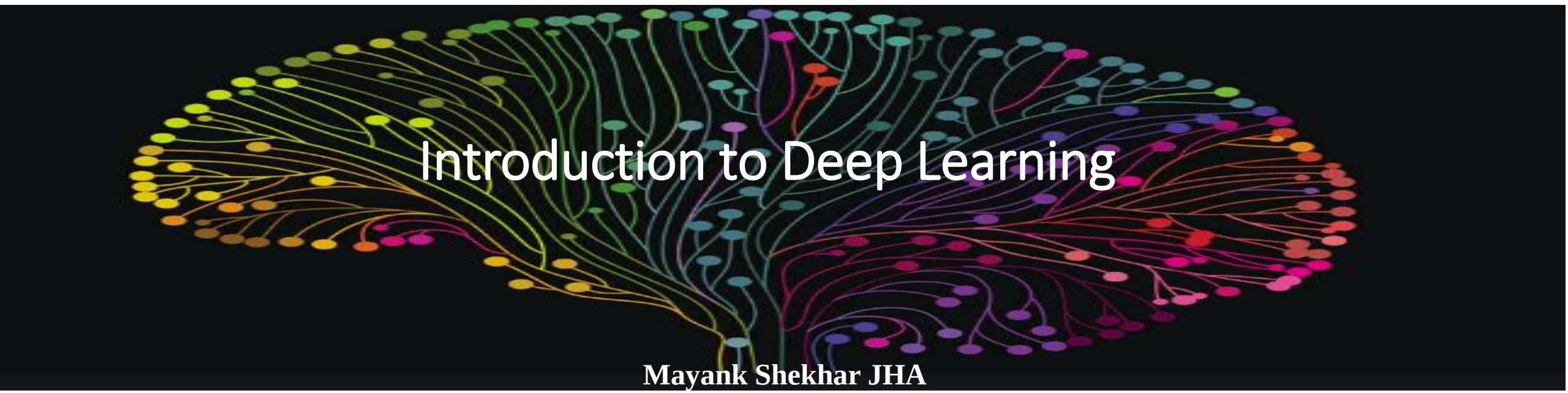


Introduction to Deep Learning

Mayank Shekhar JHA

Associate Professor

(Maitre de Conférences)

Automatic Control, Reliability of Systems

mayank-shekhar.jha{at}univ-lorraine.fr

Research

Centre de Recherche en Automatique de Nancy (CRAN) UMR 7039,
Faculté des Sciences et Technologies
Boulevard des Aiguillettes - BP 70239 - Bât. 1er cycle 54506 Vandoeuvre-lès-Nancy
Cedex France

Teaching

Polytech Nancy (ESSTIN),
2 Rue Jean Lamour 54509
Vandoeuvre-lès-Nancy,
Cedex France

Email: mayank-shekhar.jha [at] univ-lorraine.fr

Contents

Introduction

Part 1 : Neural Networks basics and Feed forward Deep NNs

Part 2: Convolutional Neural Networks (CNNs)

Part 3: Recurrent Neural Networks

Lab Sessions (Python + Pytorch) : TD 1 → ANNs

TD2 → CNNs

Home Assignments : ANNs (50%), CNNs (50%)

POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

Introduction and Few Reminders

Artificial Intelligence Domains

Types of Learning

Linear and Logistic Regression

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Artificial Intelligence (AI) Domains

AI

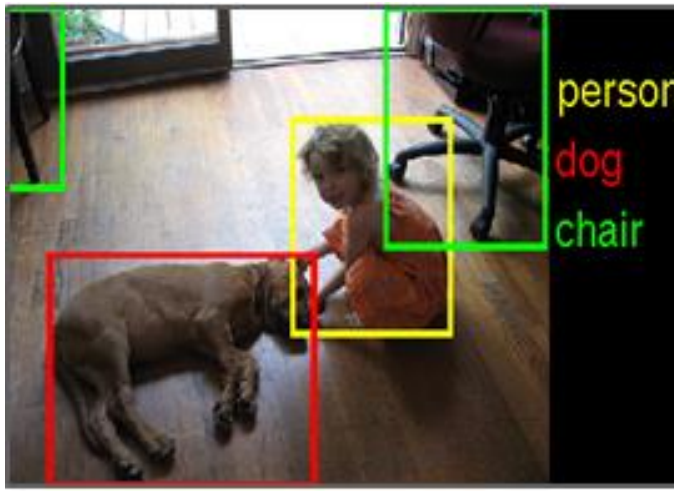
Artificial Intelligence (as of today): Detection and Exploitation of useful patterns and trends in data

