
SUPPORT VECTOR MACHINES
REPEA
POWE
LAWS
END DUALITy GAP
Map Reduce
Map Reuse
Map Recycle
BAYESIANS AGAINST DISCRIMINATION
FREE VARIABLES!!
BAN GENETIC ALGORITHMS

Google software can see your heart attack risk in your eyes

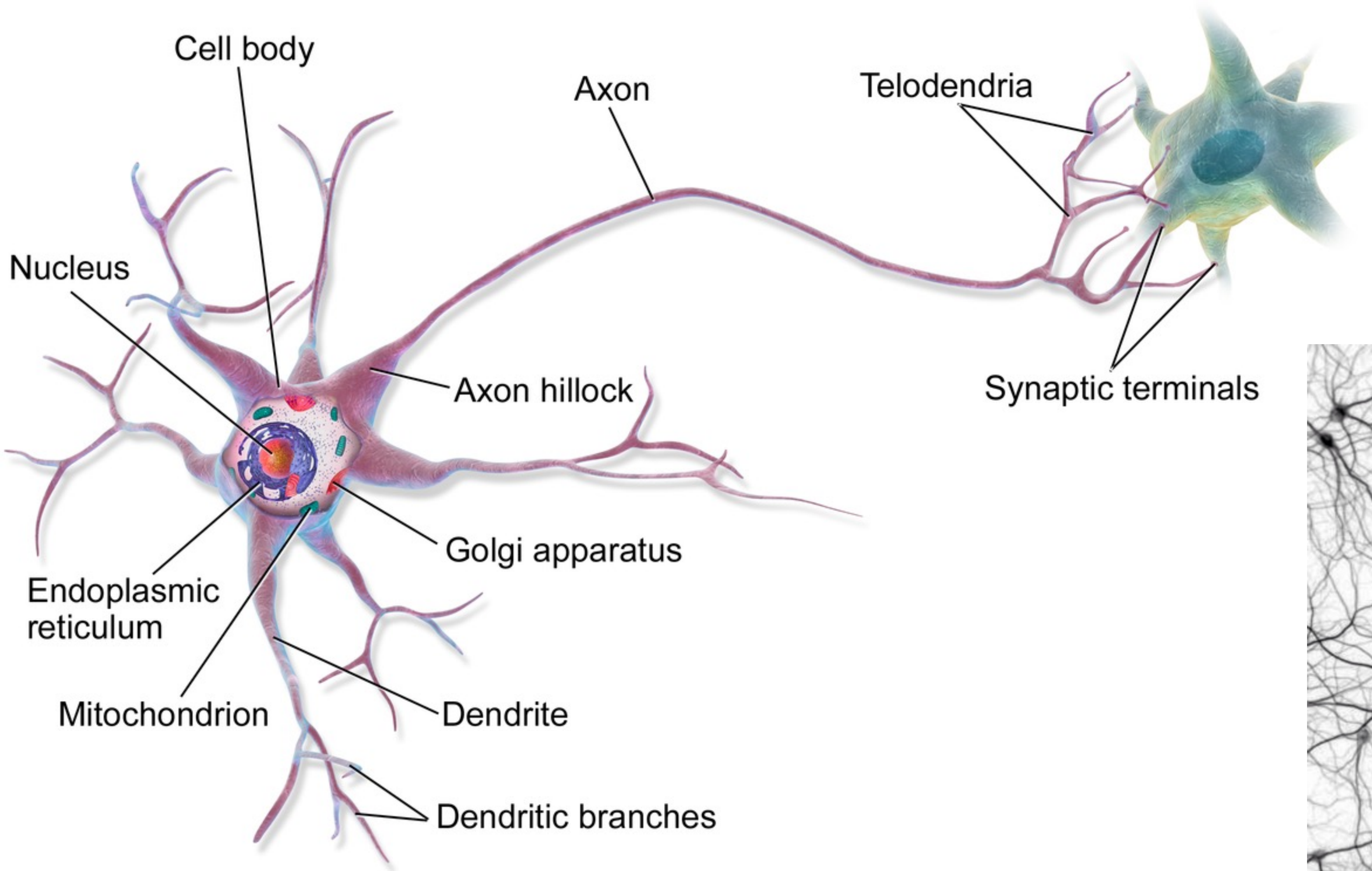
THE WASHINGTON POST | Monday, Feb. 19, 2018, 12:18 p.m.

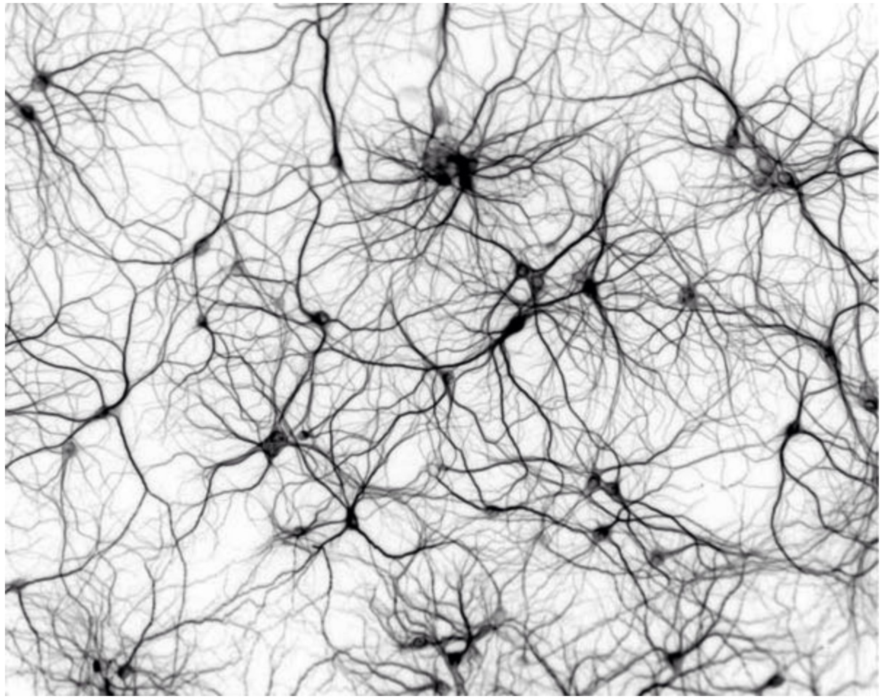
PIXABAY

Neural Networks

Nature Inspires Engineering

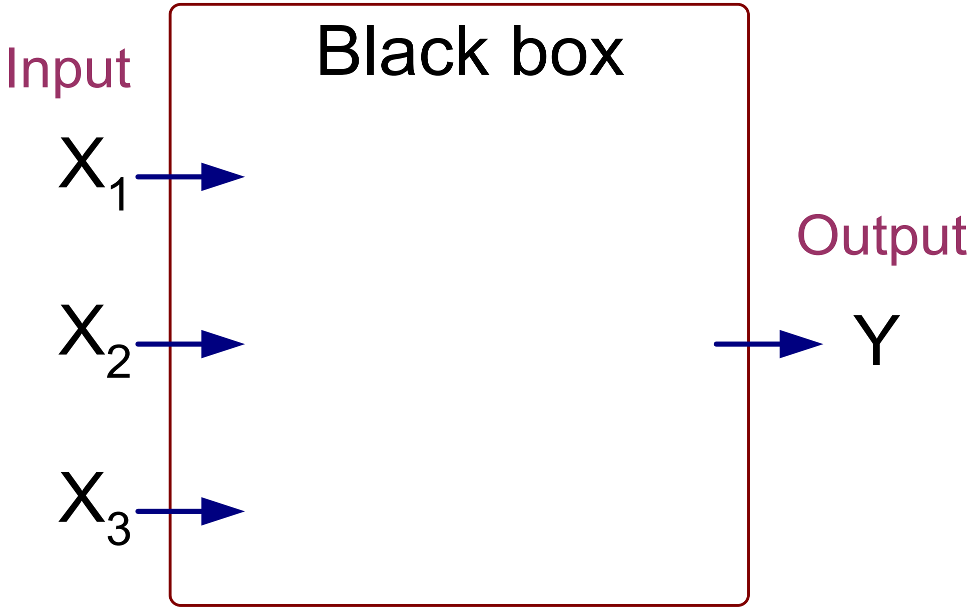
Neurons


Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic reticulum
Mitochondrion
Dendrite
Dendritic branches



Artificial Neural Networks (ANN)

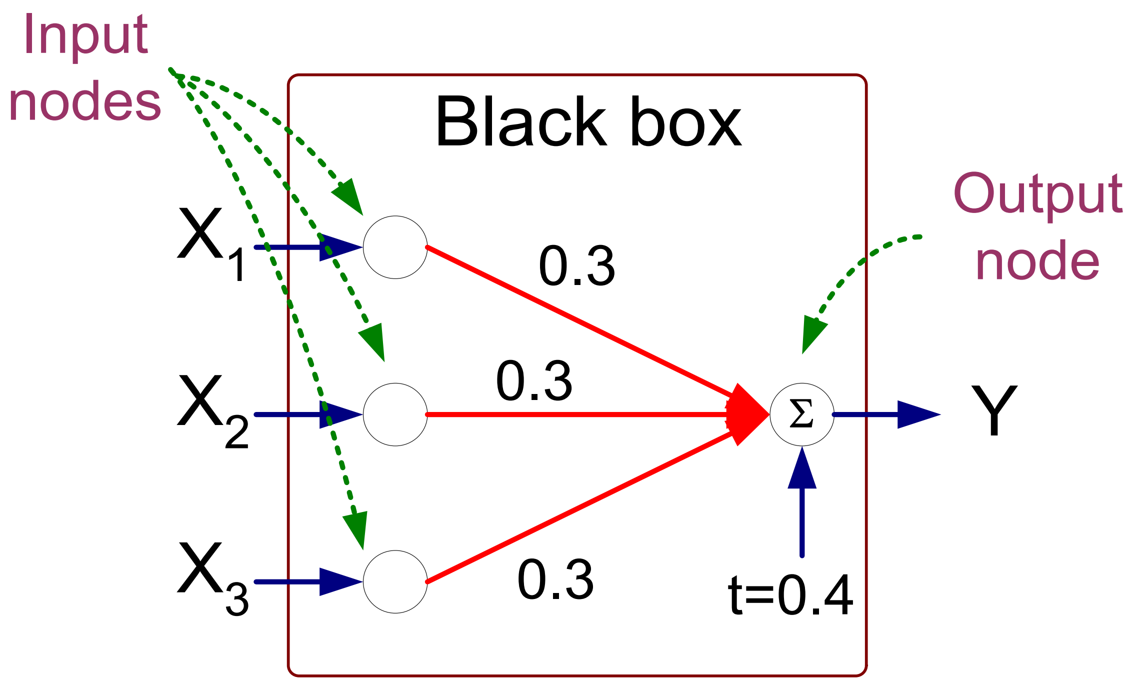
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

