Classification

- **Outline:**

  1. Introduction

  2. Artificial Neural Network (ANN)

Classification

- **Duration:** 8 hrs
- **Outline:**

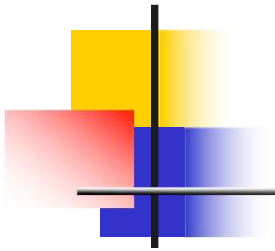
1. Introduction
2. K Nearest Neighbor (KNN)
3. **Artificial Neural Network (ANN)**
4. Support Vector Machine (SVM)

ANN

- Overview of ANN
- Components of ANN
- ANN training (forward and backward propagation)
- ANN characteristics
- ANN design

ANN

- **Overview of ANN**
- Components of ANN
- ANN training (forward and backward propagation)
- ANN characteristics
- ANN design



History

late-1800's - Neural Networks appear as an analogy to biological systems

1960's and 70's – Simple neural networks appear

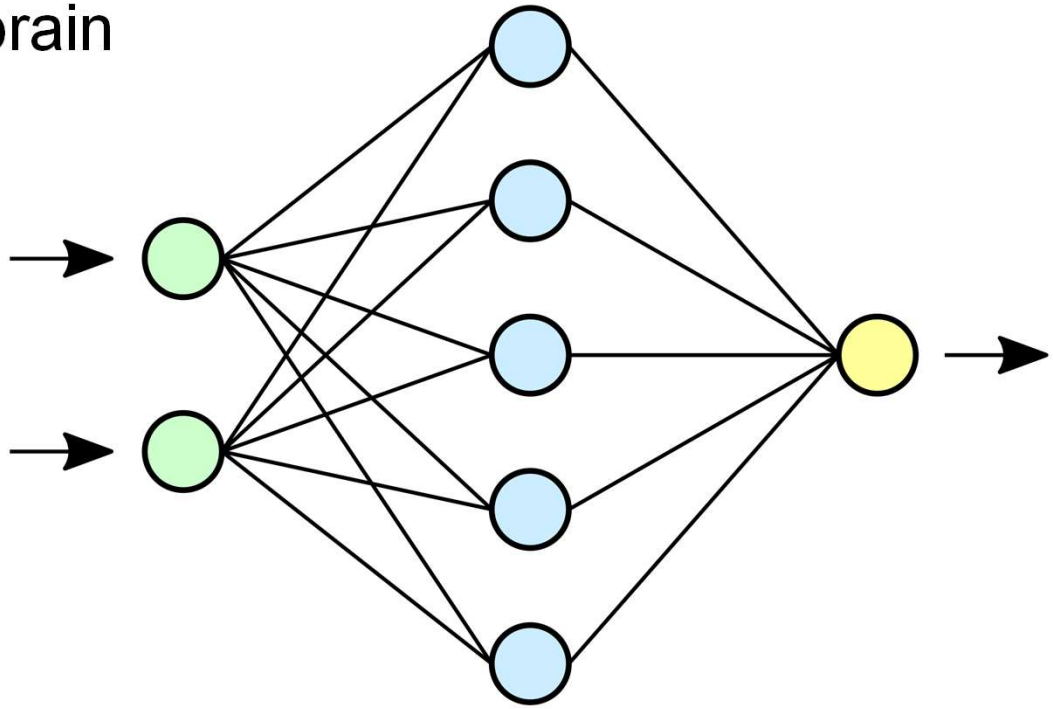
- Fall out of favor because the perceptron is not effective by itself, and there were no good algorithms for multilayer nets

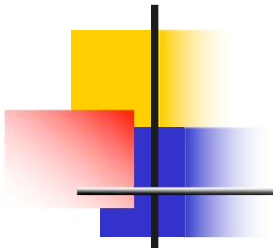
1986 – Backpropagation algorithm appears

- Neural Networks have a resurgence in popularity
- More computationally expensive

What is an ANN?

- Computing system inspired by the biological neural network
- Connection of units (or nodes) called artificial neurons
  - Each node ~ biological neuron
  - Each connection ~ synapse in brain





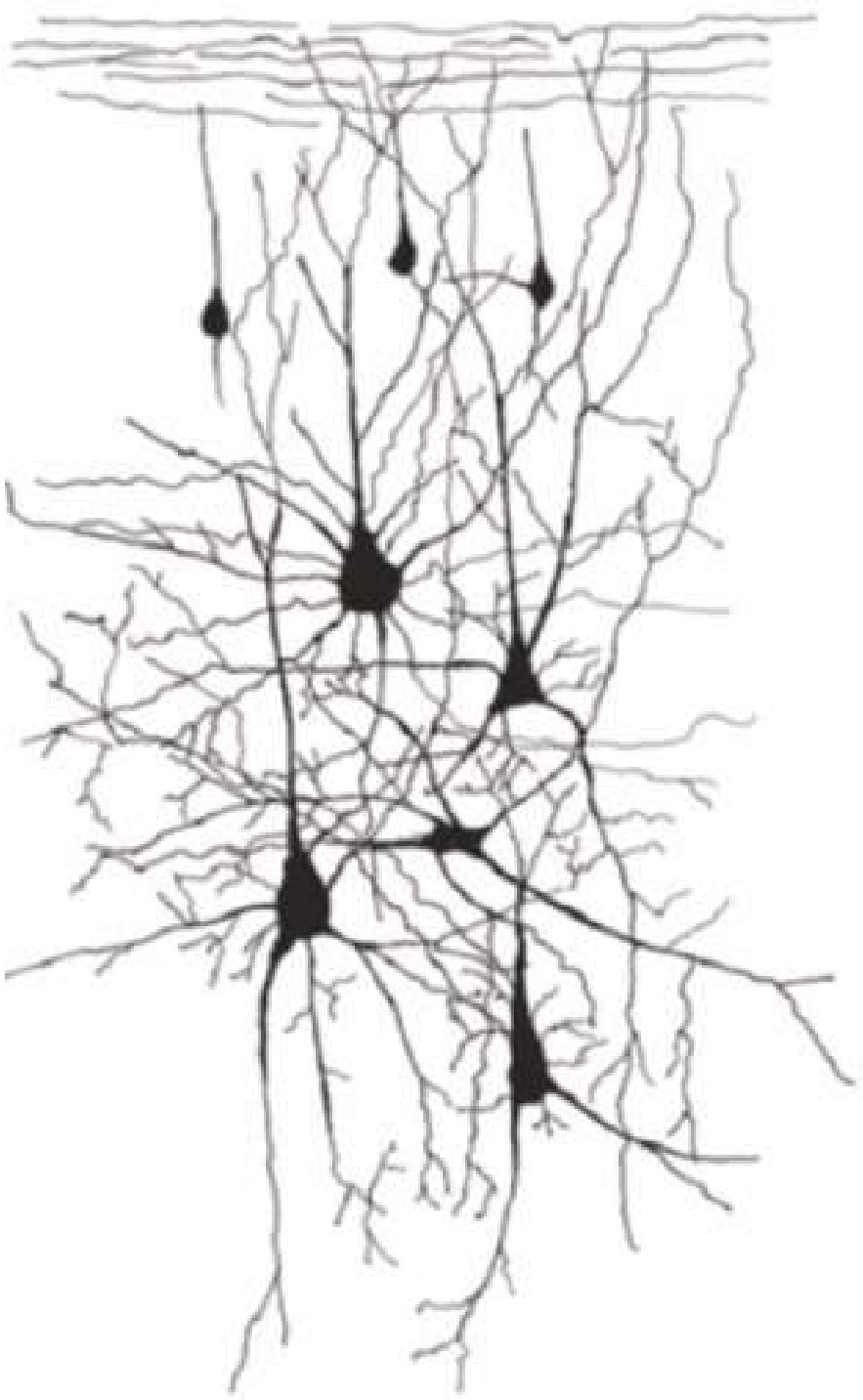
Applications of ANNs

ANNs have been widely used in various domains for:

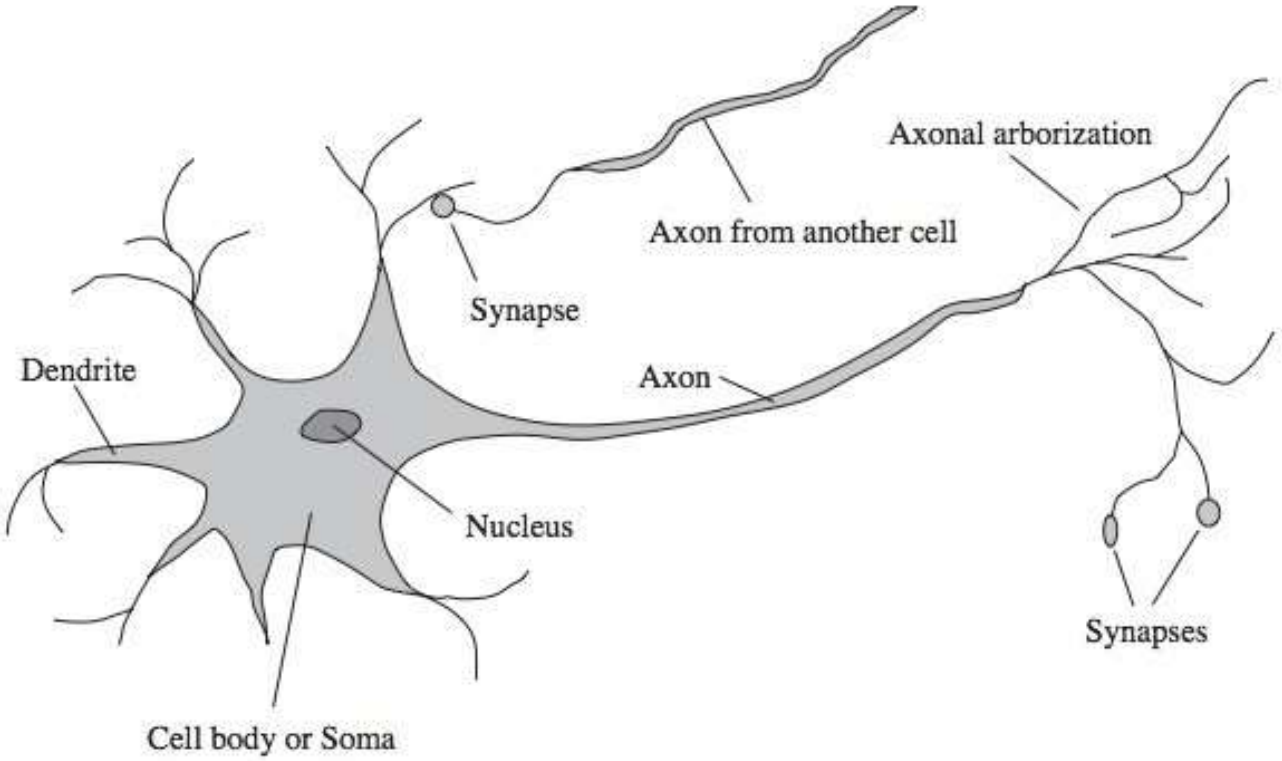
- Pattern recognition
- Function approximation
- Associative memory

Biological neural networks

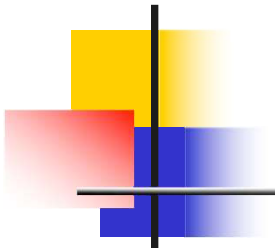
- Our brain consists of ~ 100 billion neurons
- A neuron may connect to as many as 100,000 other neurons
- Signals “move” via electrochemical signals
- **Biological neural network:** interconnected network of billions of neurons with trillions of interconnections between them



Structure of a biological neuron



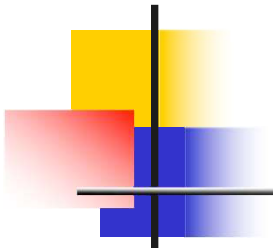
- **Dendrite:** receives signals from other neurons
- **Cell body:** sums all the incoming signals to generate input
- **Axon:** transfers signals to the other neurons when the neuron "fires" (sum reaches a threshold)
- **Synapses:** points of interconnection of one neuron with other neurons



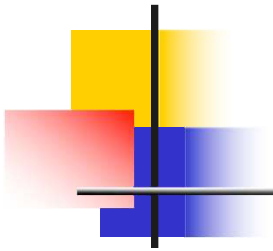
Properties

- Inputs are flexible
  - any real values
  - Highly correlated or independent
- Target function may be discrete-valued, real-valued, or vectors of discrete or real values
  - Outputs are real numbers between 0 and 1
- Resistant to errors in the training data
- Long training time
- Fast evaluation
- The function produced can be difficult for humans to interpret

When to consider neural networks

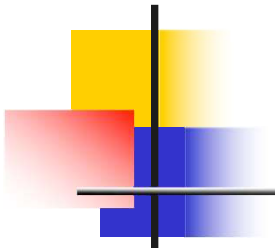


- Input is high-dimensional discrete or raw-valued
- Output is discrete or real-valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of the result is not important
- **Examples:**
- Speech phoneme recognition
- Image classification
- Financial prediction



Artificial Neural Networks

- Computing systems inspired by biological neural networks
- Consists of several processing elements that receive inputs and deliver outputs
- "Learn" to perform tasks by considering examples
- Be able to model nonlinear processes

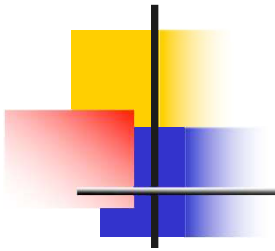


Artificial Neural Networks

- ✓ Applications:
  - ❖ Natural Language Processing
    - Text classification
    - Language generation
    - Document summarization
    - Machine translation
    - Speech recognition
    - Character recognition
    - Spell checking

Artificial Neural Networks

- ✓ Applications:
  - ❖ System identification and control
    - Vehicle control
    - Process control
    - Auto-piloting
    - Autonomous driving cars
    - Robot navigation
    - Component fault detection and simulation
    - Chip failure analysis
    - Trajectory prediction
    - Petroleum exploration



Artificial Neural Networks

- ✓ Applications:
  - ❖ Time series forecasting and prediction
    - ■ Weather forecasting
    - ■ Stock market prediction
    - ■ Cost models
  - ❖ Medical
    - ■ Cancers detection
    - ■ Medical image analysis
    - ■ EEGs
    - ■ Drug design

