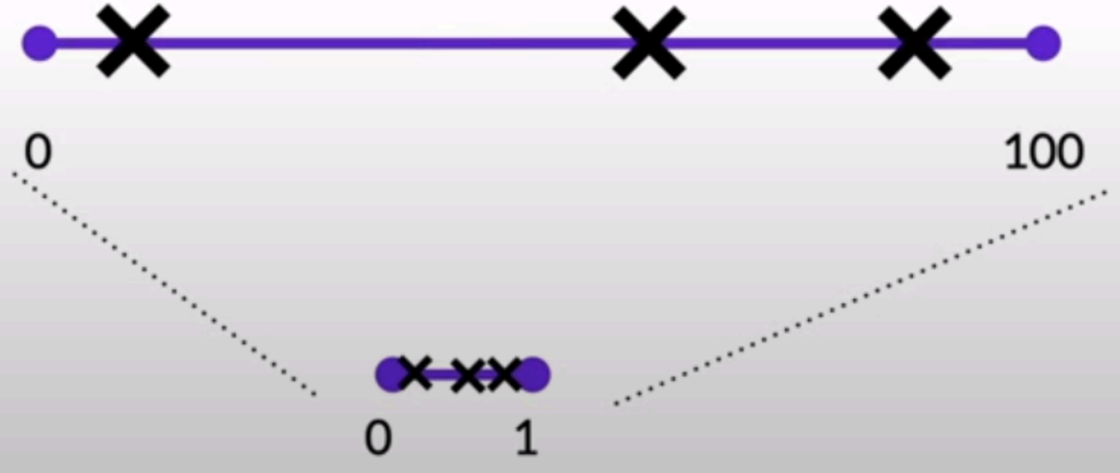
Normalization



- **Normalization - Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.**

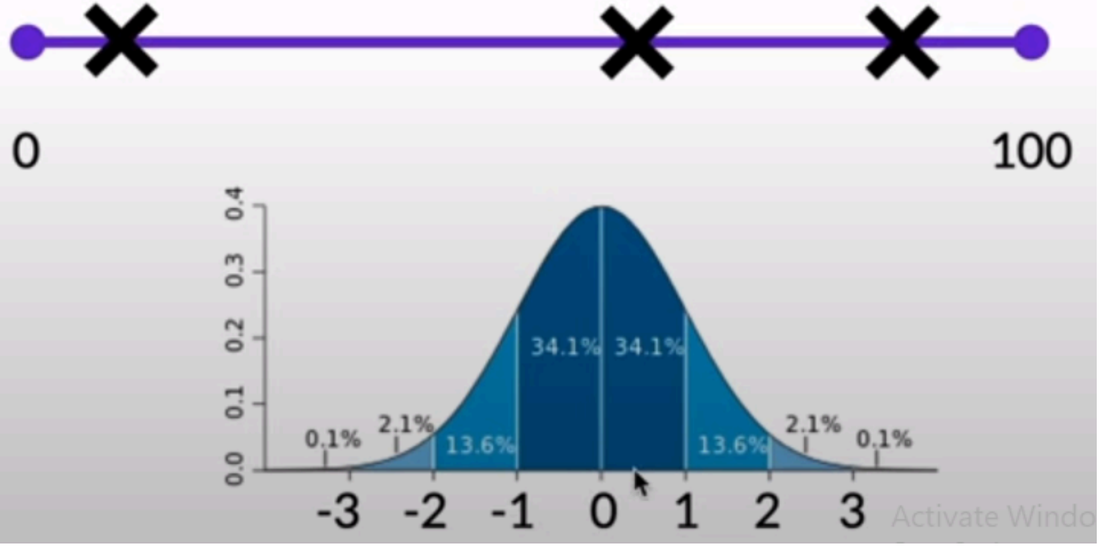
Normalization

Collapse inputs to be between 0 and 1.