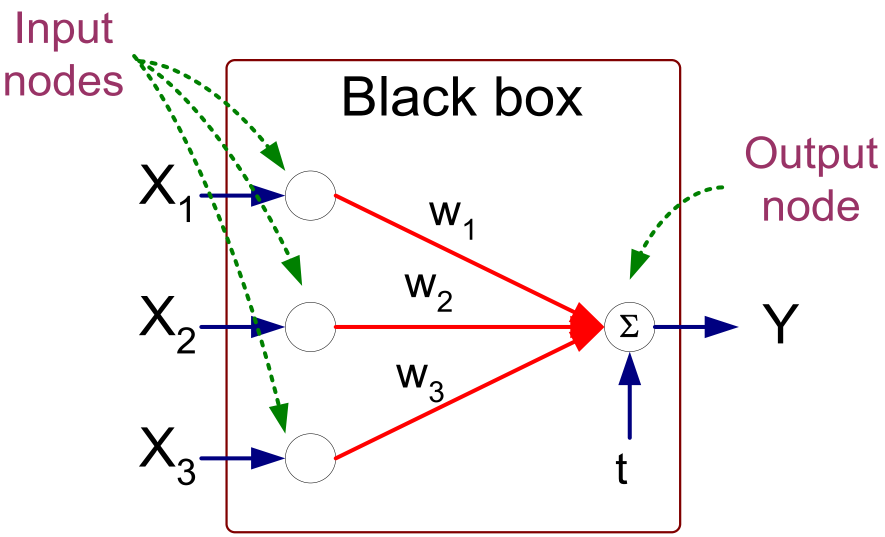


$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t



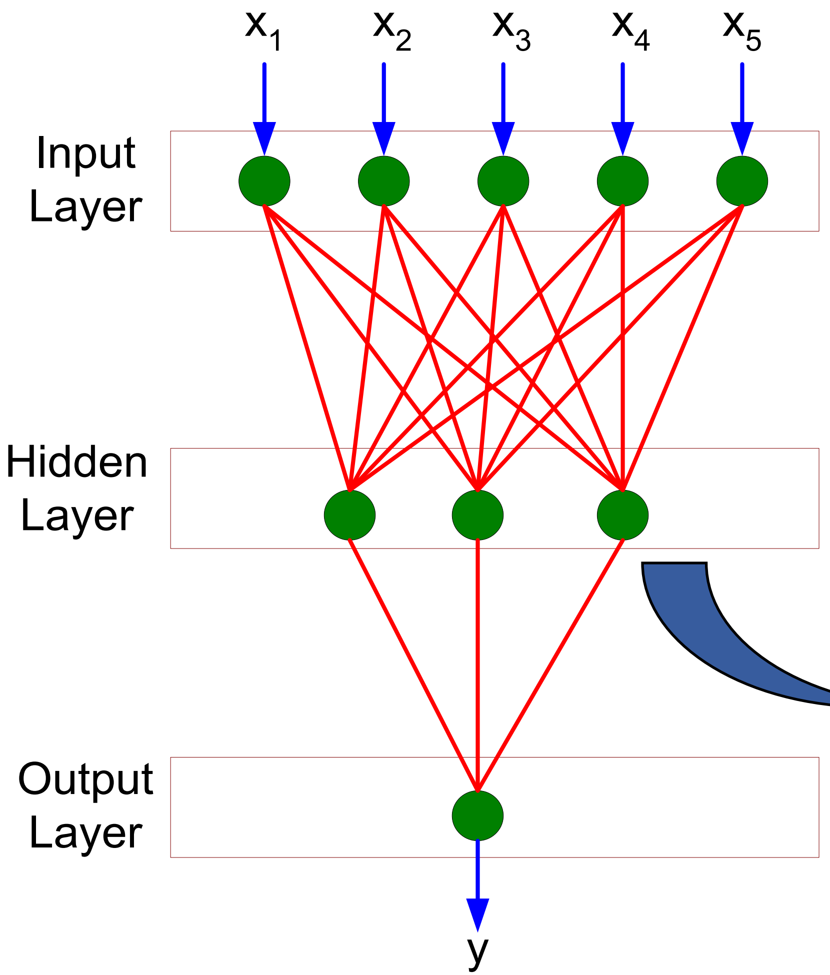
Perceptron Model

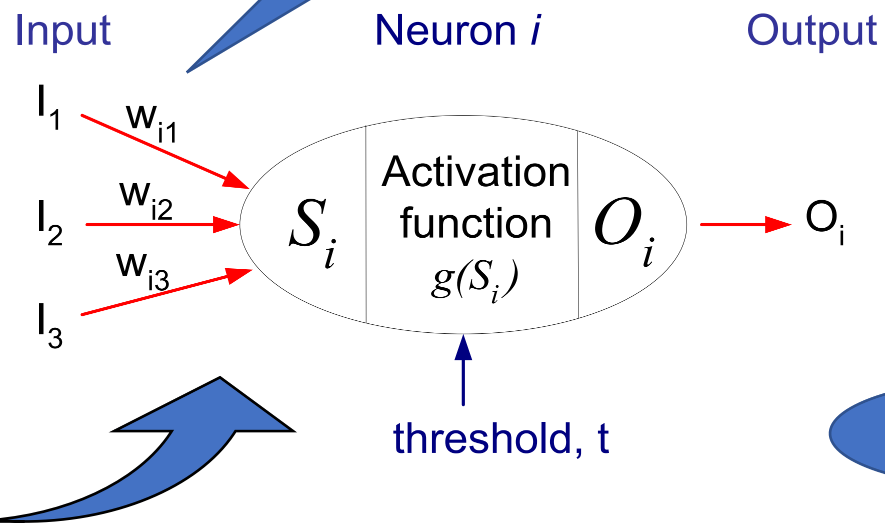
$$Y = sign(\sum_{i=1}^{d} w_i X_i - t)$$

$$= sign(\sum_{i=0}^{d} w_i X_i)$$

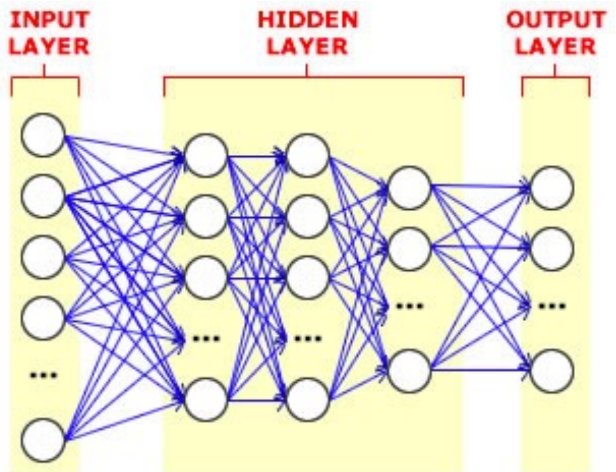
General Structure of ANN

Training ANN means learning the weights of the neurons





Can have more than one output node

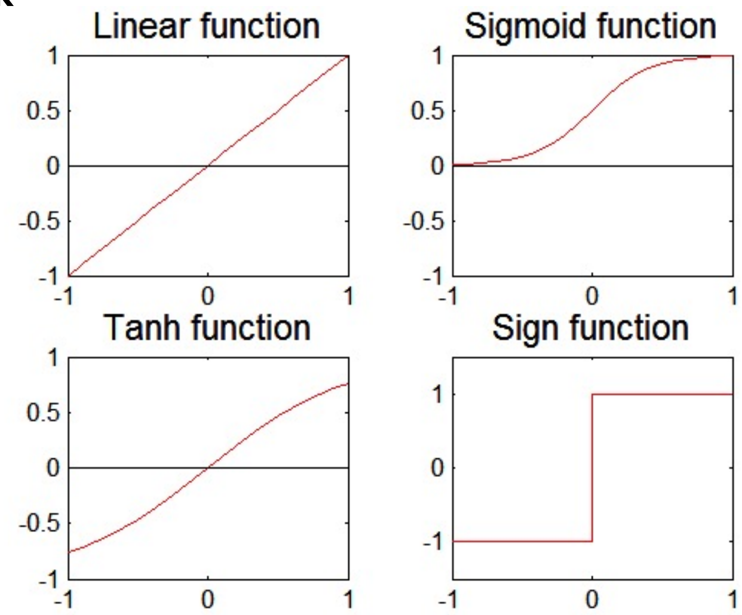


Artificial Neural Networks (ANN)

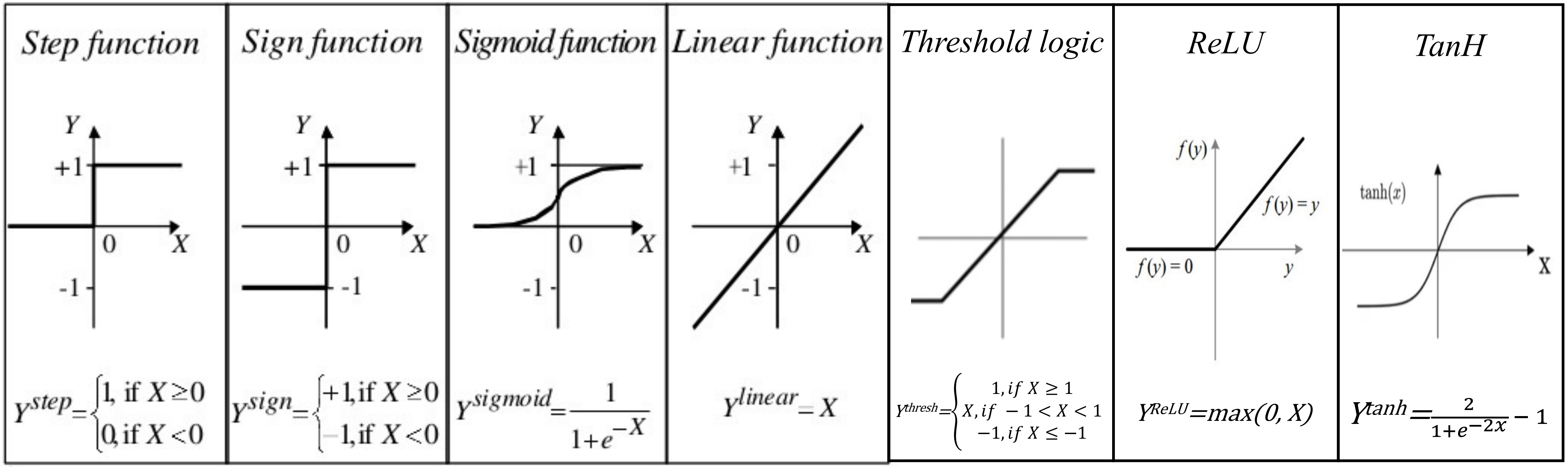
- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f(\sum_i w_i X_i)$$



Activation Functions



The bias value will shift the activation function left or right.
https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

Perceptron

- Single layer network
  - Contains only input and output nodes
- Activation function: f = sign(w•x)
- Applying model is straightforward

$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

  - $X_1 = 1$, $X_2 = 0$, $X_3 = 1$ => y = sign(0.2) = 1

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute f(w, x_i)
    - Update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

- Until stopping condition is met

Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda\big[y_i - f(w^{(k)}, x_i)\big]x_i \;\;;\;\; \lambda\text{: learning rate}$$

$$e = \left[y_i - f(w^{(k)}, x_i)\right]$$

- Intuition:
  - Update weight based on error:
  - If y=f(x,w), e=0: no update needed
  - If y>f(x,w), e=2: weight must be increased so that f(x,w) will increase
  - If y<f(x,w), e=-2: weight must be decreased so that f(x,w) will decrease

Example of Perceptron Learning

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

$$Y = sign(\sum_{i=0}^{d} w_i X_i)$$

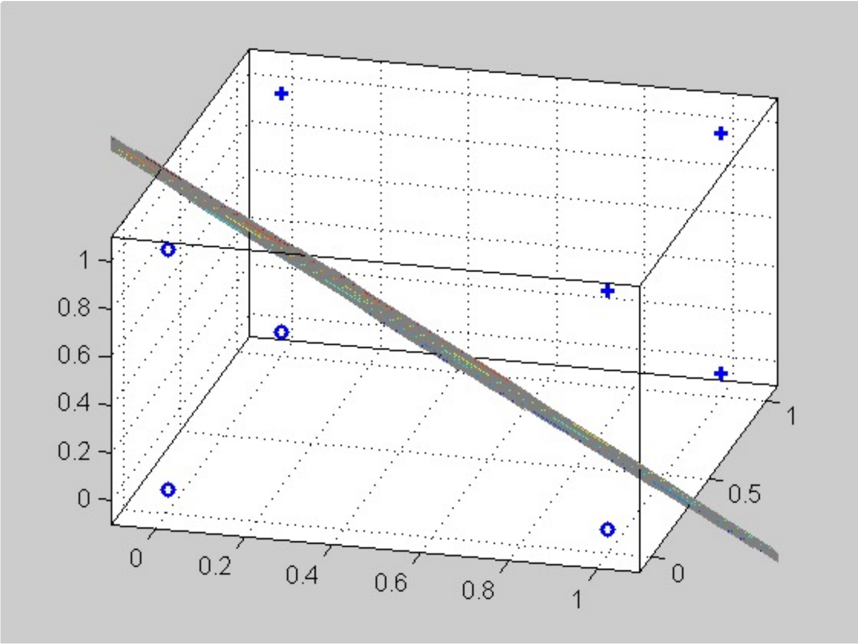
$$\lambda = 0.1$$

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Perceptron Learning Rule



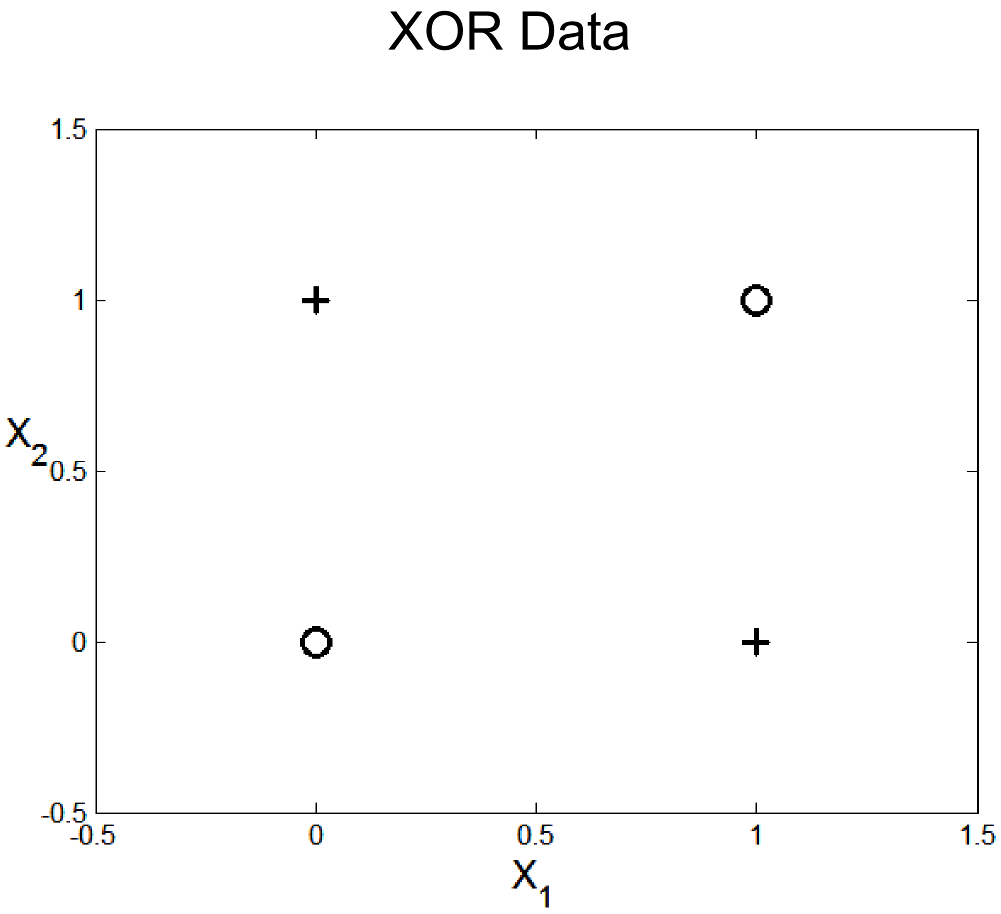
- Since f(w,x) is a linear combination of input variables, decision boundary is linear

- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

Nonlinearly Separable Data

$$y = x_1 \oplus x_2$$

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1



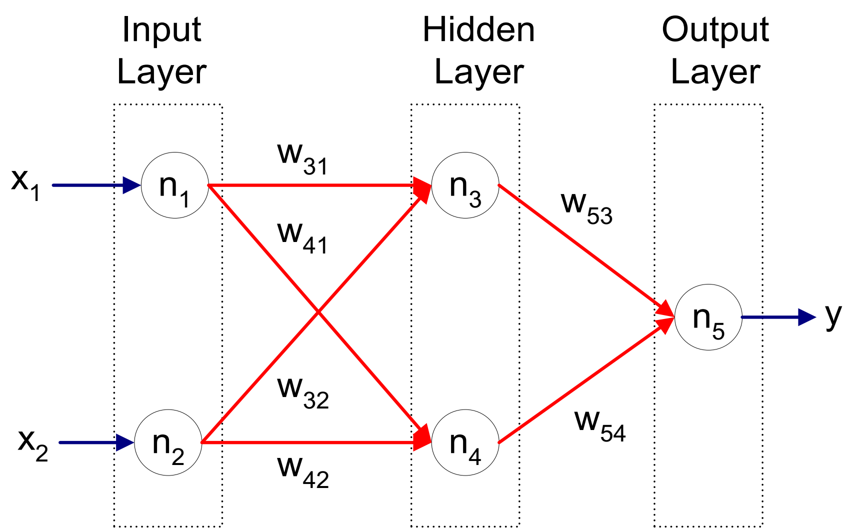
Multilayer Neural Network

- Hidden layers
  - intermediary layers between input & output layers

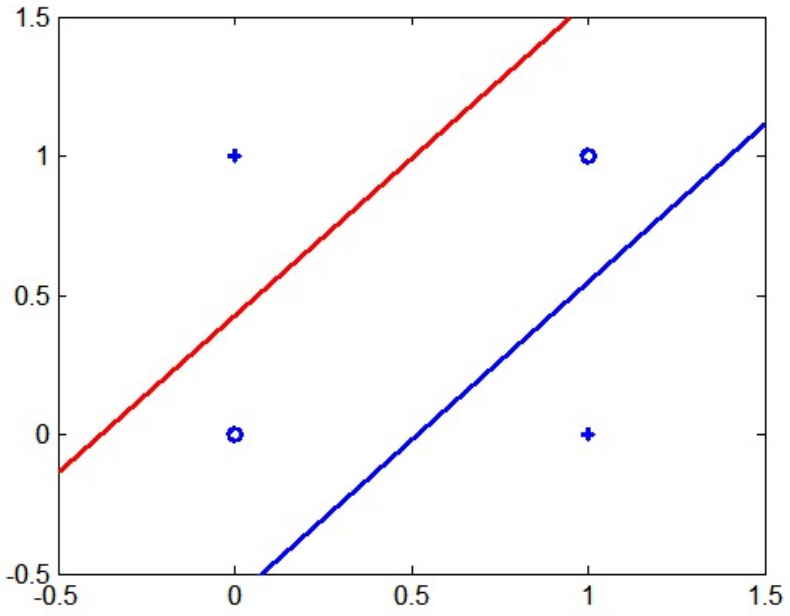
- More general activation functions (sigmoid, linear, etc)

Multi-layer Neural Network

- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces



XOR Data

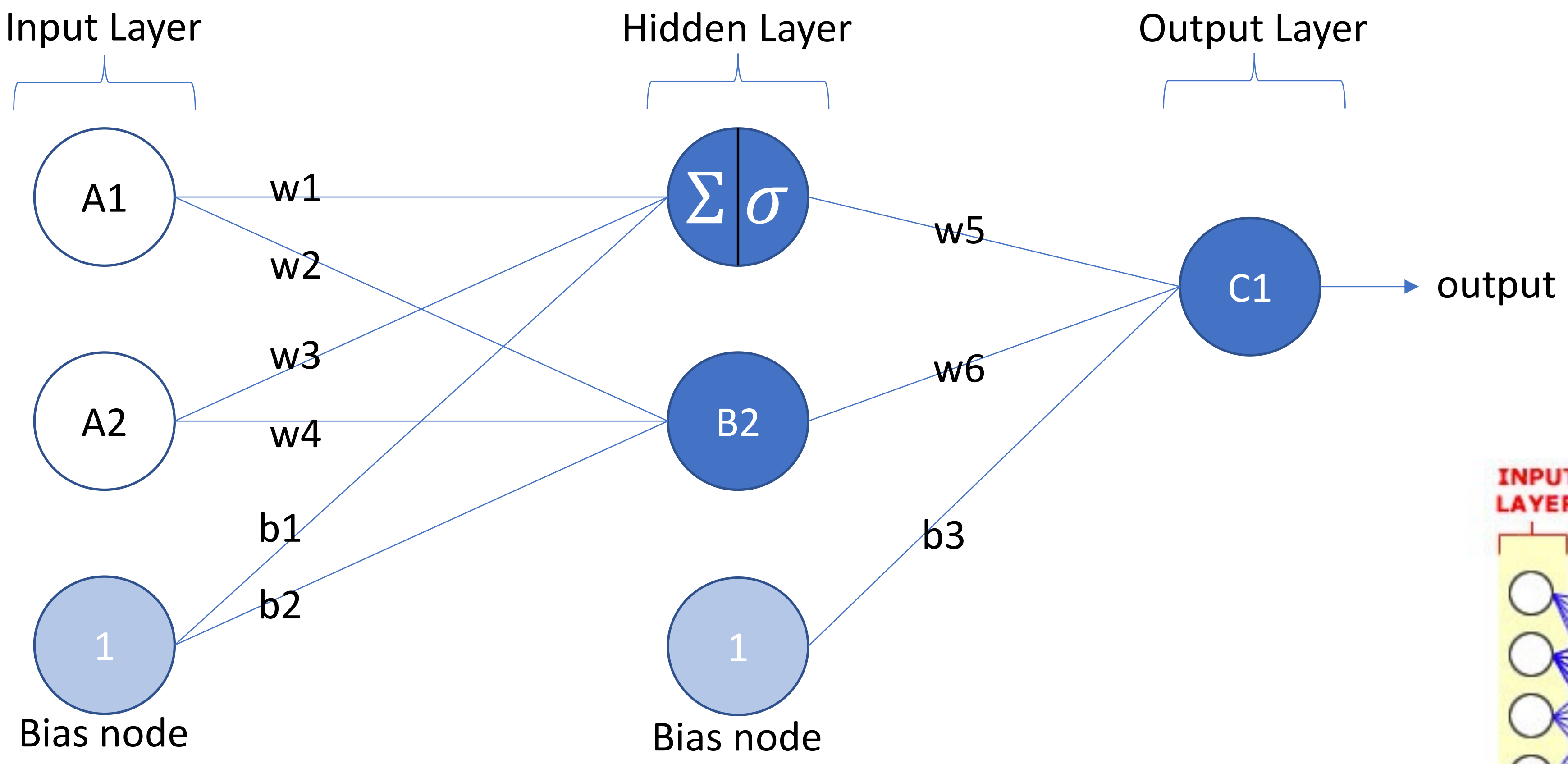


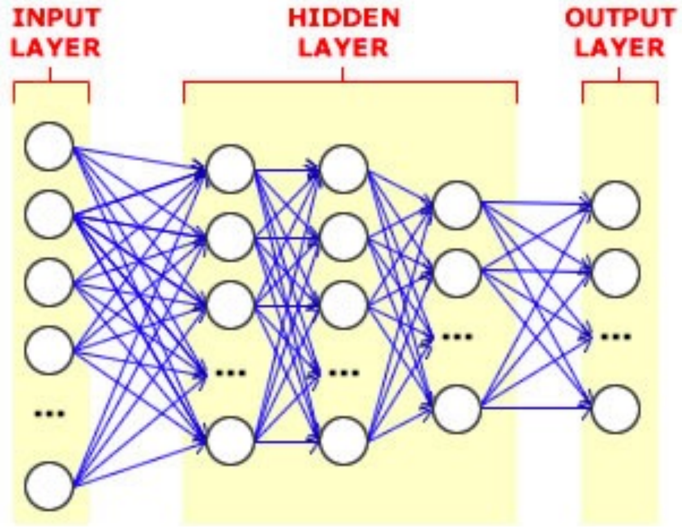
Training an Artificial Neural Network: Rinse and Repeat

- Step 1: Forward Propagation
  - Summation Operator
  - Activation Function
- Step 2: Calculate Total Error
  - Cost Function
- Step 3: Calculate Gradients
  - Partial Derivative
  - Chain Rule
- Step 4: Update Weights
  - Weight Update Formula

