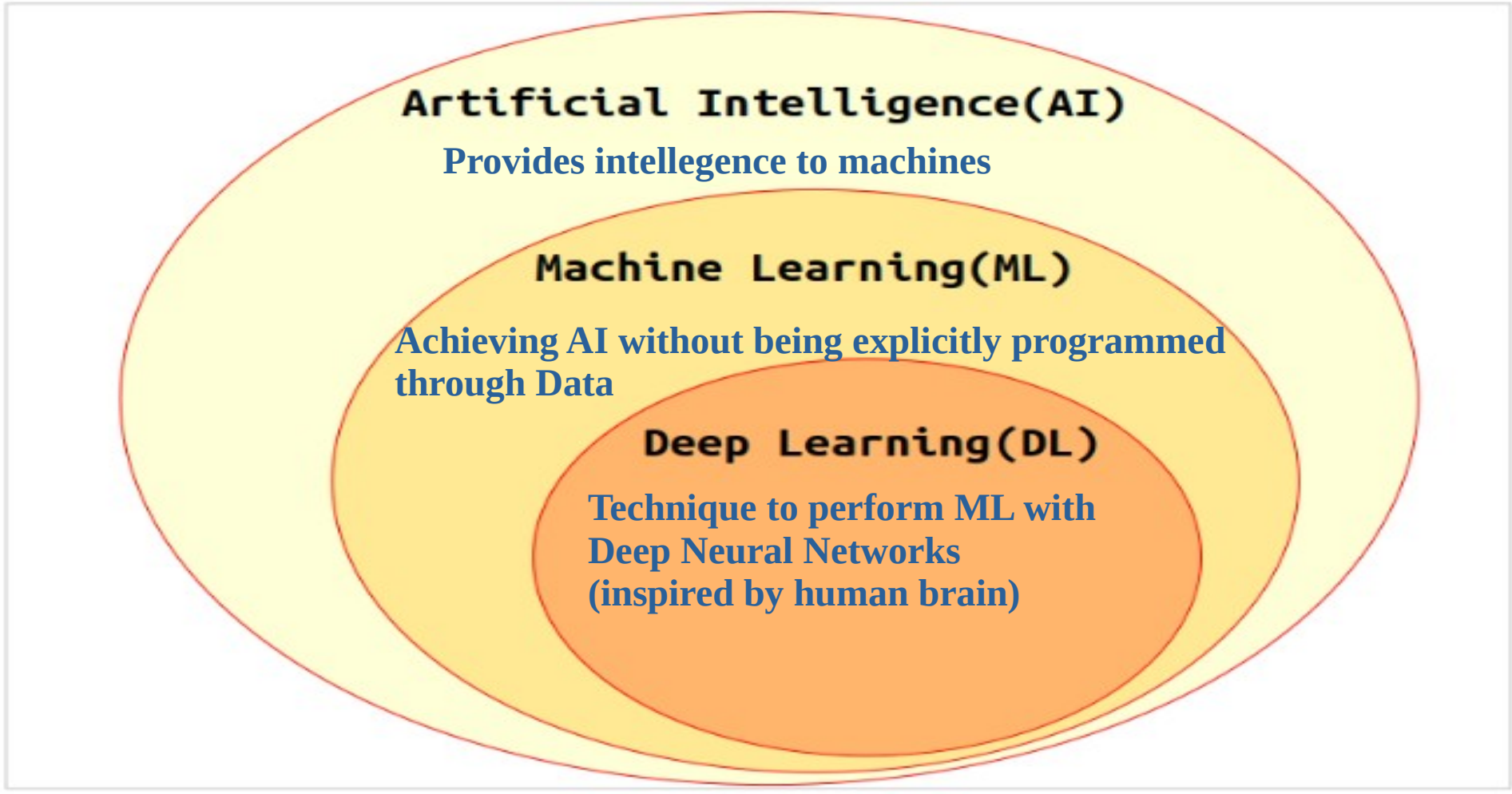
Introduction to Deep Learning

Deep Learning

- Deep learning is part of a broader family of machine learning methods based on artificial neural networks
- The word "deep" in "deep learning" refers to the number of layers through which the data is transformed
- Generally Deep Learning methods involve Neural Networks with more number of layers.

What is the Difference between AI,ML and DL?



What is the Difference between AI,ML and DL?

- AI, ML and DL does the same thing(providing intellegence to Machines).
- **Artificial Intelligence** involves machines that can perform tasks that are characteristic of human intelligence.
- like understanding language, recognizing objects and sounds, learning, and problem solving.
- **Machine Learning** is simply a way of achieving AI, the ability to learn without being explicitly programmed

What is the Difference between AI,ML and DL?

- You can get AI without using machine learning
- But this would require building millions of lines of codes with complex rules and decision-trees.
- So instead of hard coding software routines with specific instructions to accomplish a particular task.
- Machine learning is a way of **training** an algorithm so that it can learn.
- **Training** involves feeding huge amounts of data to the algorithm and allowing the algorithm to adjust itself and improve

What is the Difference between AI,ML and DL?

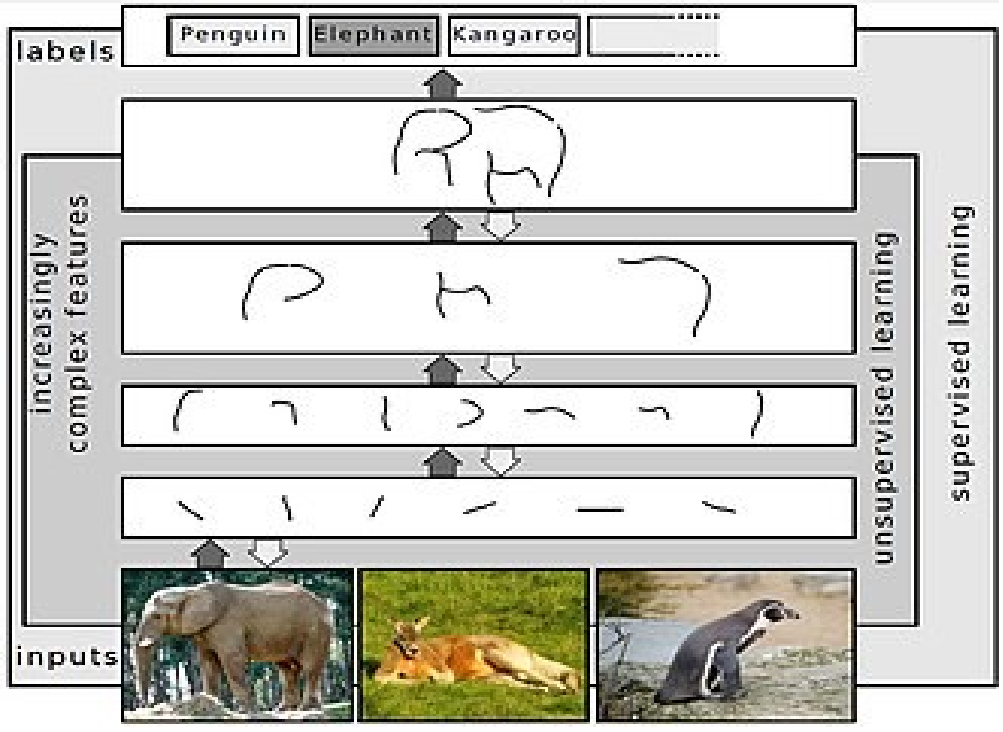
- Example: Computer Vision
- The ability of a machine to recognize an object in an image or video
- For example, the humans might tag pictures that have a cat in them versus those that do not
- algorithm tries to build a model that can accurately tag a picture as containing a cat or not as well as a human
- Once the accuracy level is high enough, the machine has now learned what a cat looks like

What is the Difference between AI,ML and DL?

- **Deep learning** is one of many approaches to machine learning
- Other approaches include decision tree learning, inductive logic programming, clustering, reinforcement learning, and Bayesian networks.
- Deep learning was inspired by the structure and function of the brain, namely the interconnecting of many neurons.
- Artificial Neural Networks (ANNs) are algorithms that mimic the human brain
- ANNs are a collection of connected units or nodes called artificial neurons arranged as layers.

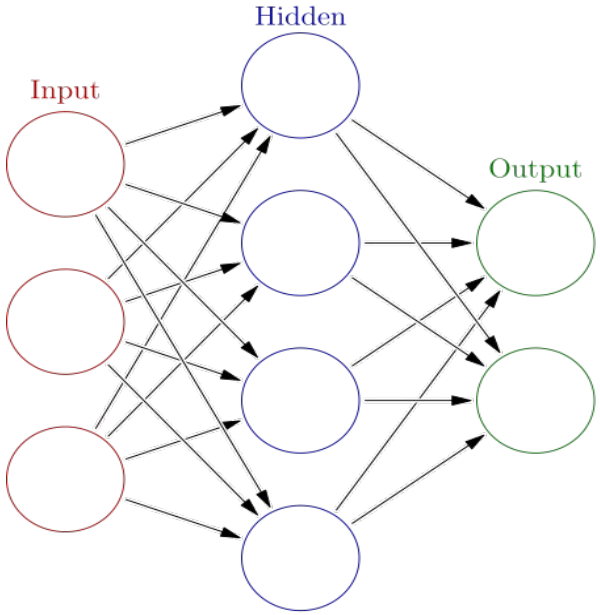
What is the Difference between AI,ML and DL?

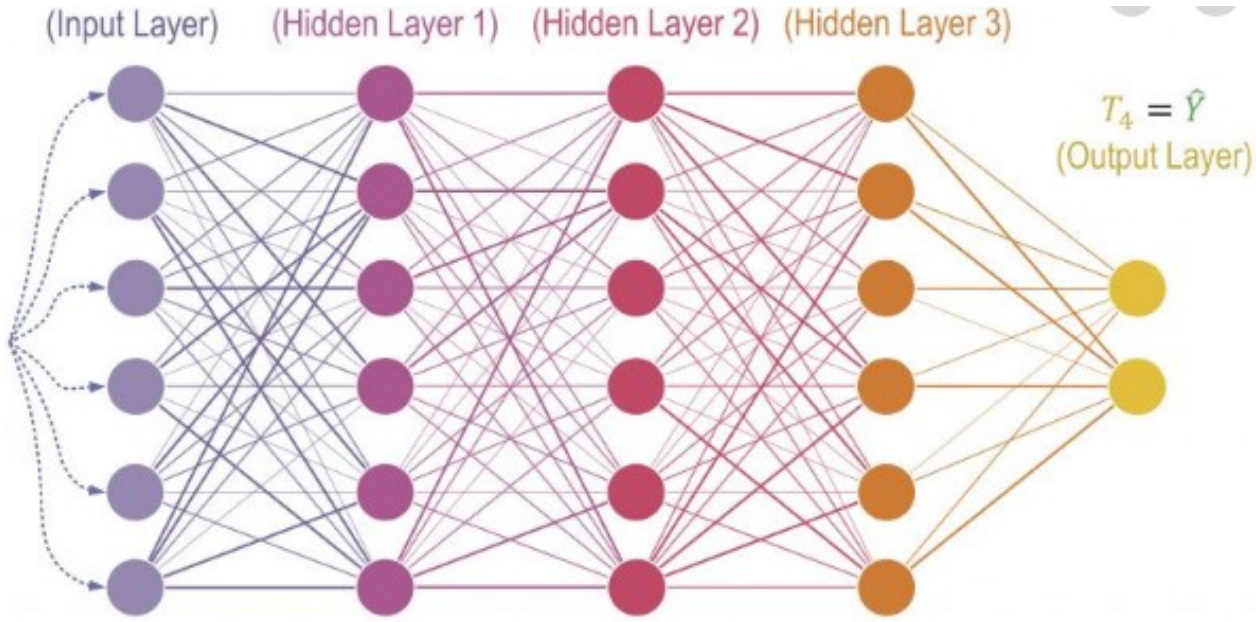
- Each layer picks out a specific feature to learn, such as curves/edges in image recognition



What is the Difference between AI,ML and DL?

- The word "deep" in "deep learning" refers to the number of layers through which the data is transforme
- Depth is created by using multiple layers as opposed to a single layer



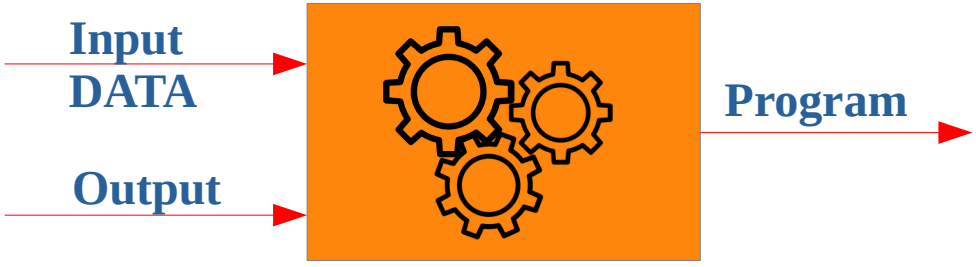


Why we need ML based systems

- There are certain tasks which can’t be programmed as there are no relavent logics describing the task.
- Even if we know the logic to wite program the program might be very complicated
- **Example:**
- Comute the probability that a credit card transaction is fraudulent.
- Fraud is moving target.
- Program needs to keep changing

Why we need ML based systems

- Insted of writing program by hand for each specific task
- We collect lot of examples that specify correct output for given input
- Machine learning Algorithm then takes these examples and produces program that does the job



Why we need ML based systems

- ML systems learn (progressively improve their ability) to do tasks by considering examples, without task specific programming.
- If data changes program can change too by training on new data
- They have found most use in applications difficult to express with a traditional computer algorithm using rule-based programming.

Problems best solved by learning

- Prediction of Future stock prices or currency exchange rates
- Prediction of Which movies will a person like?
- Recognizing Spoken words
- Recognizing facial expressions

Machine Learning Paradigms

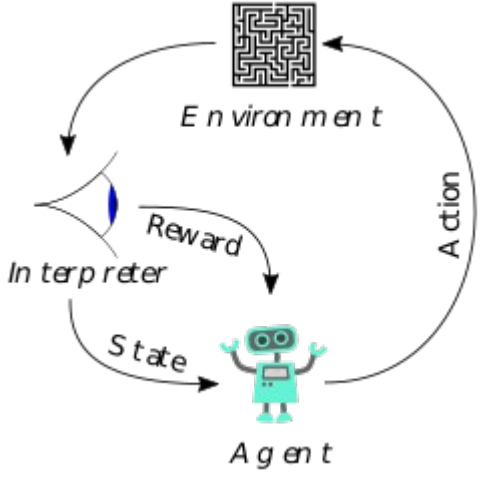
- **Supervised Learning**
  - Supervised learning is the task of inferring a function from labeled training data.
  - Each training data is a pair consisting of an input object (typically a vector) and a desired output value.
  - Regression, Decision Tree, Random Forest, KNN,SVM, Naive Bayes, Logistic Regression etc.
- **Unsupervised Learning**
  - It is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution.
  - Apriori algorithm, K-means,GAN, Autoencoders.