Artificial Neural Networks

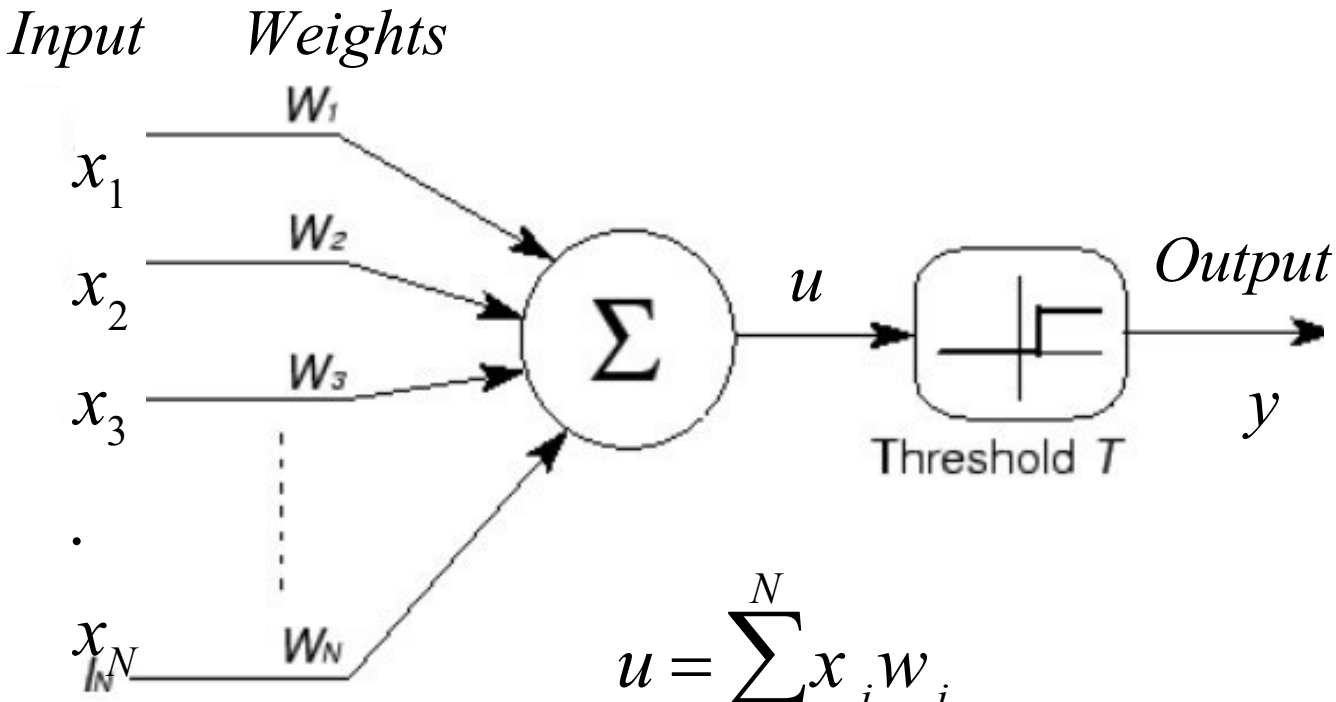
- ✓ Applications:
  - ❖ Image/video/audio analysis
    - ■ Image recognition
    - ■ Image compression
    - ■ Radar and sonar image classification
    - ■ Signature verification

Artificial Neural Networks

- Applications:
  - Education:
    - Adaptive learning software
    - Student performance modeling
    - Personality profiling
  - Bussiness analytics
    - Customer behavior modeling
    - Customer segmentation
    - Market research
  - Banking
    - Credit and loan application evaluation
    - Fraud and risk evaluation

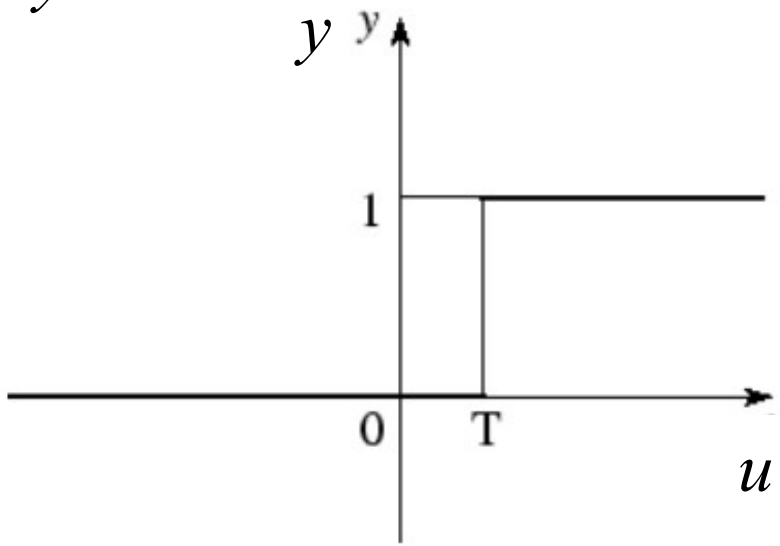
McCulloch and Pitts neuron model (1943)

- A mathematic computing paradigm that models the human neuron



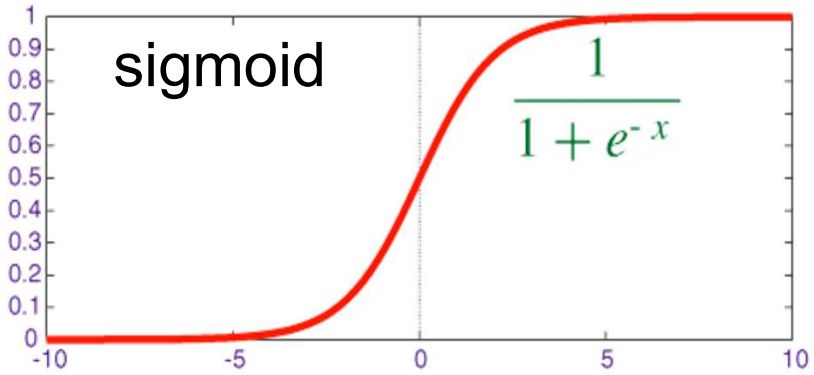
$$u = \sum_{j=1}^{N} x_j w_j$$

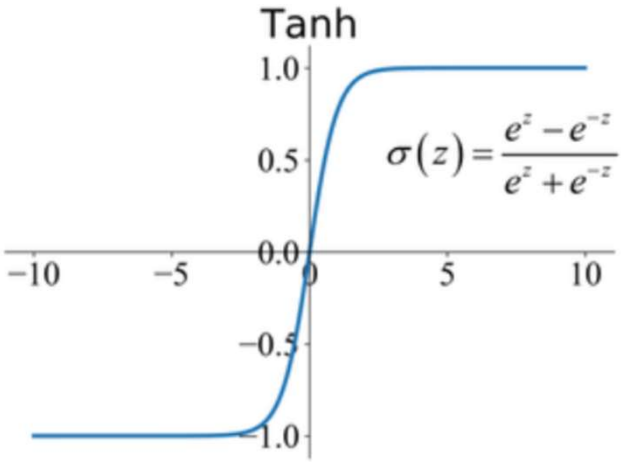
$$y = f(u)$$

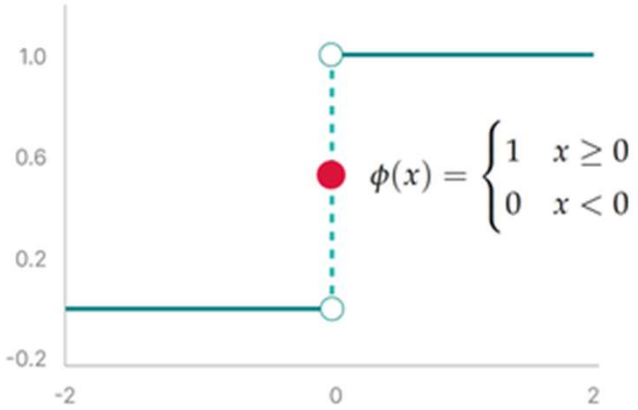


Artificial Neural Networks

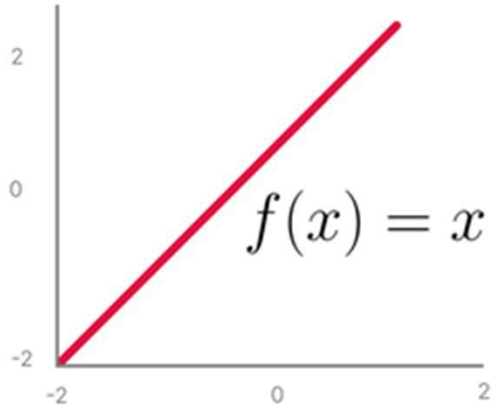
- ✓ Activation function







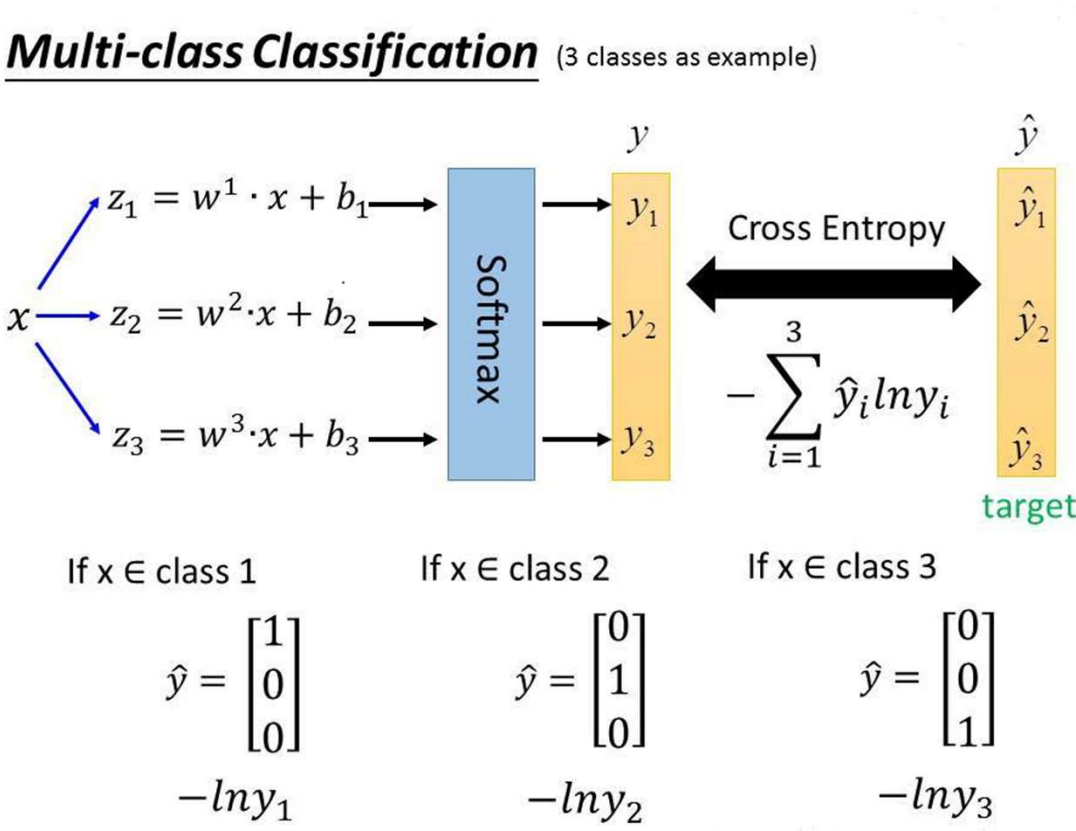
(a) Binary-step function

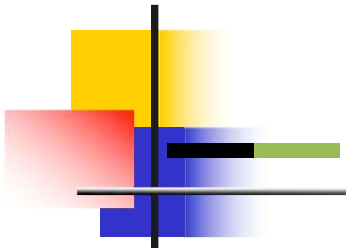


(b) Linear function

Softmax Function

Softmax function calculates the probabilities distribution of the event over ‘n’ different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes.





Pros And Cons of Softmax Activation function

Pros

1. It mimics the one hot encoded labels better than the absolute values.
2. If we use the absolute (modulus) values we would lose information, while the exponential intrinsically takes care of this.

Cons

1. The softmax function should not be used for multi-label classification.
2. the sigmoid function (discussed later) is preferred for multi-label classification.
3. The Softmax function should not be used for a regression task as well.

Softmax function

- Ordinary classifier:
  - Some rule, or function, that assigns to a sample x a class $\hat{y}$ label

$$\hat{y} = f(x)$$

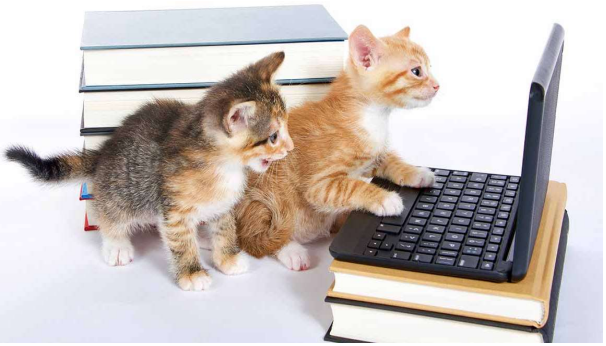
- Probabilistic classifier: $P(\mathbf{Y} \mid \mathbf{X})$
  - Conditional distributions $y \in \mathbf{Y}$: for a given $x \in \mathbf{X}$, assign probabilities to all (these probabilities sum to one).