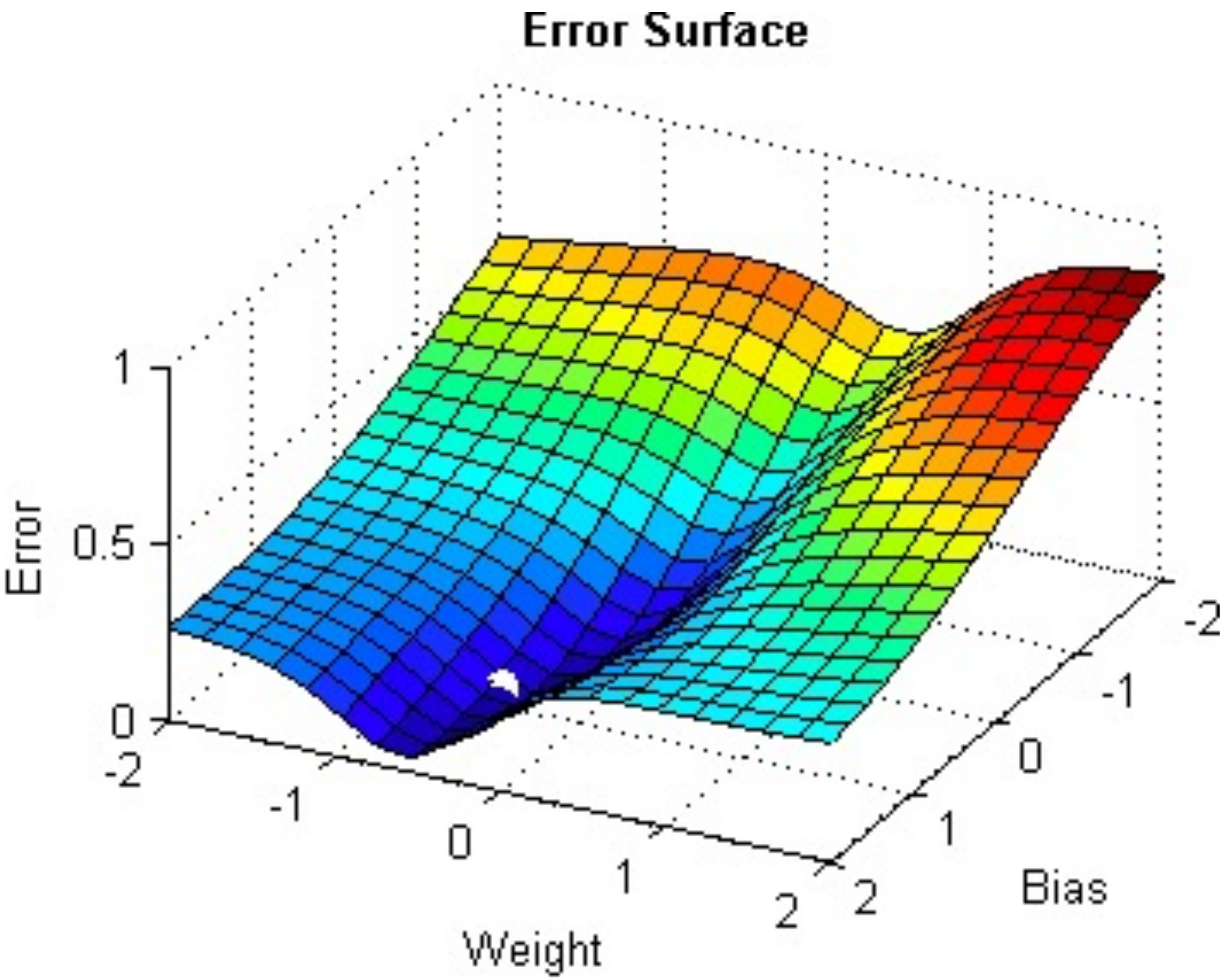
Backpropagation

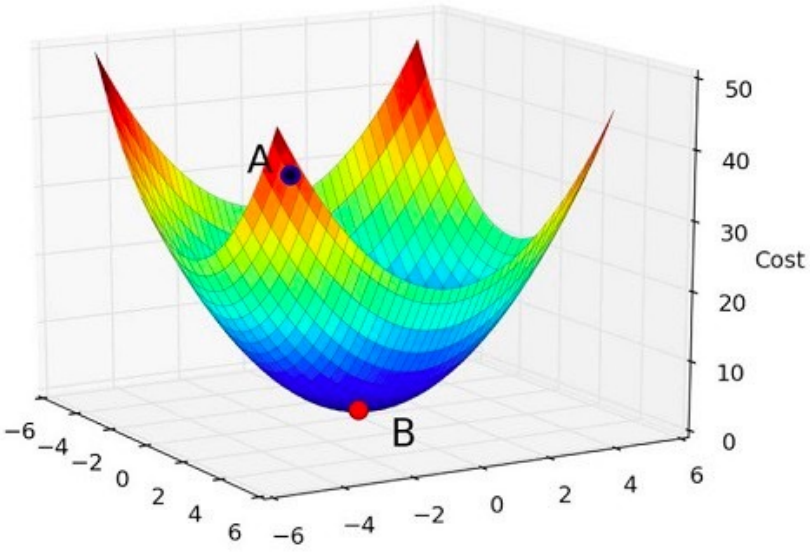
Forward Propagation Example

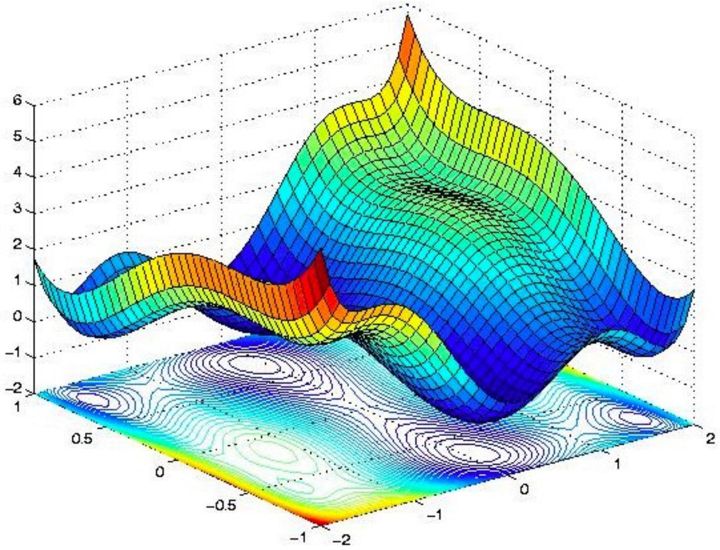




Gradient Descent

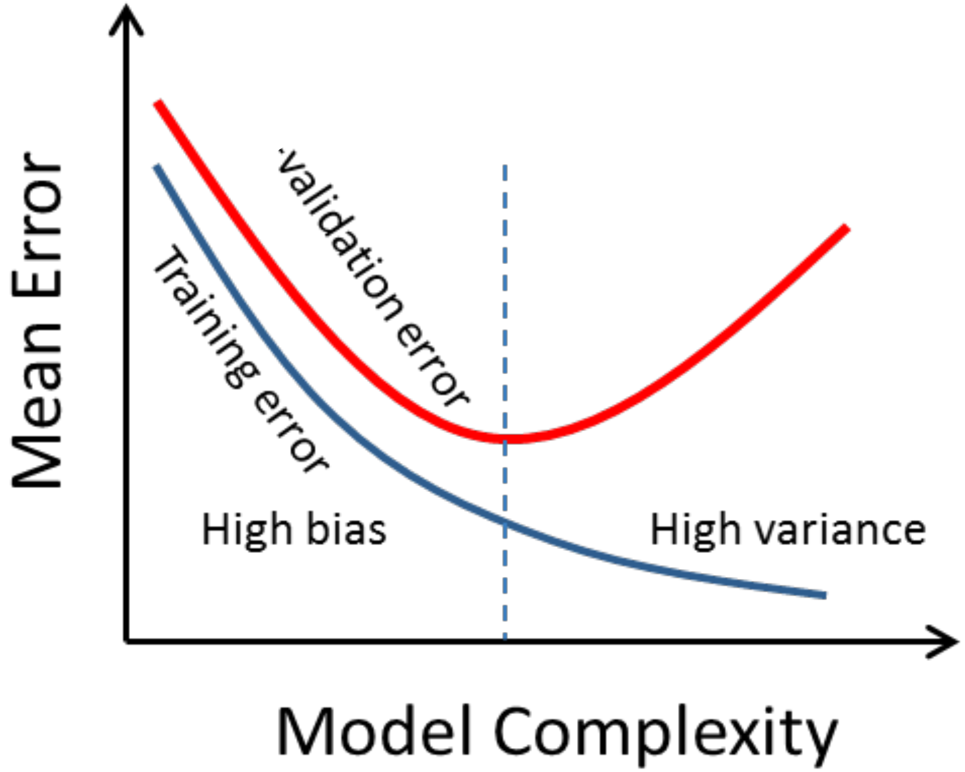






Stopping Conditions

- Error reaches some threshold
- Weights and bias terms converge
- Number of epochs reached
- Overfitting begins



Characteristics of Neural Networks

- Multi-layer neural networks with at least one hidden layer can learn complex and diverse decision boundaries
- Susceptible to overfitting
- Can handle irrelevant attributes by using zero weights; can handle redundant attributes by using similar weights
- Can get stuck at a local minima (a non-optimal solution)
- Training is time consuming and requires a lot of data
- Difficult to interpret the results
- Difficult to handle missing attributes


DEEP LEARNING,
SO HOT RIGHT NOW
imgflip.com

Deep Neural Networks

- Involve a large number of hidden layers
- Can represent features at multiple levels of abstraction
- Often require fewer nodes per layer to achieve generalization performance similar to shallow networks
- Deep networks have become the technique of choice for complex problems such as vision and language processing

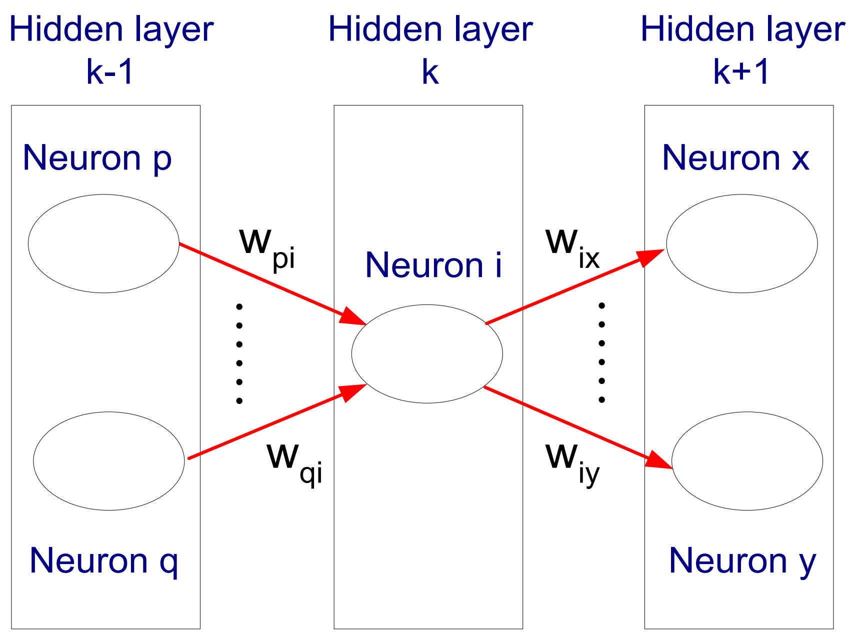
Expanded Explanation

(optional back up material)

Gradient Descent for MultiLayer NN

- For output neurons, weight update formula is the same as before (gradient descent for perceptron)

- For hidden neurons:



$$w_{pi}^{(k+1)} = w_{pi}^{(k)} + \lambda o_i (1 - o_i) \sum_{j \in \Phi_i} \delta_j w_{ij} x_{pi}$$

$$\text{Output neurons}: \delta_j = o_j (1 - o_j)(t_j - o_j)$$

$$\text{Hidden neurons}: \delta_j = o_j (1 - o_j) \sum_{k \in \Phi_j} \delta_k w_{jk}$$

Learning Multi-layer Neural Network

- Can we apply the perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term e = y-f(w,x) and updates weights accordingly
    - Problem: how to determine the true value of y for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