Machine Learning Paradigms

- **Reinforcement Learning**
  - an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent
  - The algorithms learn to react to an environment on their own

Supervised

Unsupervised

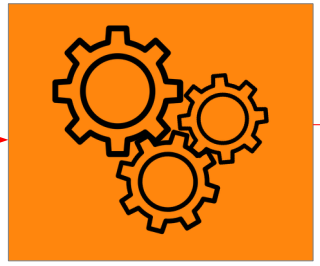


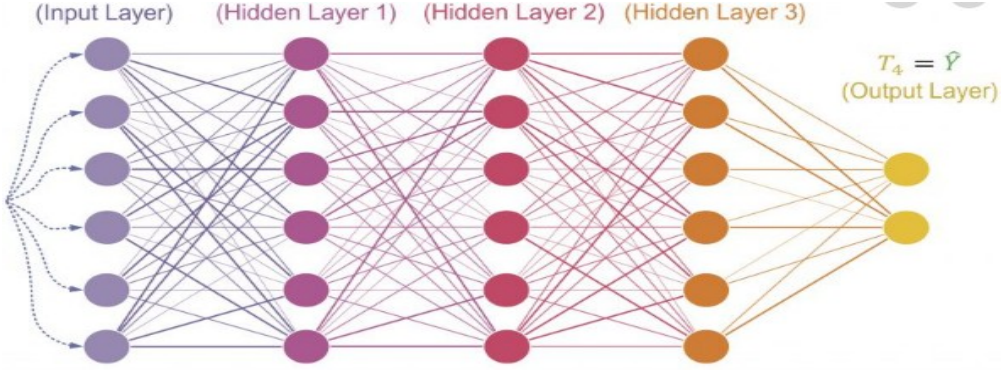
Reinforcedment

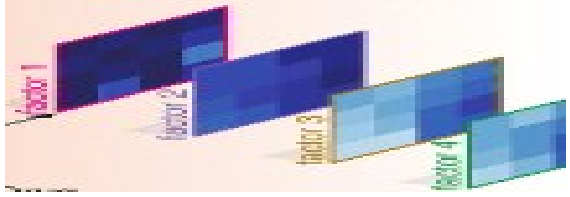
Traditional ML vs Deep Learning

MFCCs, ZCR
Spectral

Feature
Selection







Introduction to Neural Networks

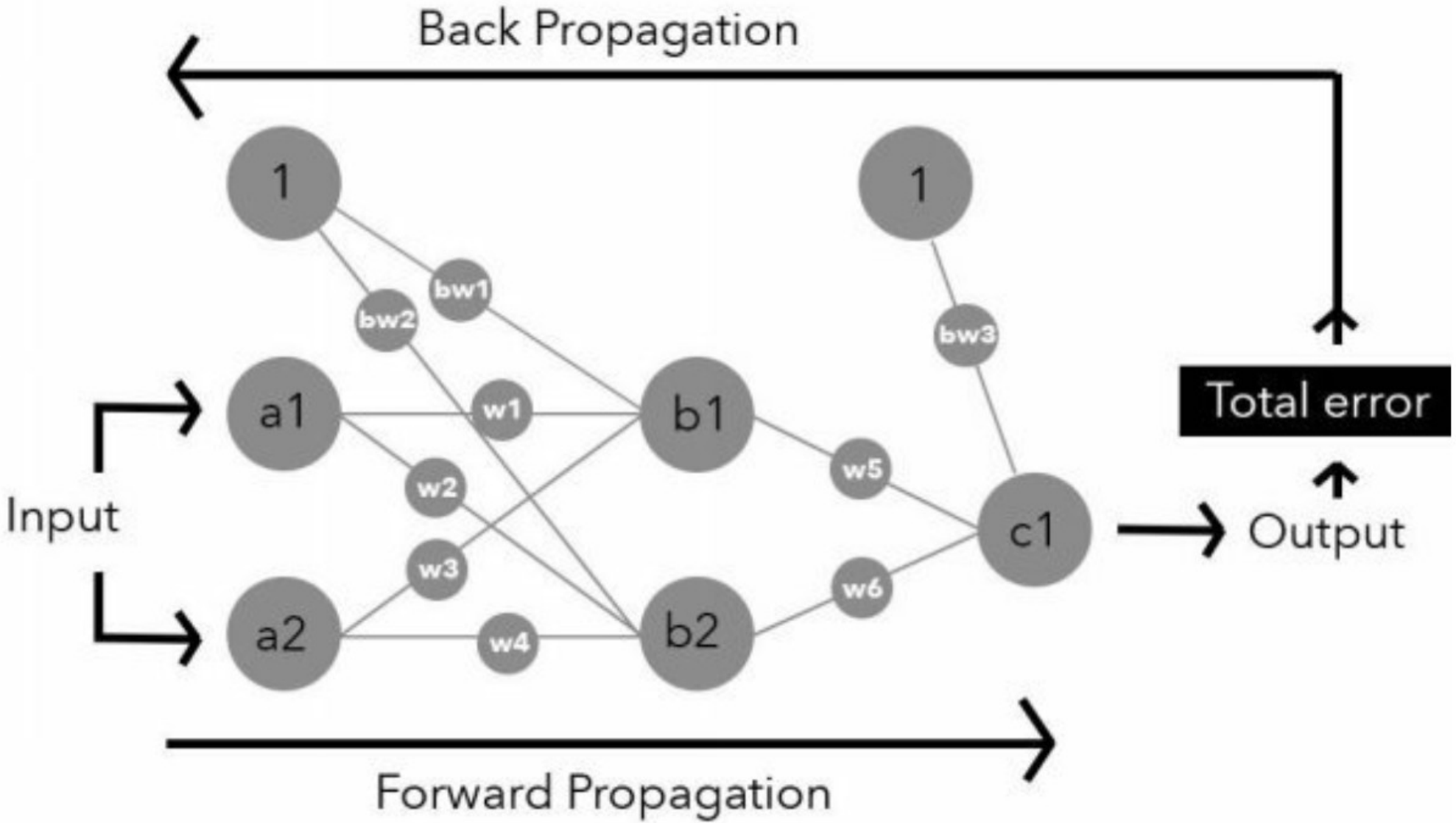
Applications of NN

- Make web searches better
- Organizing photos
- Speech translation
- Generate encryption
- Object detection
- Stock predictions

What does inside NN?

- How do they accomplish these things
- They are mysterious and mind blowing

- On a high level a network learns just like we do through trail and error
- Regardless supervised, unsupervised


Back Propagation
1
1
bw1
bw2
bw3
a1
w1
b1
w5
Total error
w2
Input
c1
Output
w3
w6
a2
w4
b2
Forward Propagation

Grandma’s legendary soup recipe

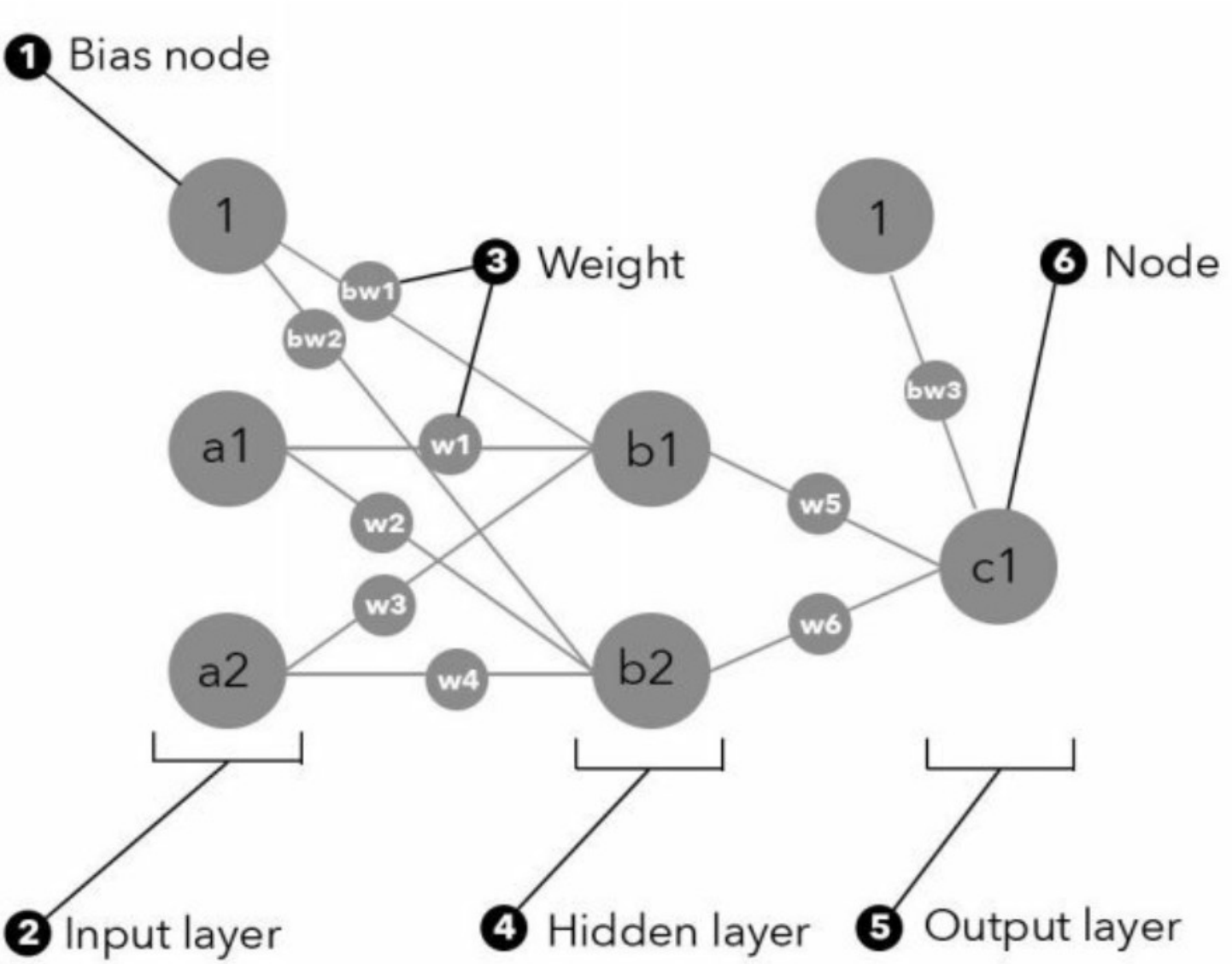
- You have 10 attempts to make the best attempt to grandma soup
- For each attempt your mom and your dad will taste the resulting soup
- And they will give feedback from your last soup
- You have to try change different quantities of ingredients to make soup quite close to grandma’s
- Repeate same process 10 times
- At the last attempt you will have soup quite close to grandma's.

Grandma’s legendary soup recipe

- The inputs are the ingredients
- The weights are the quantities of each ingredient
- The neurons are your different soup attempts.
- The feedbacks are the losses.

Creating Network Structure

- There are many ways to contruct NN
- But all have a structure that is made up of similar parts
- These parts are called **hyper parameters**
- They can help a network successfully train
- hyper parameter Types
  - Required
  - Optional


1 Bias node
1
bw1
3 Weight
bw2
1
6 Node
bw3
a1
w1
b1
w5
w2
c1
w3
w6
a2
w4
b2
2 Input layer
4 Hidden layer
5 Output layer

Creating Network Structure

- Require
  - Total no of input nodes
  - No of hidden layers
  - Total no of nodes in each hidden layer layer
  - Total no of output nodes
  - Weight(Synapse) values
  - Bias values
  - Learning ratel

Creating Network Structure

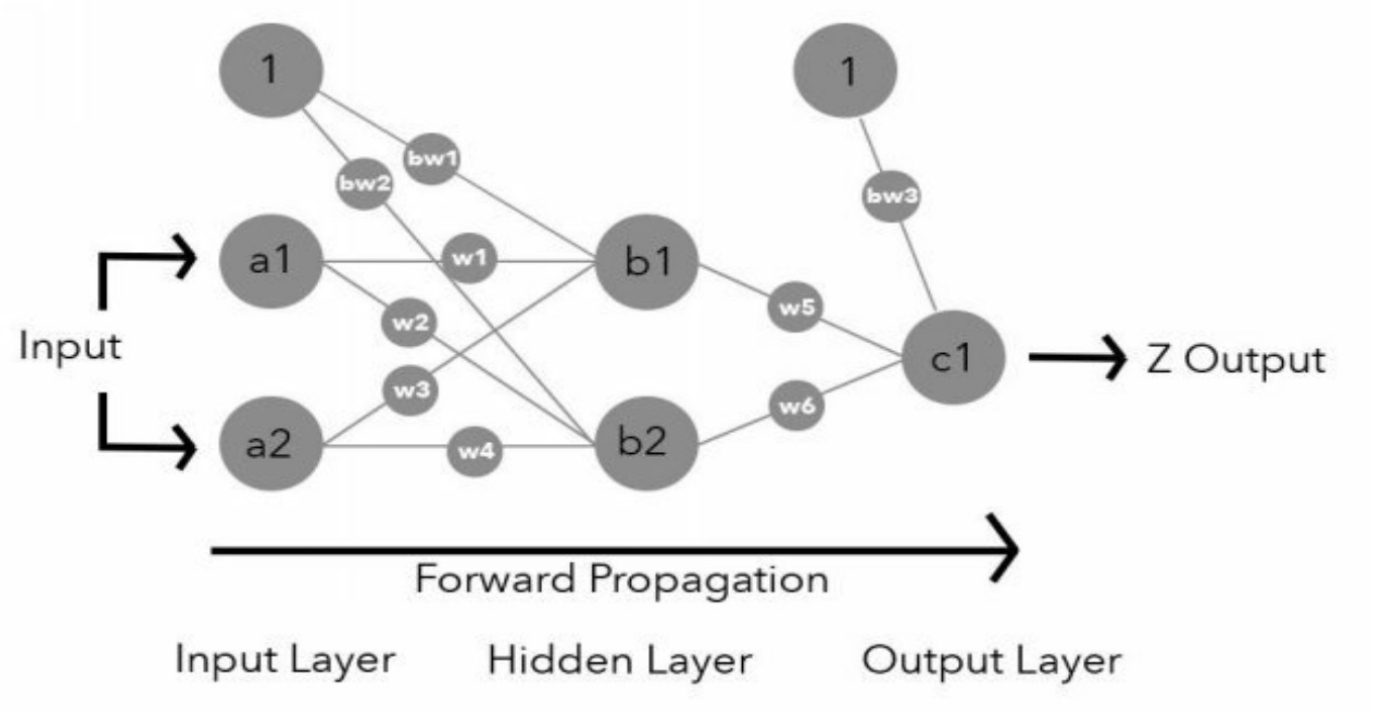
- Optinal
  - Learning rate decay
  - Momentum
  - Mini batch size
  - Weight decay
  - Droup out
- This list is not exhaustive

Stages involved in NN implementation

- **Forward Propagation**
  - **Summation operation**
  - **Activation Function**
  - **Caliculate Total Error**
    - Cost function
- **Back Propagation**
  - **Caliculate Gradients**
    - Partial Derivatives
    - Chain Rule
  - **Updating Weights**
    - Weight update formula