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0.20 \\ 0.35 \\ 0.05 \\ 0.40 \end{bmatrix}$$

Softmax function

- Example: Image classification problem
  - 4 categories: dog, cat, kite, car
  - One-hot encoding

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$



Input → Black → Output

$$\mathbf{z} = \begin{bmatrix}z_1\\z_2\\z_3\\z_4\end{bmatrix} = \begin{bmatrix}0.8\\1.4\\-0.8\\0.2\end{bmatrix}$$

Softma x →

$$\mathbf{y} = \begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix} = \begin{bmatrix}0.28\\0.51\\0.06\\0.15\end{bmatrix}$$

Softmax function

- Softmax function:
  - Takes as input a vector of *k* real numbers
  - Normalizes it into a probability distribution consisting of *k* probabilities proportional to the exponentials of the input numbers

$$\begin{bmatrix} 0.8 \\ 1.4 \\ -0.8 \\ 0.2 \end{bmatrix}$$

$$\begin{bmatrix} 0.28 \\ 0.51 \\ 0.06 \\ 0.15 \end{bmatrix}$$

Softmax function

- Input:
  - Some vector components could be negative, or greater than one
  - Might not sum to 1

$$\begin{bmatrix} 0.8 \\ 1.4 \\ -0.8 \\ 0.2 \end{bmatrix}$$

- Output:
  - Each component in the interval (0,1)
  - The components add up to 1
  - Larger input components correspond to larger probabilities
  - Can be interpreted as probabilities

$$\begin{bmatrix} 0.28 \\ 0.51 \\ 0.06 \\ 0.15 \end{bmatrix}$$

Softmax function

- Softmax function:

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \forall i = 1, 2, \ldots, C$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.4 \\ -0.8 \\ 0.2 \end{bmatrix} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0.28 \\ 0.51 \\ 0.06 \\ 0.15 \end{bmatrix}$$

Softmax function

Difference Between Sigmoid Function and Softmax Function

	Softmax Function	Sigmoid Function
1	Used for multi-classification in logis tic regression model.	Used for binary classification in logistic regression model.
2	The probabilities sum will be 1	The probabilities sum need not be 1.
3	Used in the different layers of neur al networks.	Used as activation function while buil ding neural networks.
4	The high value will have the higher probability than other values.	The high value will have the high prob ability but not the higher probability.

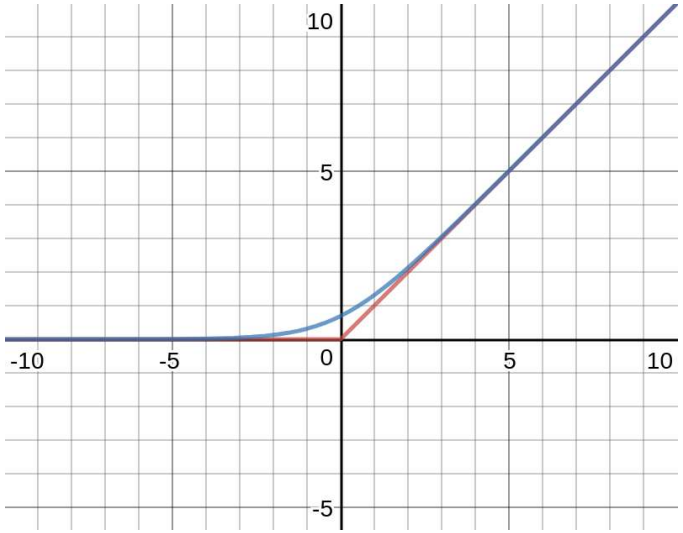
$$softmax(x) = \frac{x_i}{\sum_{j=0}^{k} e^{x_j}} \ (i = 0, 1, \ldots k)$$

$$sigmoid(x_i) = \frac{1}{1 + e^{(-x_i)}}$$

Softmax function

- Where comes the name “softmax”?
  - Smooth approximation to the maximum function?
  - Smooth approximation to the arg max function

softargmax



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

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.4 \\ -0.8 \\ 0.2 \end{bmatrix} \qquad \arg\max(\mathbf{z}) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0.28 \\ 0.51 \\ 0.06 \\ 0.15 \end{bmatrix}$$

The Nonlinear Activation Functions are mainly divided on the basis of their range or curves-

Sigmoid or Logistic Activation Function

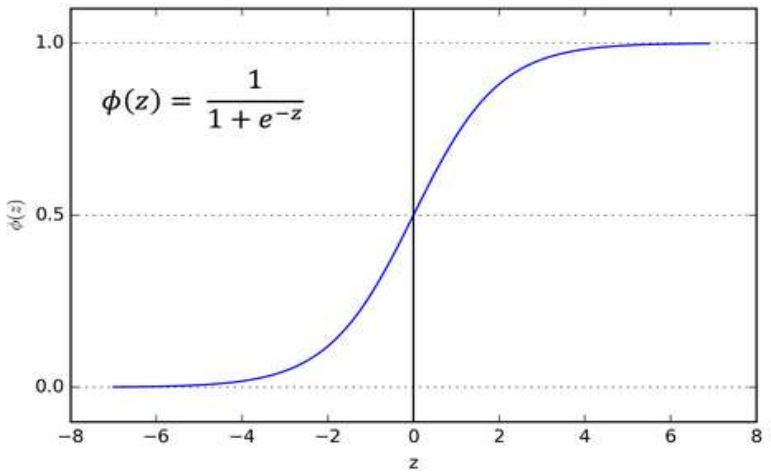
The Sigmoid Function curve looks like a S-shape.

The main reason why we use sigmoid function is because it exists between **(0 to 1).** Therefore, it is especially used for models where we have to **predict the probability** as an output.Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

The beauty of an exponent is that the value never reaches zero nor exceed 1 in the above equation.

The large negative numbers are scaled towards 0 and large positive numbers are scaled towards 1.



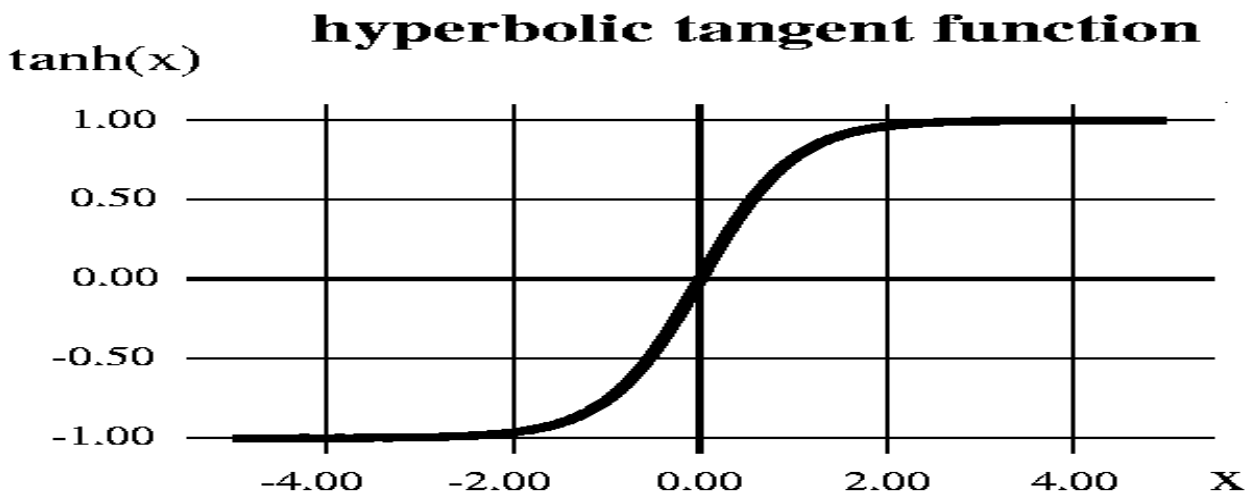
Hyperbolic Tangent function- Tanh :

It's mathamatical formula is

f(x) = 1—exp(-2x) / 1 + exp(-2x).

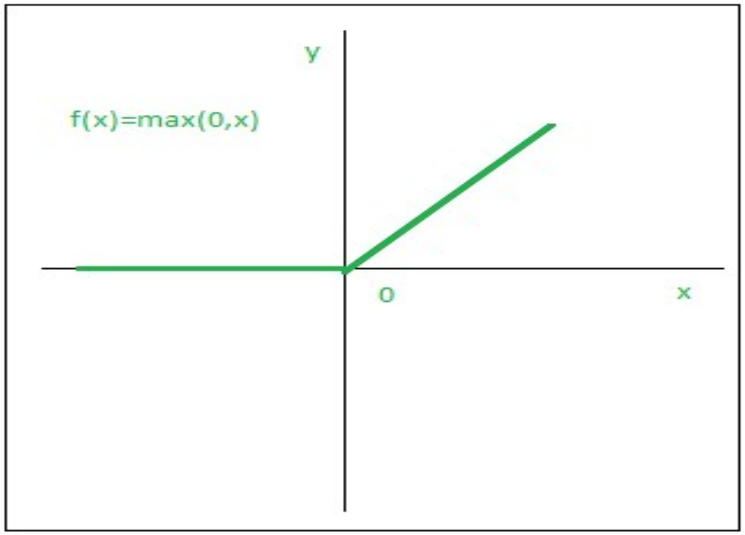
Now it's output is zero centered because its range in between -1 to 1 i.e -1 < output < 1 . Hence optimization is *easier* in this method hence in practice it is always preferred over Sigmoid function .

But still it suffers from Vanishing gradient problem.



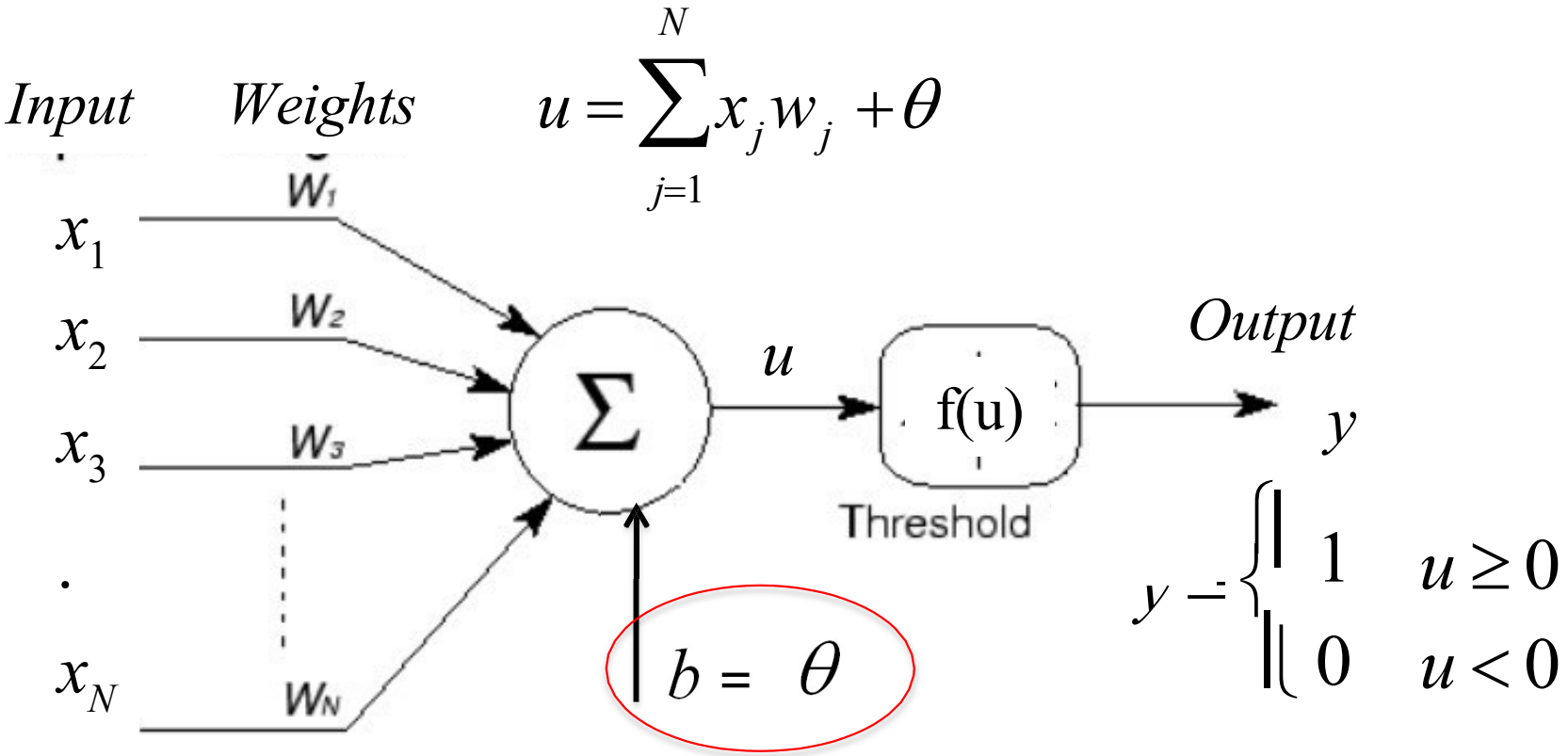
ReLu- Rectified Linear units

It was recently proved that it had *6 times improvement in convergence* from Tanh function.
R(x) = max(0,x) i.e if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x.
Hence it avoids and rectifies **vanishing gradient** problem.
But its limitation is that it should only be used within Hidden layers of a Neural Network Model.
Another problem with ReLu is that some gradients can be fragile during training and can die.
It can cause a weight update which will makes it never activate on any data point again.
Simply saying that ReLu could result in Dead Neurons**.**

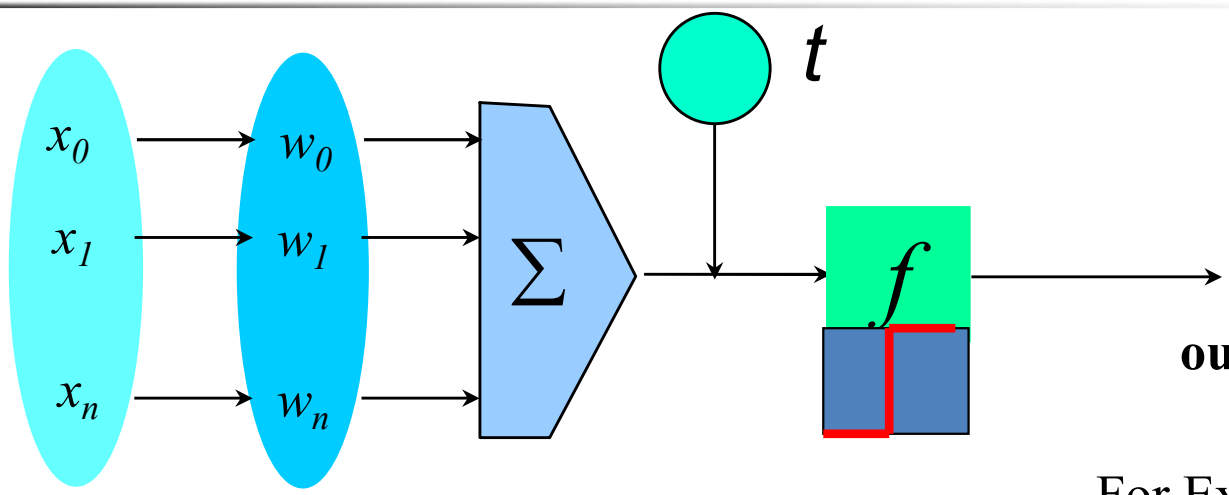


Perceptron neuron model

- An enhanced version of McCulloch-Pitts model:
  - Merge Hebbian learning rule of adjusting weights
  - Add bias