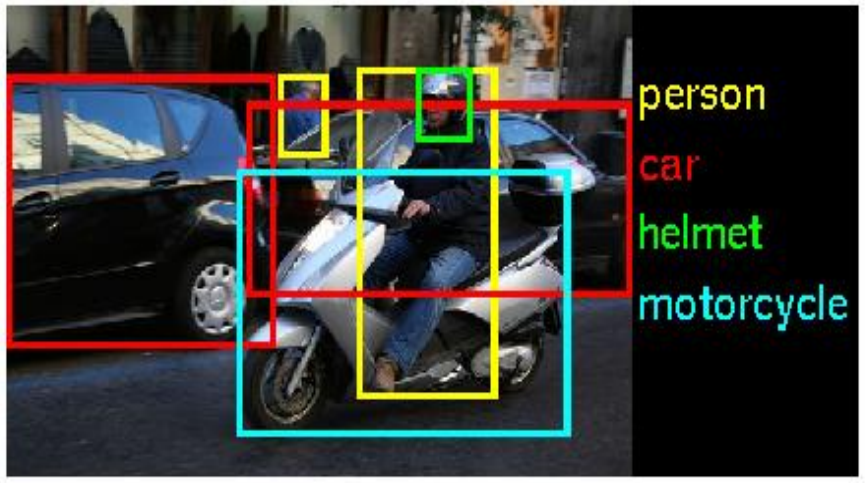
→Decisions

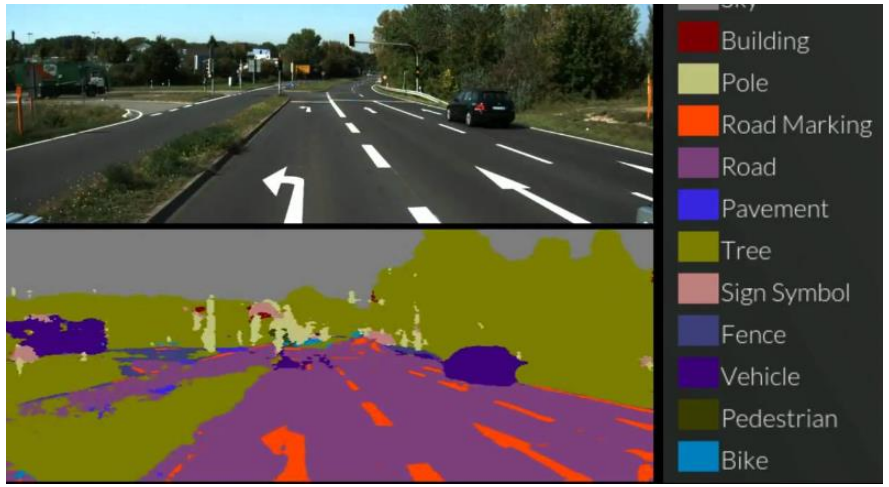
→ Predictions

→Automated Actions

Major Domains C1: Computer vision & Self Driving Cars



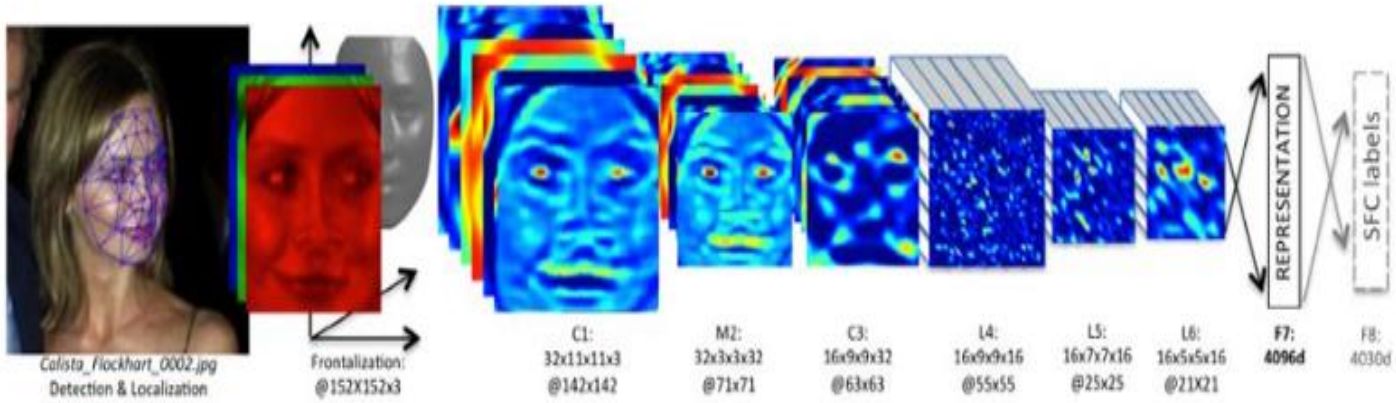




Source : Nvidia, L. Fridman et al.

Domains of AI

C2. Image processing: Shape & Object Detection



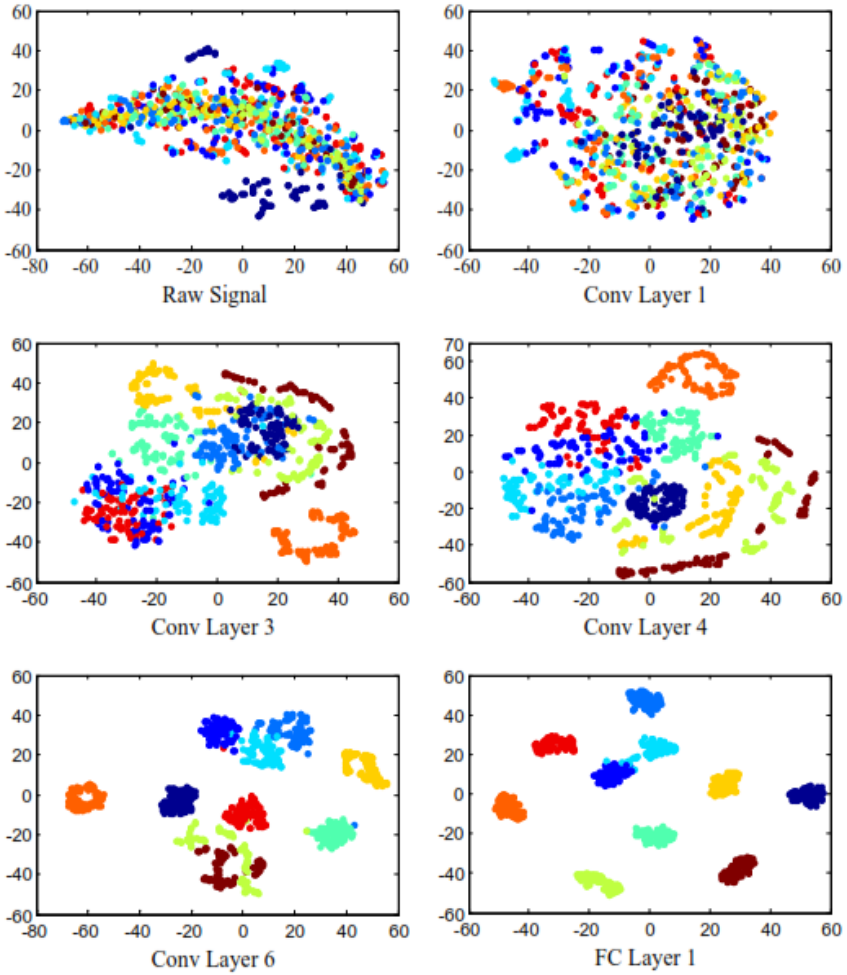
Face detection and Recognition.