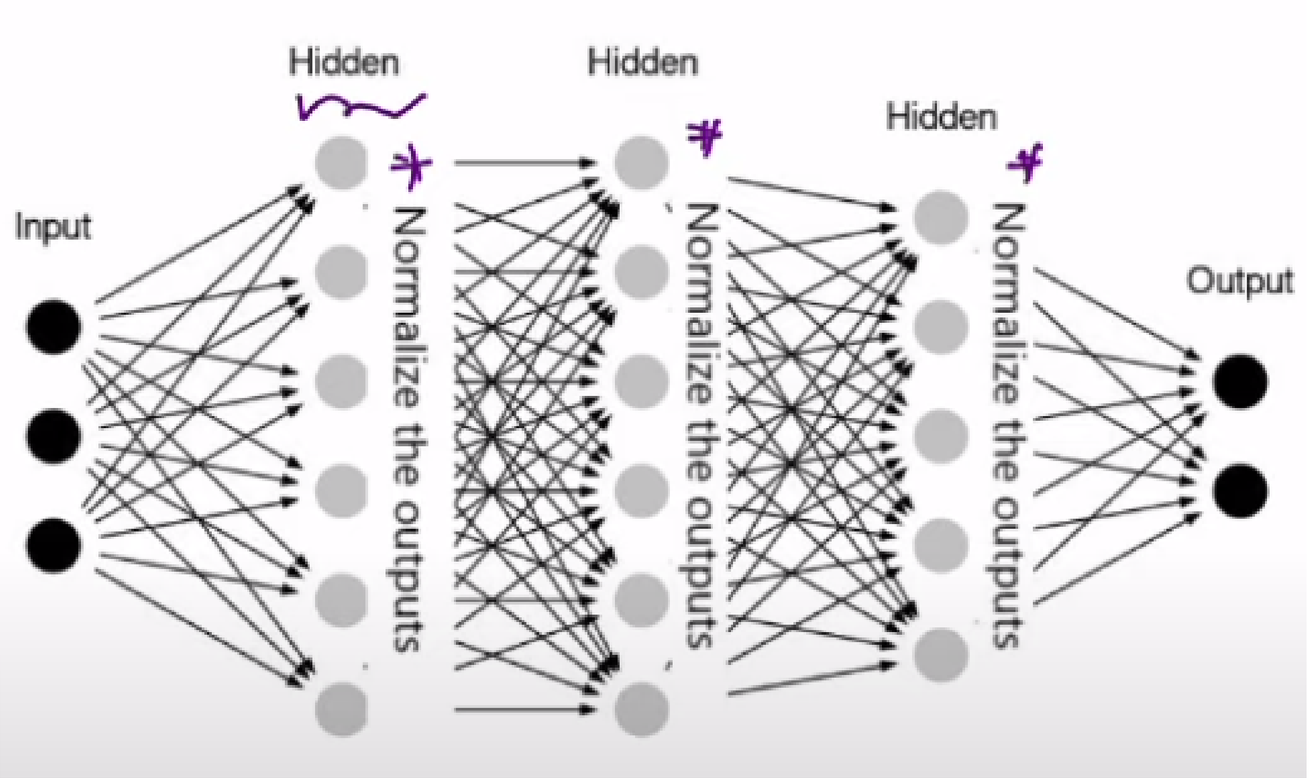
Standardization

Make mean 0 and variance 1.