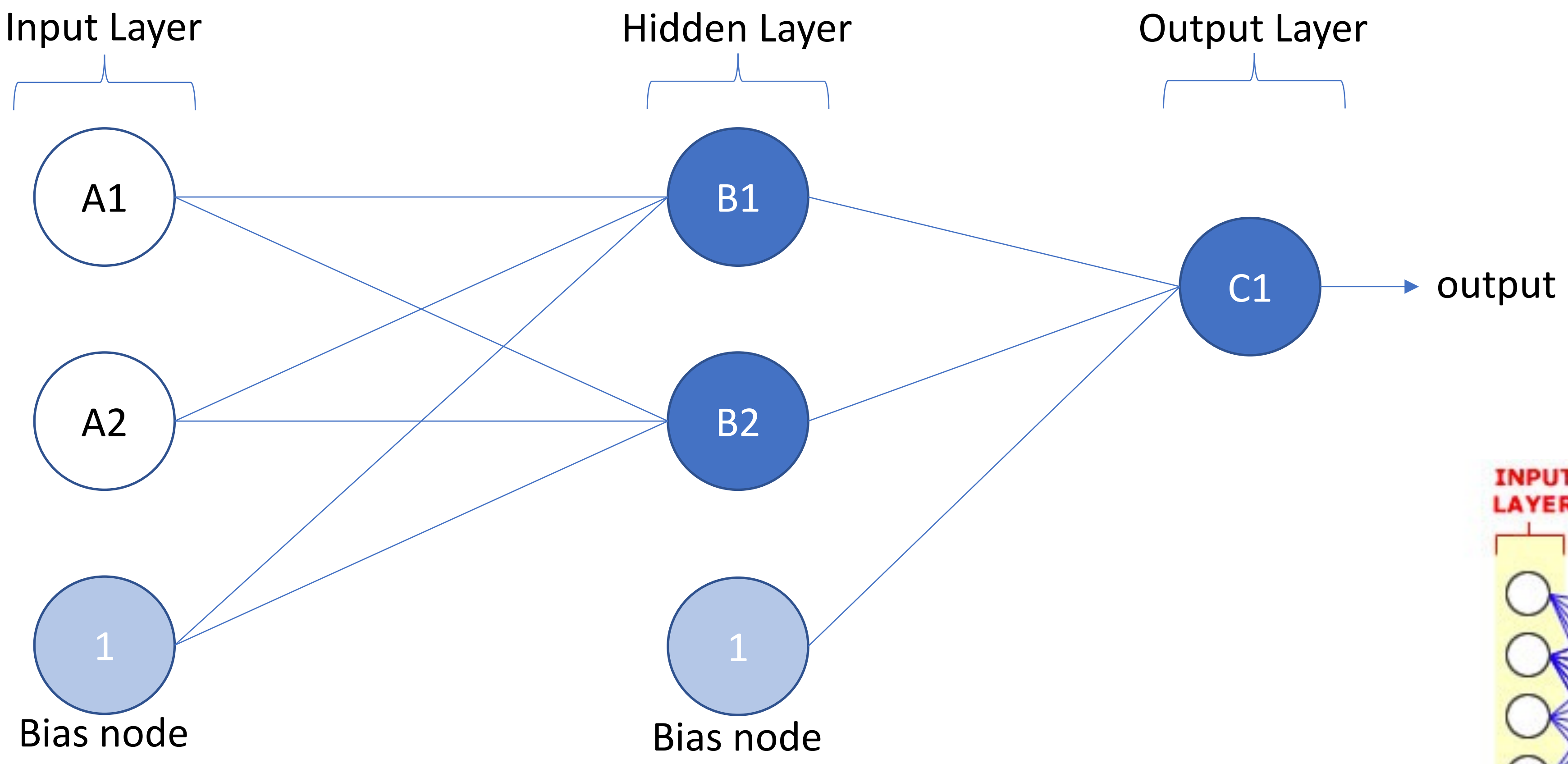
Gradient Descent for Multilayer NN

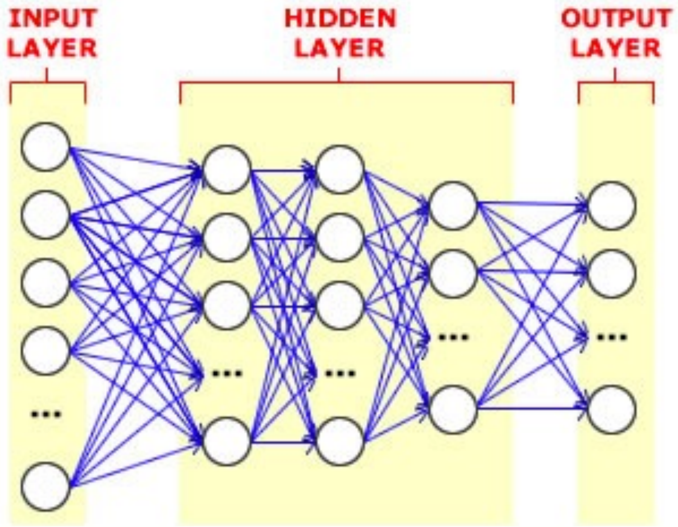
- Weight update: $$w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$$
- Loss/Error function: $$E = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - f\left(\sum_j w_j x_{ij}\right)\right)^2$$
- Activation function f must be differentiable
- For sigmoid function:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda \sum_i \overbrace{(t_i - o_i)}^{\text{From loss}} \underbrace{o_i(1-o_i)}_{\text{From activation (sigmoid)}} x_{ij}$$

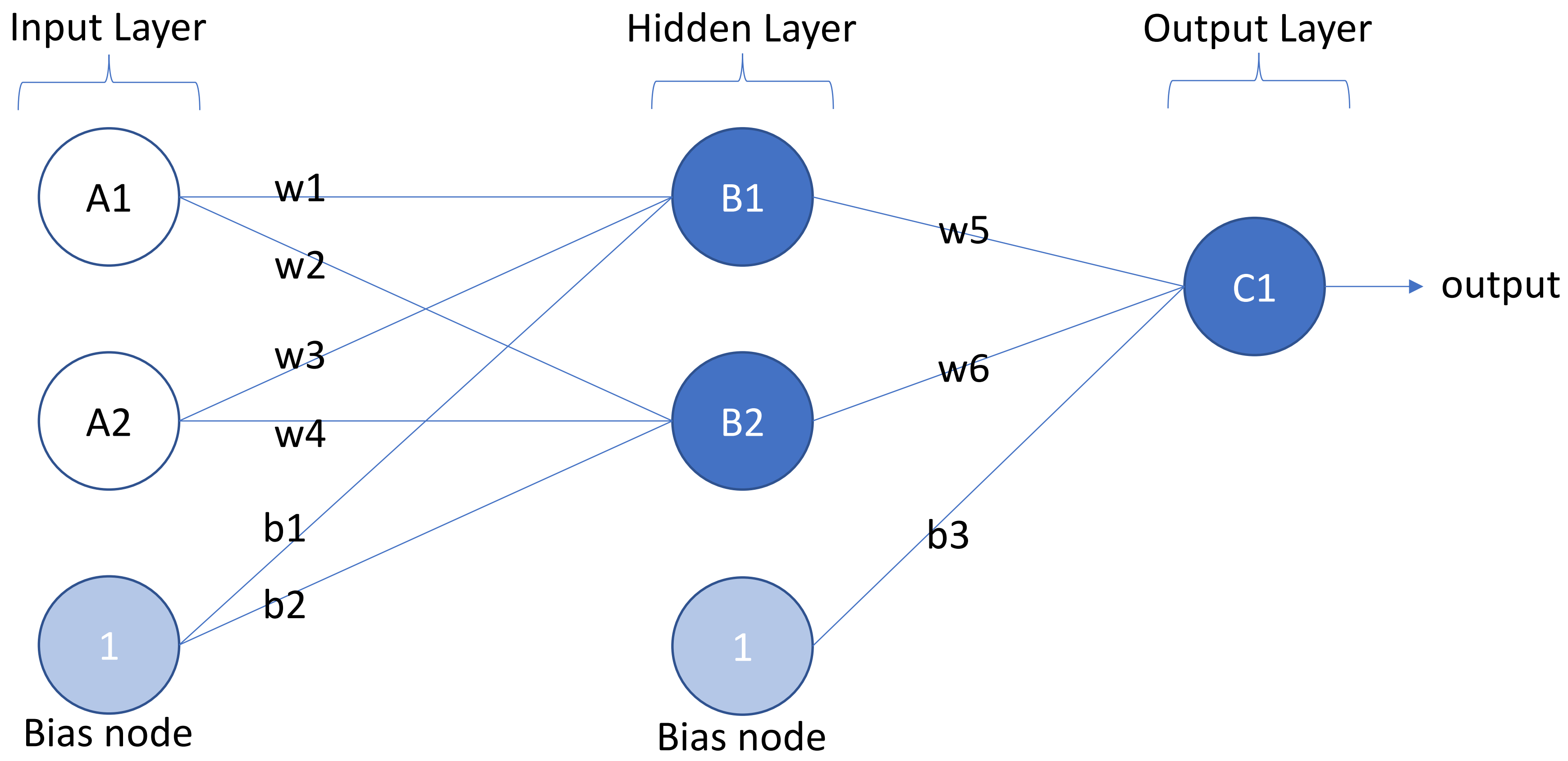
- Stochastic gradient descent (update the weight immediately)

Artificial Neural Network A-Z Example





Weights A-Z Example

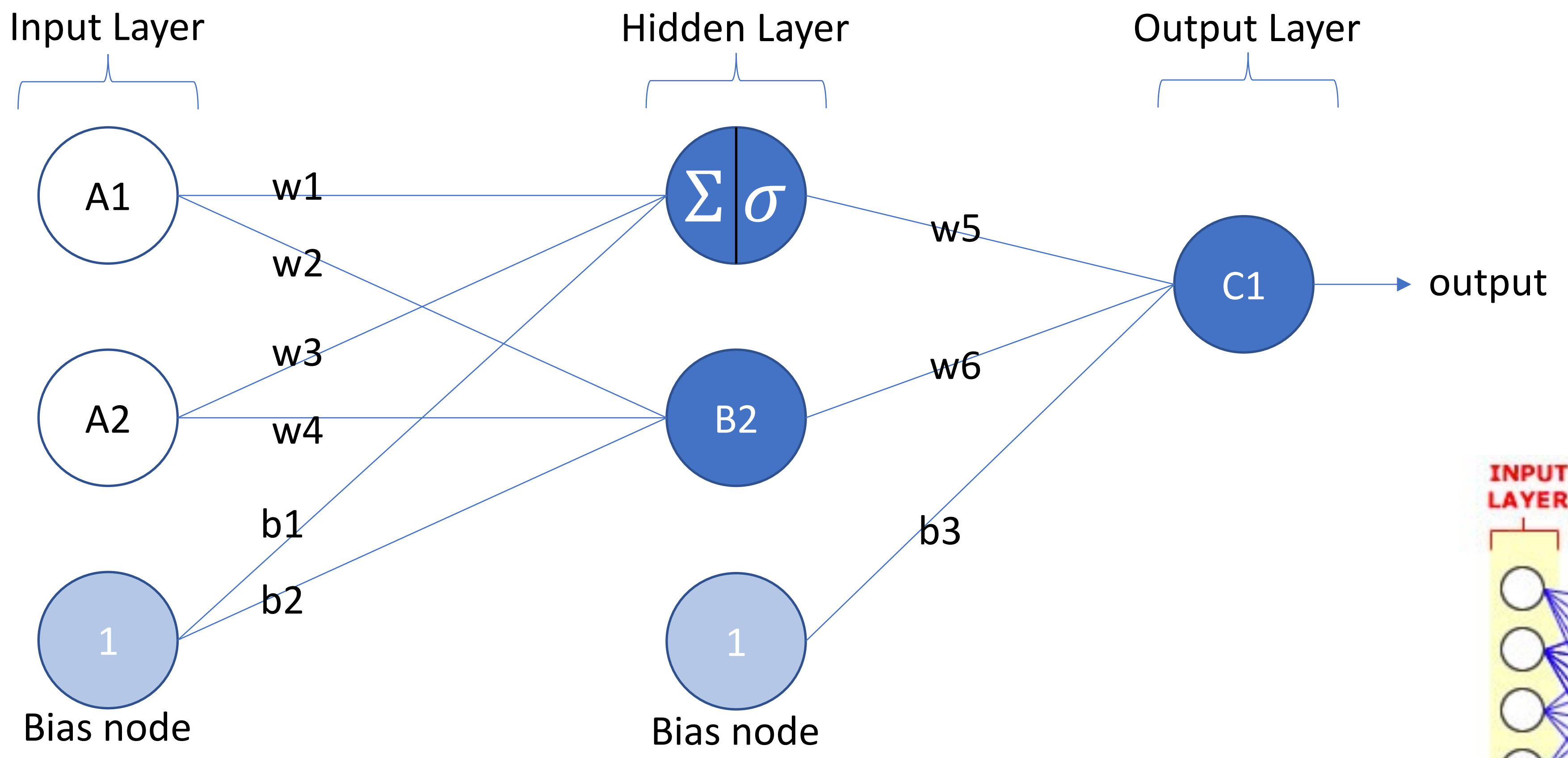


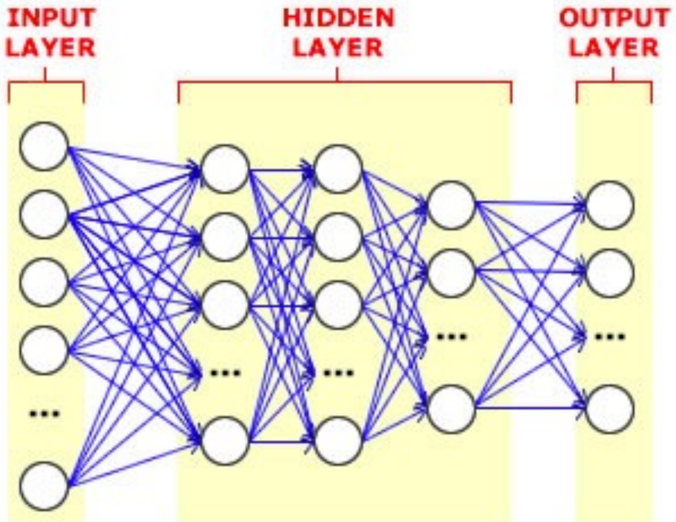
Artificial Neural Nets A-Z Example

- **Step 1: Forward Propagation**
  - **Summation Operator**
  - **Activation Function**
- Step 2: Calculate Total Error
  - Cost Function
- Step 3: Calculate Gradients
  - Partial Derivative
  - Chain Rule
- Step 4: Update Weights
  - Weight Update Formula

Backpropagation

Forward Propagation A-Z Example

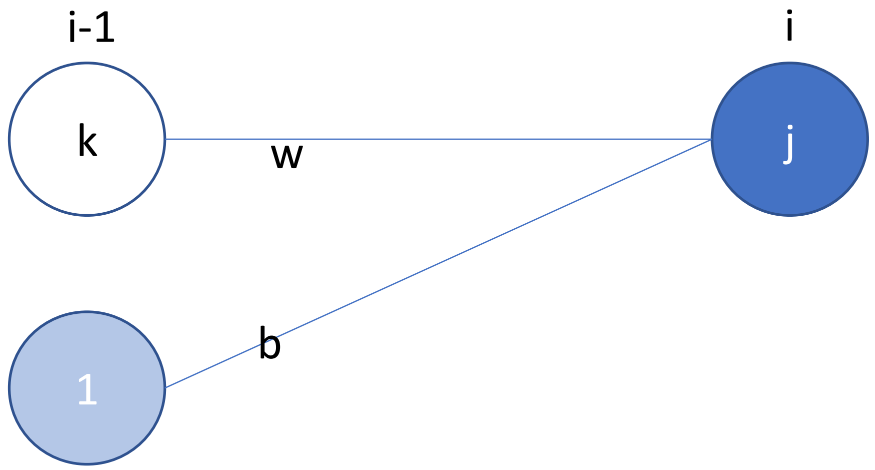




Summation Operator A-Z Example

- Call w^i_{jk} the weight of the connection to the j^{th} node in the i^{th} layer from the k^{th} node in the previous layer
- Call a^i_k the output (activation) of the k^{th} node in the i^{th} layer
- Call b^i_j the bias of the j^{th} node in the i^{th} layer