Detection  Recognition "XYZ"

DeepFace: Closing the Gap to Human-Level Performance in Face Verification
Taigman, Yang, Ranzato, & Wolf (Facebook, Tel Aviv), CVPR 2014

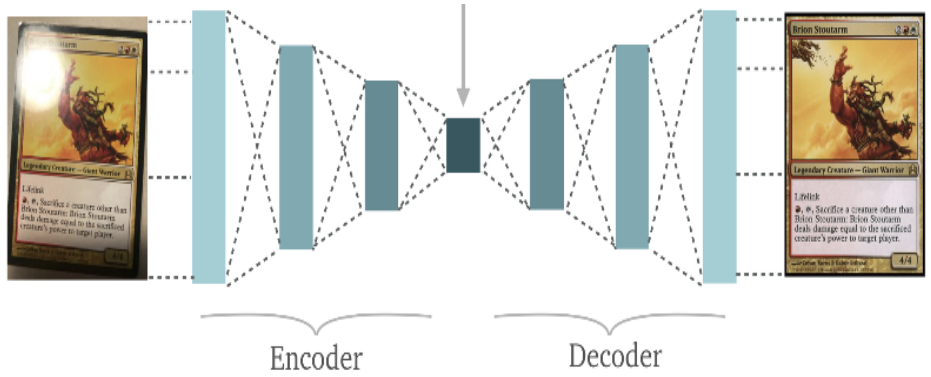
Predictive Maintenance
Fault Detection (Roller Bearing)

Zhang et al.

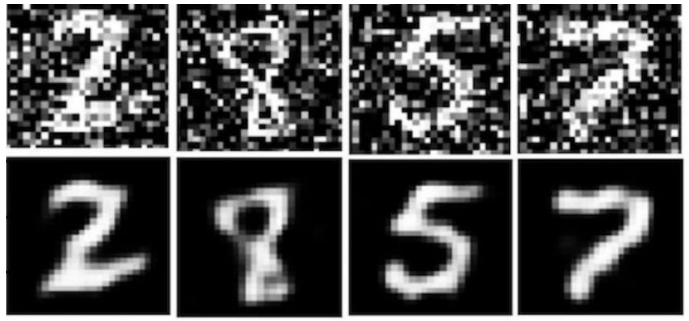


Domains of AI

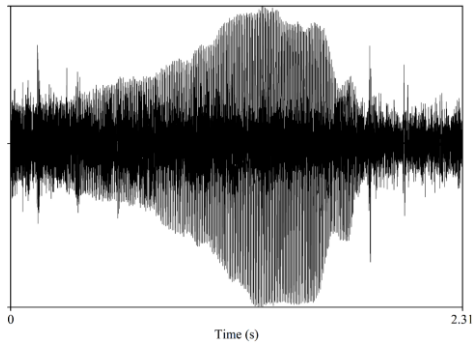
C3. Filtering and Denoising : Auto encoders



Link: Denoising autoencoder for Image classification



End to End learning: Fault detection and Prediction:
Unknown Model, Environment. (JHA et al. 2017)



Inputs

Learning in Black Box

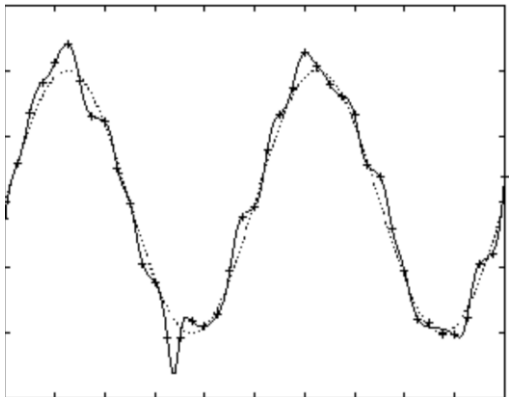
Black Box
- Feature extraction,
- selection,
- Unsupervised Learning

Decisions

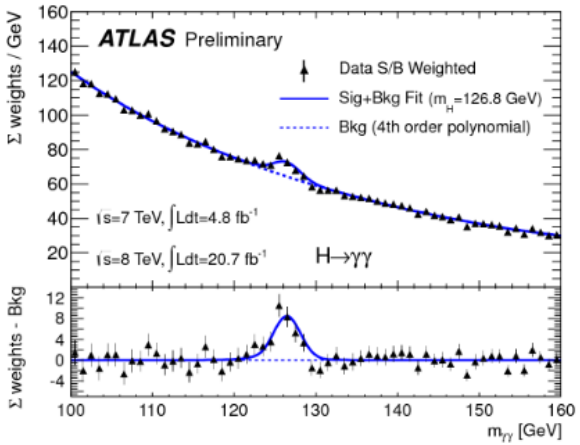
Denoising Autoencoders: MATLAB Central

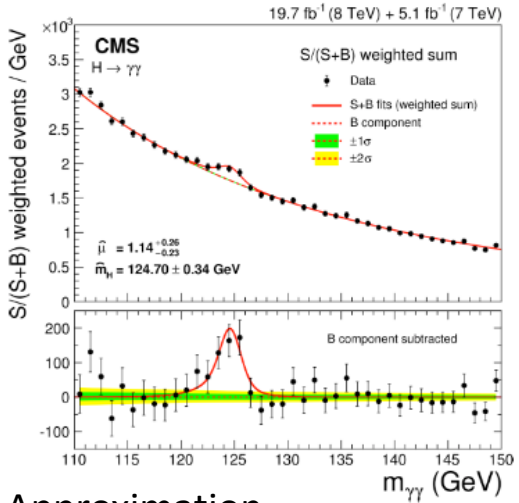
Domains of AI

C3. Particle Physics, Intelligent control (adaptive) of systems, Robotics: **Function Approximation**