A Neuron (= a perceptron)



Input vector x **weight vector w** **weighted sum** **Activation function**

For Example

$$y = \text{sign}(\sum_{i=0}^{n} w_i x_i - t)$$

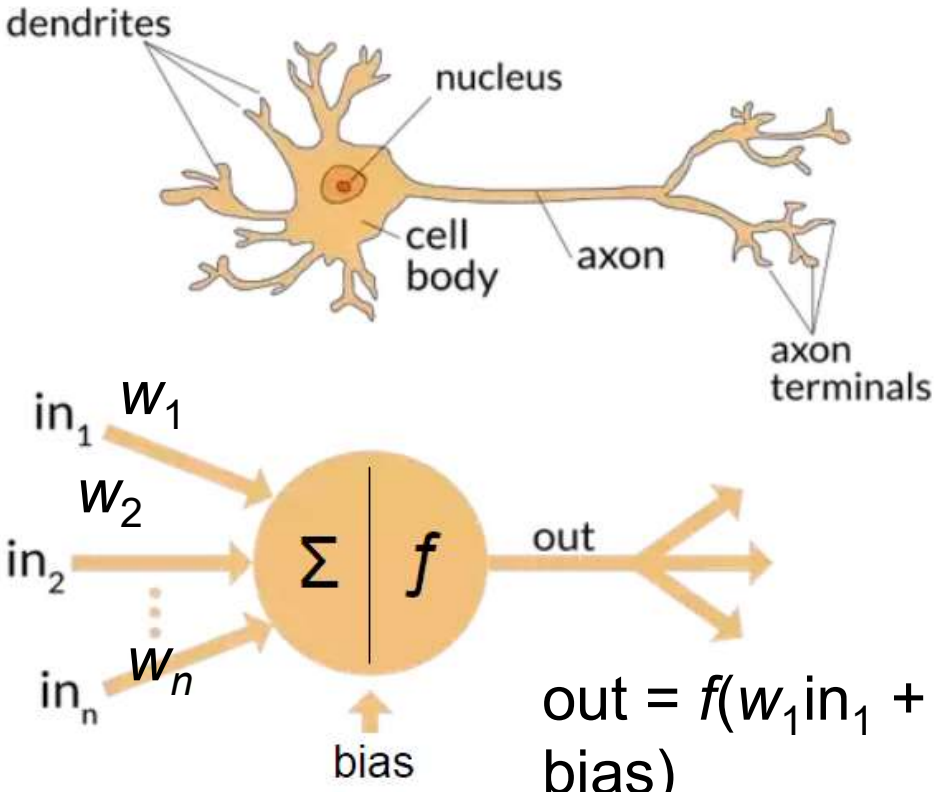
The *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

ANN

- Overview of ANN
- **Components of ANN**
- ANN training (forward and backward propagation)
- ANN characteristics
- ANN design

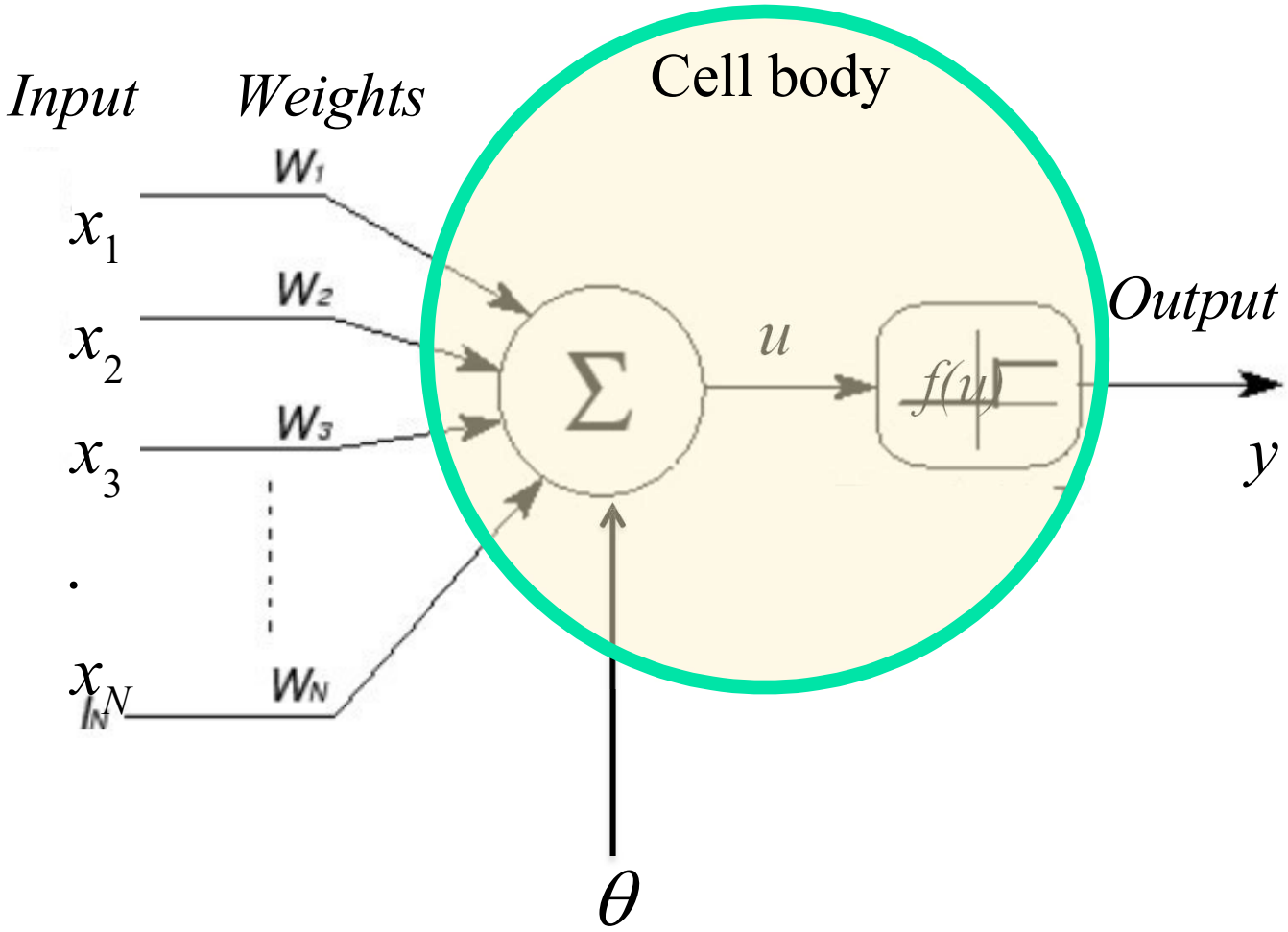
Artificial Neural Networks

- ✓ Artificial neural network



$$out = f(w_1 in_1 + w_2 in_2 + \ldots + w_n in_n + bias)$$

General neuron model



$\{w_j;\ 1 \le j \le N\}$: *synaptic weights*

θ : *threshold*

Net function

$$u = \sum_{j=1}^{N} x_j w_j + \theta$$

Activation function

$$y = f(u)$$

Ex:

$$y = f(u) = \frac{1}{1+e^{-u}}$$

Popular net functions

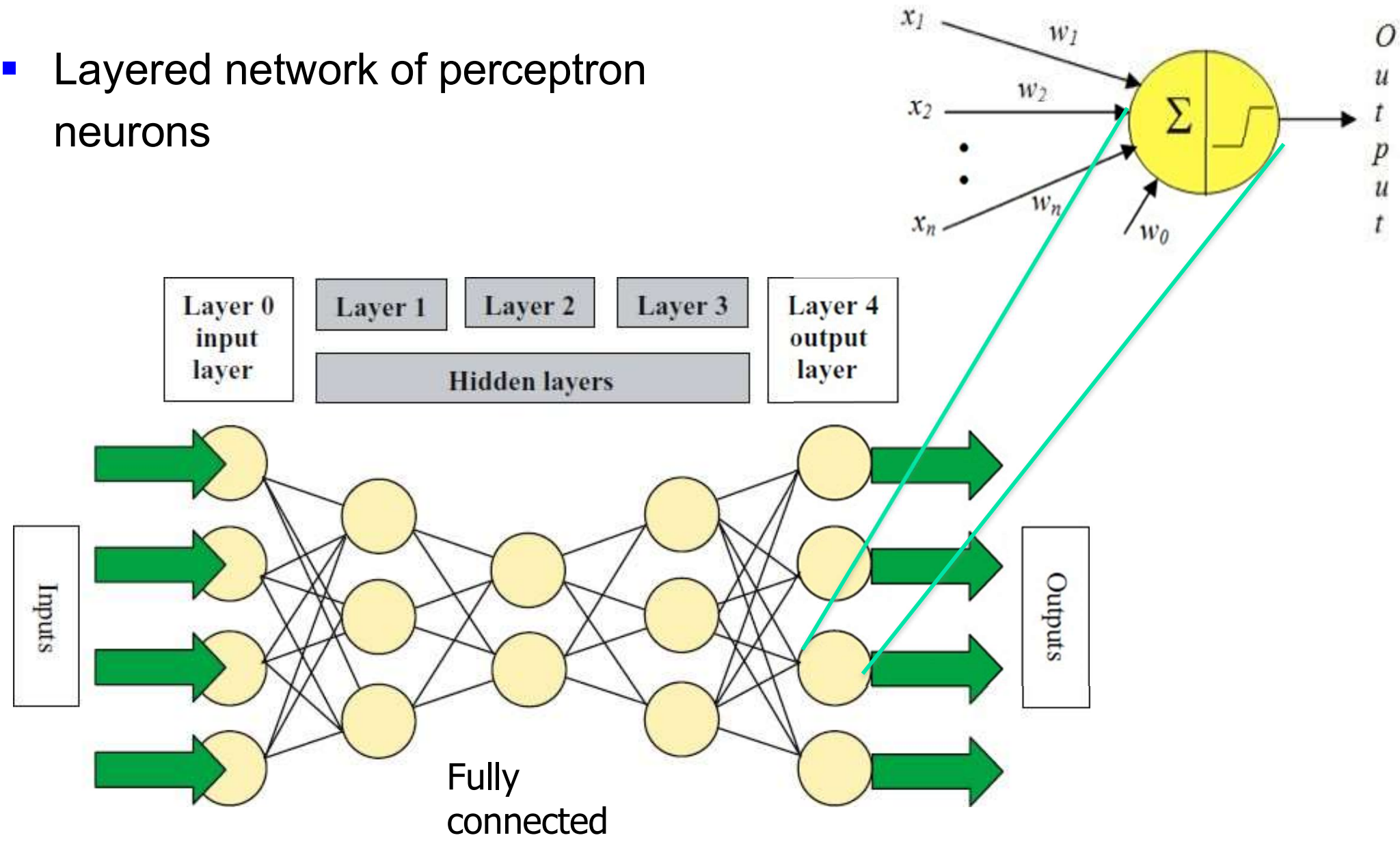
Net functions	*Formula*	*Comments*
Linear	$u = \sum_{j=1}^{N} w_j y_j + \theta$	Most commonly used.
Higher order (2nd order formula exhibited)	$u = \sum_{j=1}^{N} \sum_{k=1}^{N} w_{jk} y_j y_k + \theta$	u_i is a weighted linear combination of higher order polynomial terms of input variable. The number of input terms equals to N^d where d is the order of the polynomial.
Delta (Σ-Π)	$u = \prod_{j=1}^{N} w_j y_j$	Seldom used

Popular activation functions

Activation function	*Formula* $a = f(u)$	*Derivatives* $\frac{df(u)}{du}$	*Comments*
Sigmoid	$f(u) = \frac{1}{1+e^{-u/T}}$	$f(u)[1-f(u)]/T$	Commonly used. Derivative can be computed from *f(u)* directly. T: temperature parameter.
Hyperbolic tangent	$f(u) = \tanh\left(\frac{u}{T}\right)$	$(1-[f(u)]^2)/T$	
Inverse tangent	$f(u) = \frac{2}{\pi}\tan^{-1}\left(\frac{u}{T}\right)$	$\frac{2}{\pi T}\cdot\frac{1}{1+(u/T)^2}$	Less frequently used.
Threshold	$f(u) = \begin{cases} 1 & u > 0; \\ -1 & u < 0. \end{cases}$	Derivatives does not exist at $u = 0$.	
Gaussian radial basis	$f(u) = \exp[-\|u-m\|^2/\sigma^2]$	$-2(u-m)\cdot f(u)/\sigma^2$	Used for radial basis neural network. m and σ^2 are parameters to be specified.
Linear	$f(u) = au + b$	a	

Multilayer perceptron model (MLP)