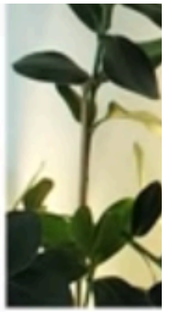


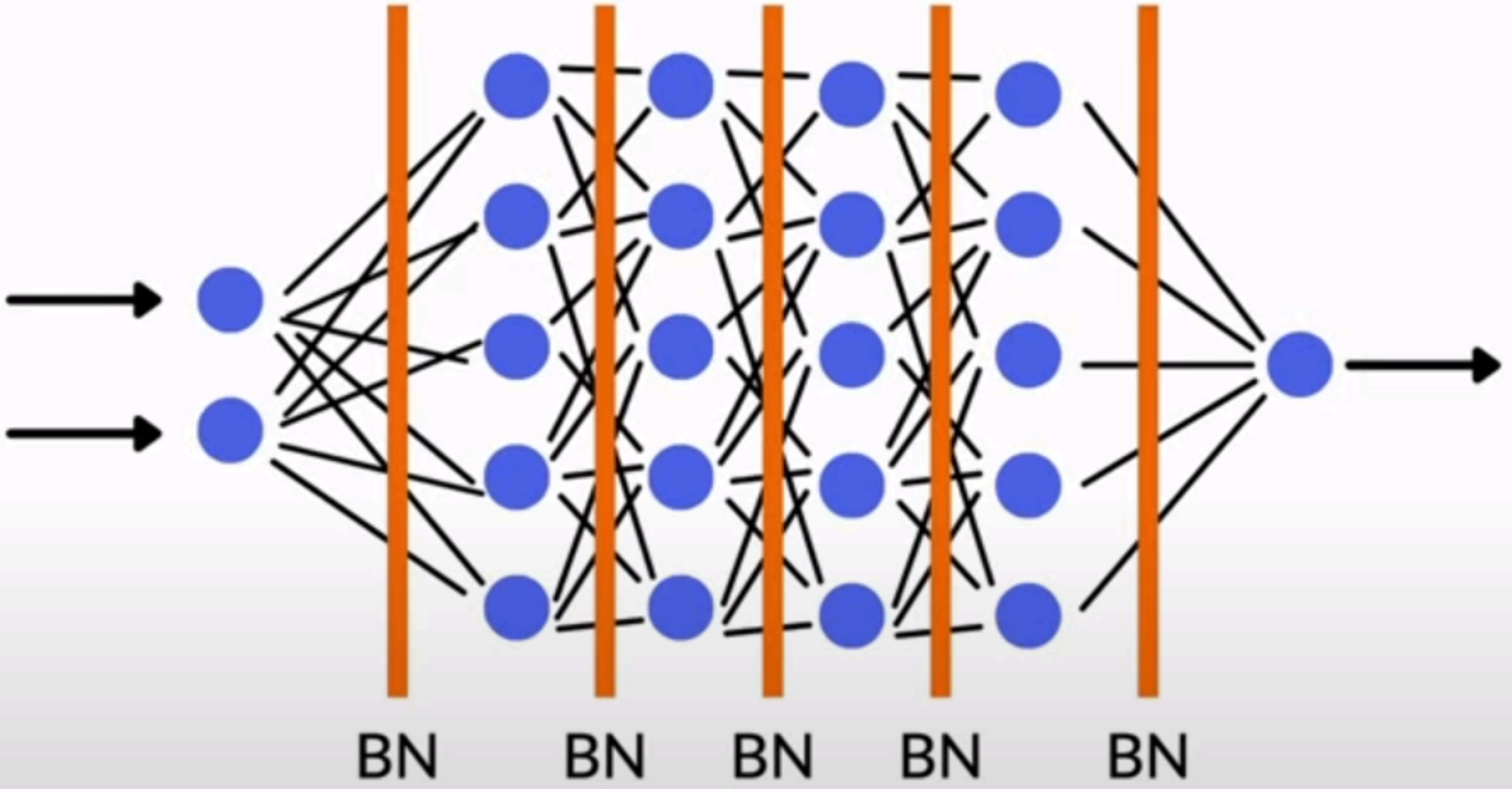
BATCH NORMALIZATION

- Batch normalization is a method used to train Artificial Neural Networks **faster** and more **stable** through adding extra layers in a deep neural network (normalization of the layers' inputs).