Forward Propagation

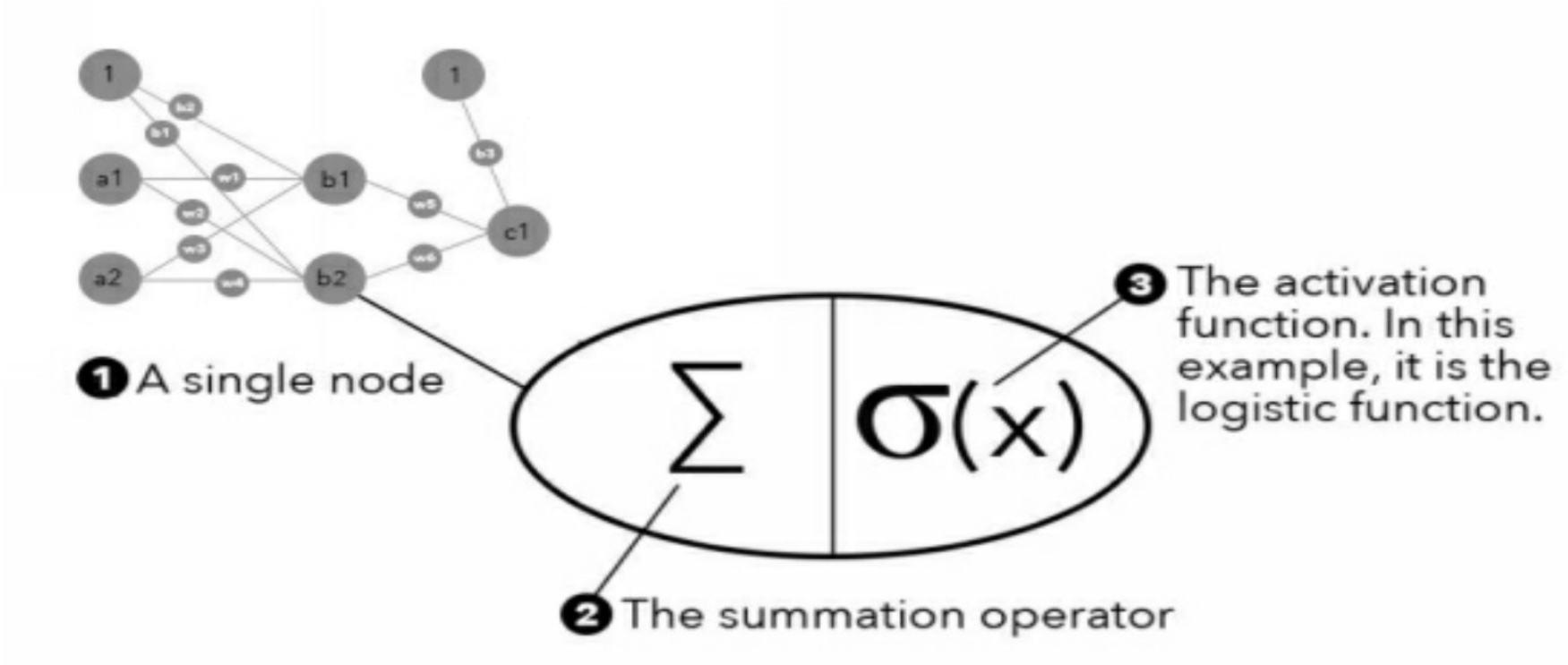
- The process of how input moves through the network to become output is called **Forward Propagation**



Forward Propagation

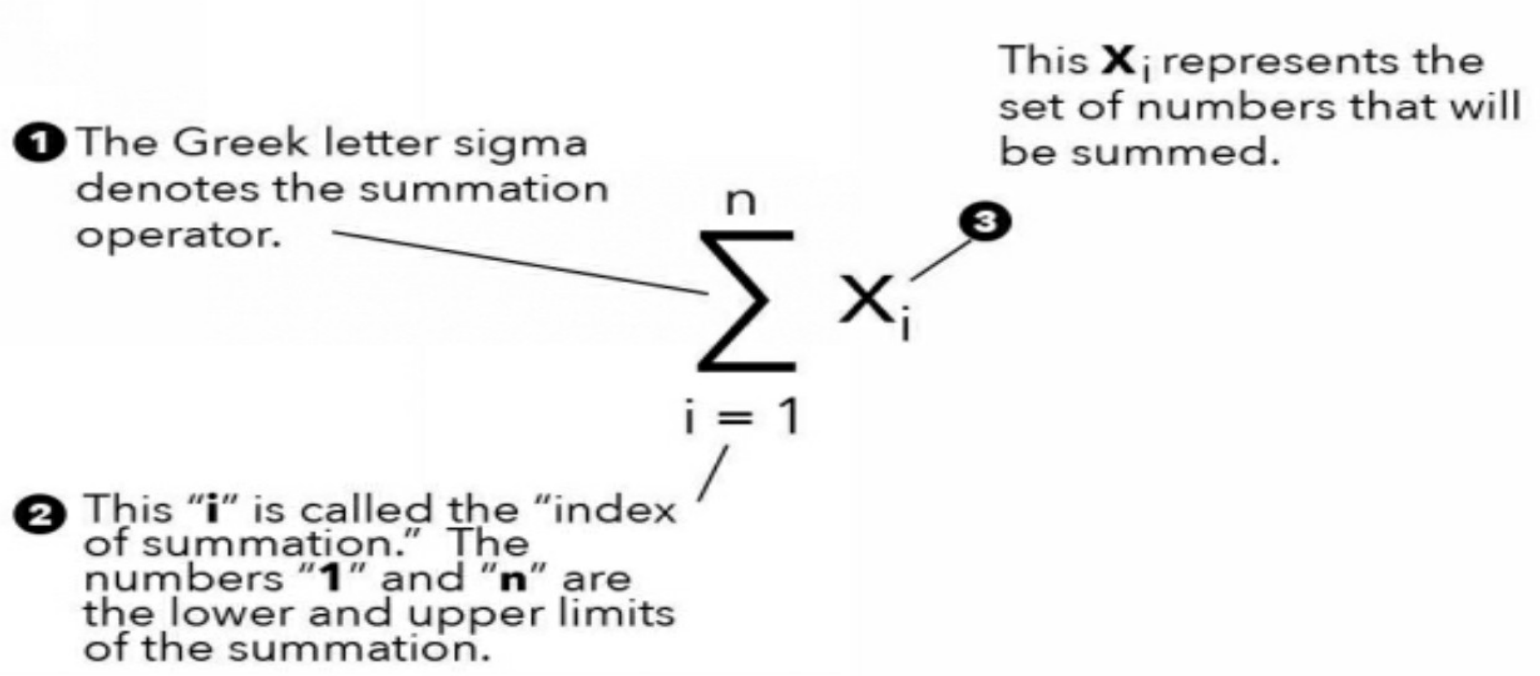
- When input is passed into the network it moves from one layer to next untill it passes through output layer
- Two mathematical functions are used to make all of this possible
- Mathematical functions
  - Summation operator
  - Activation function

Single Node


1
1
a1
a2
b1
b2
c1
1 A single node
2 The summation operator
3 The activation function. In this example, it is the logistic function.
∑
σ(x)

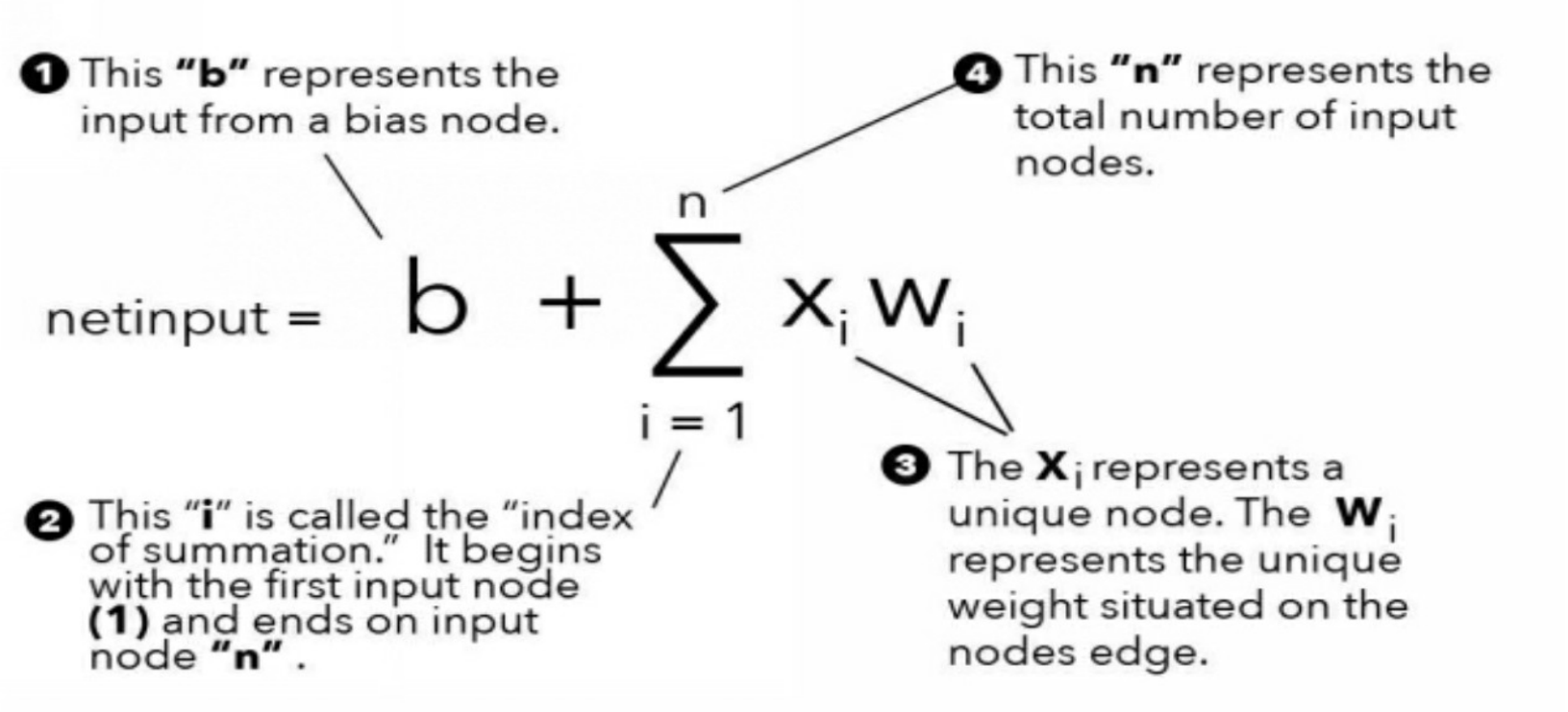
Summation operator

- It sums all of a node's inputs to create a net input
- General Summation Operator



Summation operator

- Caliculationg the net input of a node



Summation Operator

- Bias values is always set to 1(one)
- So output of bias is always one
- Bias is always multiplied with edge weight to become input for a node
- The final output of a bias to a node is often its edge weight value

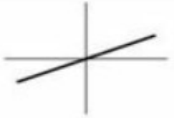
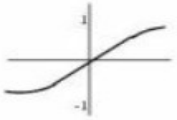
Why this Summation Operator?

- Each hidden and output has multiple input values
- For this input values successfully move through the network and create an output they must be summed and turned into **single value**
- To do this summation operaor make use of matrices
- Output is dot-product

Activation function

- It recieves output of the summation function and transform it into final output of a node
- It squashes input into certain range(i.e how much a node should fire)

Types of Activation functions

Activation Function	Graph	Equation
Linear		$f(x) = x$
Step (Heaviside)		$f(x) = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0, \end{cases}$
Hyperbolic tangent		$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
Rectified Linear Unit (ReLU)		$f(x) = \max(0, x)$

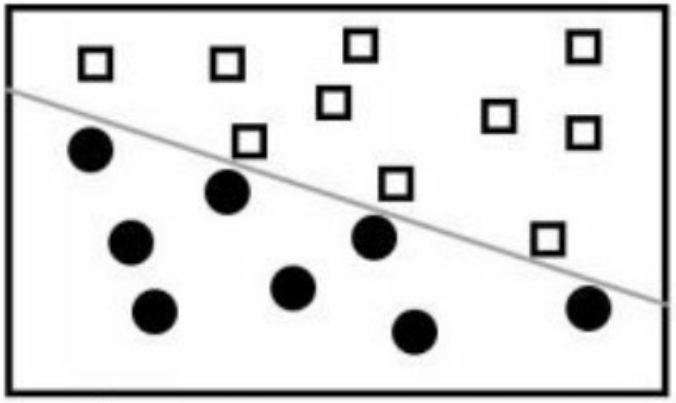
❶ The **"x"** in every activation function equation represents the input of each function. The input is the *net input* calculated by the summation operator.

❷ Every **"e"** in this equation stands for a mathematical constant that is approxiamtely 2.71828.

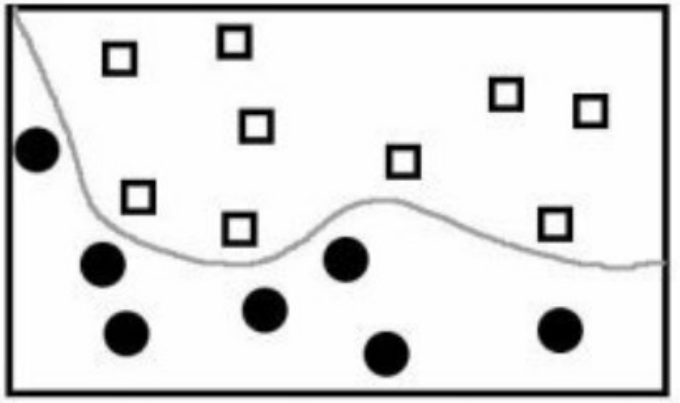
Why Activation Functions?

- **Reason 1: Introducing Non-Linearity**
- Neural networks are often used to solve non-linear problems
- Non linear problems can’t be expressed by linear functions

  like y = ax+b, ax+by+c = 0
- ax2 + bx + c = y or xy + 3x + 4=0 are non-linear



❶ There are two classes in this example. As you can see, they can be linearly divided.



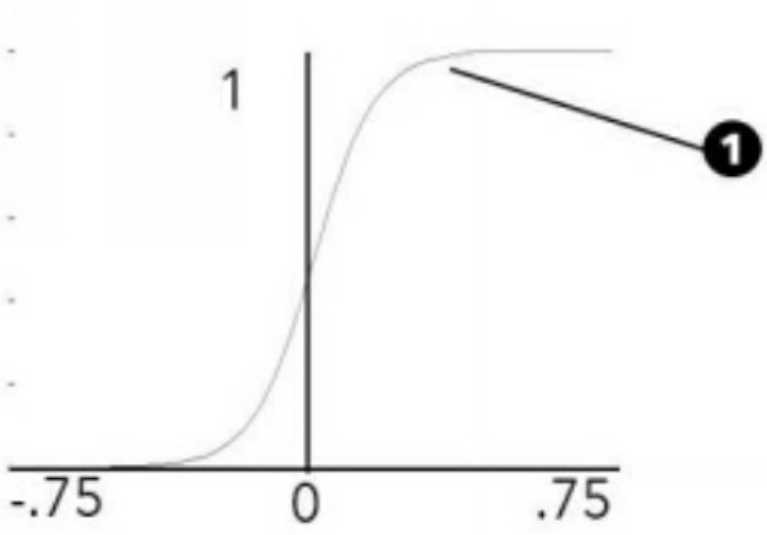
❷ There are two classes in this example,but they cannot be divided linearly. Non-linearity must be introduced into the problem to successfully classify. The logistic (Sigmoid) and tanh functions both introduce this, and when used in conjunction with a multi-layer neural network can solve these problems.

Why Activation Function

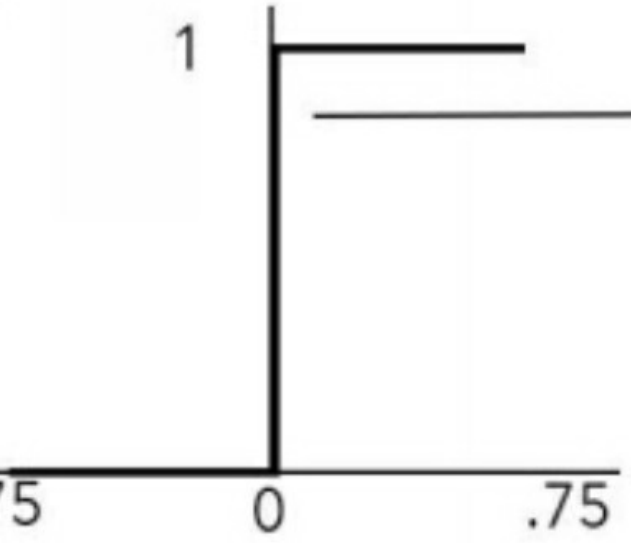
- Activation function such as tanh, logistic(sigmoid) break the linearity enable solve more complex problems.
- **Reason 2: Limiting Output**
- The tanh, sigmoid functions limit the output of a node to certain range
  - Tanh produces output between -1 and 1
  - The Logistic function produces output between 0 and 1
- Advantage of limiting output in certain range weight adjustment, faster training

Advantages of Limiting Output

- Smoothness enables small changes in the weights and bias to produce a small change in the output.
- You can view this fine tuning that makes the task of “learning” much smoother and easier



1 The curved line represents the range of output of a logistic function. It's nice and smooth, and enables a small change in input to create a small change in output.