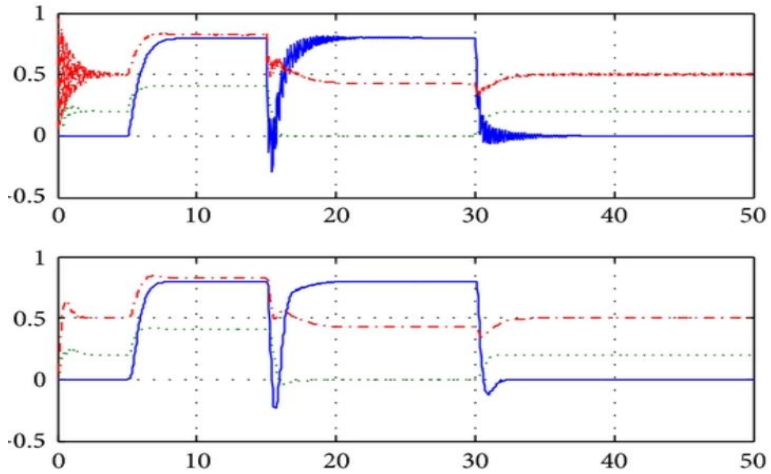
- Universal function approximators
- Efficient approximation of unknown dynamics .

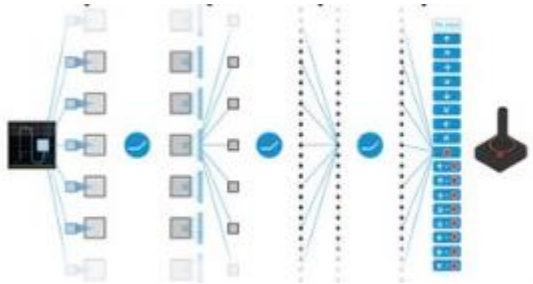


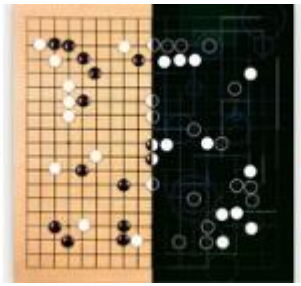


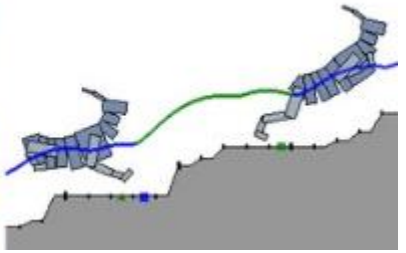
Deep learning enabled function Approximation in LHC physics.

Sirunyan et al. 2019, Physical Review letters









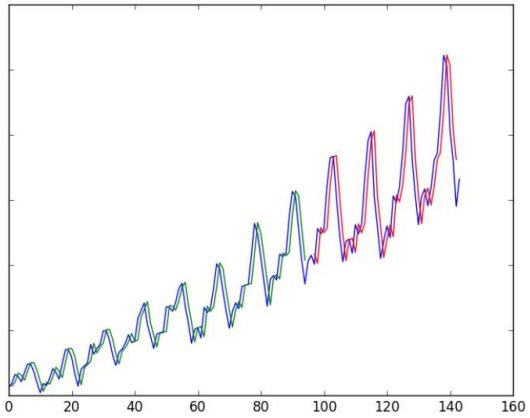
Deep learning **based function approximation of Extremely large state space** (World)