Batch Normalization

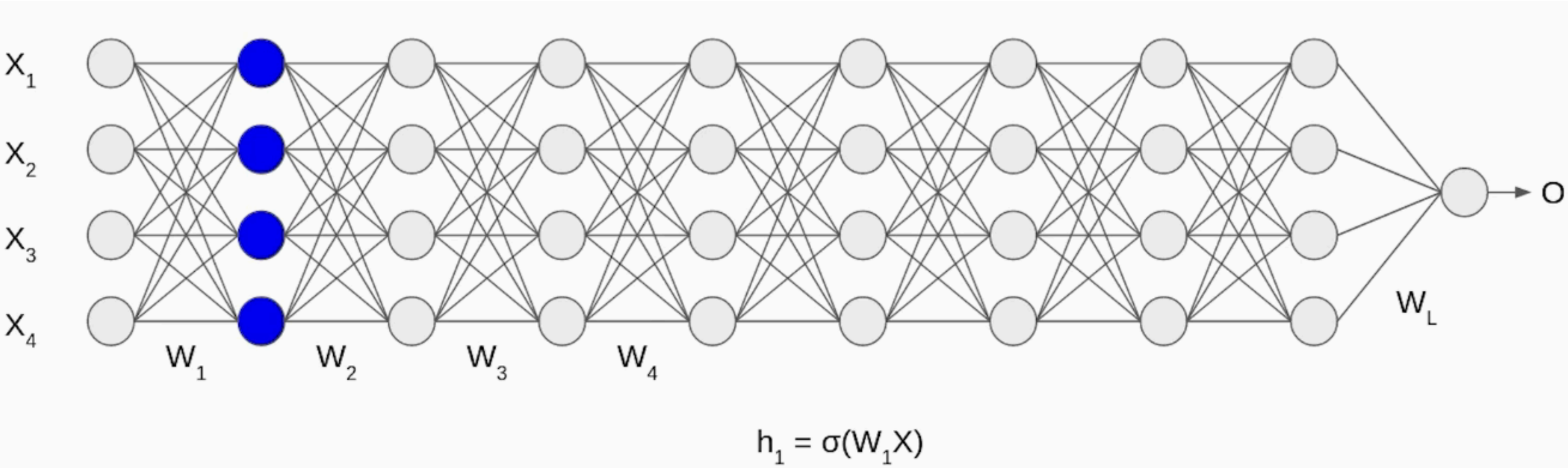




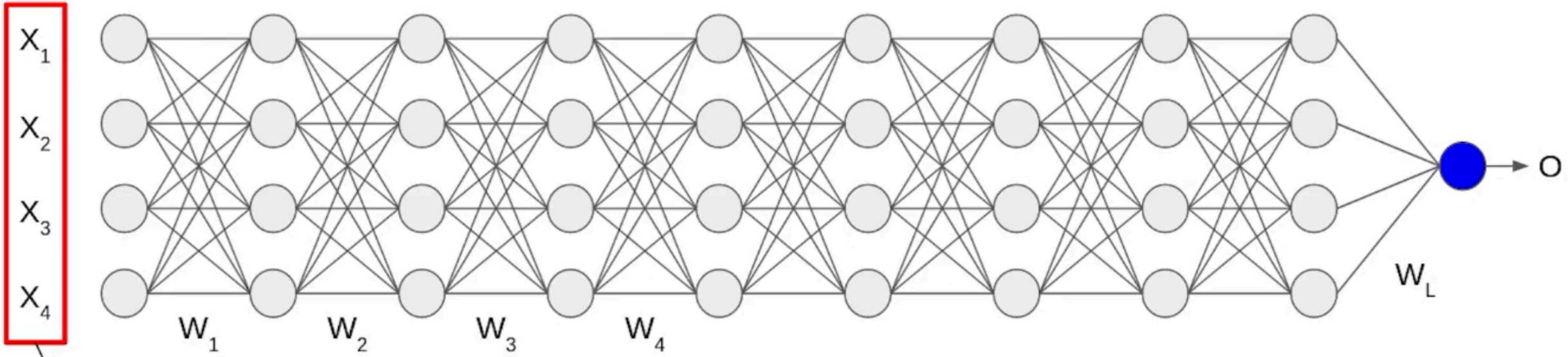
BATCH NORMALIZATION

- The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.
- **“Batch” in batch normalization:**
    - A typical neural network is trained using a collected set of input data called **batch**.
    - Similarly, the **normalizing process** in batch normalization takes place in **batches**, not as a single input.

- Batch normalization is a technique used in neural networks to improve training stability and speed.
- It normalizes the inputs of each layer in a network to have a mean of zero and a standard deviation of one.
- This normalization is applied over batches of data during training.
- Normalization can be done before or after applying activation function



- **Initially, our inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage.**
- **When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.**
- **Similarly, this transformation will take place for the second layer and go till the last layer L.**



Normalize the inputs

$$h_1 = \sigma(W_1 X)$$

$$h_2 = \sigma(W_2 h_1) = \sigma(W_2 \sigma(W_1 X))$$

$$O = \sigma(W_L h_{L-1})$$

How does Batch Normalization work?

- Batch Normalization is a two-step process.
- First, the input is i) normalized, and later ii) rescaling and offsetting are performed.

i) <u>Normalization of the Input:</u>

- Normalization is the process of transforming the data to have a mean zero and standard deviation one.

a) we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \sum h_i$$

Here, m is the number of neurons at layer h.

How does Batch Normalization work?

b. The next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

- **We have the mean and the standard deviation ready.**
- **We will normalize the hidden activations using these values.**
- **For this, we will subtract the mean from each input and divide the whole value with the sum of the standard deviation and the smoothing term (ε).**

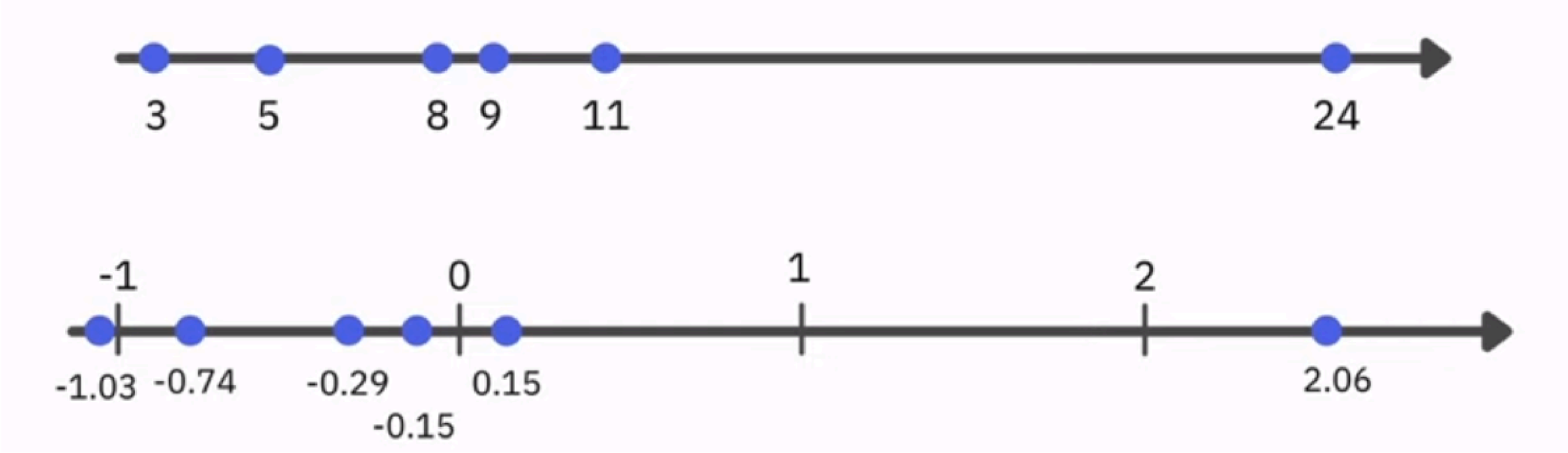
$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