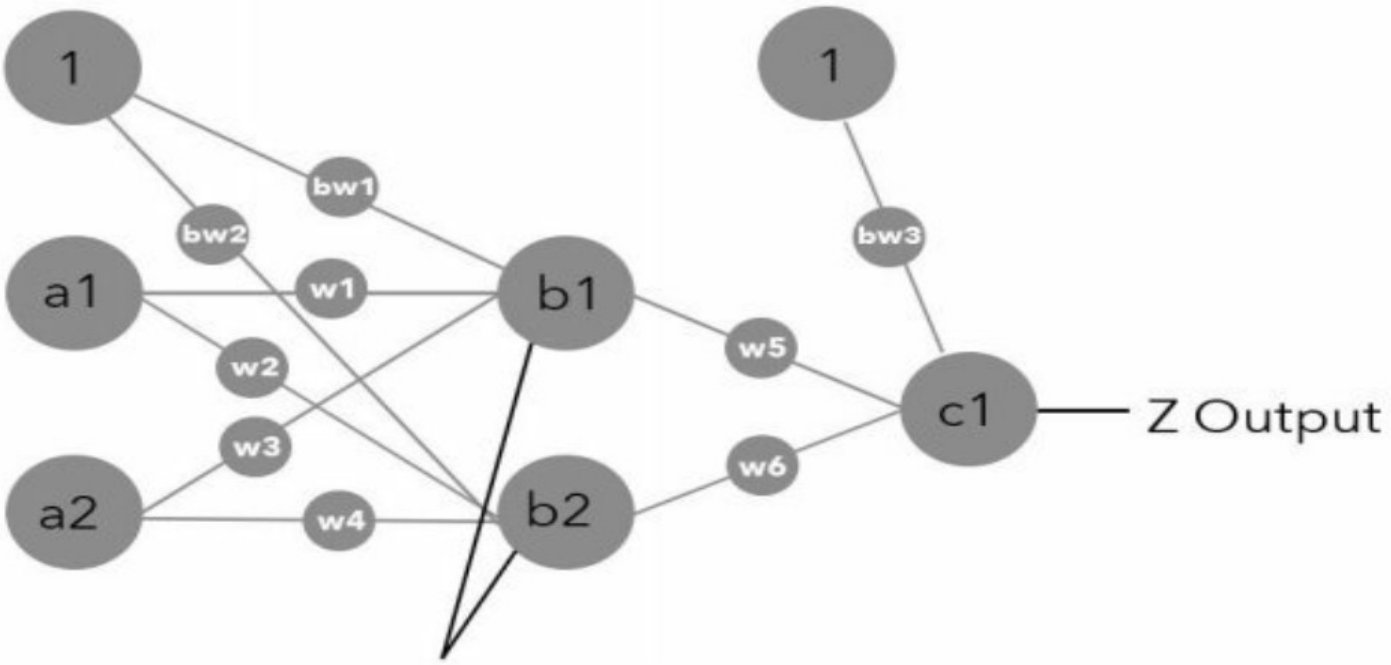


❶ The **bolded** step represents the range of output for a Step (Heaviside) function. All output is either 0 (zero) or 1. A small change in input can create a large change in output.

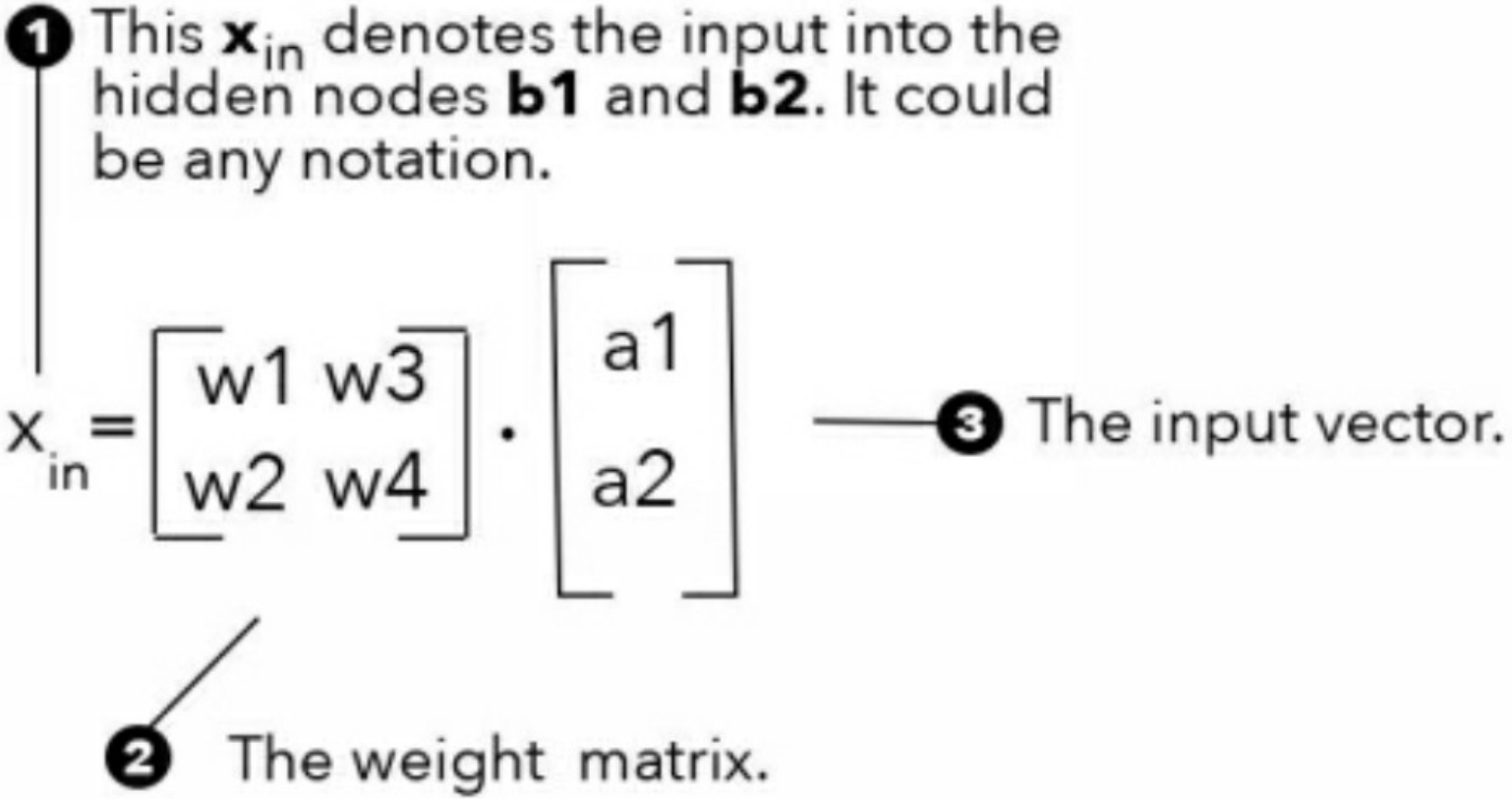
Advantages of Limiting Output

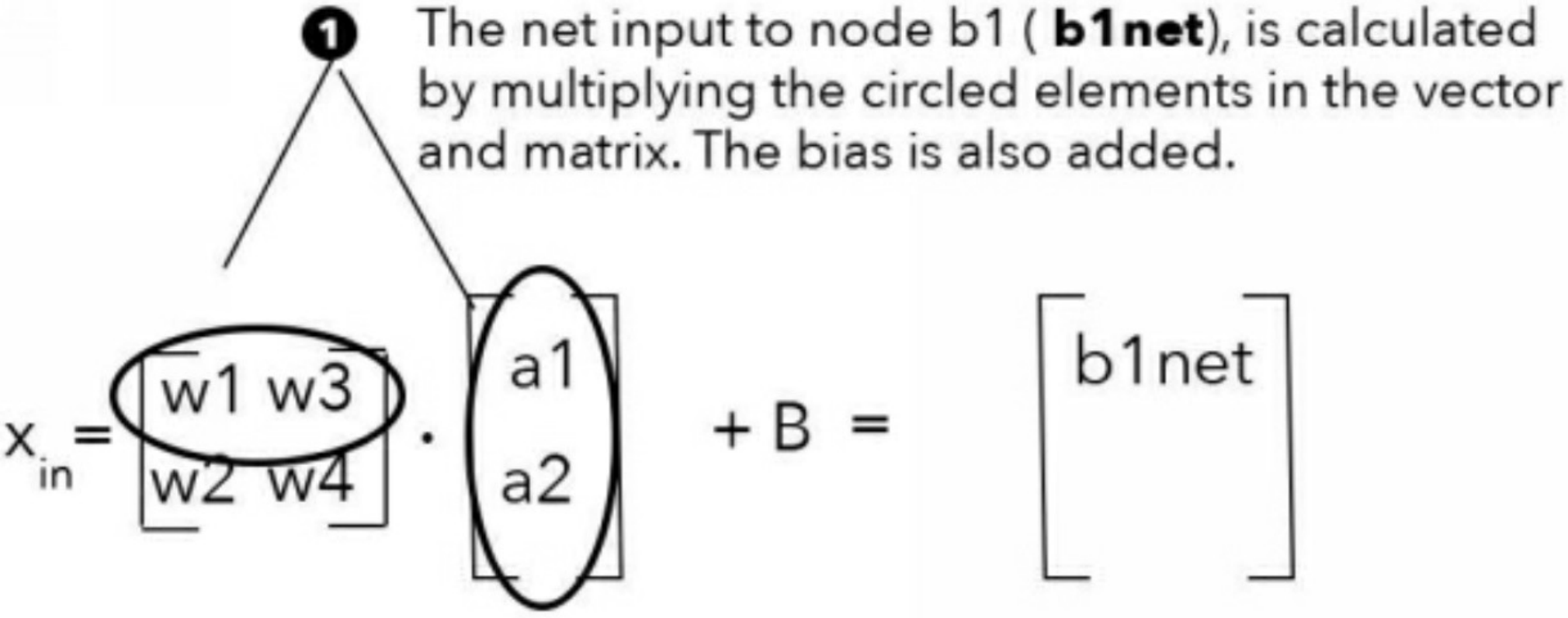
- The step function produces 1 or 0
- This means that small change in weight and bias is not reflected by a small change in output
- This can make learning difficult

Forward Propagation



❶ To calculate the net input of **b1** and **b2**, we need to multiply **a1** and **a2** by their respective weights and then sum the answers into **b1** and **b2**, respectively. The bias also needs to be added.


1 This x_{in} denotes the input into the hidden nodes **b1** and **b2**. It could be any notation.
$x_{in} = \begin{bmatrix} w1 & w3 \\ w2 & w4 \end{bmatrix} \cdot \begin{bmatrix} a1 \\ a2 \end{bmatrix}$
3 The input vector.
2 The weight matrix.


1
The net input to node b1 (b1net), is calculated by multiplying the circled elements in the vector and matrix. The bias is also added.
x_{in} =
w1 w3
w2 w4
·
a1
a2
+ B =
b1net

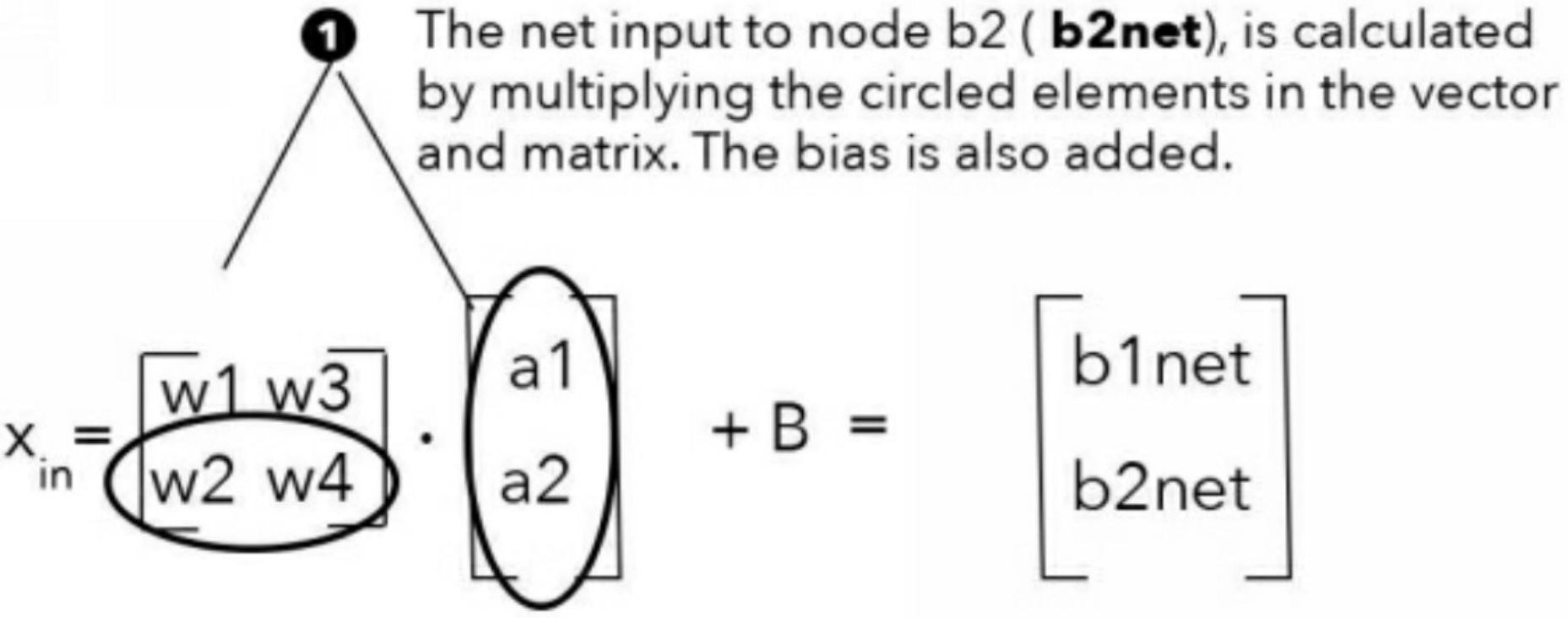
Forward Propagation

- If we write in formula(Summation Operator)

  b1net = (a1*w1)+(a2*w3)+bias

  b1net = (a1*w1)+(a2*w3)+(bias *bw1) since bias is 1

  b1net = (a1*w1)+(a2*w3)+bw1


1
The net input to node b2 (b2net), is calculated by multiplying the circled elements in the vector and matrix. The bias is also added.
x_{in} =
w1 w3
w2 w4
·
a1
a2
+ B =
b1net
b2net

Forward Propagation

- If we write in formula(Summation Operator)