Human Level control through Deep Learning

Mnih et al. 2015, *Nature*

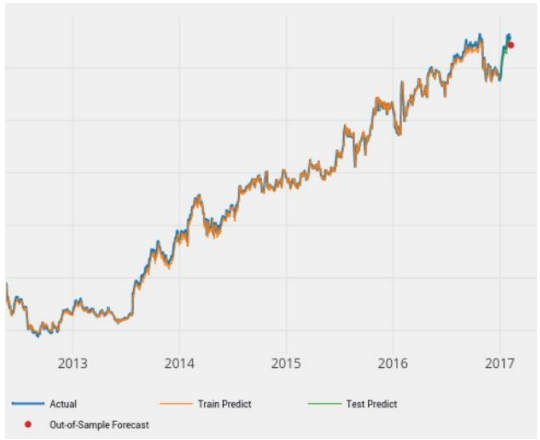
Domains of AI

R1: Time Series Forecasting, Trend Prediction, Event Prediction

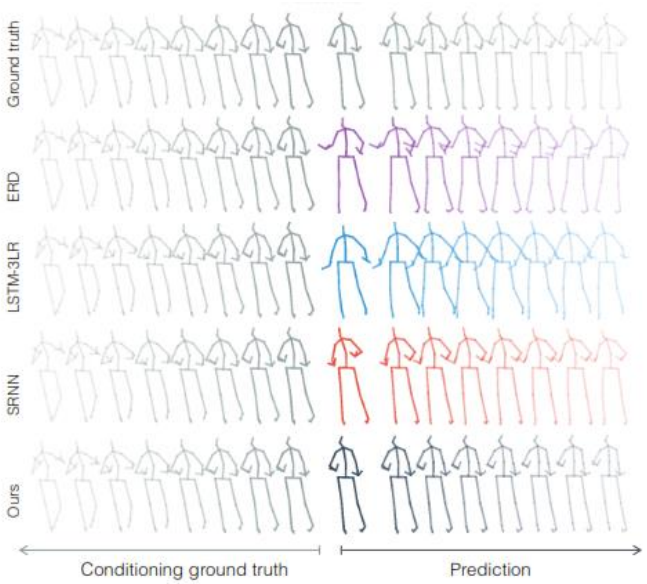


Long terms traffic Speed prediction

Ma, Xiaolei, et al

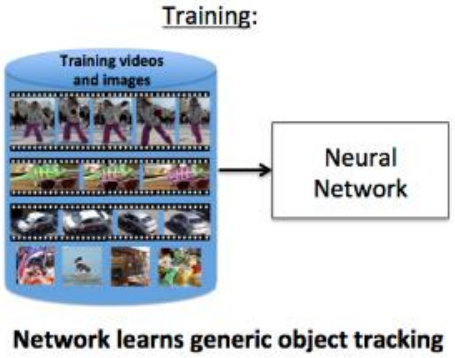


Financial market prediction (Dixon et al.)

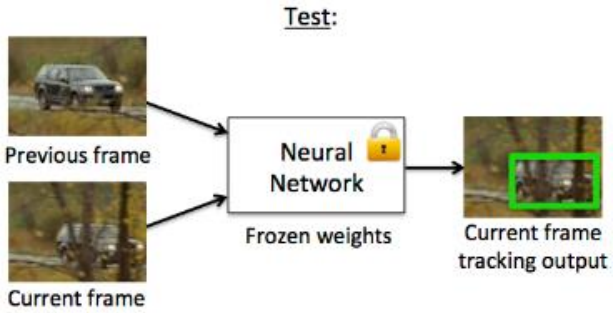


Human Motion Prediction

Martinez et al., 2016



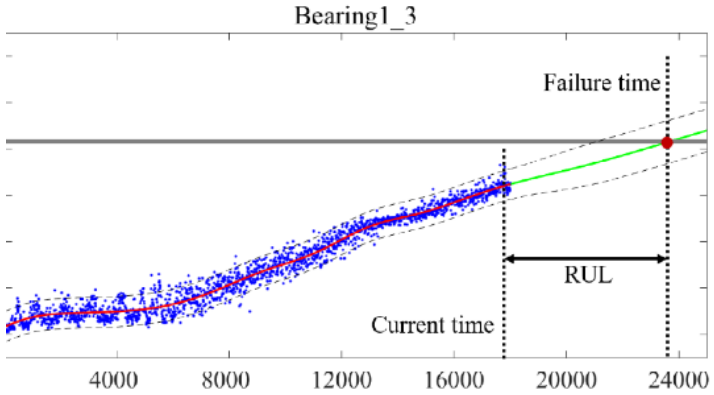
Network learns generic object tracking



Network tracks novel objects (no finetuning)

Video Frame tracking and Prediction

Held et al.



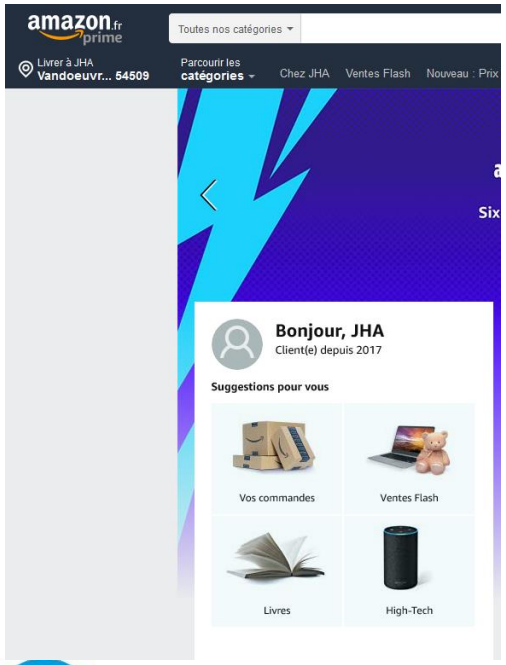
Component Failure Prediction (Yoo et al., 2018)

Domains of AI

R2 : Recommendation Systems

- Candidate Profiling,
- Scoring , similarity measures,
- Prediction

NETFLIX



DeepL

Google Translate

- google Translate
- voice recognition
- text prediction
- voice to text and vice versa
- echo cancellation

Google home Mini
Alexa

Sequence prediction often involves forecasting the next value in a real valued sequence.

POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN

Domains of AI

Reinforcement Learning: Towards human level :

control ((Finding the optimal way of doing a given task)

prediction

Adaptation (Robots That Can Adapt like Animals, *Nature*)