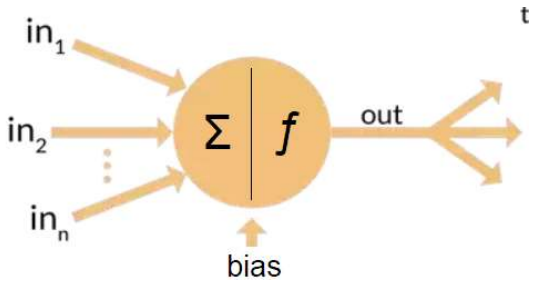
- Layered network of perceptron neurons

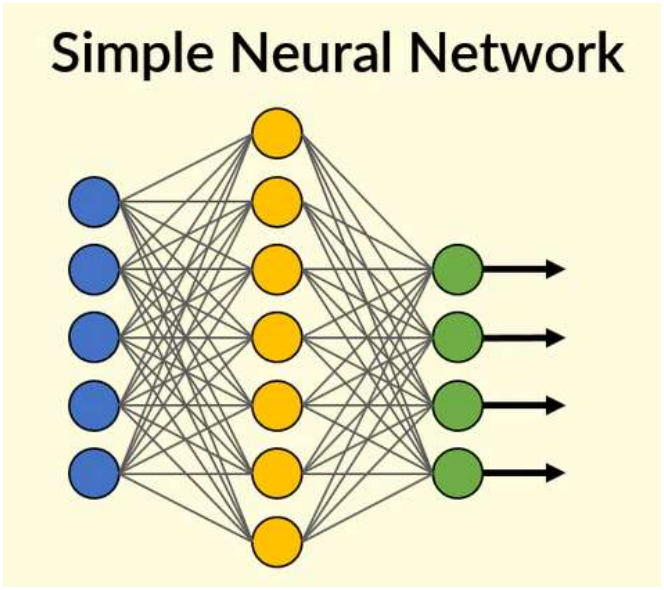


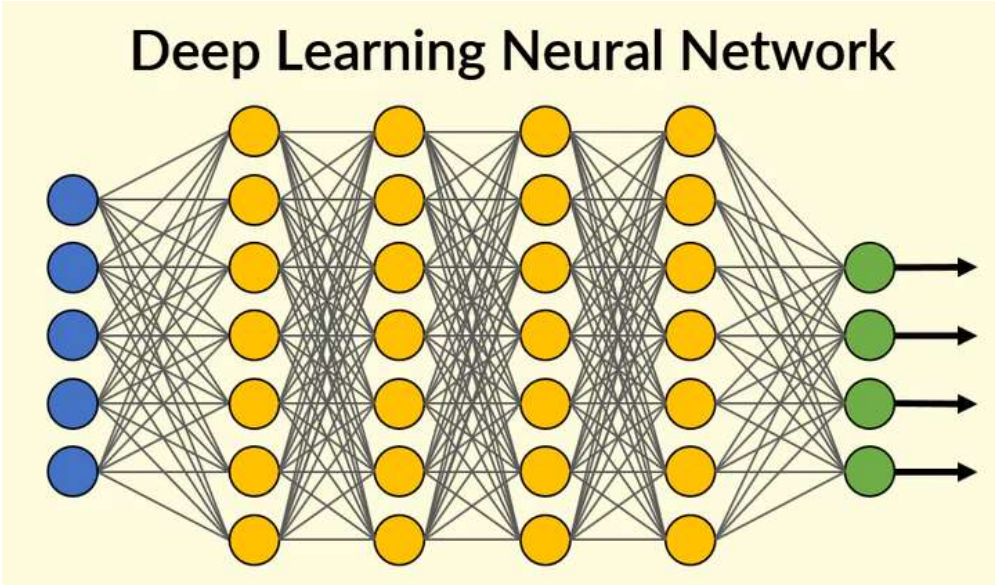
ANN

- Overview of ANN
- Components of ANN
- **ANN training (forward and backward propagation)**
- ANN characteristics
- ANN design

Artificial Neural Networks



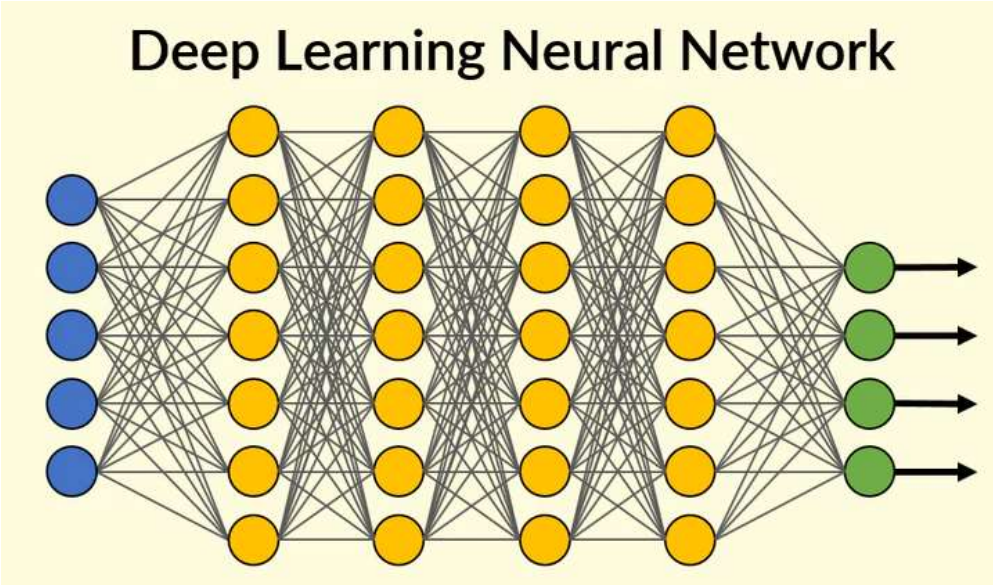




Input Layer Hidden Layer Output Layer

Artificial Neural Networks

- ✓ Example: self-driving car
  - ❖ Input (raw data/features)
  - ❖ Output (steering angle, throttle, brake)
  - ❖ Hidden layers



Artificial Neural Networks

- ✓ Useful links
  - Link 1
  - Link 2
  - Link 3

ANN training process?

- **Calibrating all of the weights** by repeating forward-backward propagation steps until the output is predicted accurately
- **Forward propagation:**
  - Applying a set of weights to the input data
  - Calculating the output
- **Backward propagation:**
  - Measuring the error of the output (difference between desired output and actual output)
  - Adjusting the weights to decrease the error in the next step

ANN training example – epoch 1

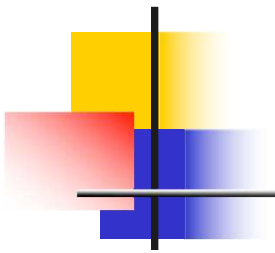
INPUT DATA	DESIRED VALUE	ACTUAL OUTPUT
	0	0.21
	0	0.156
	1	0.78
	1	0.83

ANN training example – epoch 2

INPUT DATA	DESIRED VALUE	ACTUAL OUTPUT
	0	0.194
	0	0.143
	1	0.802
	1	0.895

ANN training example – epoch n

INPUT DATA	DESIRED VALUE	ACTUAL OUTPUT
	0	0.119
	0	0.056
	1	0.884
	1	0.926

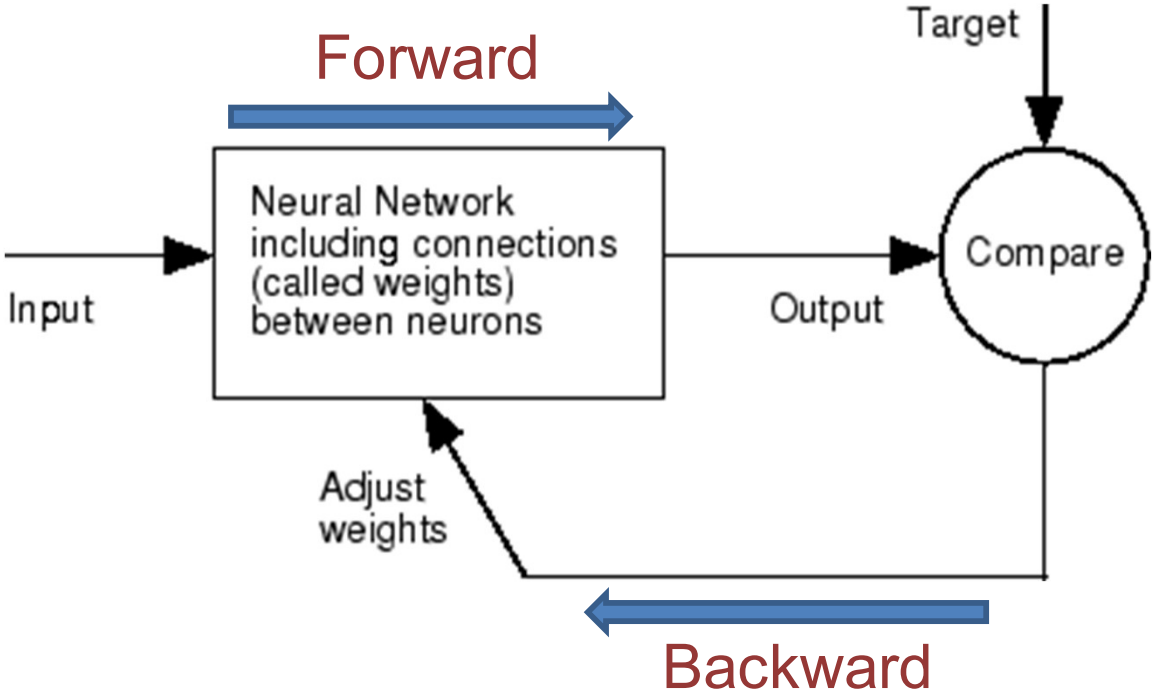


Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights (to small random #s) and biases in the network
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Artificial Neural Networks

- ✓ Error back propagation
  - ❖ Framework



  - ❖ Loss function:
    - ➢ Squared L^2 norm
    - ➢ Angle between 2 vectors
    - ➢ Cross entropy

L1/L2 Regularization

L2 adds "***squared magnitude***" of coefficient as penalty term to the loss function.

$$Loss = Loss + \lambda \sum \beta^2$$

L1 adds "***absolute value of magnitude***" of coefficient as penalty term to the loss function.

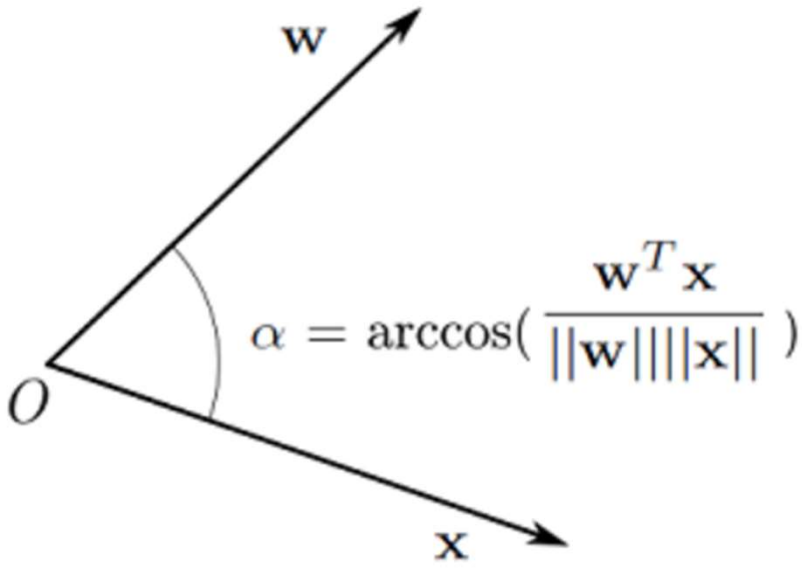
$$Loss = Loss + \lambda \sum |\beta|$$

Weight Penalties → Smaller Weights → Simpler Model → Less Overfit

Artificial Neural Networks

- Error back propagation
  - Loss function:
    - Euclidean norm (2-norm)

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}$$



    - Angle between 2 vectors

    - Cross entropy

$$H(p, q) = -\sum_i p_i \log q_i$$

Error back propagation learning

- Step 1: initialization
- Step 2: output calculating
- Step 3: error calculating

$$E = \sum_{k=1}^{K} [e(k)]^2 = \sum_{k=1}^{K} [d(k) - z(k)]^2$$

d – desired output values (target)

z – actual outputs

- Step 4: weight updating then go back to step (2) until the stop condition is satisfied

Weight updating

- To achieve the minimum error $\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta\mathbf{W}(t)$

Algorithm	$\Delta \boldsymbol{W}(t)$	*Comments*
Steepest descend gradient method	$= -\eta\, \boldsymbol{g}(t) = -\eta\ dE/d\boldsymbol{W}$	$\boldsymbol{g}$ *is known as the gradient vector.* η *is the step size or learning rate. This is also known as error back-propagation learning*
Newton's method	$= -\boldsymbol{H}^{-1}\boldsymbol{g}(t)$ $= -[d^2E/d\boldsymbol{W}^2]^{-1}(dE/d\boldsymbol{W})$	$\boldsymbol{H}$ *is known as the Hessian matrix. There are several different ways to estimate it.*
Conjugate-Gradient method	$= \eta\, \boldsymbol{p}(t)$ *where* $\boldsymbol{p}(t+1) = -\boldsymbol{g}(t+1) + \beta\, \boldsymbol{p}(t)$	