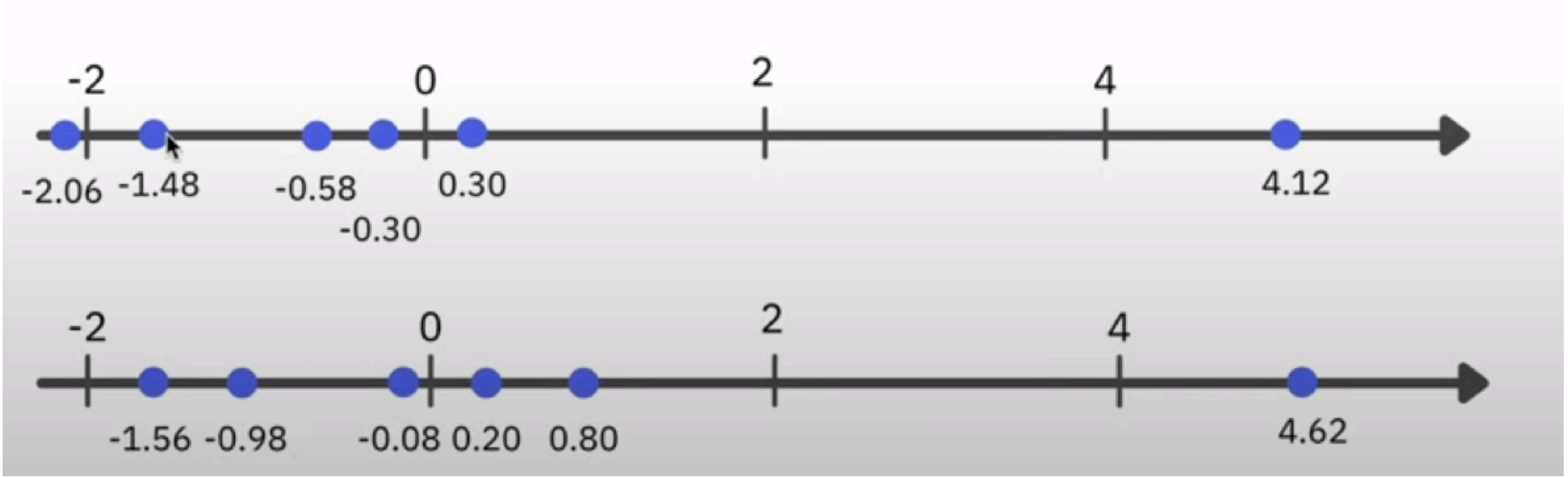
Rescaling & Offsetting

- In the final operation, the re-scaling and offsetting of the input take place.
- Here two components of the BN algorithm come into the picture, γ(gamma) and β (beta).
- These parameters are used for re-scaling (γ) and shifting(β) of the vector containing values from the previous operations.
- These two are learnable parameters, during the training neural network ensures the optimal values of γ and β are used.
- That will enable the accurate normalization of each batch.

$$h_i = \gamma h_{i(norm)} + \beta$$

- **Instead of only normalizing our inputs and feeding the data into our network, with batch normalization, we normalize all the outputs of all the layers in our network.**
- **In between each layer, we have a batch normalization.**