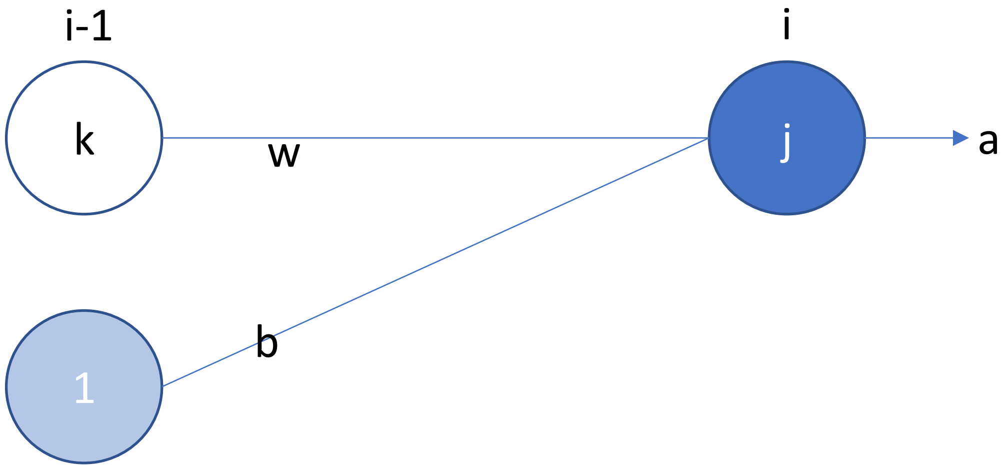
$$net_j = \left(\sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j\right)$$



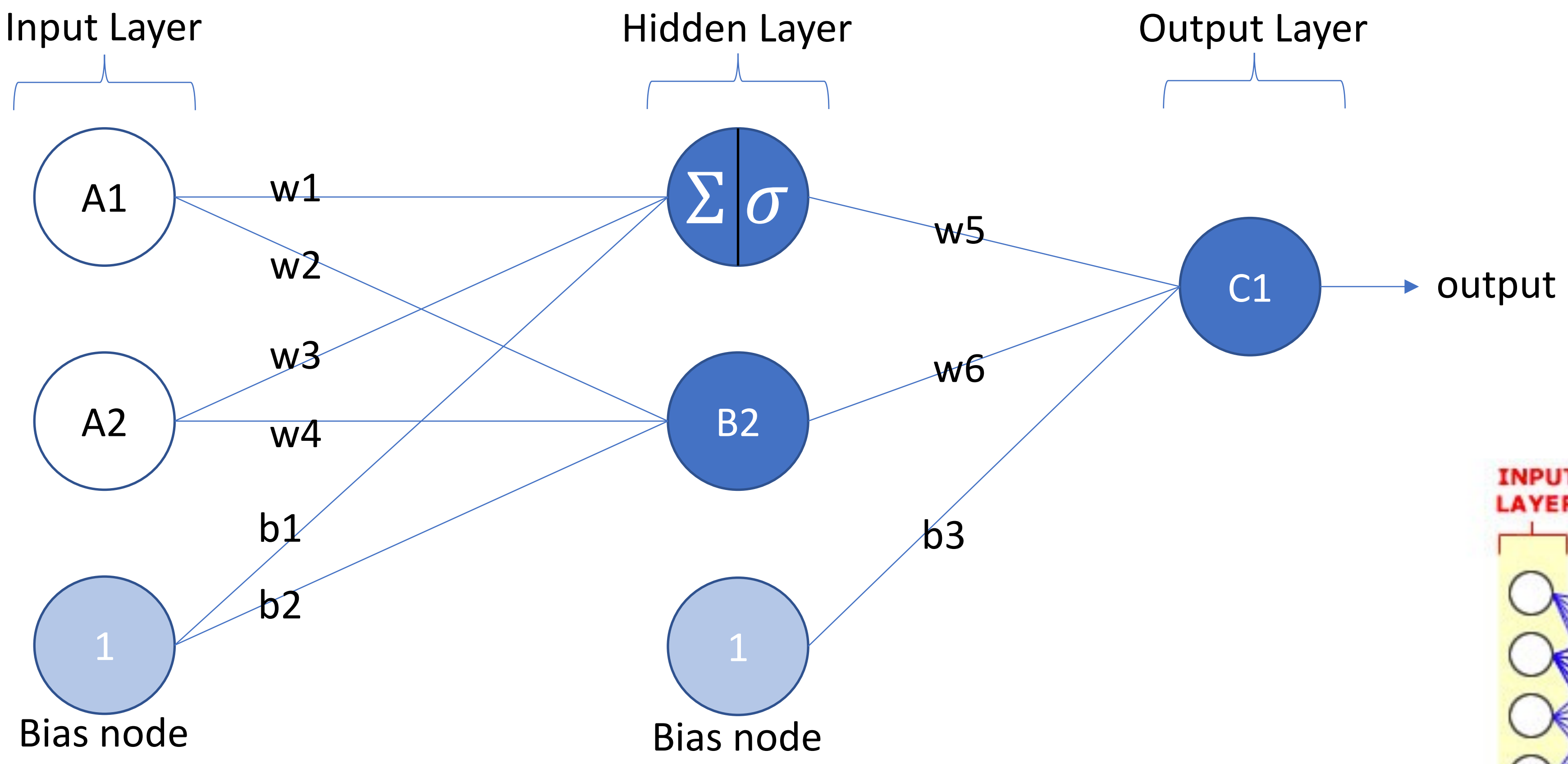
Activation A-Z Example

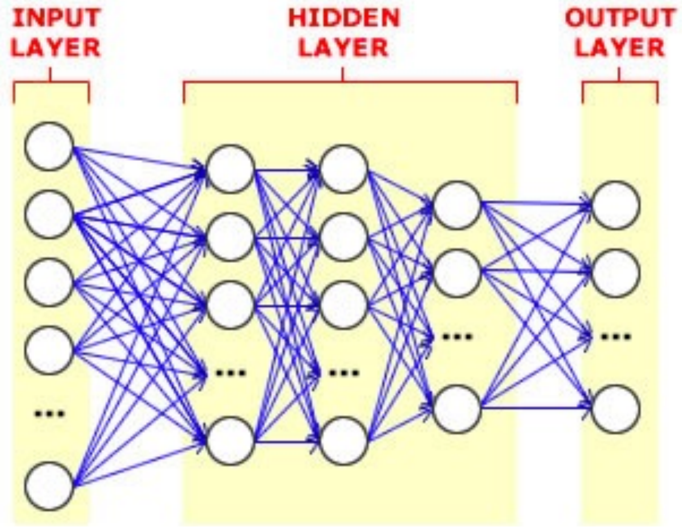
- Call a^i_j the output (activation) of the j^{th} neuron in the i^{th} layer
- Sigma is the activation function

$$a^i_j = \sigma\Big(\sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j\Big)$$



Forward Propagation A-Z Example





Artificial Neural Nets A-Z Example

- Step 1: Forward Propagation
  - Summation Operator
  - Activation Function
- **Step 2: Calculate Total Error**
  - **Cost Function**
- Step 3: Calculate Gradients
  - Partial Derivative
  - Chain Rule
- Step 4: Update Weights
  - Weight Update Formula

Backpropagation

Cost Function (or Loss Function) A-Z Example

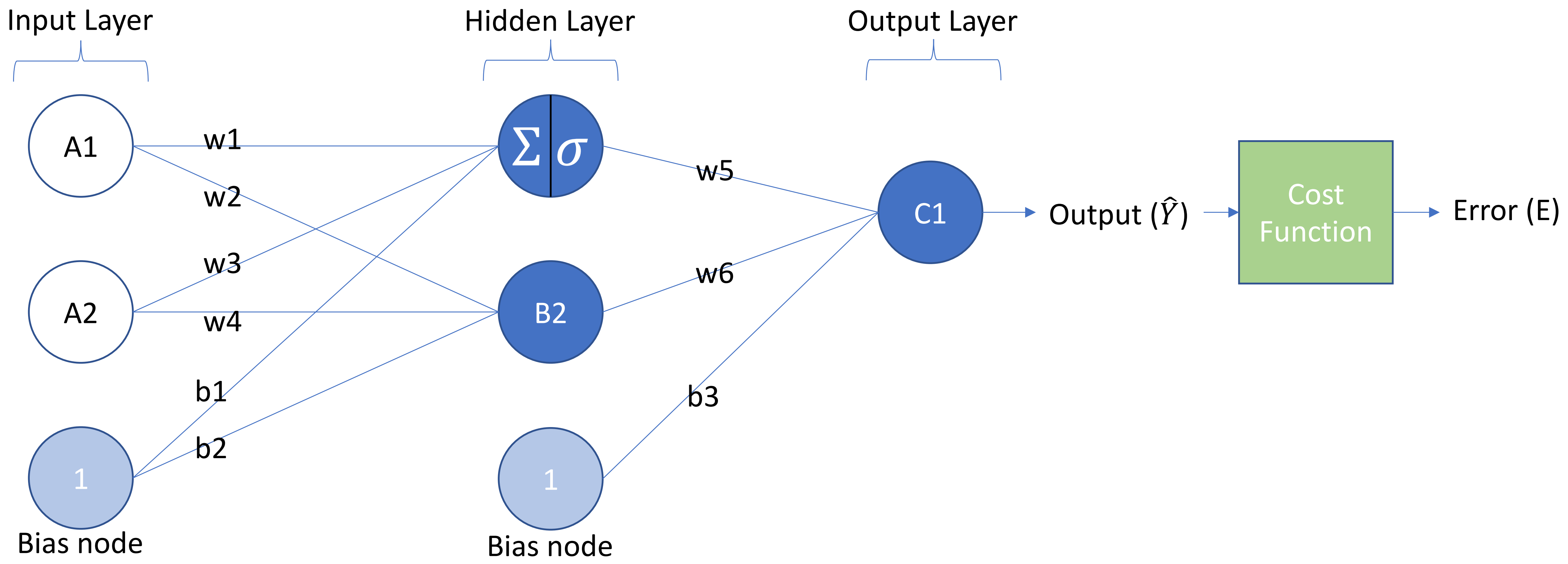
- Mean Squared Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

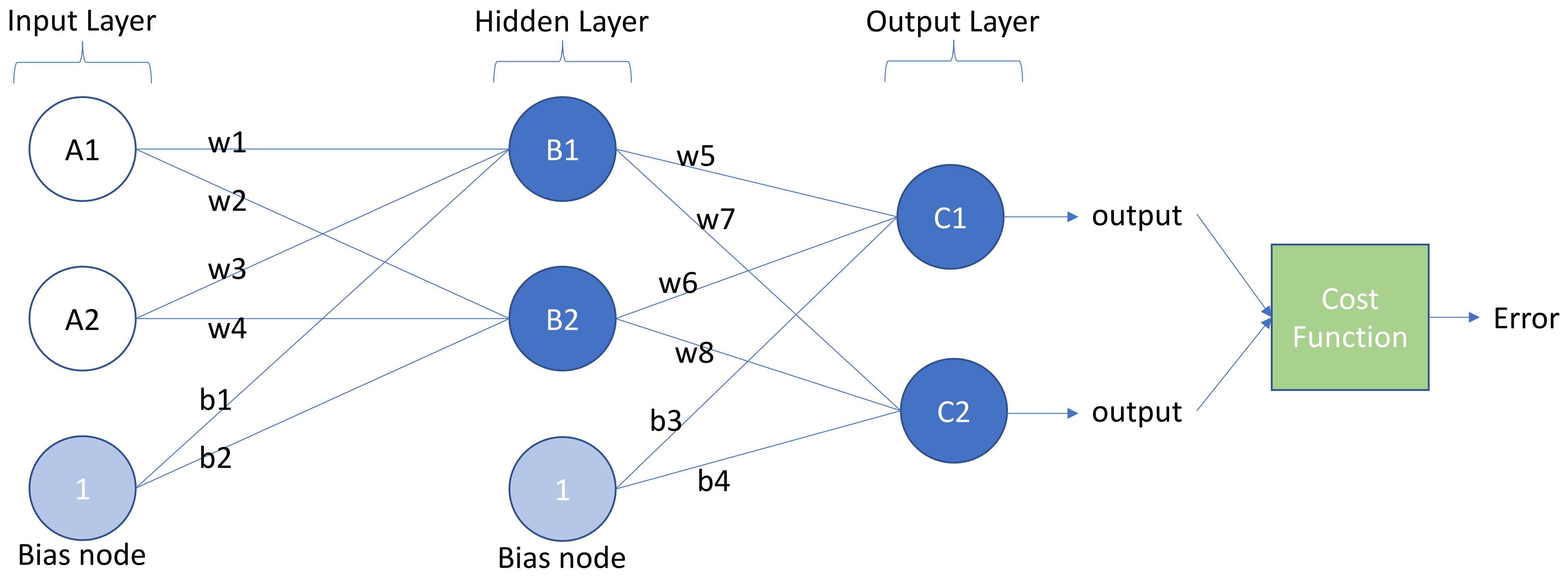
n = number of training examples
$\hat{Y}_i$ = predicted output for example i
Y_i = actual output for example i