η – *learning rate*

W – *weight*

g - *gradient vector*

E – *error*

Artificial Neural Networks

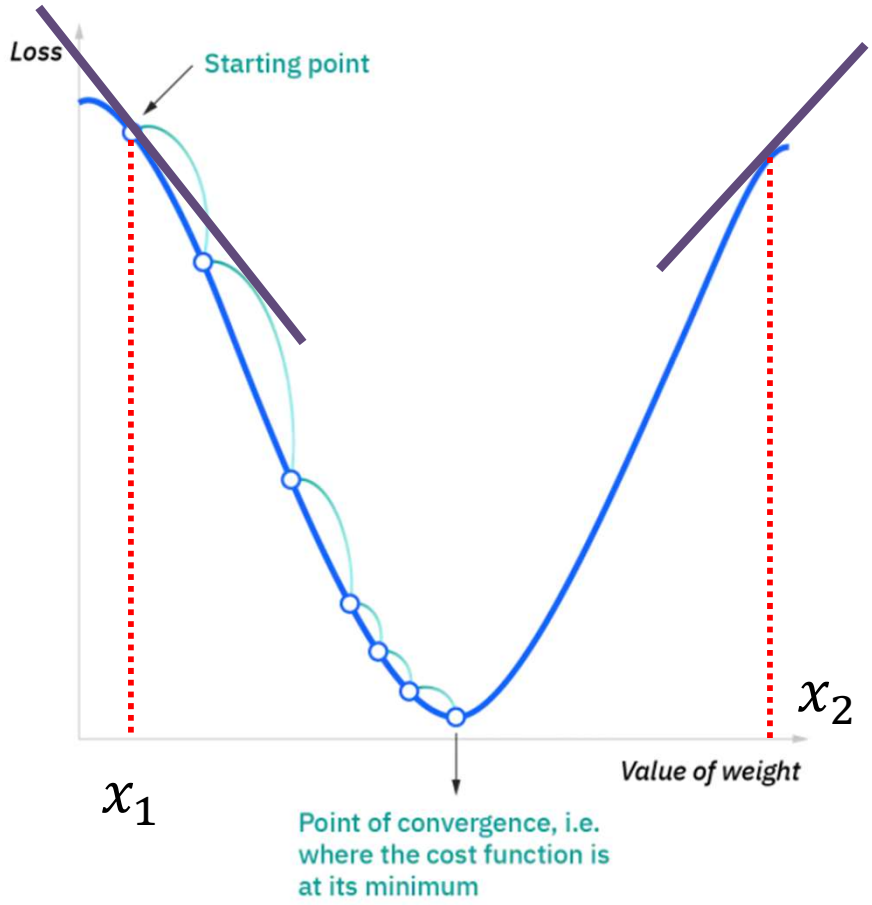
- ✓ Error back propagation
  - ❖ Optimization problem
  - ❖ Gradient descent

$$x \leftarrow x - \frac{df}{dx}$$

$$x \leftarrow x - \mu \frac{df}{dx}$$

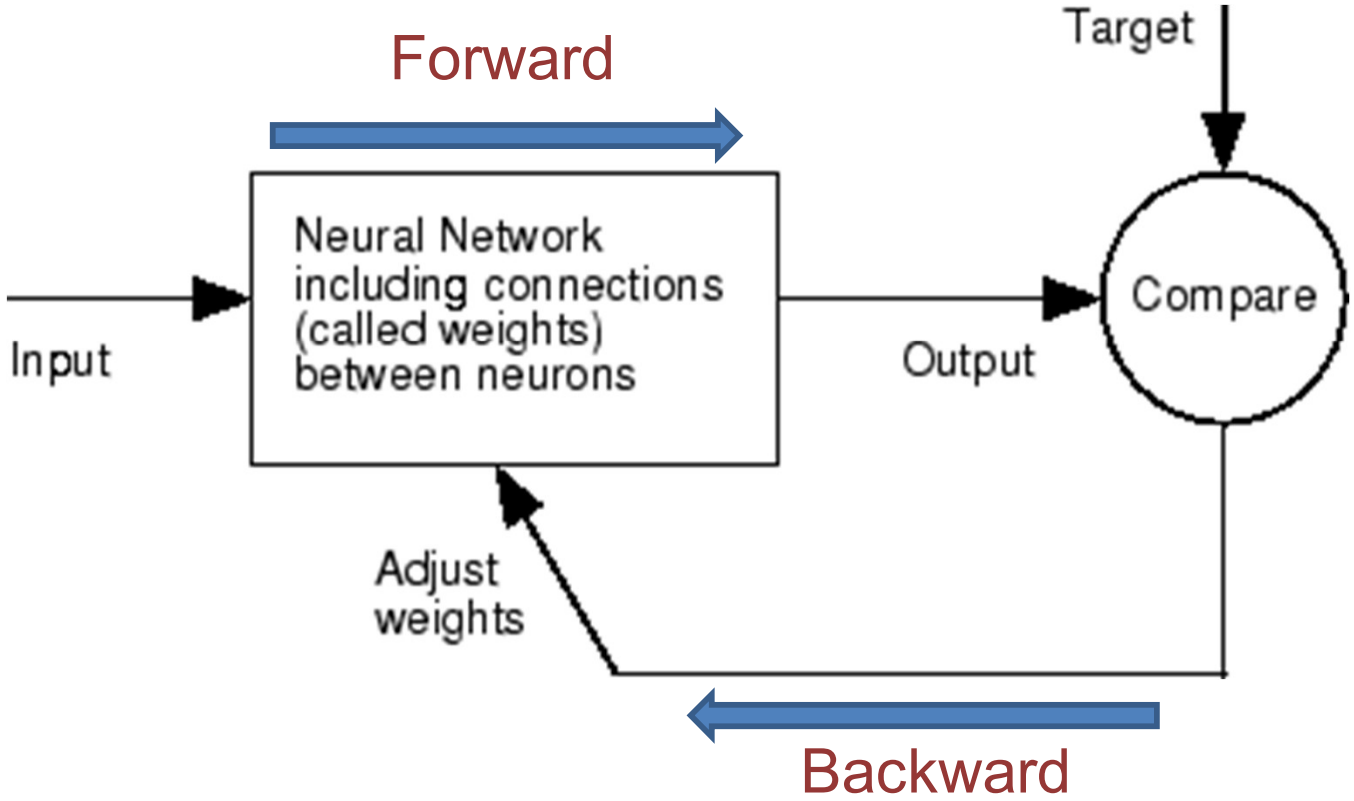
μ: learning rate

$$w \leftarrow w - \mu \frac{\partial L}{\partial w}$$



Artificial Neural Networks

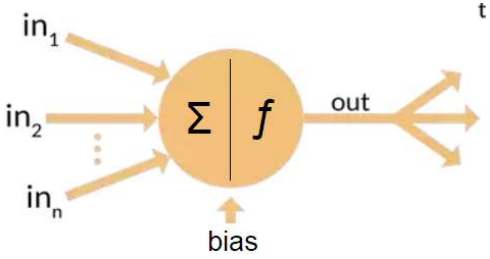
- ✓ Error back propagation
  - ❖ Example

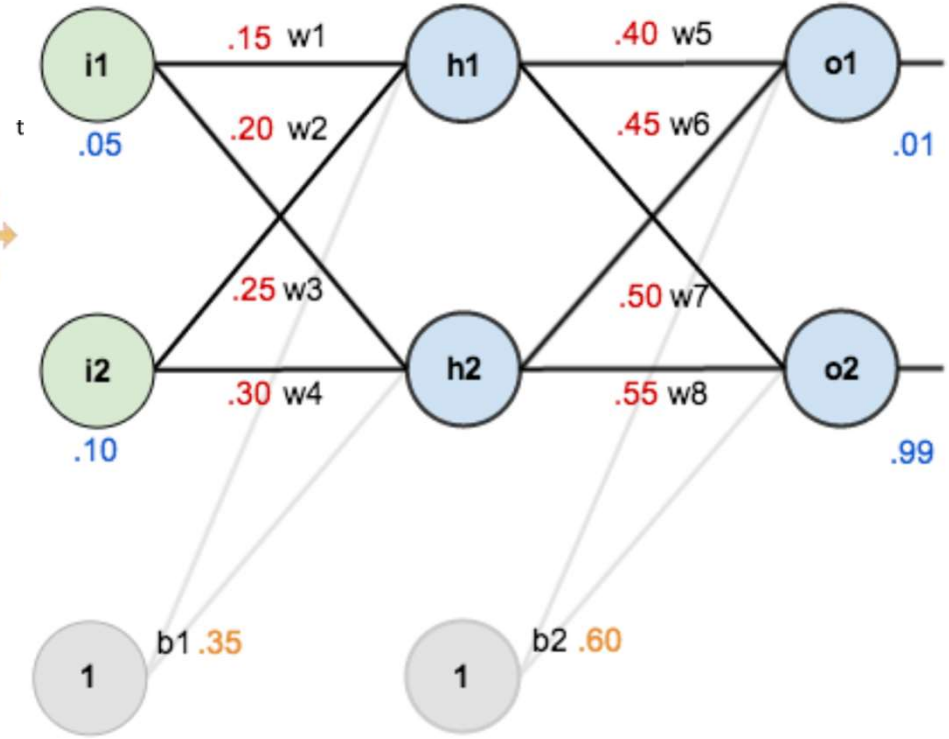


$$w \leftarrow w - \mu \frac{\partial L}{\partial w}$$

Artificial Neural Networks

- ✓ Error back propagation
  - ❖ Example





$$net_{h_1} = w_1 i_1 + w_2 i_2 + b_1$$

$$out_{h_1} = f(net_{h_1})$$

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

$$E = \frac{1}{2}\left\{\left(target_{o_1} - out_{o_1}\right)^2 + \left(target_{o_2} - out_{o_2}\right)^2\right\}$$

Artificial Neural Networks

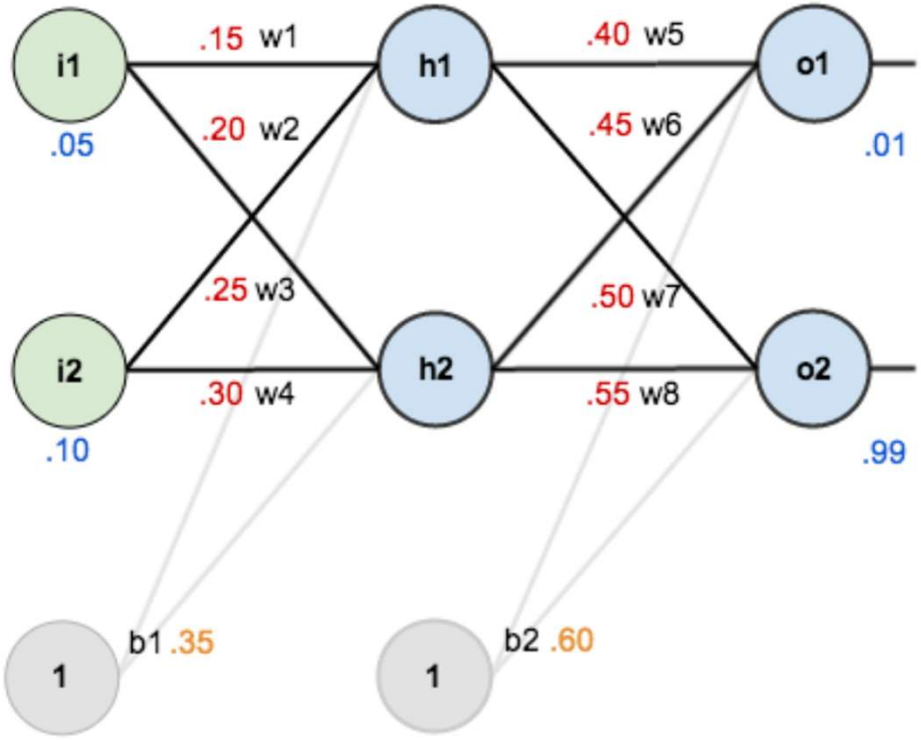
- ✓ Error back propagation

$$w \leftarrow w - \mu \frac{\partial L}{\partial w}$$

$$\frac{\partial E}{\partial w_5}$$

  - ❖ Chain rule

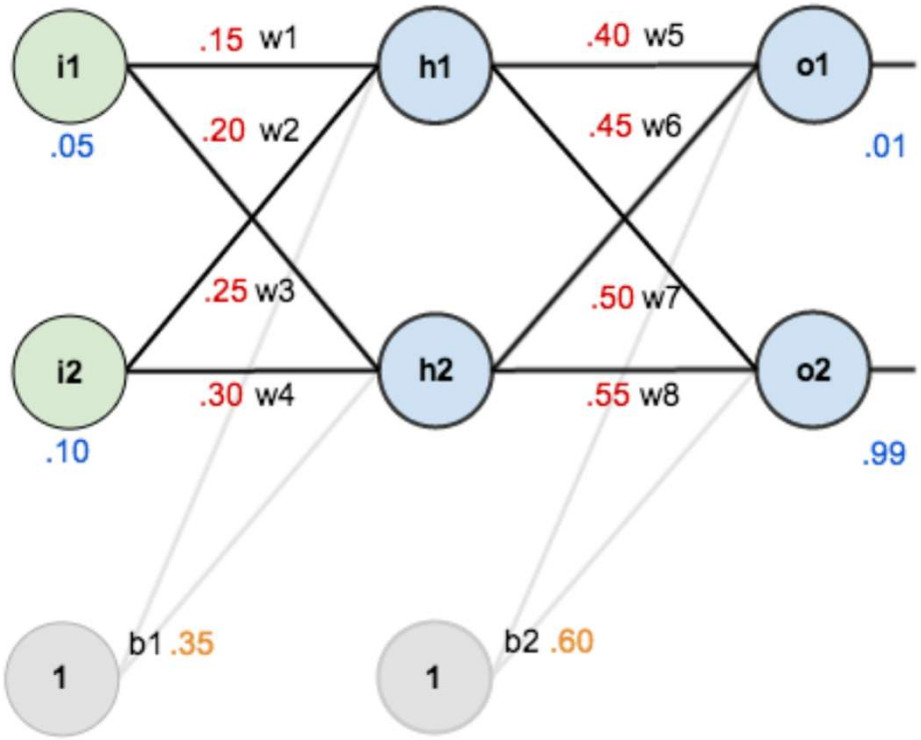
$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial ne_{o_1}} \frac{\partial net_{o_1}}{\partial w_5}$$