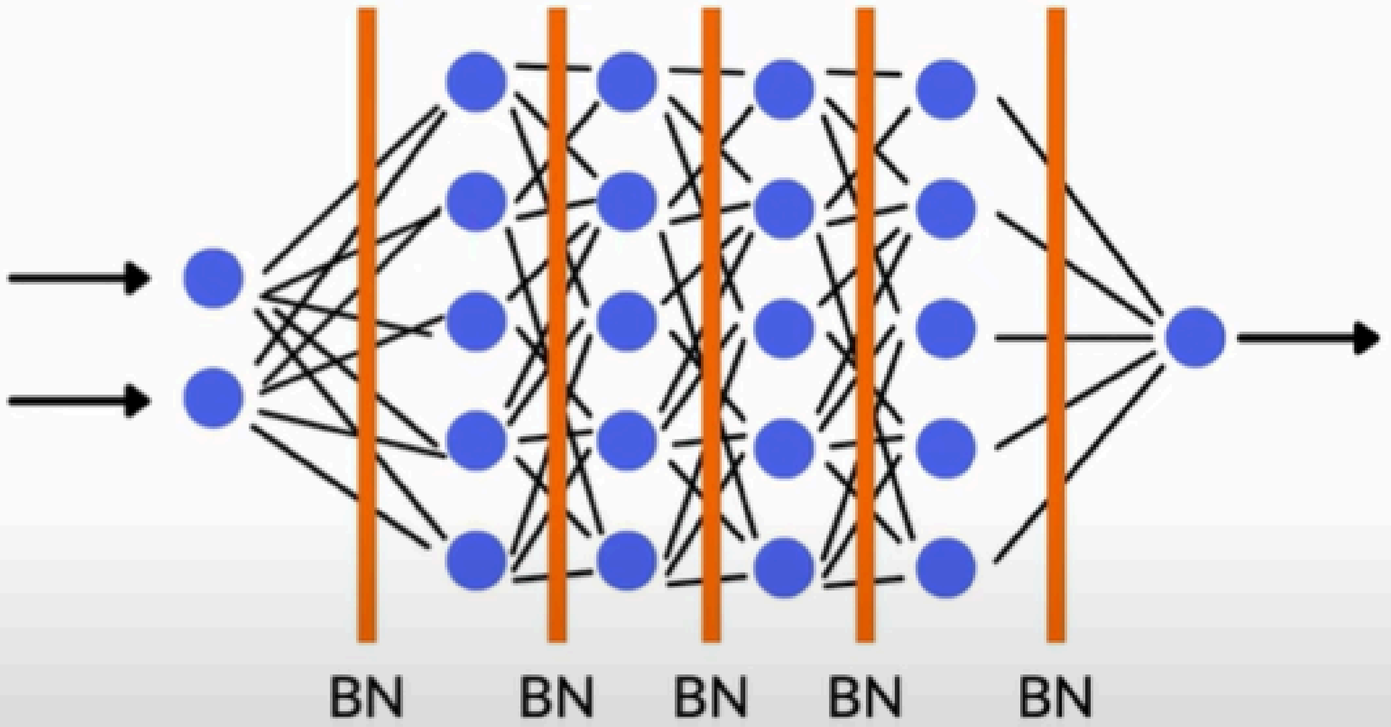


$$z^{(i)} = \gamma \otimes \hat{x}^{(i)} + \beta$$

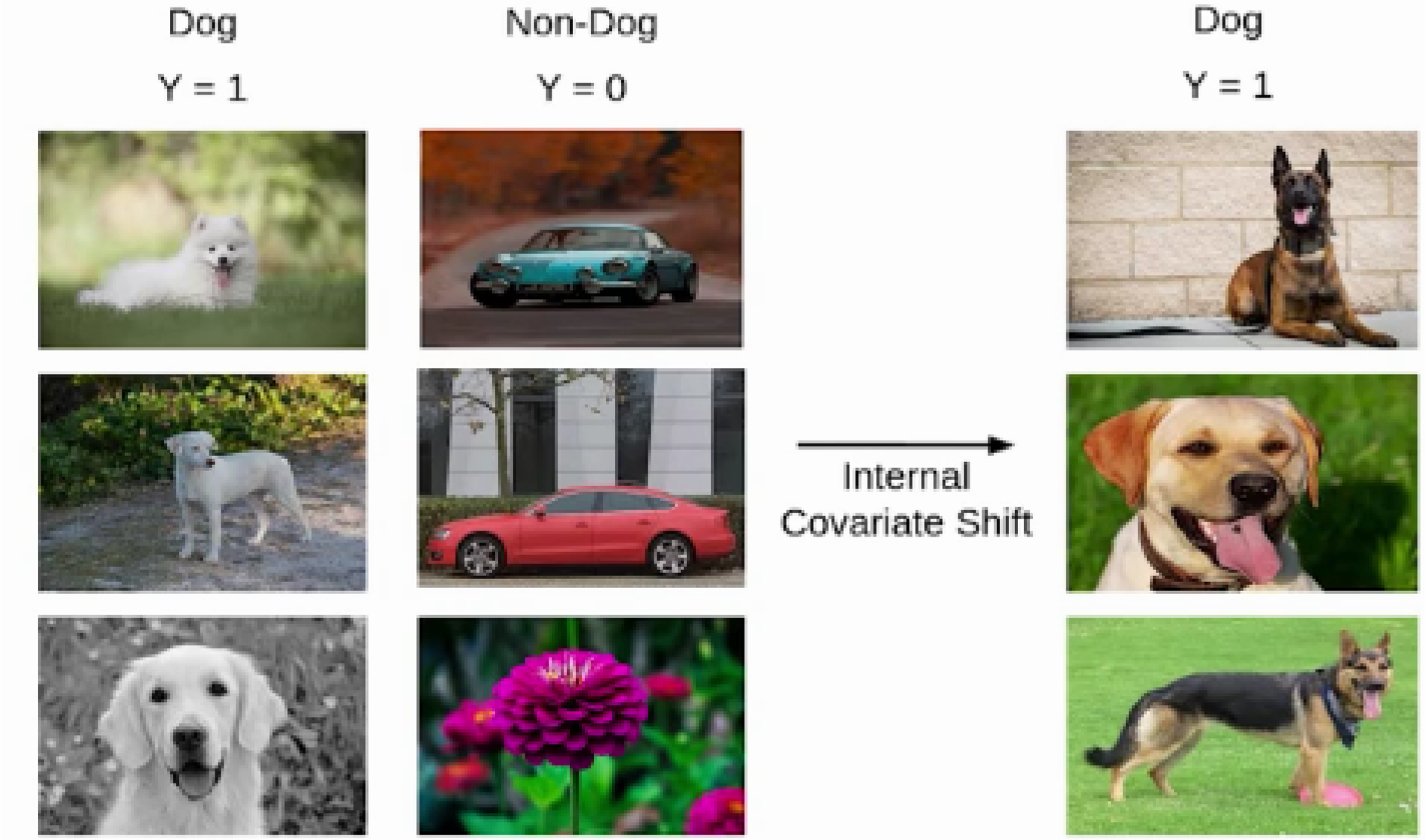
2 0.5

Advantages of Batch Normalization

- Epochs take longer due to the amount of computations but convergence will be faster
- Achieves same accuracy faster
- Can lead to better performance
- No need to have a standardization layer
- Reduces the need for other regularization



Internal Covariate Shift



- Suppose we are training an image classification model, that classifies the images into Dog or Not Dog.
- Let’s say we have the images of white dogs only, these images will have certain distribution as well.
- Using these images model will update its parameters.
- Later, if we get a new set of images, consisting of non-white dogs. These new images will have a slightly different distribution from the previous images.
- Now the model will change its parameters according to these new images.
- Hence the distribution of the hidden activation will also change.
- This change in hidden activation is known as an internal covariate shift.
- Batch normalization solves the problem of Internal Covariate Shift.

REGULARIZATION

Regularization