[A list of cost functions used in neural networks]

Calculating Error A-Z Example



Calculating Error with multiple outputs A-Z Example

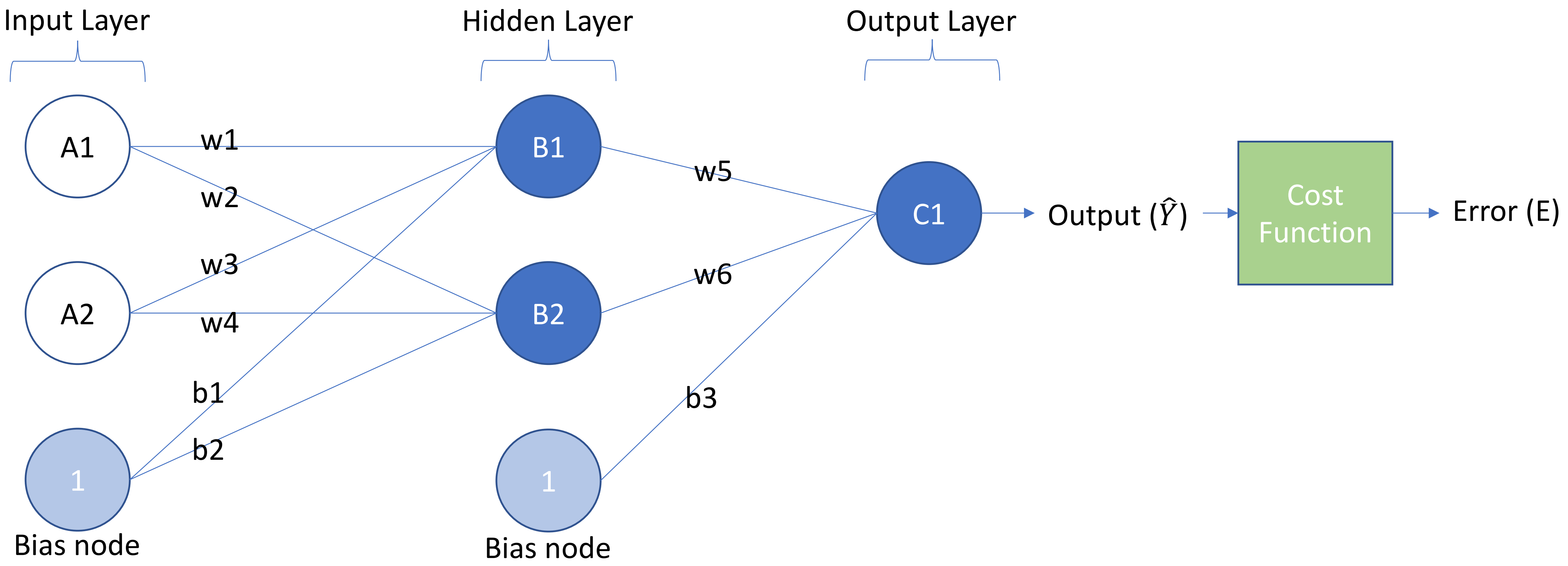


Artificial Neural Nets A-Z Example

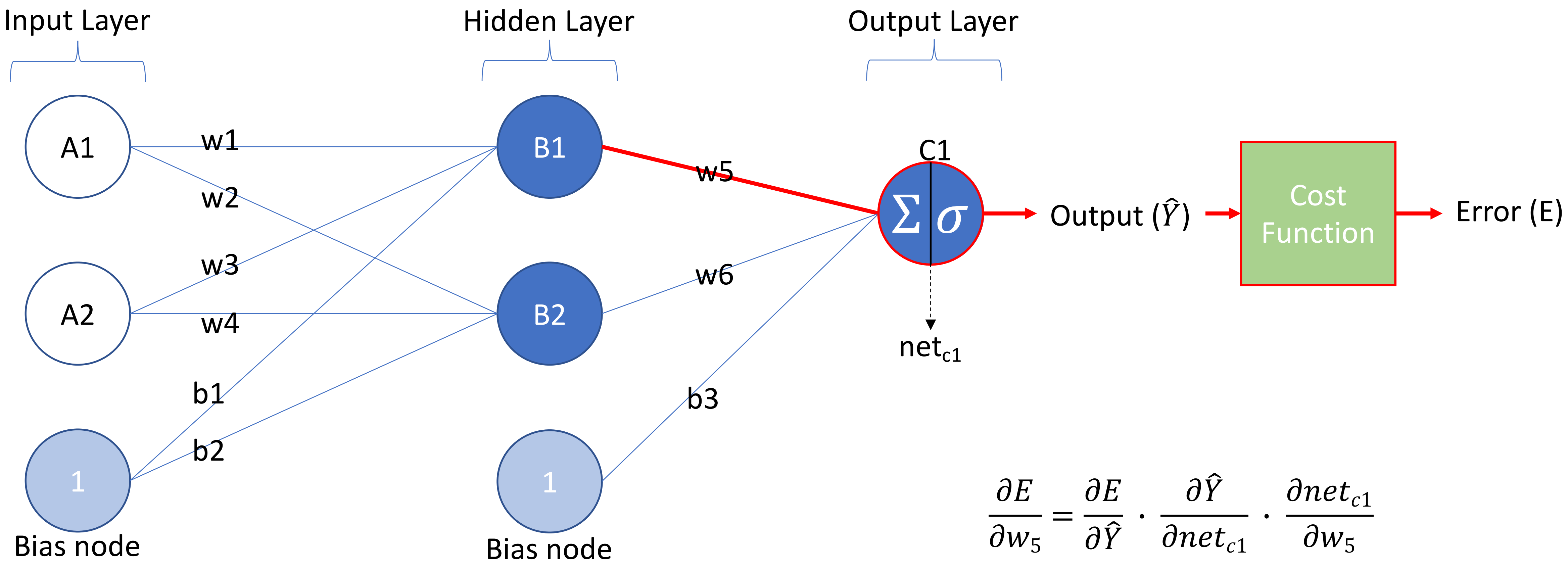
- Step 1: Forward Propagation
  - Summation Operator
  - Activation Function
- Step 2: Calculate Total Error
  - Cost Function
- **Step 3: Calculate Gradients**
  - **Partial Derivative**
  - **Chain Rule**
- Step 4: Update Weights
  - Weight Update Formula

Backpropagation

Calculating Gradients A-Z Example



Calculating Gradients A-Z Example



$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial net_{c1}} \cdot \frac{\partial net_{c1}}{\partial w_5}$$

Calculating Gradients A-Z Example

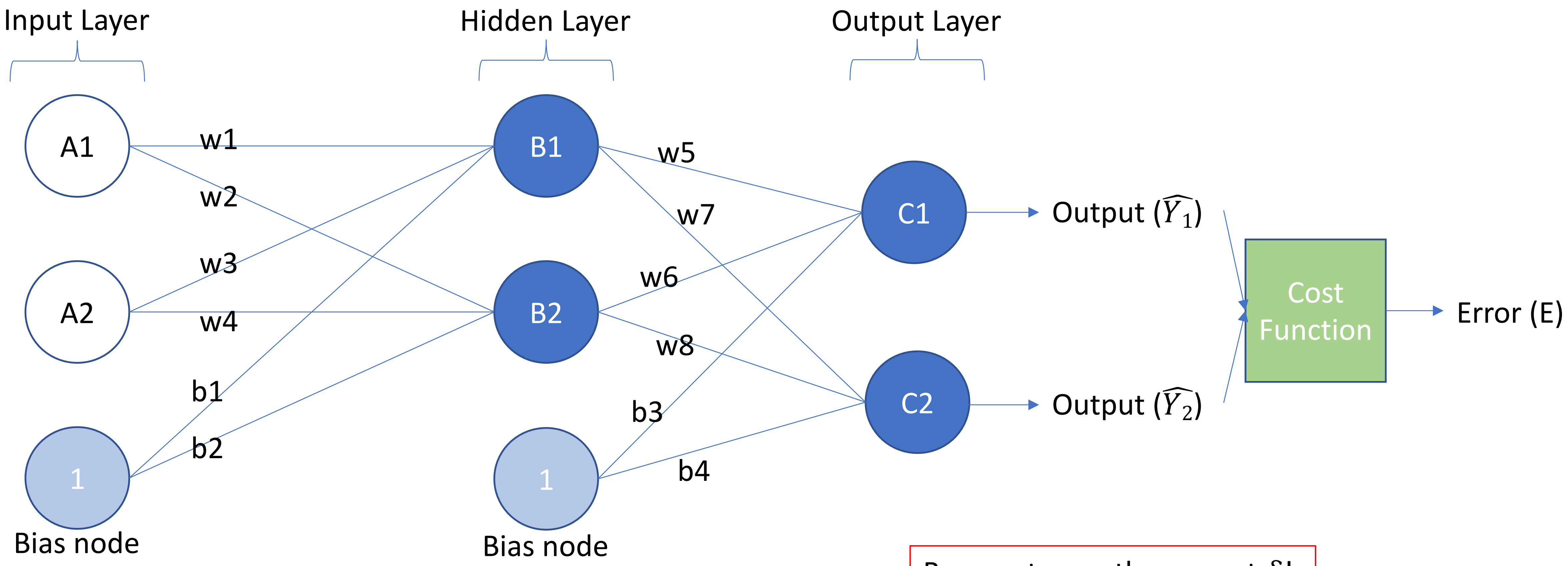
$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial net_{C1}} \cdot \frac{\partial net_{C1}}{\partial w_5}$$

$$\frac{\partial E}{\partial w_5} = (\hat{Y} - Y) \cdot \hat{Y}\left(1 - \hat{Y}\right) \cdot B1_{out}$$

$$\frac{\partial E}{\partial w_5} = \delta_{c1} \, . \, B1_{out}$$

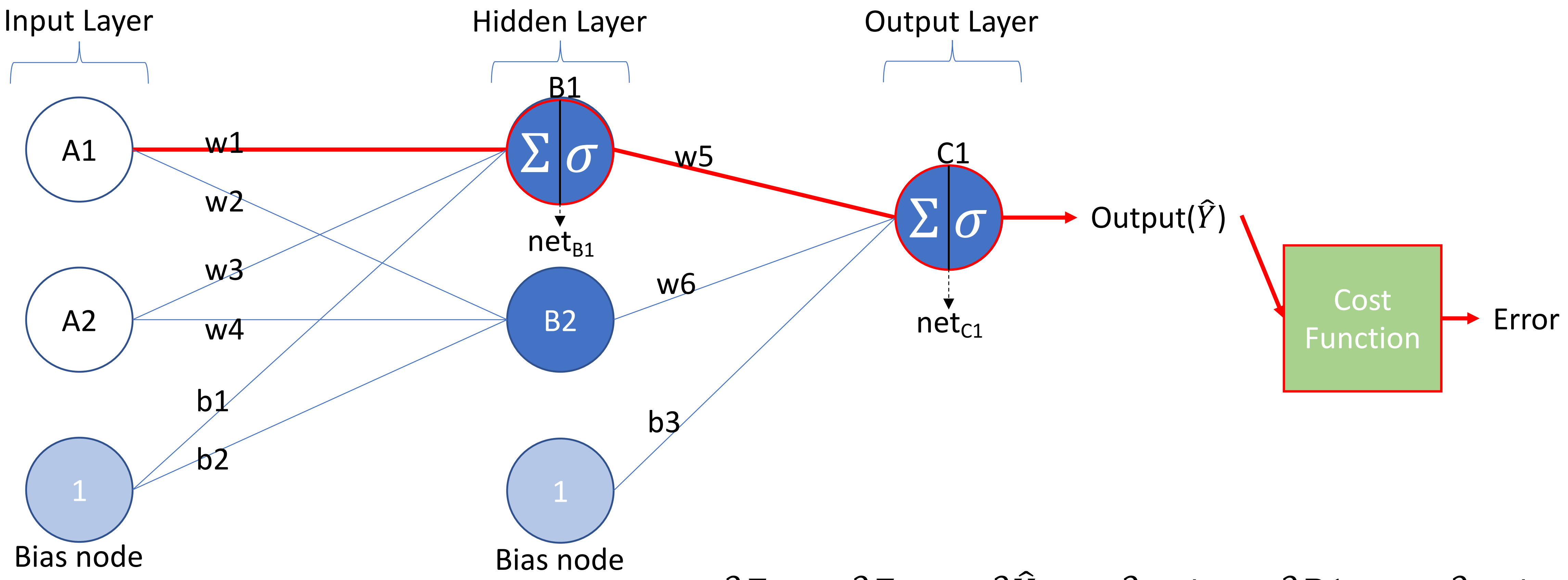
Where $\delta = \underbrace{(\hat{Y} - Y)}_{\text{Derivative of the cost function}} \cdot \underbrace{\hat{Y}\left(1 - \hat{Y}\right)}_{\text{Derivative of the activation function}}$