Artificial Neural Networks

- ✓ Error back propagation
  - ❖ Example

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial w_5}$$



$$net_{o_1} = w_5 out_{h_1} + w_6 out_{h_2} + b_2$$

$$out_{o_1} = f(net_{o_1})$$

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

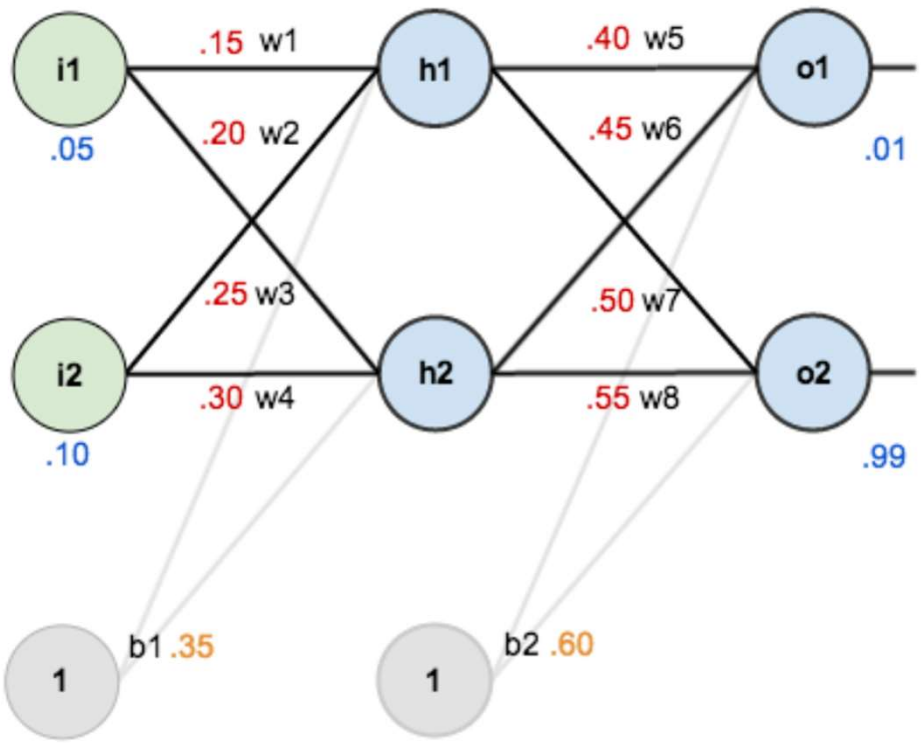
$$E = \frac{1}{2}\left\{\left(target_{o_1} - out_{o_1}\right)^2 + \left(target_{o_2} - out_{o_2}\right)^2\right\}$$

Artificial Neural Networks

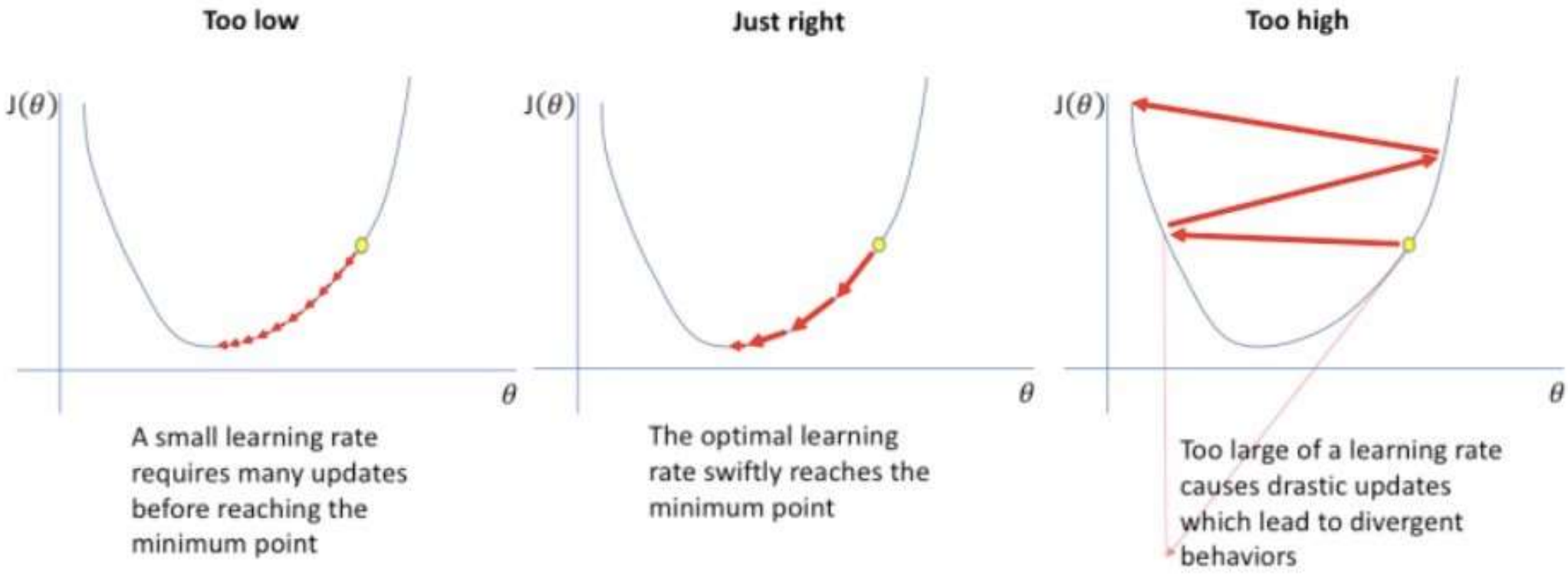
- ✓ Error back propagation
  - ❖ Chain rule

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial w_5}$$

w_5 ➡ net_{O1} ➡ out_{O1} ➡ E



Learning rate choosing



Stopping conditions

- **Average squared error change:** the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
- **Generalization based criterion:** after each epoch the ANN is tested for generalization. If the generalization performance is adequate then stop.
- **Good generalization:** the I/O mapping is nearly correct for new data

ANN

- Overview of ANN
- Components of ANN
- ANN training (forward and backward propagation)
- **ANN characteristics**
- ANN design

ANN characteristics

- Parameters, hyperparameters
- Shallow NN, deep NN
- Underfitting, overfitting
- Generalization

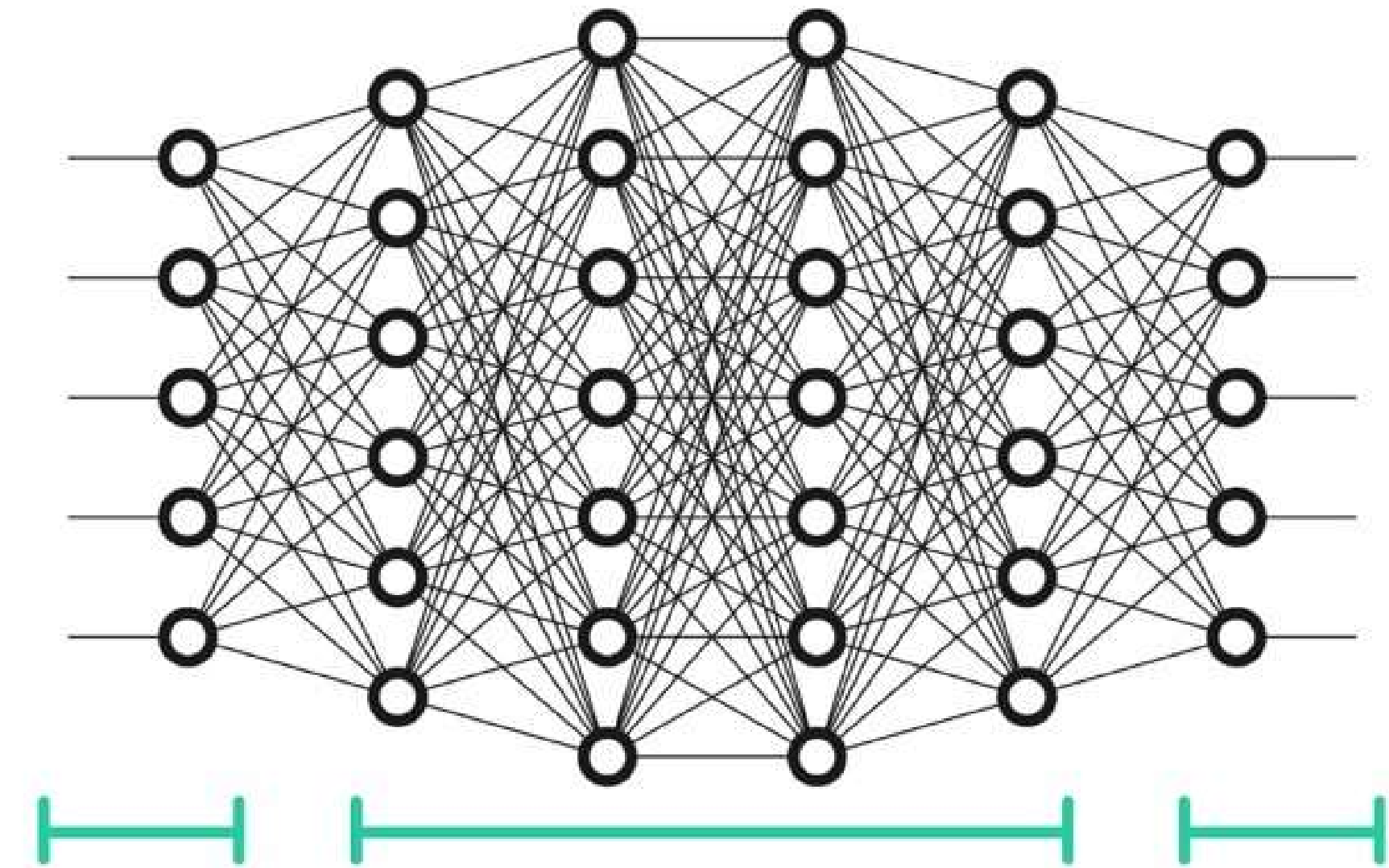
ANN parameters

- **Parameters:** changing while training ANN
  - Weights
  - Biases
- **Hyperparameters:** constant parameters related to ANN configuration defined before training ANN
  - Learning rate
  - Number of hidden layers
  - Net function
  - Activation function,
  - Number of examples in the training dataset…

Shallow NN and deep NN

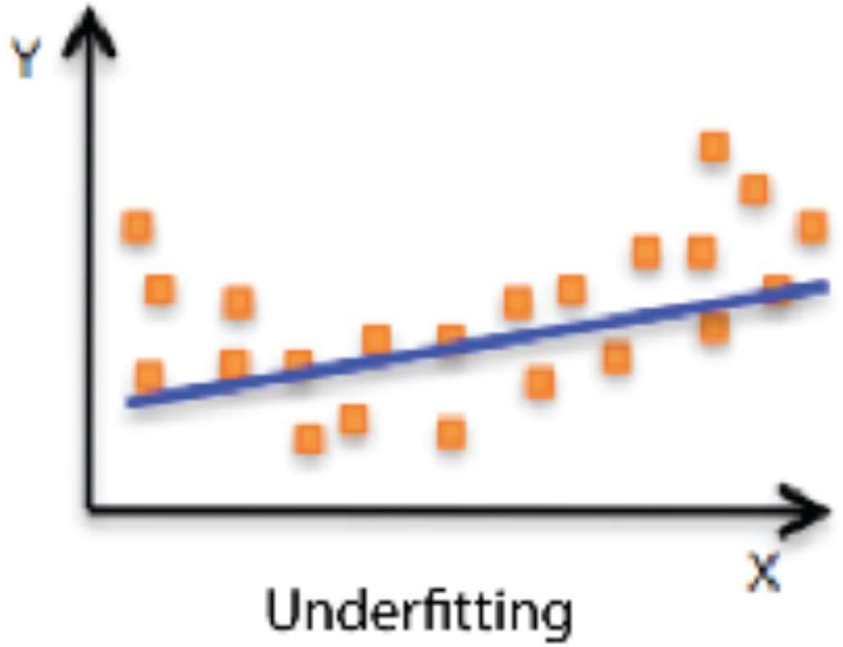
- **Shallow NN:**
  - One hidden layer
  - Used for simple problems
- **Deep NN:**
  - Many hidden layers
  - Used for complex problems
  - Each layer is used for a specific role in the entire problem

Deep neural network



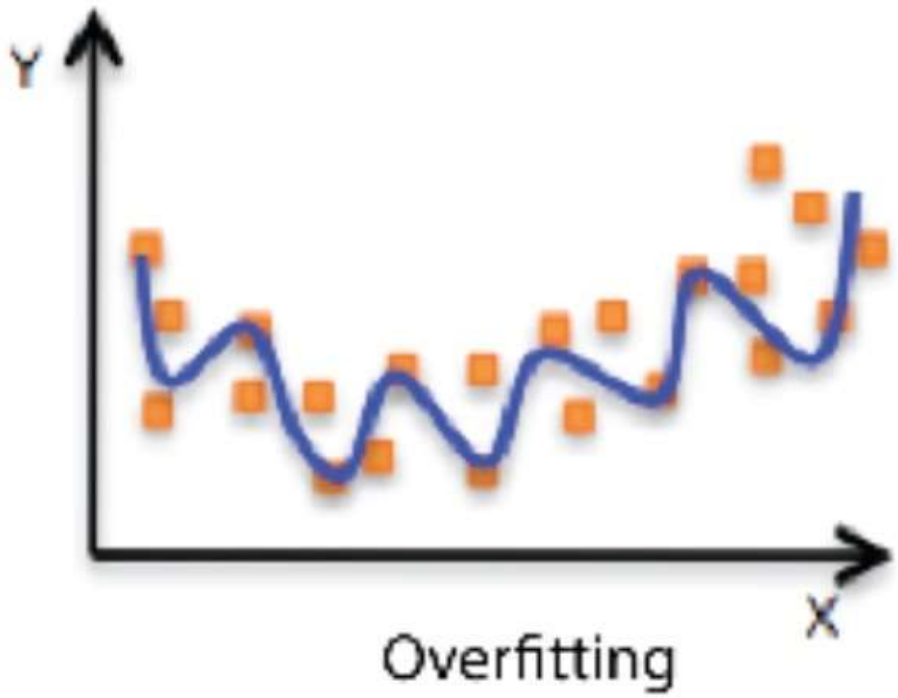
Model fitting

- **Underfitting (high bias):**
  - Model is too simple for data
  - Train error is large, vali/test error is large too.
  - Model can do accurate predictions, but the initial assumption about the data is incorrect