Built new moves

POLYTECH NANCY

Email: *mayank-shekhar.jha [at] univ-lorraine.fr*

Types of Learning

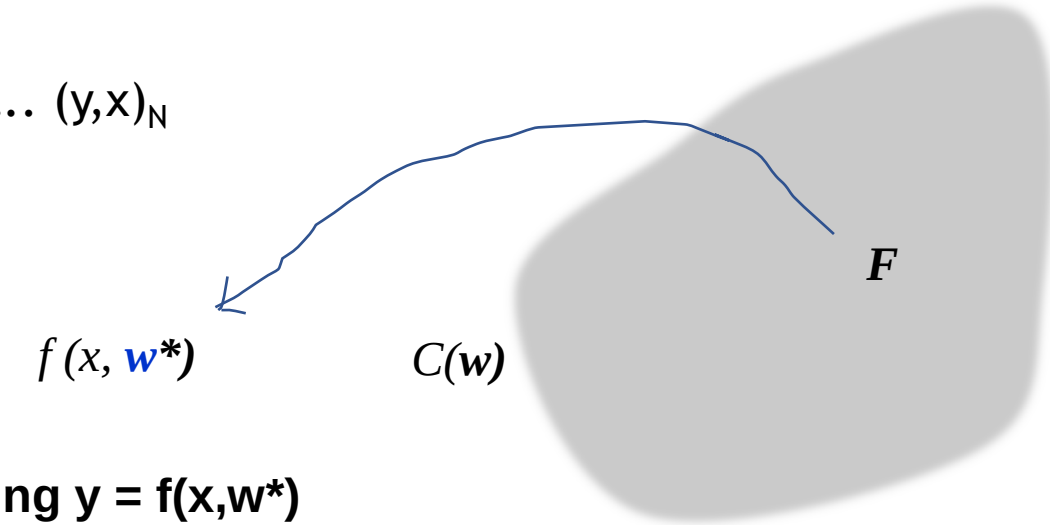
Learning : Supervised vs Unsupervised

Machine Learning: Study of algorithms that improve their **performance,** for a given **task**, with more **experience.**

Supervised, you know y. You know the true !

Training data: $\{y,x\}=(y,x)_1, (y,x)_2,\ldots. (y,x)_N$

Function space: F(x,w)
and constraints on function



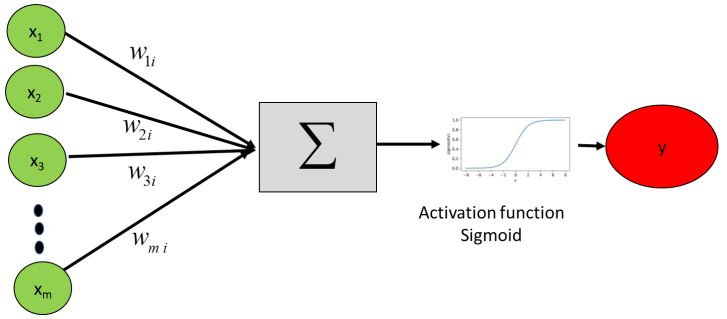
y = m1x1² + m2x2 + m3x3²
y = f(x, [m1, m2, m3]T)
y = f(x,w)

Teach a machine to learn the **mapping y = f(x,w*)**

Optimal parameters or "BEST "parameters

Objectif : find the f fonction AND the OPTIMAL parameters

Learning : Supervised



Training data: $\{y,x\}=(y,x)_1, (y,x)_2,\ldots. (y,x)_N$

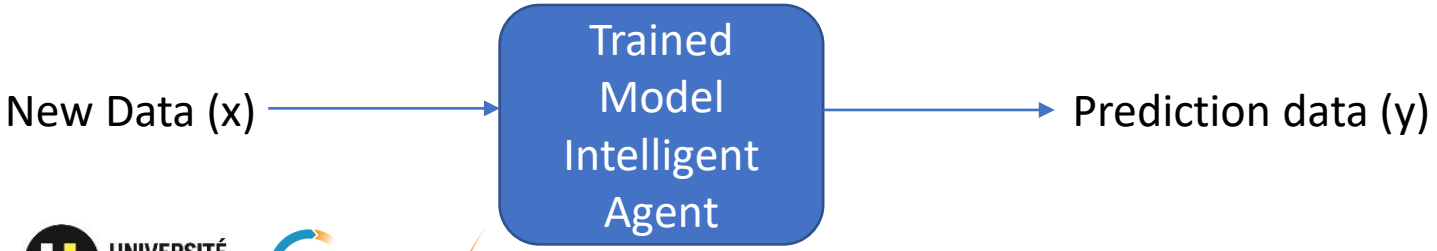
Function space: F(x,w)
and constraints on function

Teach a machine to learn the **mapping y = f(x,w*)**

Supervised learning:

- Training of intelligent agent under 'supervision'.
- Model known, environment known.
- Data sources, labels known!
- An algorithm is employed to learn the mapping function from the input variable (x) to the output variable (y) and optimal function parameters: that is $y=f(x, \boldsymbol{w^*})$

- Objective: Mapping function estimated accurately→ Agent Intelligent! WHY??



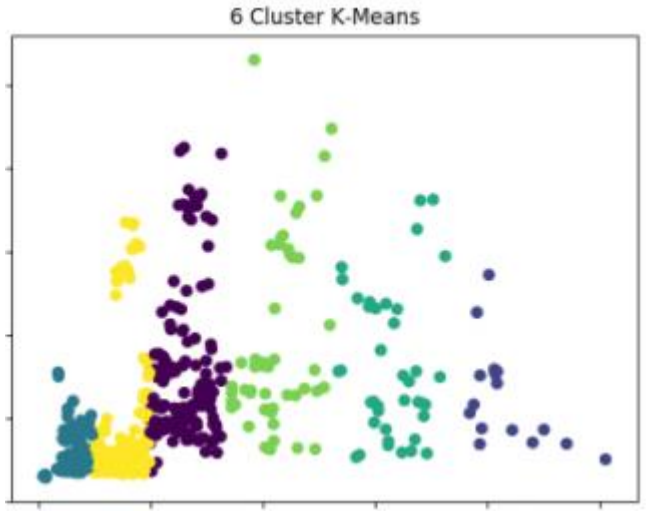
Learning : Unsupervised

Unsupervised learning = Available input data (X) and NO output .

- LEARNING DONE IN AUTONOMOUS WAY.