Calculating Gradients with multiple outputs A-Z Example



Be sure to use the correct δ!
w5, w6, b3 will use C1's delta
w7, w8, b4 will use C2's delta

Calculating Gradients for inner layers A-Z Example



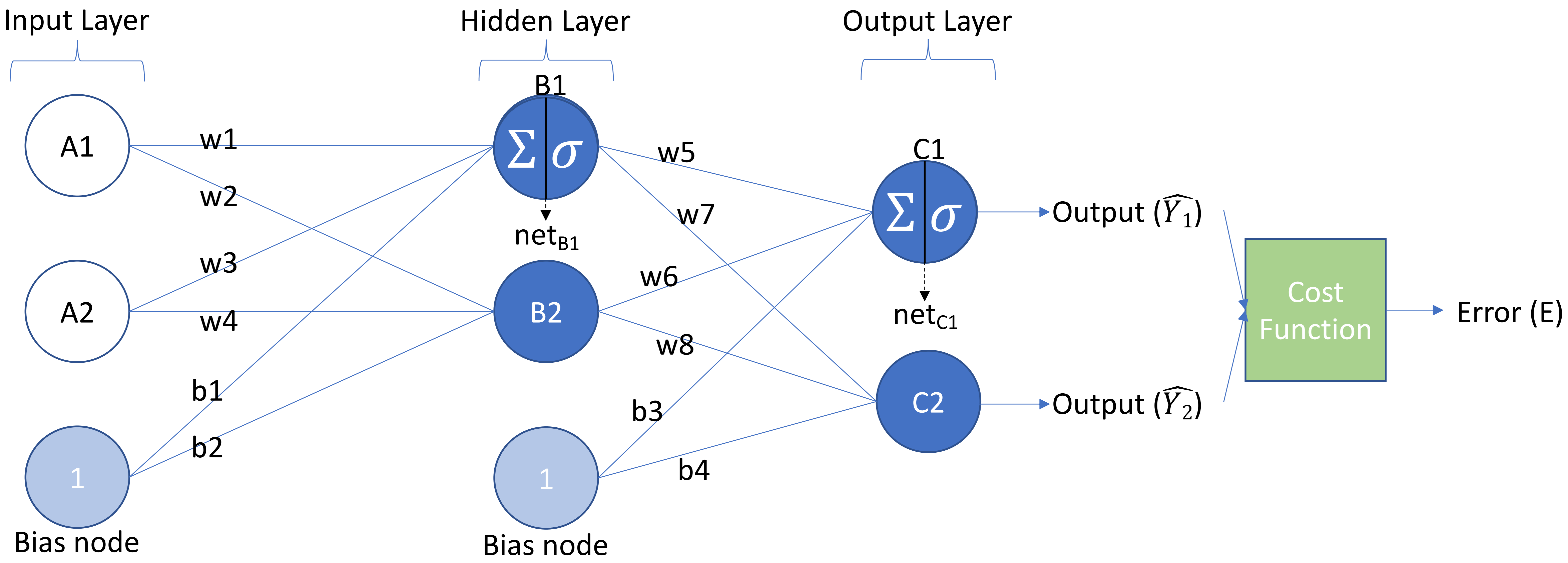
$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial net_{C1}} \cdot \frac{\partial net_{C1}}{\partial B1_{out}} \cdot \frac{\partial B1_{out}}{\partial net_{B1}} \cdot \frac{\partial net_{B1}}{\partial W_1}$$

Calculating Gradients A-Z Example

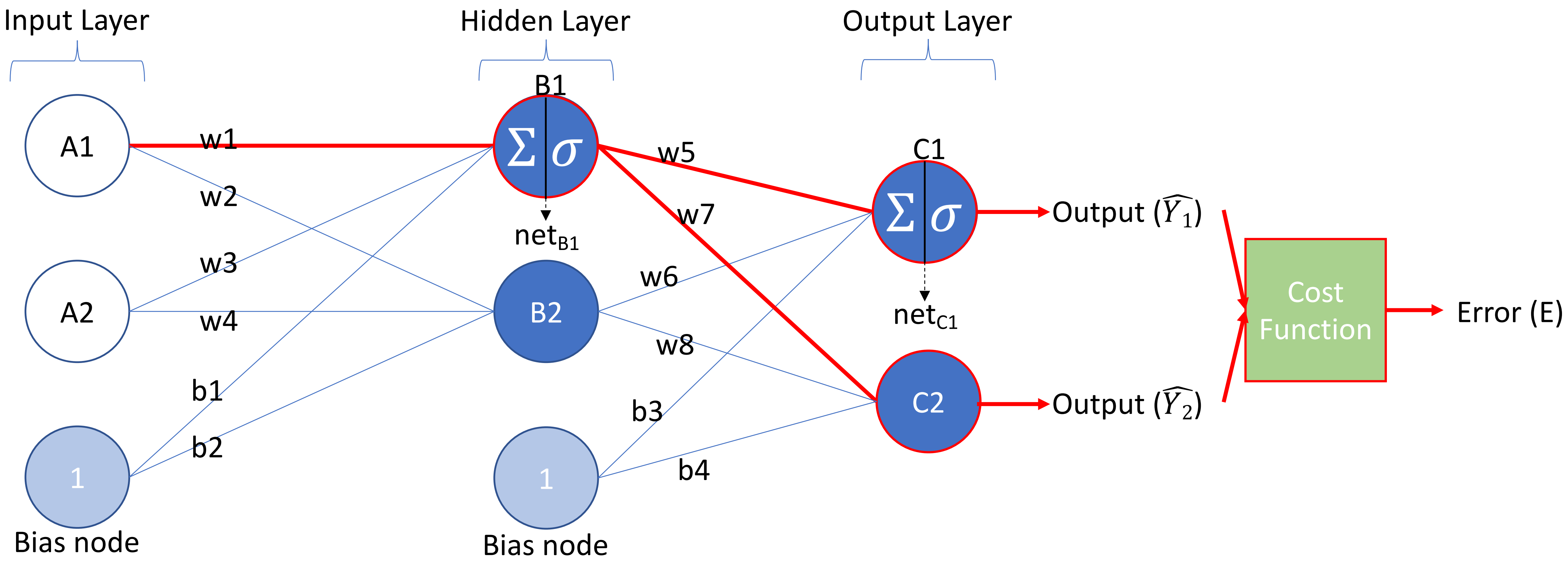
$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial net_{C1}} \cdot \frac{\partial net_{C1}}{\partial B1_{out}} \cdot \frac{\partial B1_{out}}{\partial net_{B1}} \cdot \frac{\partial net_{B1}}{\partial W_1}$$

$$\frac{\partial E}{\partial w_1} = (\hat{Y} - Y) \cdot \hat{Y}\left(1 - \hat{Y}\right) \cdot w_5 \cdot B1_{out}(1 - B1_{out}) \cdot A1_{out}$$

Calculating Gradients for inner layers A-Z Example



Calculating Gradients for inner layers A-Z Example



$$\frac{\partial E}{\partial W_1} = \Sigma\left(\frac{\partial E}{\partial \hat{Y}_i} \cdot \frac{\partial \hat{Y}_i}{\partial net_{Ci}} \cdot \frac{\partial net_{Ci}}{\partial B1_{out}}\right) \cdot \frac{\partial B1_{out}}{\partial net_{B1}} \cdot \frac{\partial net_{B1}}{\partial W_1}$$

Calculating Gradients A-Z Example

$$\frac{\partial E}{\partial w_1} = \Sigma(\frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial net_{Ci}} \cdot \frac{\partial net_{Ci}}{\partial out_{B1}}) \cdot \frac{\partial out_{B1}}{\partial net_{B1}} \cdot \frac{\partial net_{B1}}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = \Sigma\left(\left(\widehat{Y}_i - Y_i\right) \cdot \hat{Y}_i\left(1 - \hat{Y}_i\right) w_j\right) \cdot B1_{out(}1 - B1_{out)} \cdot A1_{out}$$

$$\frac{\partial E}{\partial w_1} = \Sigma(\delta_{ci}\, w_j) \cdot B1_{out(}1 - B1_{out)} \cdot A1_{out}$$

Where $\delta_{ci} = (\widehat{Y}_i - Y_i) \cdot \widehat{Y}_i\left(1 - \widehat{Y}_i\right)$

$$\frac{\partial E}{\partial w_1} = \delta_{B1} \,.\, A1_{out}$$

Where $\delta_{Bi} = \Sigma(\delta_c\, w_j) \cdot Bi_{out}(1 - Biout)$

Artificial Neural Nets A-Z Example

- Step 1: Forward Propagation
  - Summation Operator
  - Activation Function
- Step 2: Calculate Total Error
  - Cost Function
- Step 3: Calculate Gradients
  - Partial Derivative
  - Chain Rule
- **Step 4: Update Weights**
  - **Weight Update Formula**

Backpropagation

Updating Weights A-Z Example

$$w_{new} = w - \alpha \frac{\partial E}{\partial w}$$

Where α is the learning rate

Design Issues in ANN

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - k or $\log_2$ k nodes for each categorical attribute with k values
- Number of nodes in output layer
  - One output for binary class problem
  - k or $\log_2$ k nodes for k-class problem
- Number of nodes in hidden layer
- Initial weights and biases

Characteristics of Neural Networks

- Multi-layer neural networks with at least one hidden layer can learn complex and diverse decision boundaries
- Susceptible to overfitting
- Can handle irrelevant attributes by using zero weights; can handle redundant attributes by using similar weights
- Can get stuck at a local minima (a non-optimal solution)
- Training is time consuming and requires a lot of data
- Difficult to interpret the results
- Difficult to handle missing attributes


DEEP LEARNING,
SO HOT RIGHT NOW
imgflip.com

Recent Noteworthy Developments in ANN

- Use in deep learning and unsupervised feature learning
  - Seek to automatically learn a good representation of the input from unlabeled data
- Google Brain project
  - Learned the concept of a ‘cat’ by looking at unlabeled pictures from YouTube
  - One billion connection network

Deep Neural Networks

- Involve a large number of hidden layers
- Can represent features at multiple levels of abstraction
- Often require fewer nodes per layer to achieve generalization performance similar to shallow networks
- Deep networks have become the technique of choice for complex problems such as vision and language processing

Deep Nets: Challenges and Solutions

- Challenges
  - Slow convergence
  - Sensitivity to initial values of model parameters
  - The larger number of nodes makes deep networks susceptible to overfitting

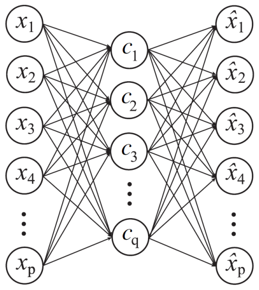
- Solutions
  - Large training data sets
  - Advances in computational power, e.g., GPUs
  - Algorithmic advances
    - New architectures and activation units
    - Better parameter and hyper-parameter selection
    - Regularization

Deep Learning Characteristics

- Pre-training allow deep learning models to reuse previous learning.
  - The learned parameters of the original task are used as initial parameter choices for the target task
  - Particularly useful when the target application has a smaller number of labeled training instances than the one used for pre-training

- Deep learning techniques for regularization help in reducing the model complexity
  - Lower model complexity promotes good generalization performance
  - The dropout method is one regularization approach
  - Regularization is especially important when we have
    - high-dimensional data
    - a small number of training labels
    - the classification problem is inherently difficult.

Deep Learning Characteristics …



Single-layer Autoencoder

- Using an autoencoder for pretraining can
  - Help eliminate irrelevant attributes
  - Reduce the impact of redundant attributes.
- ANN models, especially deep models, can find inferior and locally optimal solutions,
  - Deep learning techniques have been proposed to ensure adequate learning of an ANN
  - Example: Skip connections
- Specialized ANN architectures have been designed to handle various data sets.
  - Convolutional Neural Networks} (CNN) handle two-dimensional gridded data and are used for image processing
  - Recurrent Neural Network handles sequences and are used to process speech and language