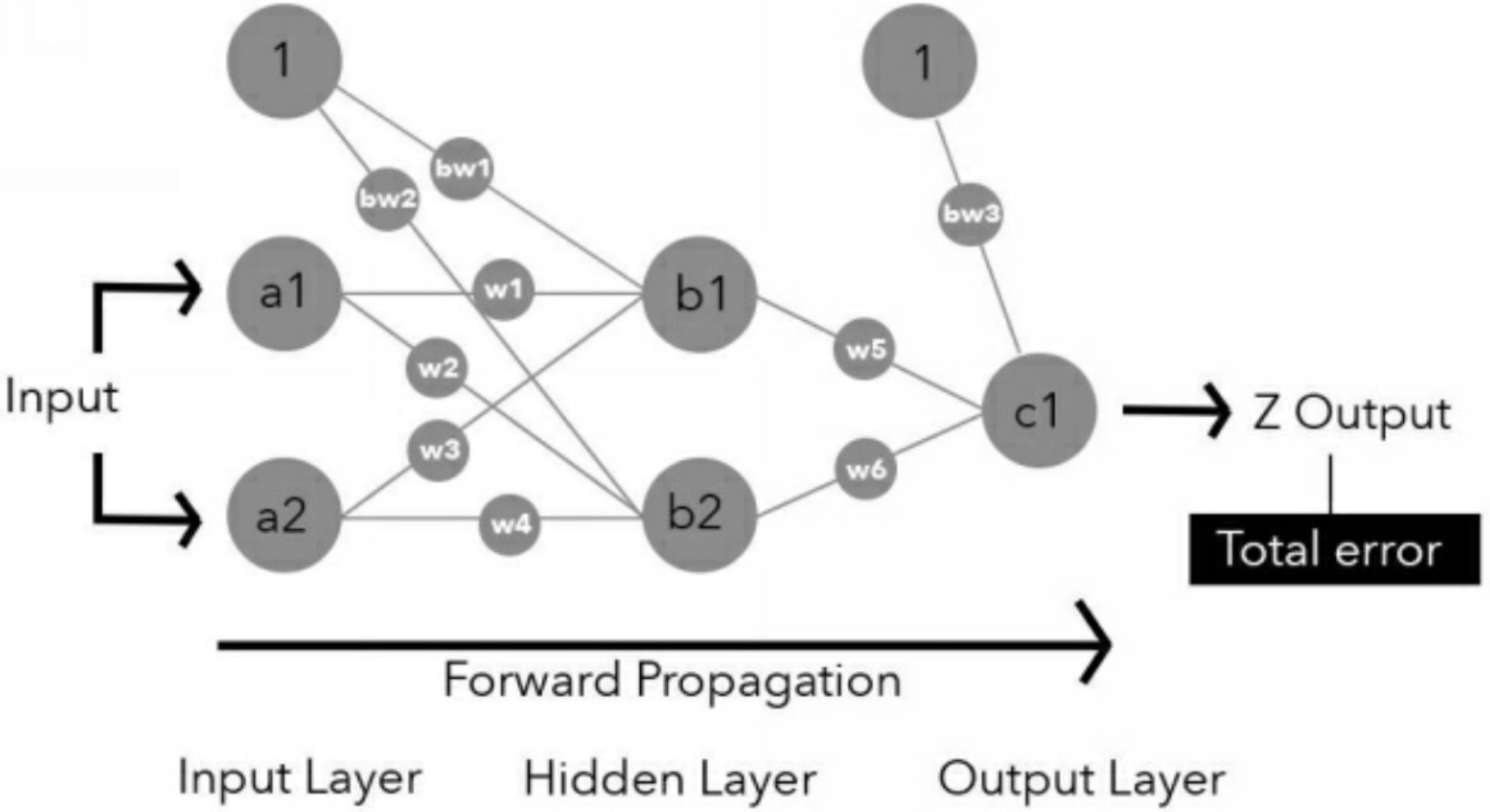
  b2net = (a1*w2)+(a2*w4)+bias

  b2net = (a1*w2)+(a2*w4)+(bias *bw2) since bias is 1

  b2net = (a1*w2)+(a2*w4)+bw2

Forward Propagation

- Total net input is then fed into an activarion function
- Which transform the net input into new output
- This output sent over one or more edges and multiplies by a weight
- Cycle continues untill the output layer caliculations are completed


1
1
bw1
bw2
bw3
a1
w1
b1
w5
w2
Input
c1
Z Output
w3
w6
a2
w4
b2
Total error
Forward Propagation
Input Layer
Hidden Layer
Output Layer

Forward Propagation

- We sucessfully moved to output layer and computed Output
- Next step is to compute **total error** of the network
- Whic will enable the network to adjust its weights and learn.
- Total Error is the difference between networks **actual output** and **target output**.

Forward Propagation

- Once total error computed two things can happen
- Converge
  - The network has successfully converged(ie. Reached global minimum or local minum closed to global minimum)
  - At this point network ceases training
- Failed to Converge
  - The network has failed to converged(ie. global minimum not reached or local minimum not even close to global minimun)
  - Network continues to train

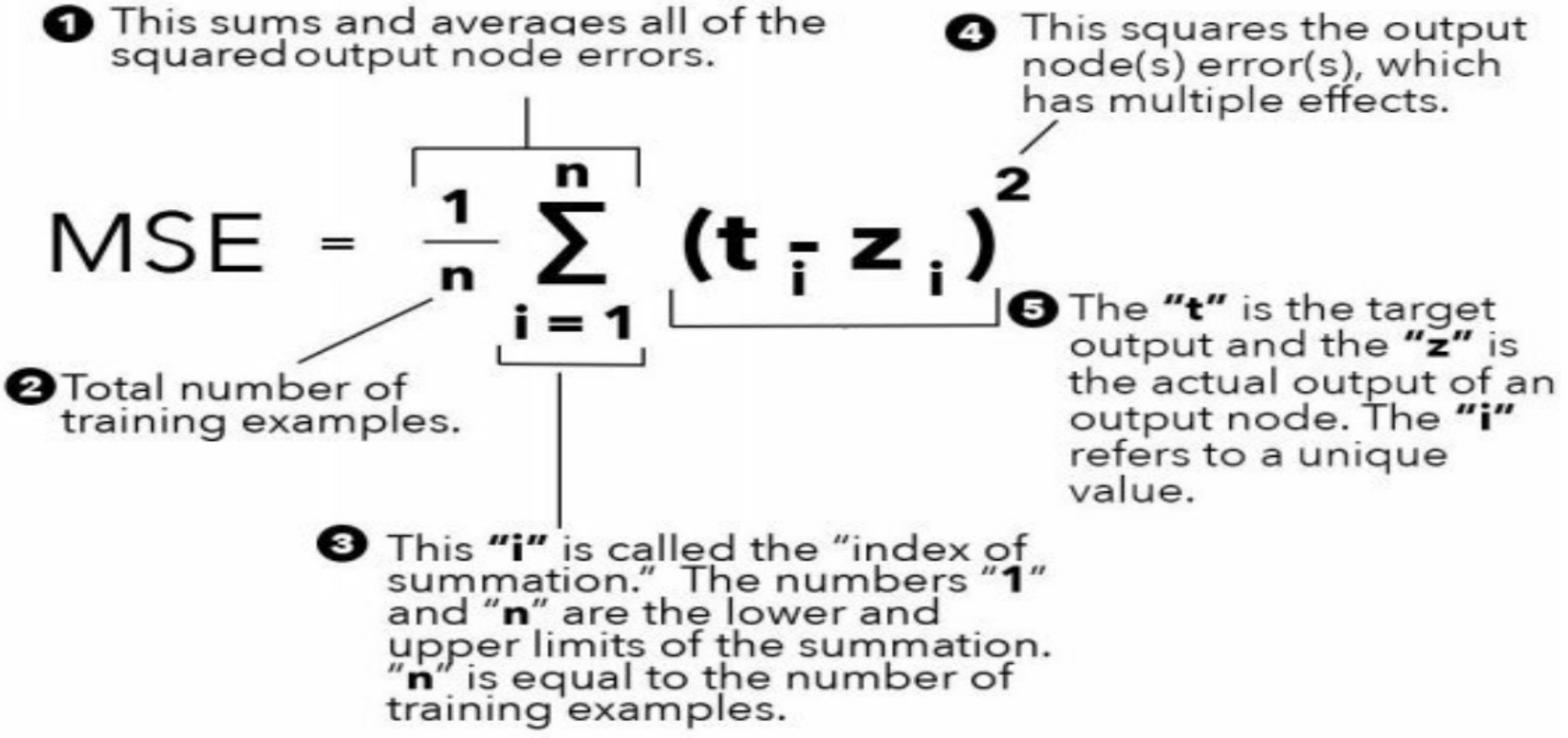
Cost Functions

- Also refered as loss function or error function
- Cost function transforms everything that occur in network into a number that reprasents **total error**
- It is measuring the how wrong the network is
- Cost function is often refered as **objective function**
- This is because the objective of the network is to minimize the cost function

Types of Cost Functions

- There are may types of cost functions some are:
  - Mean Squared Error
  - Squared Error
  - Root Mean Squared Error
  - Sum of Squares Error
  - Cross-Entropy
  - Exponential
  - Hellinger Distance
  - Kullback-Leibler Divergence

Mean Squared Error(MSE)



Cost Functions

Squared Error(SE)

$$SE = \frac{1}{2}\sum_{i=1}^{n}(t_i - z_i)^2$$

1 Multiplied by 1/2 instead of 1/n.

Root Mean Square(RMS)

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(t_i - z_i)^2}$$

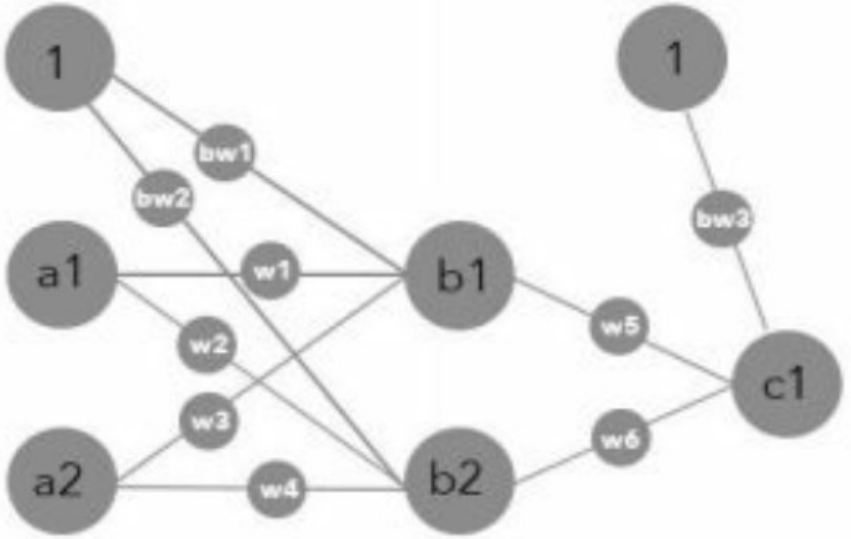
- Sum of Squares Error(SSE)

$$SSE = \sum_{i=1}^{n}(t_i - z_i)^2$$

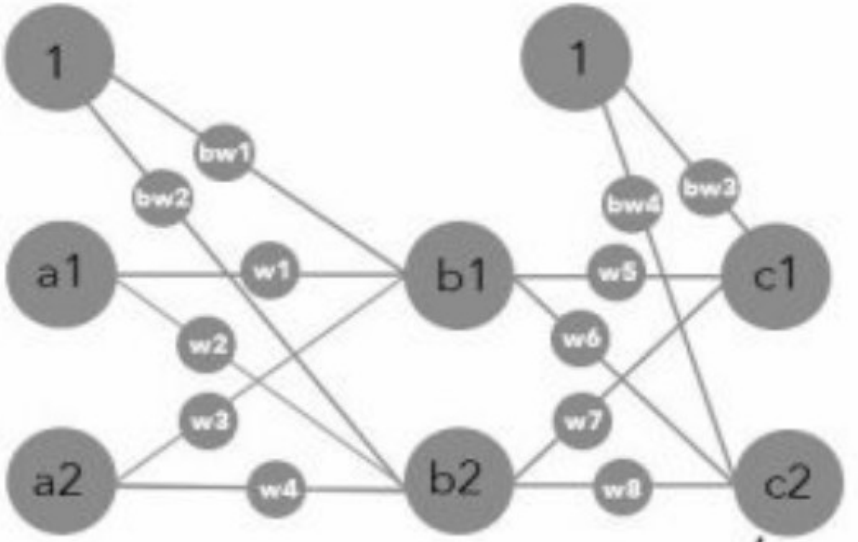
Why Cost function?

- It provides the total error or difference between the **target output** and n/w **computed output**
- **1. Caliculating the total error of each output node**
  - $\mathbf{(t_i - z_i)^2}$
  - In order to find the total error the **local error** of each training example must be first caliculated
  - **Local error** is the difference between a single training example's n/w **output** and **target output**
  - If there are multiple output layer nodes this means there are multiple outputs and target outputs
  - Hence local error caliculated at each node and results are summed to create the **final local error** for **single training example**
  - $\mathbf{\sum (t_i - z_i)^2}$

Single Node Output



Multiple Node Outputs



❶ In this example there are multiple outputs. Therefore, the local error of each output is calculated, and the results are summed to create a final local error.

Mean Squared Error(MSE)

- Single output(Single local Error)

  $$e = (5-1)^2$$

  - Actual output minus the target output

- Two output(Two local Error)

  $$e1 = (5-1)^2$$

  - The local error of node c1

  $$e2 = (3-2)^2$$

  - The local error of node c2

Mean Squared Error(MSE)

- **2. Square each local error**
- Two uses with Squaring
  - **First**, Difference caliculated is same whether it is positive or negetive
  - Avoid different signs from cancelling each other which can lead to a misreprasentation of wrong the network is
  - **Second,** squaring also helps the network converge faster

Mean Squared Error(MSE)

- **3. Summing the local errors of all training examples**

$$\sum_{i=1}^{n}$$

- In this step all the final local errors of every training example are summed
- The **n** is training examples not individual nodes

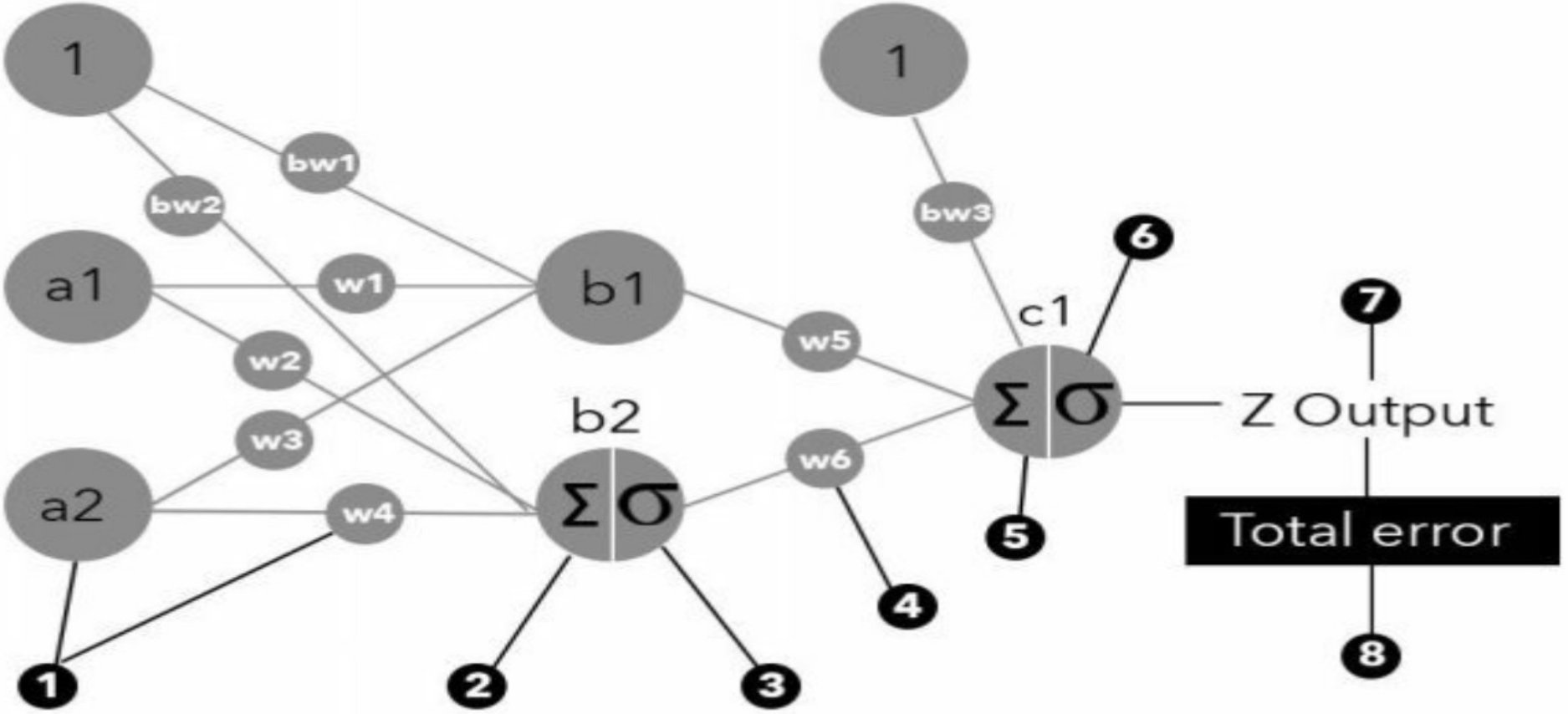
Mean Squared Error(MSE)