- The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

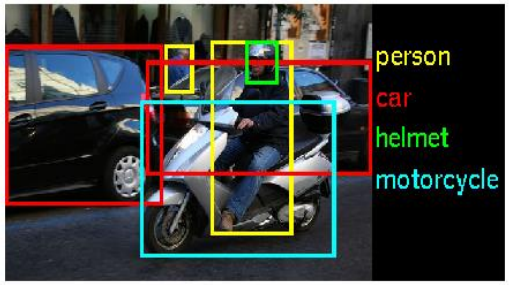
Example: K-mean clustering (using distance measures , similarity index, other ranking algos)

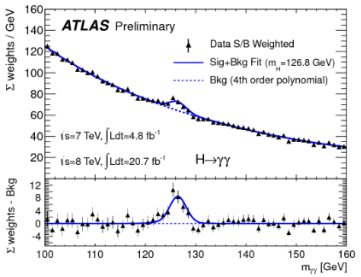


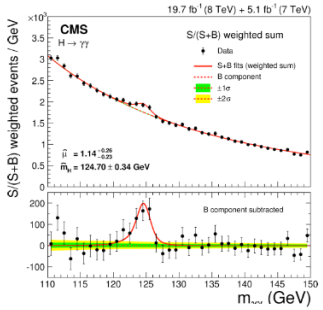
There is no correct answer and there is no teacher.

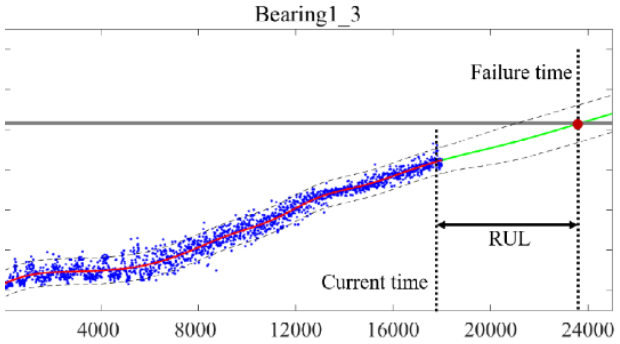
Algorithms are left to their own to discover and present the interesting structure in the data.