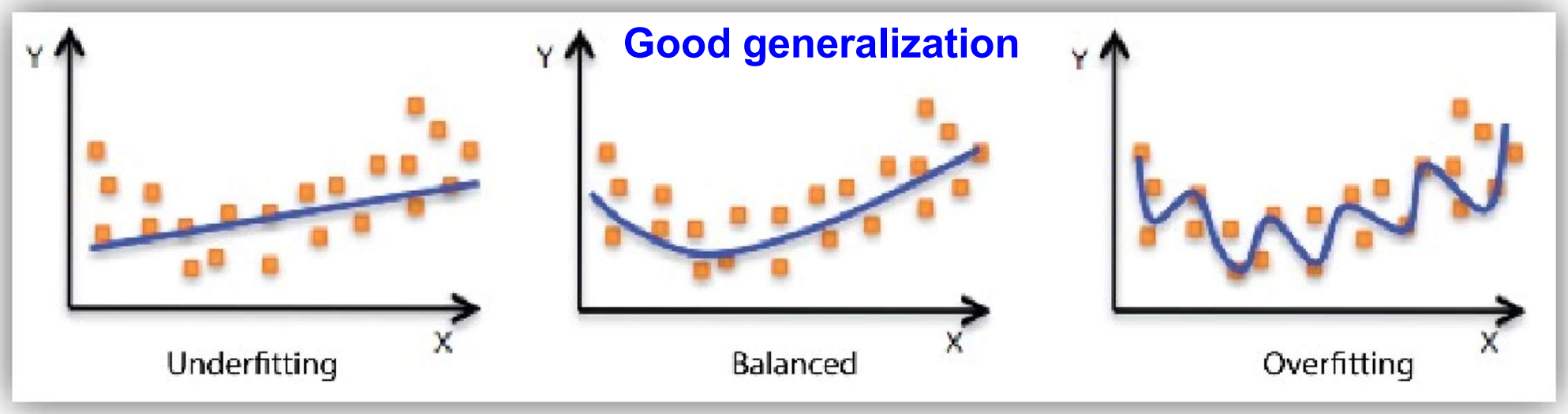


Model fitting (cont)

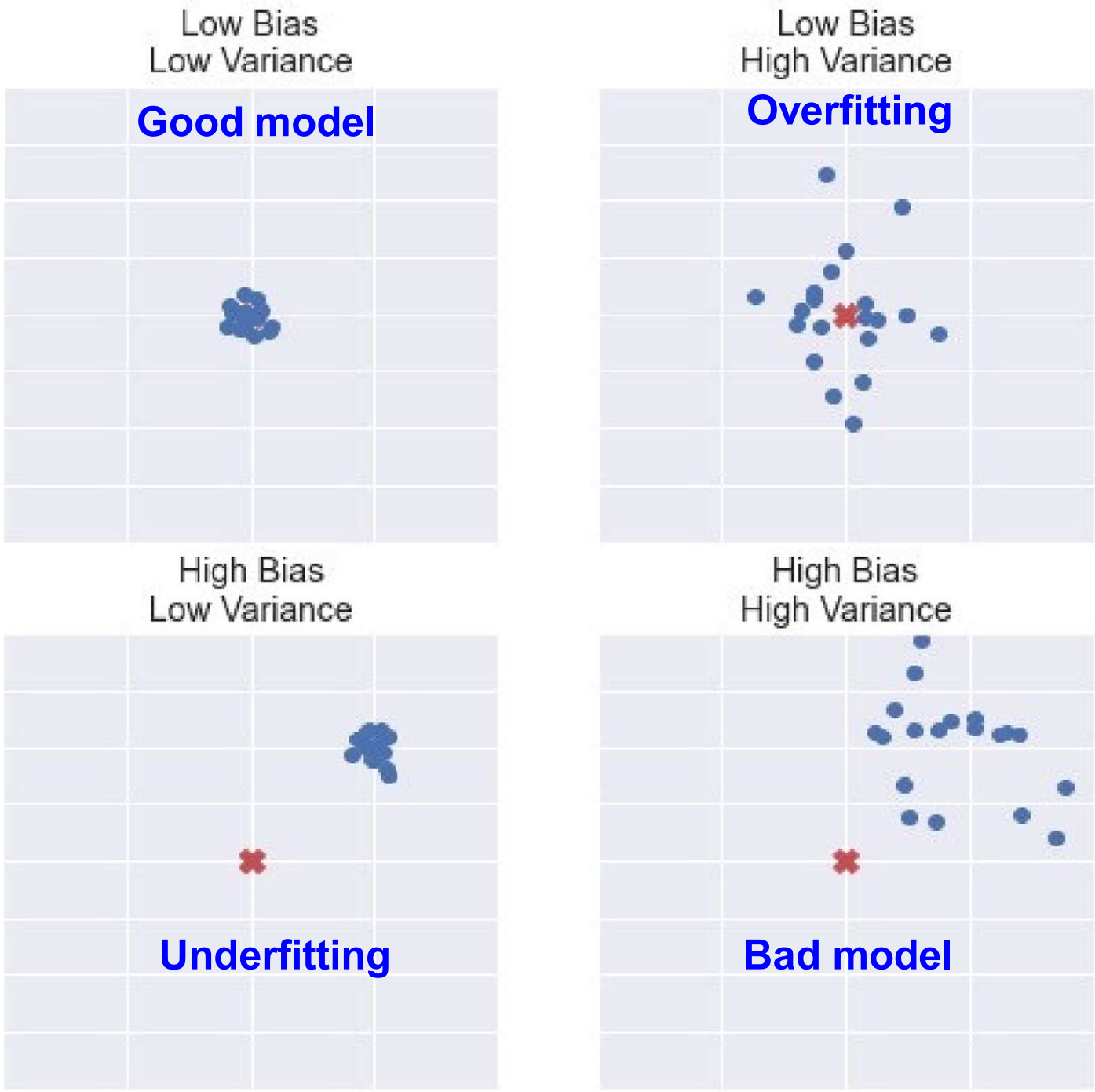
- **Overfitting (high variance):**
  - Model is too complex for data
  - Model memorizes the training data rather than generalize the data → error on training set is small, error on testing set is large

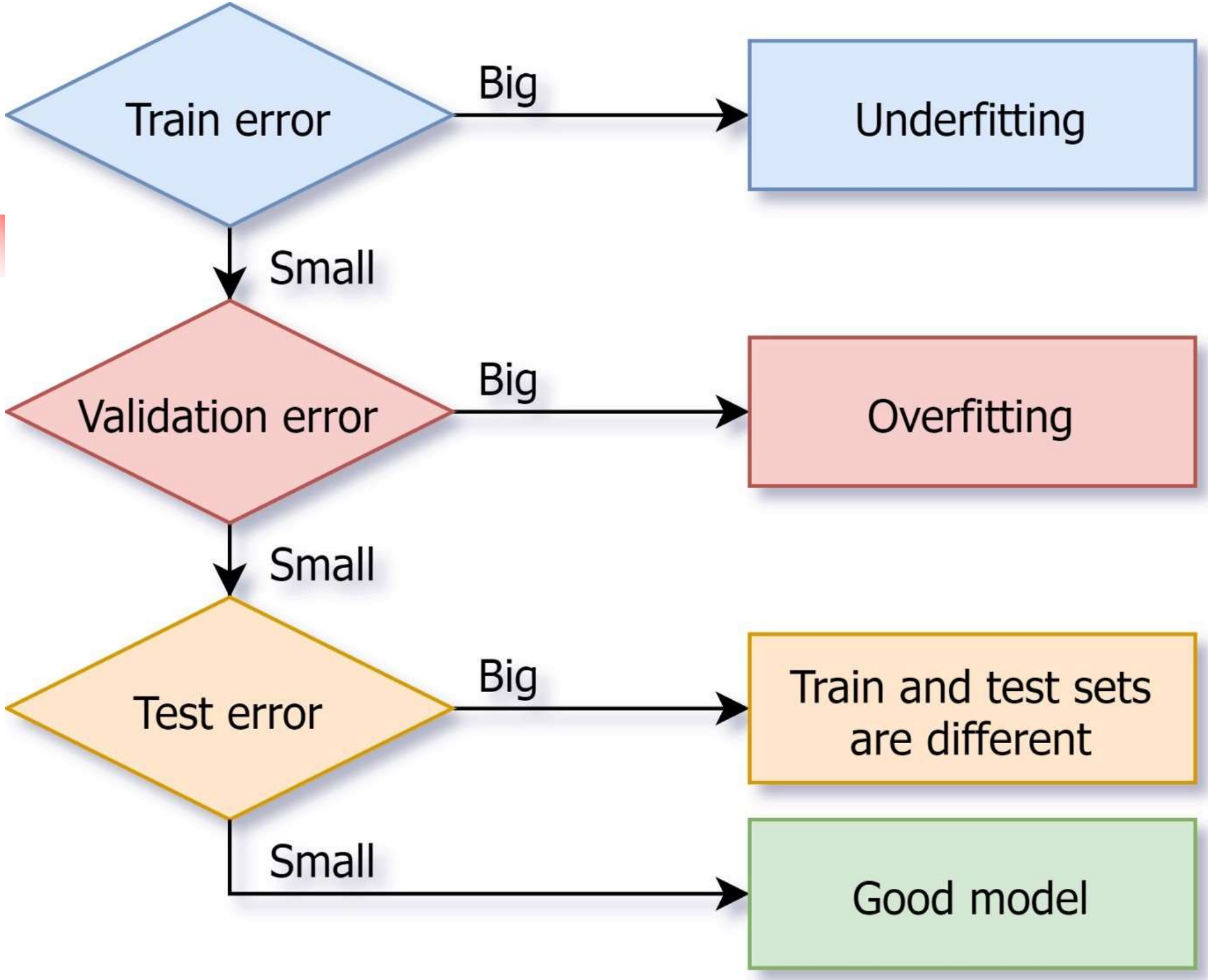


Model fitting (cont)



Model fitting (cont)




Train error
Big
Underfitting
Small
Validation error
Big
Overfitting
Small
Test error
Big
Train and test sets are different
Small
Good model

How to avoid underfitting?

- Try more complex model
  - More powerful model with a larger number of parameters
  - More layers
  - More neurons per layer
- Try larger quantity of features
  - Get additional features
  - Feature engineering
- Data cleaning, cross validation (hold-out, K-fold, LOOCV)

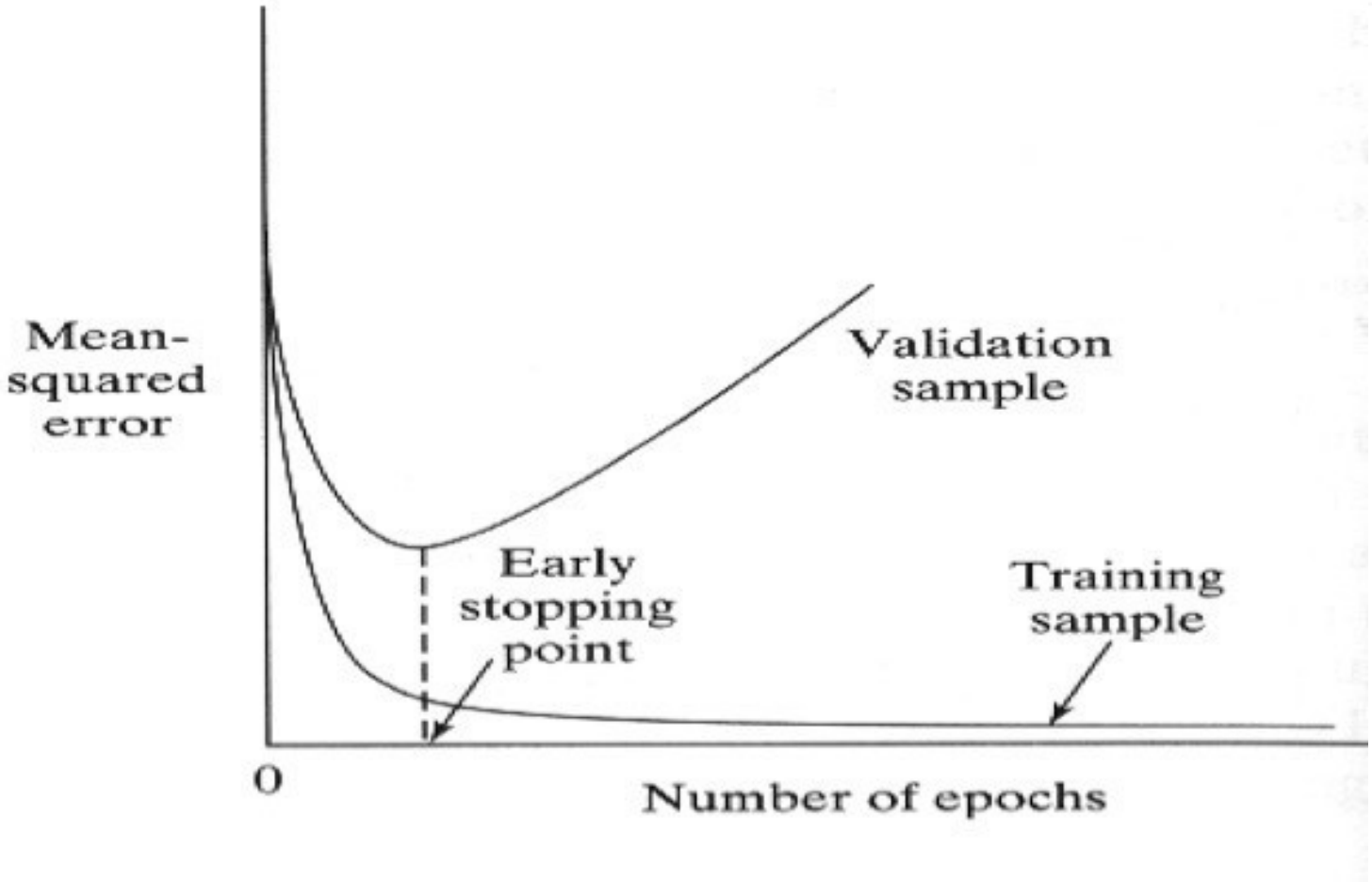
How to avoid overfitting?

- Try more simple model
- Less powerful model with a fewer number of parameters
- Less layers, less neurons per layer
- Try a smaller quantity of features
- Remove additional features
- Feature selection

How to avoid overfitting? (cont)

- Enlarge data
  - Data cleaning
  - Cross validation (hold-out, K-fold, LOOCV)
  - Data augmentation (rotate, flip, scale,…)
- More regularization
  - Early stopping
  - Drop out
  - L1, L2 regularization

Early stopping



Generalization

- **Good generalization:** the I/O mapping is nearly correct for new data