- **4. Multiply and average**

  1/n

- 1 is devided by total number of training examples, **n**
- The result is multiplied by sum of all local erros
- This normalizes the sum
- If there is a single training example no need to devide and multiply

Forward Propagation : Big Picture



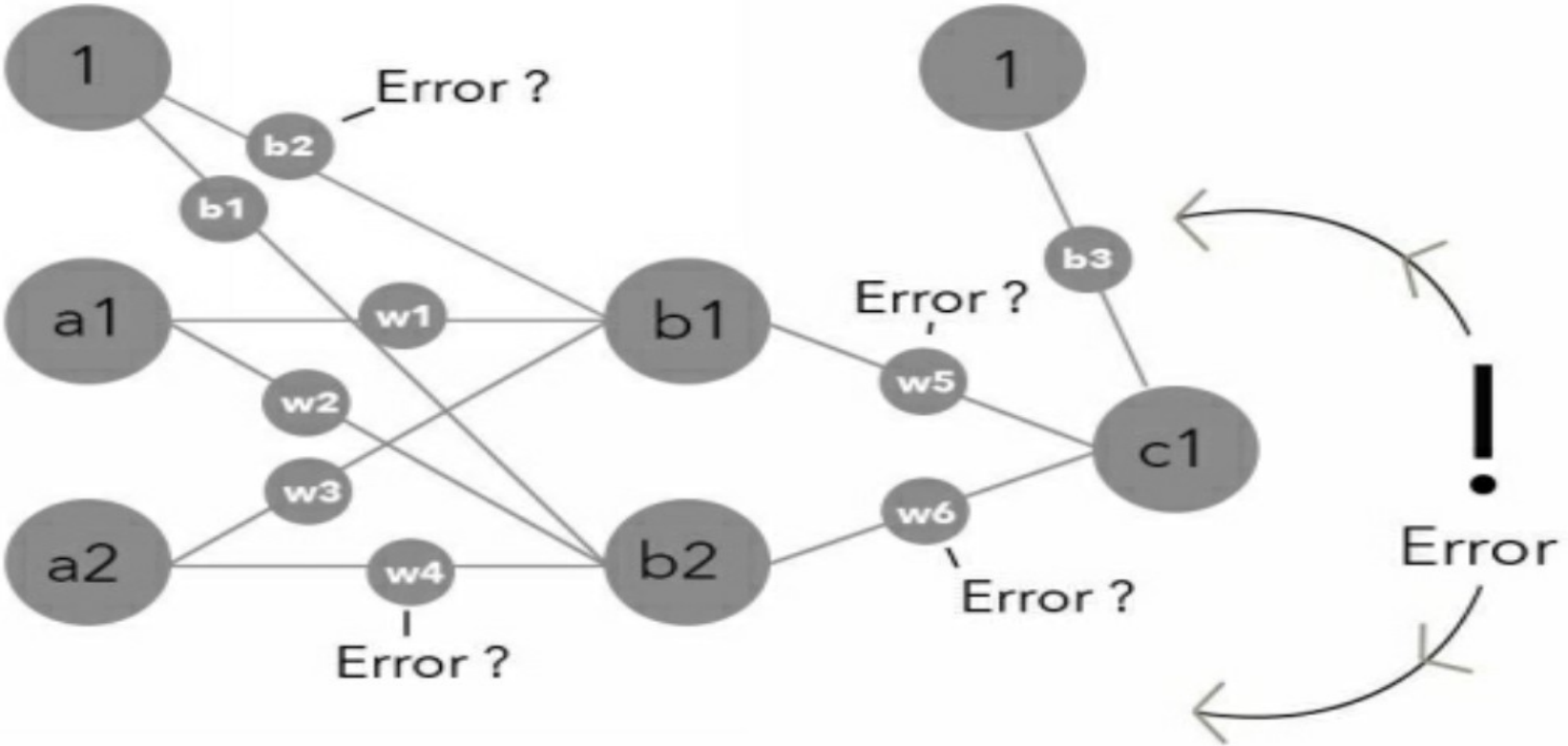
Forward Propagation : Big Picture

- **Step1**: each input is multiplied by a weight as it travels over an edge to node in fallowing layer
- **Step2**: all inputs to node, including the bias are summed using summation operator
- **Step3-4**: output of **b2** multiplied by weight **w6**
- **Step5:** the result is summed with all other inputs to create the net input of node c, **netc**
- **Step6: Netc** becomes the input of activation function
- **Step7:** The activation fun output is the final outputbof the network
- **Step8:** the cost fun is applied to output of network, and a total errorbis caliculated.

Back Propagation

- **Caliculating Gradients**
  - Inprevious step we caliculated total error of network
  - Find how this error is spred across every weight in the network
  - So that we can adjust the weights to minimize the error
  - To do this we are going to caliculate the error of every weight in the network
  - Error of weight is technically gradient, and we will use the chain rule to caliculate this
  - The network try to minimize total error

Back Propagation



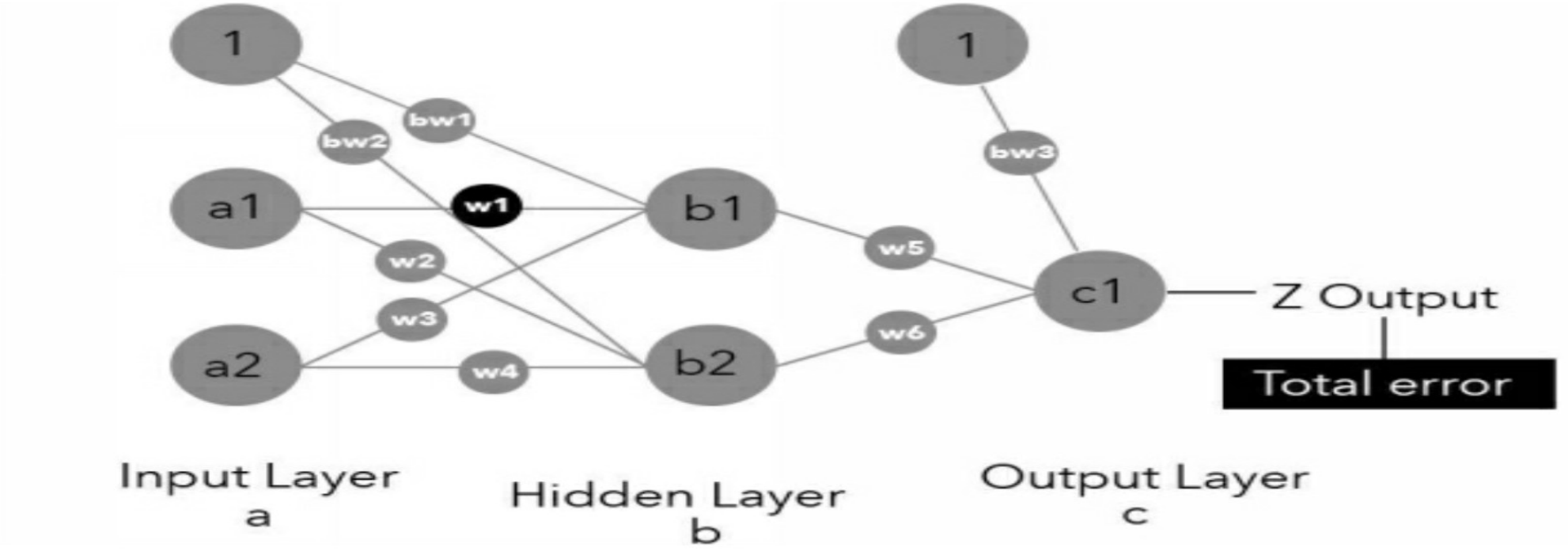
Mathematical Functions

1) Partial Derivative
2) Chain Rule

- What is derivative?
  - Rate of change of a function at single point
- What is partial derivative?
  - It is the derivative of a function which has two or more variables **but with respect to only one variable**
  - All other variables are treated as constant.

Partial Derivative

- Partial derivative enables you to measure how a single variable (out of many) impacts another single variable.



Partial Derivative

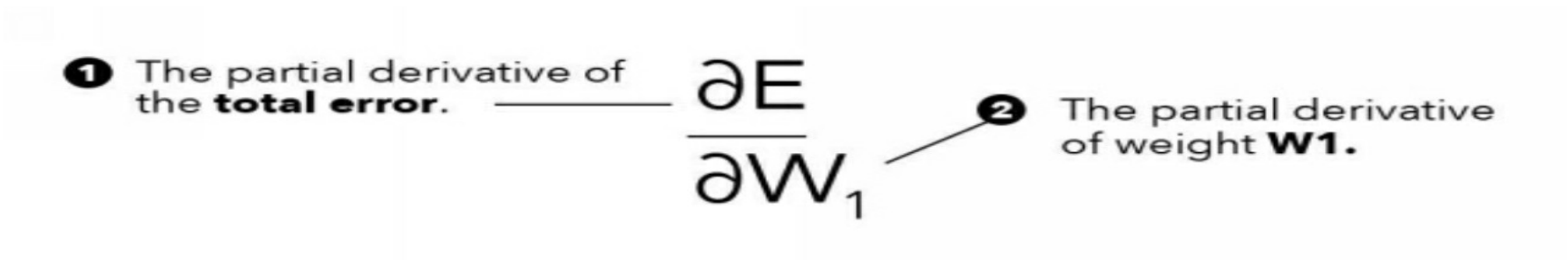
- Take **w1** as example in above fig.
- The partial derivative allows us to discover how a change in weight **w1** affects the **total error**
- While all other weights remains constant
- The same can be caliculated for **w2,w3,w4,bw1 etc.**
-

$$\frac{\partial f}{\partial x}$$

Used instead of "d" in usual $\frac{df}{dx}$ notation to emphasize that this is a partial derivative.

Partial Derivative

- Measuring a change in the **total error** by change in the weight **w1** wolud look like this



Chain Rule

- The chain rule is used to find the partial derivative when an equation consists of a function inside another function.
- In Otherway It is used to diferentiate the function of another function

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

Gradient Importance

- Lets express our network in function

$$Z = Sig(w^2 Sig(w^1 a + b_1) + b_2)$$

- Output of our network is product of weights and input
- The input is fixed so to minimize the total error our only option is to adjust the weights
- How much do we adjust weights?
- Do we increase or decrease weight?

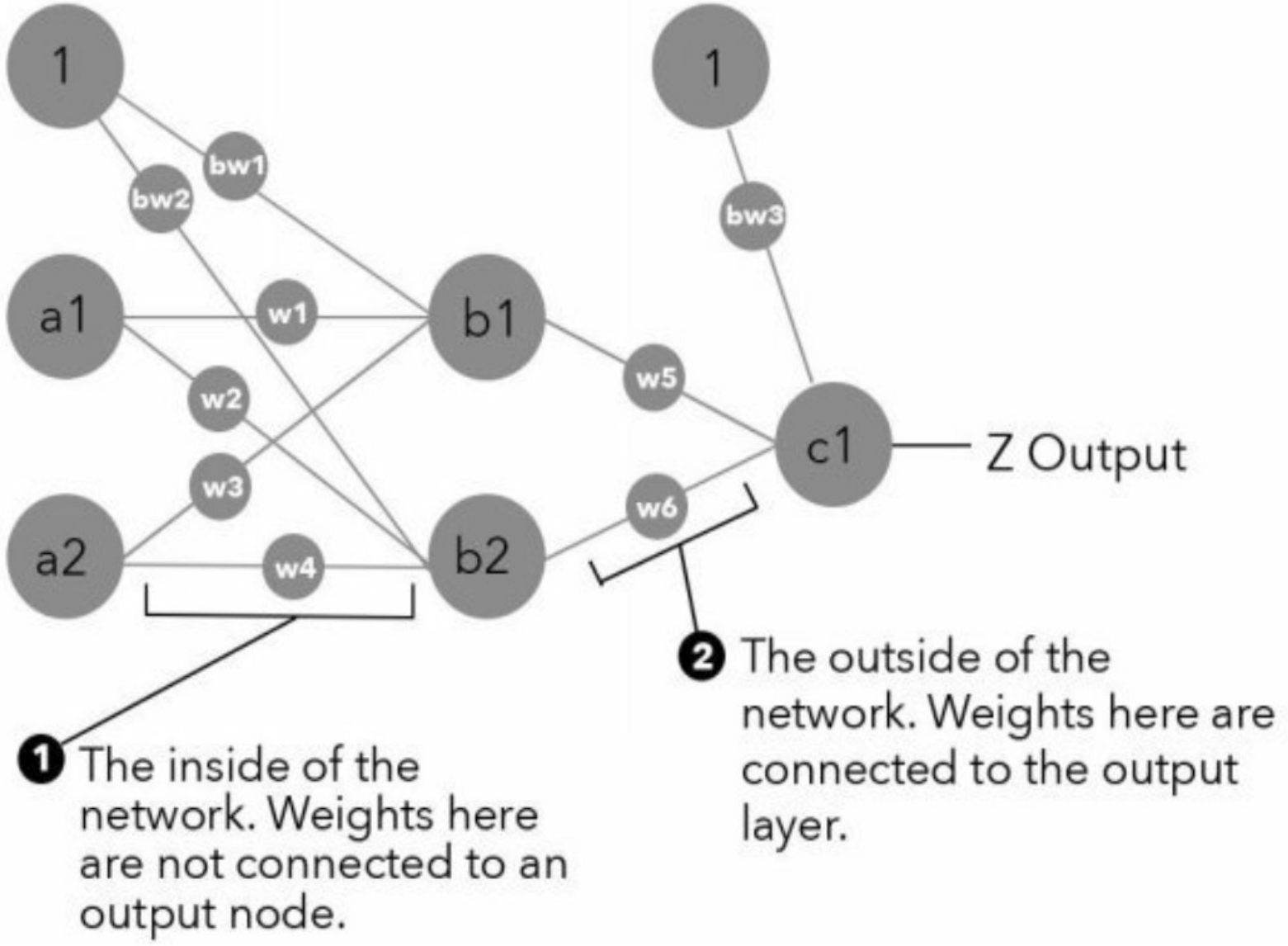
Gradient Importance

- What is the combination of weights that will produce a minimal **total error?**
- Answer is **Gradient Descent** which is an optimization method that helps us find the exact combination of weights and ultiimately discover a **global minimum(total error)**.
- To perform **Gradient Descent** we need partial derivatives of each weight
- Then use the partial derivatives to update weights

Gradient Descent

- There are two different type of weights
  - Hidden weights
  - Between input/hidden nodes
  - Otput weights
  - Between hidden to output nodes

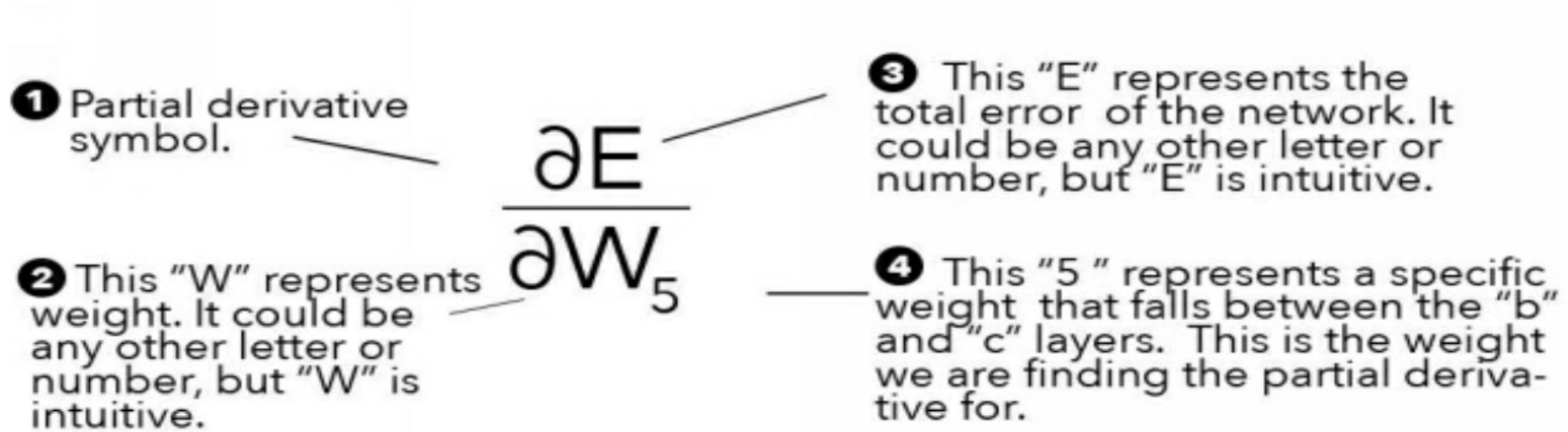
- How to perform Gradient Descent?