- Regularization techniques help improve a neural network's generalization ability by reducing overfitting.
- It involves adding a penalty term to the loss function during training.
- This penalty discourages the model from becoming too complex or having large parameter values, which helps in controlling the model’s ability to fit noise in the training data.
- Regularization in deep learning methods include L1 and L2 regularization, dropout, early stopping, and more.
- By applying regularization, models become more robust and better at making accurate predictions on unseen data.

L2 & L1 regularization

- L1 and L2 are the most common types of regularization.
- These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) +
Regularization term

- Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models.
- Therefore, it will also reduce overfitting to quite an extent.

L2 & L1 regularization

- In L2,

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

- **Here, lambda is the regularization parameter.**
- **It is the hyperparameter whose value is optimized for better results.**
- **L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).**

L2 & L1 regularization

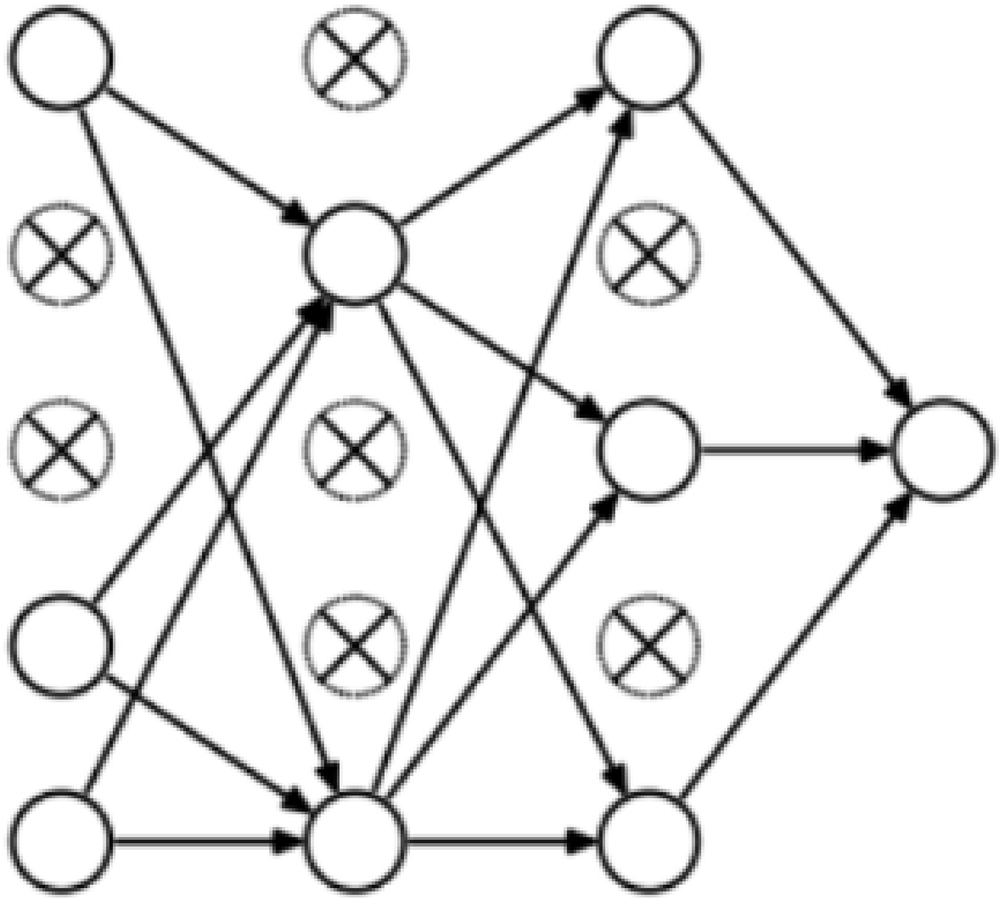
- In L1, $$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

- **Here, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here.**
- **Hence, it is very useful when we are trying to compress our model.**
- **Otherwise, we usually prefer L2 over it.**

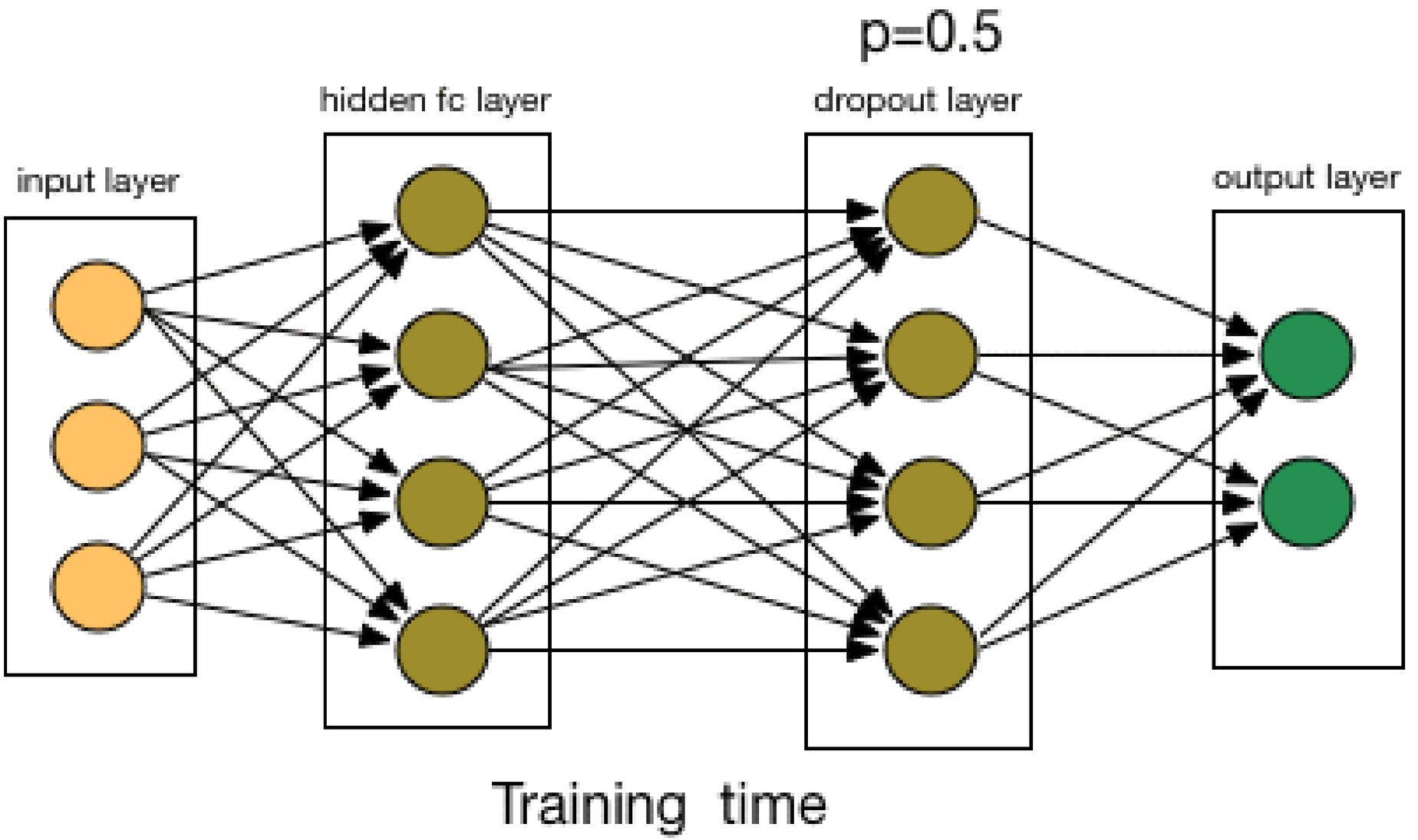
DROP OUT

Drop out

- This is the one of the most interesting and frequently used regularization technique in the field of deep learning.
- At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections.



- This probability of choosing how many nodes should be dropped is the hyperparameter of the dropout function.
- Dropout can be applied to both the hidden layers as well as the input layers.
- Dropout performs better than a normal neural network model.


p=0.5
hidden fc layer
dropout layer
input layer
output layer
Training time