1
bw1
bw2
a1
w1
b1
w2
w5
w3
a2
w4
b2
w6
1
bw3
c1
Z Output
1 The inside of the network. Weights here are not connected to an output node.
2 The outside of the network. Weights here are connected to the output layer.

Cliculating Gradients

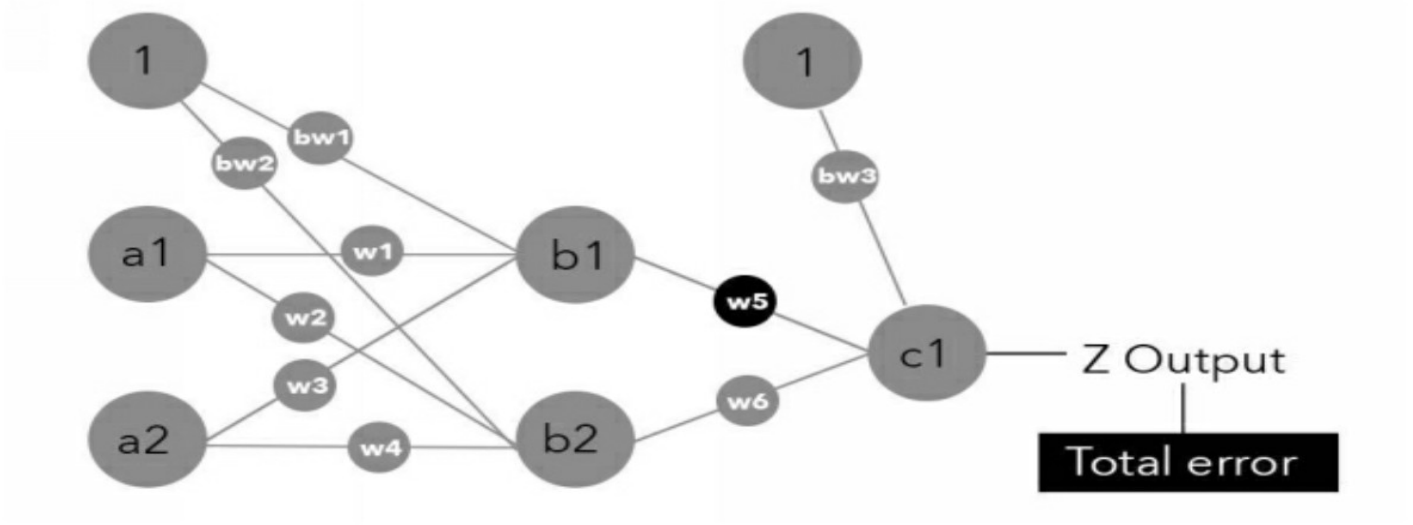
1) Caliculating partial derivatives of output Layer Weights
2) Caliculating partial derivatives of output Layer Bias Weights
3) Caliculating partial derivatives of hidden Layer Weights
4) Caliculating partial derivatives of hidden Layer Bias Weights

Partial derivatives of output weights

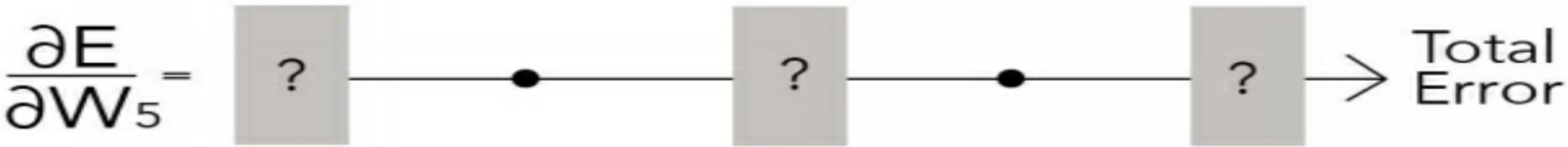


Partial derivatives of output weights

- Partial derivateive of the **total error** with respect to **w5**
- how much does change in in **w5** effect the **total error**



Partial derivatives of output weights



Partial derivatives of output weights

- But **w5** not directly connected to **total error**
- **W5** is connected through a series of **partial derivatives** to the **total error**
- Here **total error** is computed by appling a **cost function** to the output **z**
- So first, we need to know howmuch a change in the **z** affects the **total error**
- Second, we need to know howmuch a change in the **w5** affects the **z**

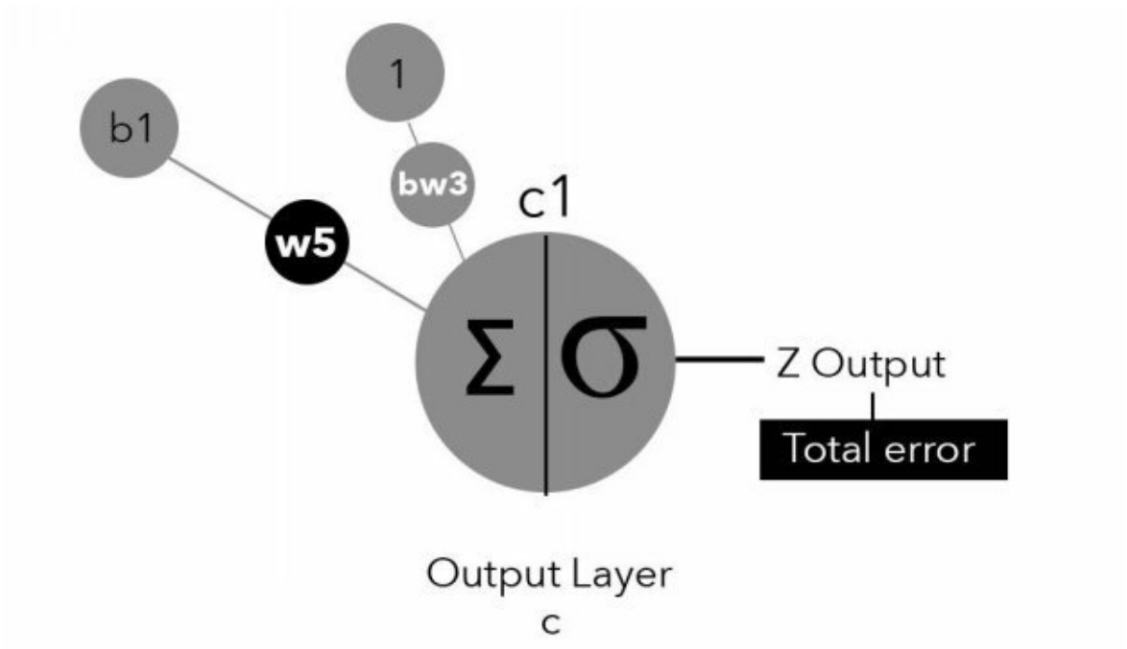
Partial derivatives of output weights

1. The partial derivative of the **total error** with respect to the output **Z**.

$$\frac{\partial E}{\partial W_5} = \frac{\partial E}{\partial Z} \frac{\partial Z}{\partial W_5}$$

2. The partial derivative of the output **Z** with respect to weight **w5.**

Partial derivatives of output weights



Partial derivatives of output weights

- Output node **c1** is devided into two sections
- **Summation operator** which sums the net input into the node.
- **Logistic activation function** which takes output from summation operator and applies a function to it before passing it out as the final output **z**
- There is still another intermediate variable inside of it
- Which tells us how the net input into node **c1** affect **z**

Partial derivatives of output weights

$$\frac{\partial Z}{\partial W_5} = \frac{\partial Z}{\partial Znet_c} \frac{\partial Znet_c}{\partial W_5}$$

❶ The partial derivative of the total error with respect to the output **Z**.

❸ The partial derivative of **netc** with respect to weight **w5.**.

$$\frac{\partial E}{\partial W_5} = \frac{\partial E}{\partial Z} \frac{\partial Z}{\partial net_c} \frac{\partial net_c}{\partial W_5}$$

❷ The partial derivative of **Z** with respect to **netc.**

Cliculating Partial Derivatives

$$\frac{\partial E}{\partial Z}$$

- $$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - z_i)^2$$

$$\frac{\partial E}{\partial Z} = -(t - z)$$

❶ This **"t"** represents the target output of a particular node.

❷ This **"z"** represents the actual output of a particular node.

$$\frac{\partial E}{\partial Z} = (z - t)$$

Cliculating Partial Derivatives

$$\frac{\partial Z}{\partial net_c}$$

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

Cliculating Partial Derivatives

$$\frac{\partial Z}{\partial net_c} = z(1 - z)$$

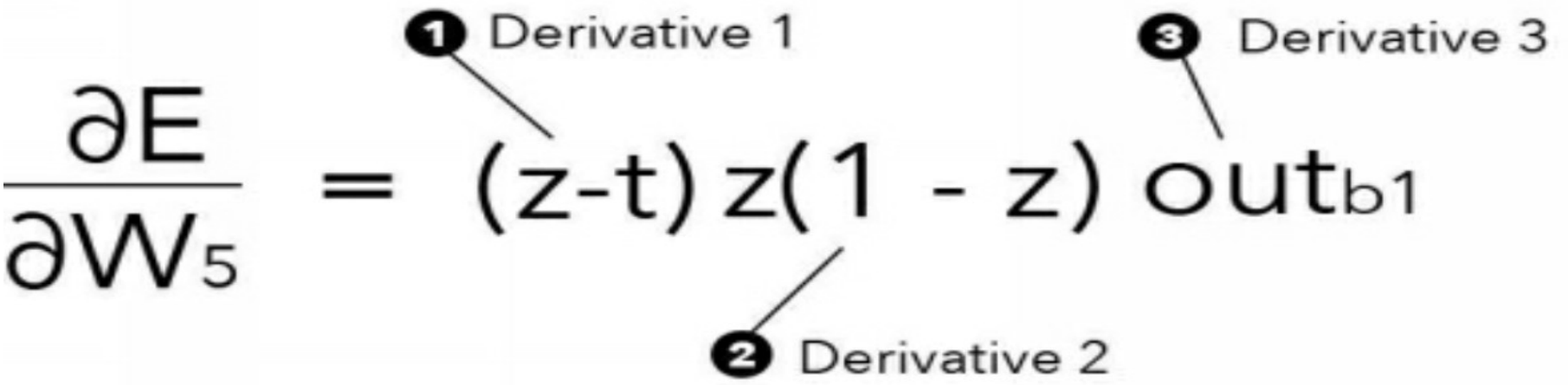
❶ Each **"z"** represents the actual output of a particular node.

Cliculating Partial Derivatives

$$\frac{\partial net_c}{\partial W_5}$$

- $$\frac{\partial net_c}{\partial W_5} = out_{b1}$$

Cliculating Partial Derivatives



Cliculating Partial Derivatives

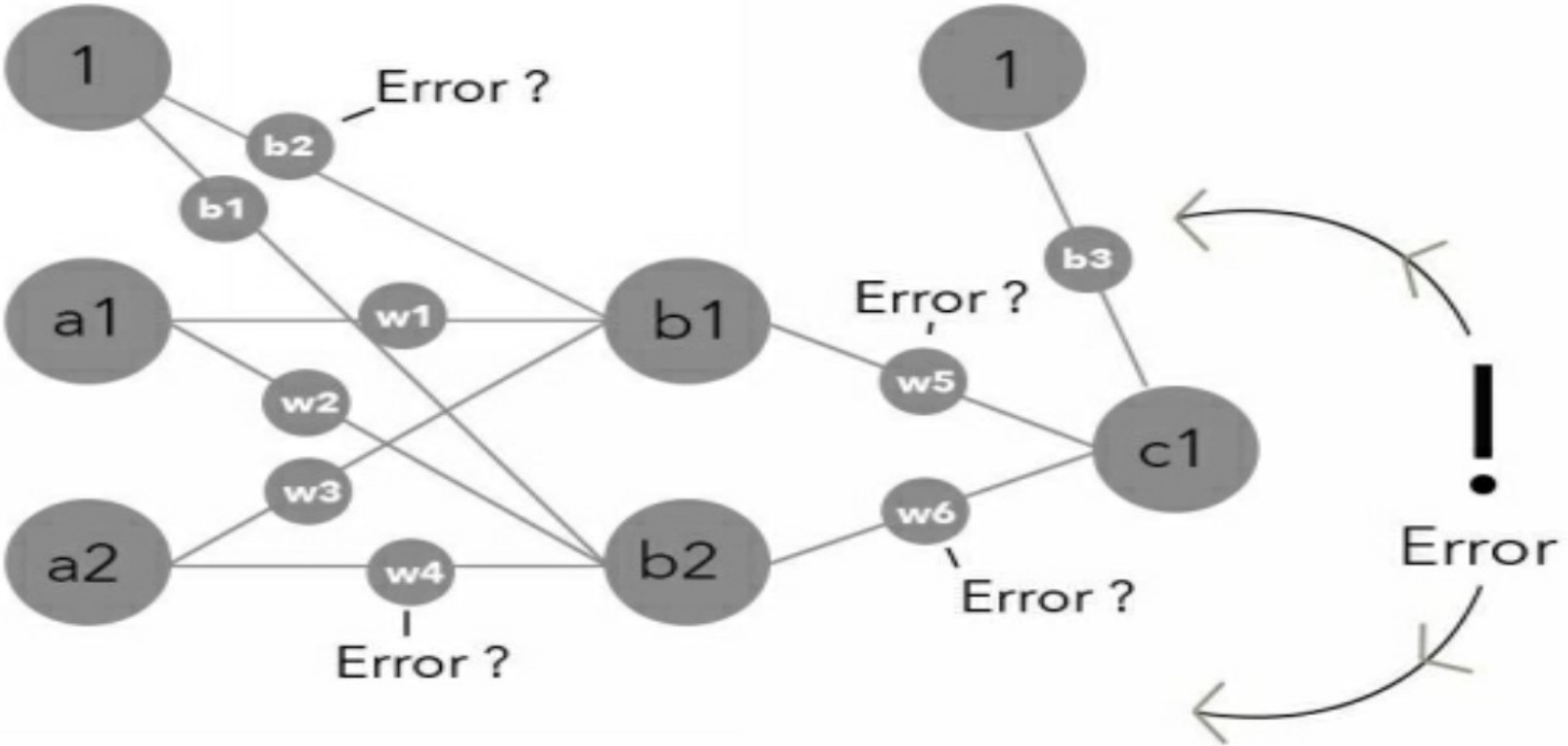
$$\delta_z = (z-t)\ z(1-z)$$

$$\frac{\partial E}{\partial W_5} = \delta_z \, out_{b1}$$

❶ This δ_z is the node delta for the output layer. See below for further explanation.

❷ This represents the output of the node **b1**.

Back Propagation

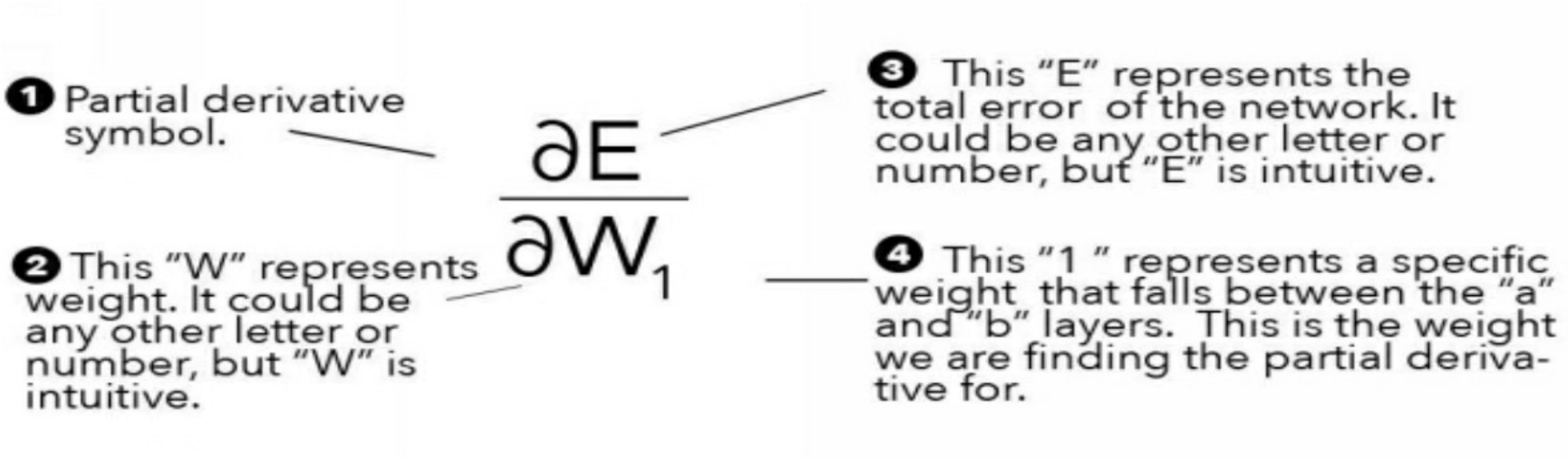


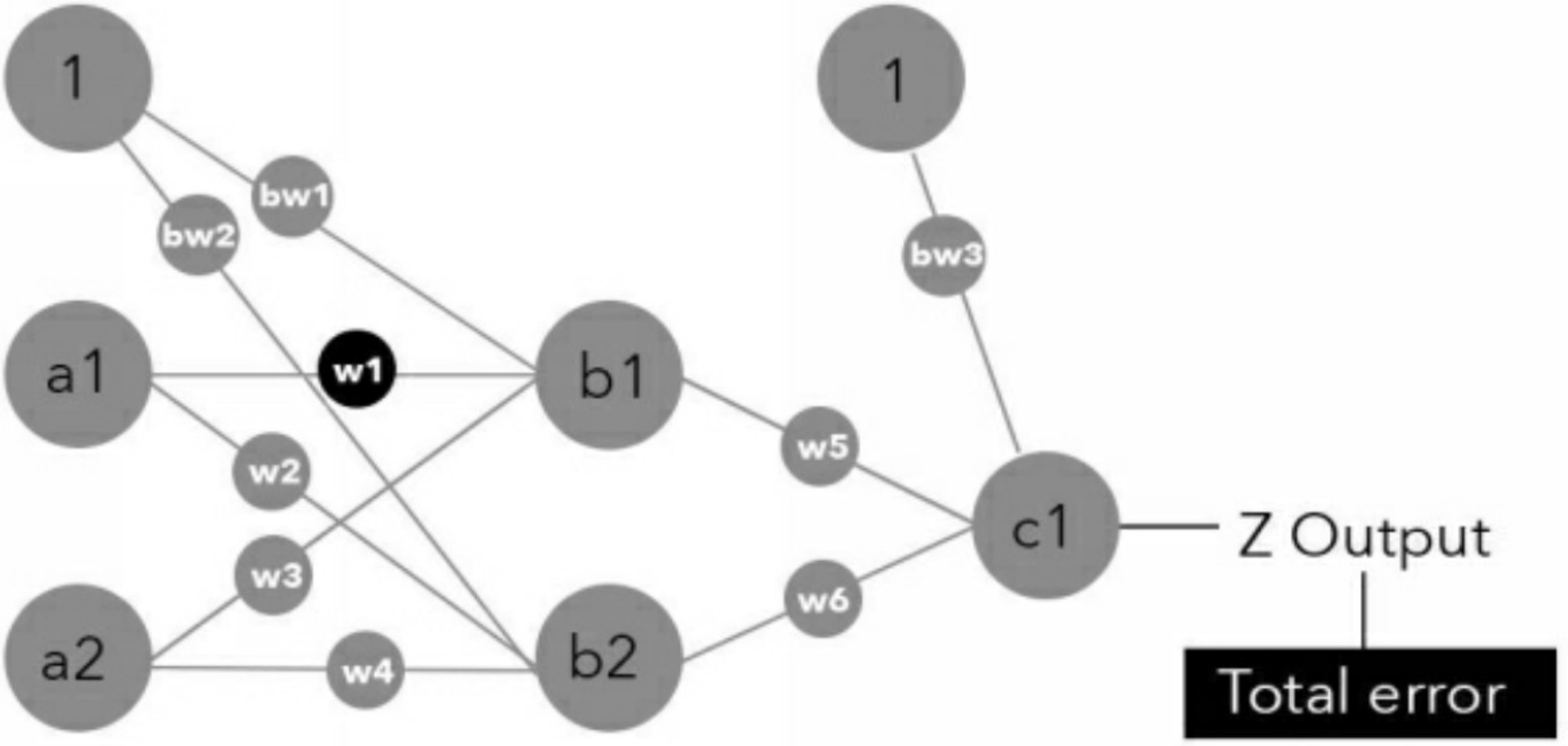
Caliculating partial derivatives of bias

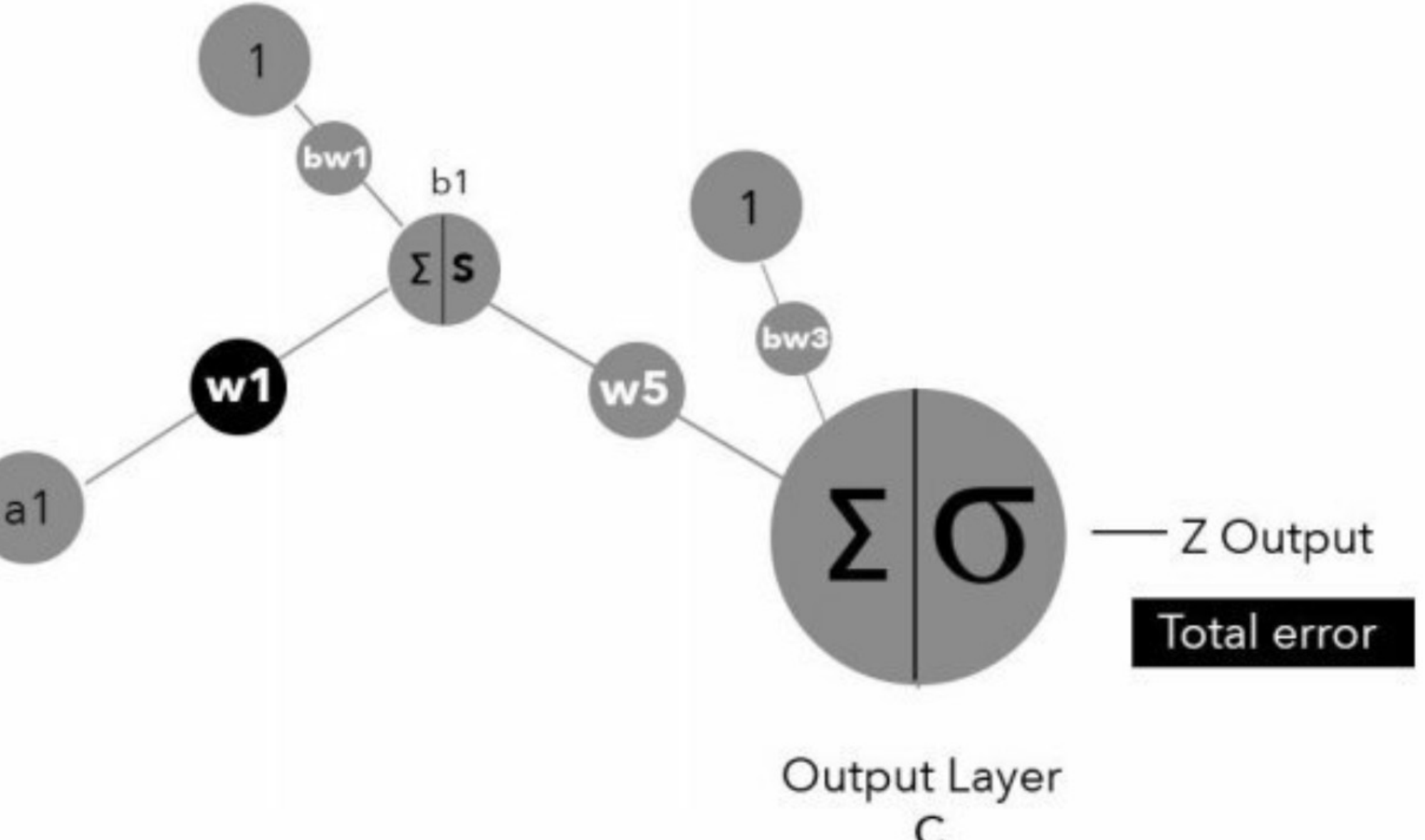
- No outb1?
- Bias is no connected to previous layer

$$\frac{\partial E}{\partial B_{w3}} = (z\text{-}t)\, z(1 - z)$$

Caliculating partial derivatives of hidden weights




1
bw1
bw2
a1
w1
b1
w2
w5
c1
Z Output
w3
a2
w4
b2
w6
1
bw3
Total error


1
bw1
b1
Σ S
1
bw3
w1
w5
a1
Σ σ
Z Output
Total error
Output Layer

❶ The partial derivative of the **total error** with respect to **netb1**.

$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial net_{b1}} \frac{\partial net_{b1}}{\partial W_1}$$

❷ The partial derivative of **netb1** with respect to weight **w1.**

Caliculating partial derivatives of hidden weights

$$\frac{\partial E}{\partial net_{b1}} = \frac{\partial E}{\partial net_{c1}} \frac{\partial net_{c1}}{\partial net_{b1}}$$

$$\frac{\partial net_{c1}}{\partial net_{b1}} = \frac{\partial net_{c1}}{\partial out_{b1}} \frac{\partial out_{b1}}{\partial net_{b1}}$$

$$\frac{\partial E}{\partial W_1} = \overset{\textcircled{1}}{\frac{\partial E}{\partial net_{c1}}} \overset{\textcircled{2}}{\frac{\partial net_{c1}}{\partial out_{b1}}} \overset{\textcircled{3}}{\frac{\partial out_{b1}}{\partial net_{b1}}} \overset{\textcircled{4}}{\frac{\partial net_{b1}}{\partial W_1}}$$

Caliculating partial derivatives of hidden weights

$$\frac{\partial E}{\partial net_{c1}}$$

Caliculating partial derivatives of hidden weights

$$\frac{\partial E}{\partial net_{c1}} = \delta_z$$

$$\delta_z = (z\text{-}t)\ z(1 - z)$$

Caliculating partial derivatives of hidden weights

$$\frac{\partial E}{\partial net_{c1}} = \delta_z$$

$$\delta_z = (z\text{-}t)\ z(1 - z)$$

Caliculating partial derivatives of hidden weights

$$\frac{\partial net_{c1}}{\partial out_{b1}}$$

$$\frac{\partial net_{c1}}{\partial out_{b1}} = W_5$$

Caliculating partial derivatives of hidden weights

$$\frac{\partial out_{b1}}{\partial net_{b1}}$$

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

Caliculating partial derivatives of hidden weights

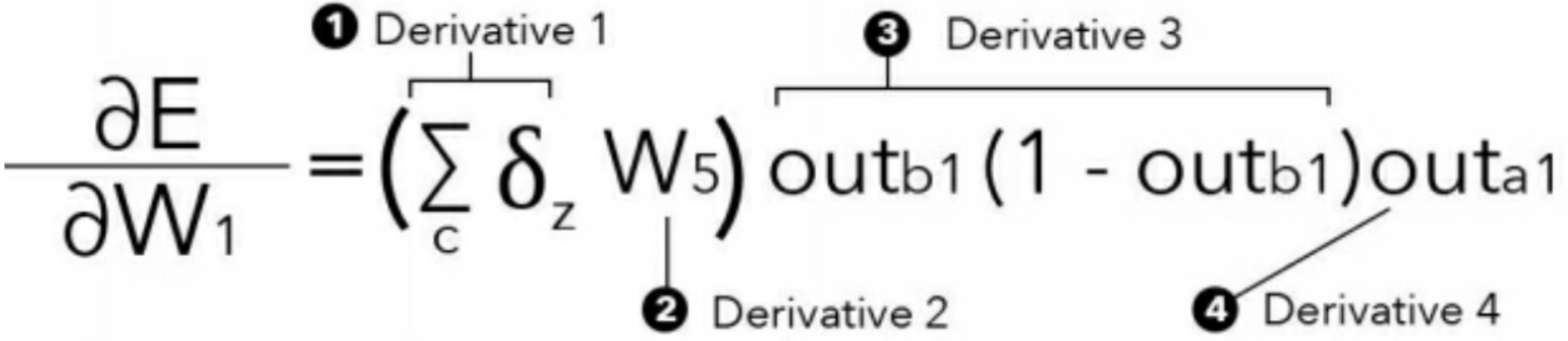
$$\frac{\partial out_{b1}}{\partial net_{b1}} = out_{b1}(1 - out_{b1})$$

Caliculating partial derivatives of hidden weights

$$\frac{\partial net_{b1}}{\partial W_1}$$

$$\frac{\partial net_{b1}}{\partial W_1} = out_{a1}$$

Caliculating partial derivatives of hidden weights



Caliculating partial derivatives of hidden bias

$$\frac{\partial E}{\partial B_{w1}} = \left(\sum_{c} \delta_z W_5\right) out_{b1}(1 - out_{b1})$$

Weight updation