Remark: Most learning (in practice) **: supervised.**

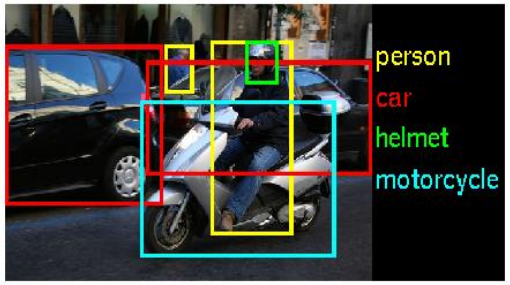


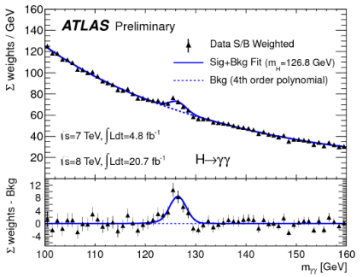


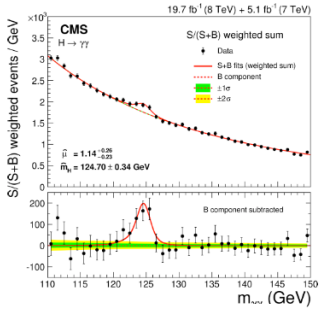


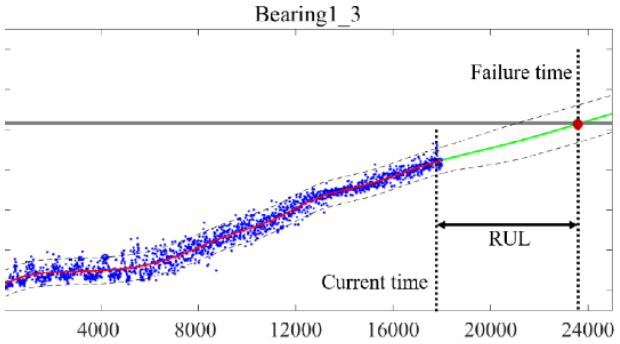


Remark: Most learning (in practice) : **supervised.**

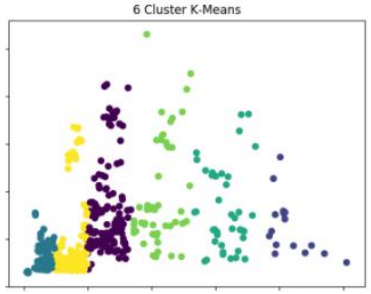


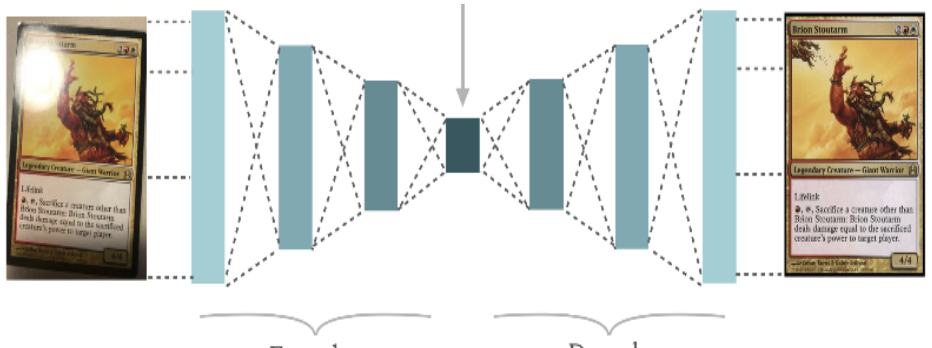


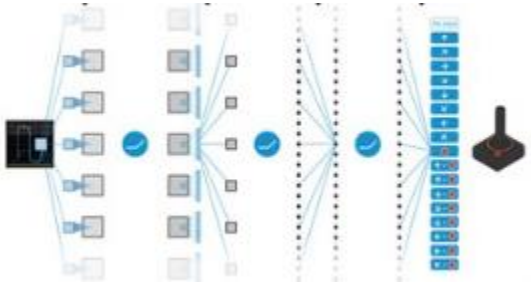


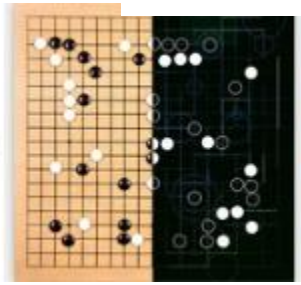


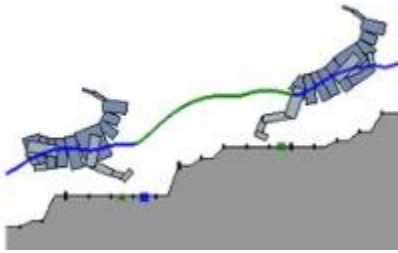
Remark: Most learning (in research) : **Unsupervised, RL**



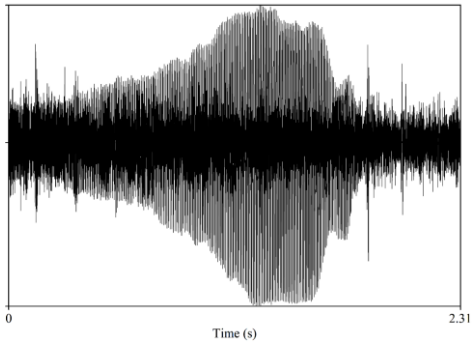








End to End Learning in Black Box

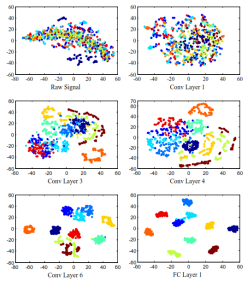


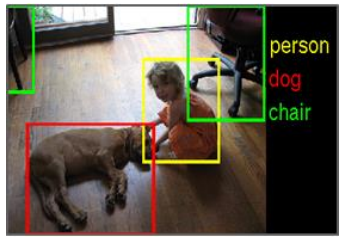
Black Box

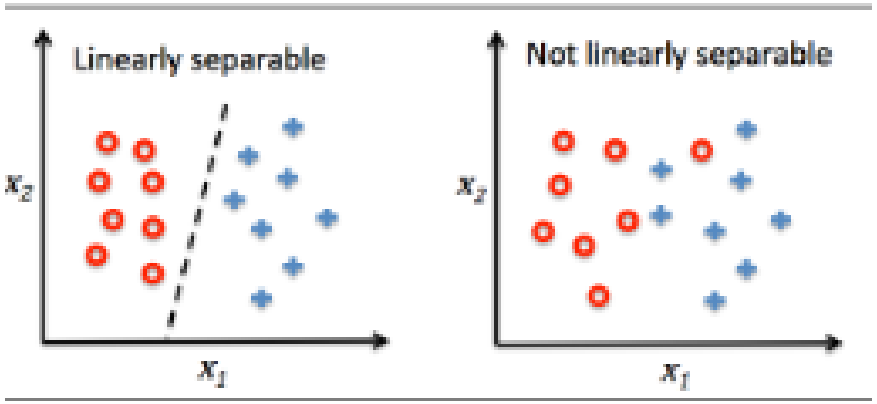
- Feature extraction,
- selection,
- Unsupervised Learning

Decisions

Basic Processes: Classification and Regression



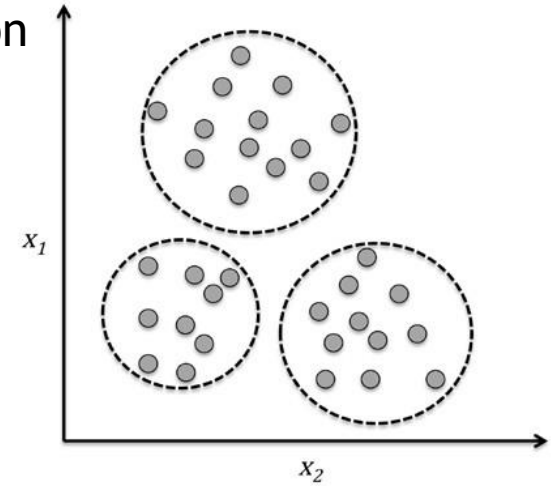




Classification : Prediction of Categorical variables (Labels)

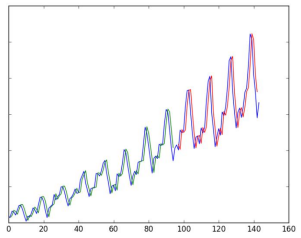
Multi-class Classification

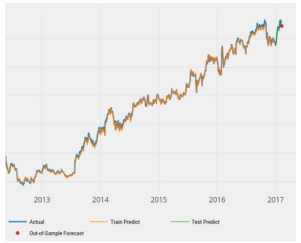
- Inter class: Maximum separation
- Inter class: Minimal variance

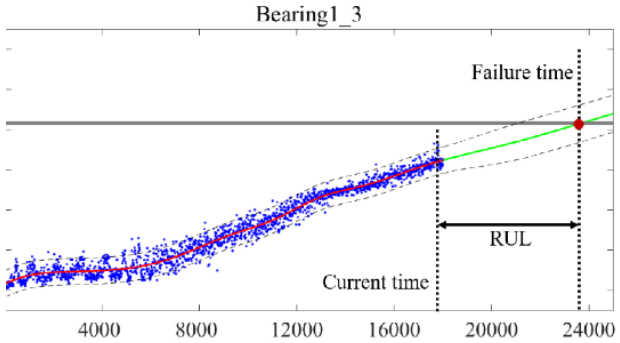


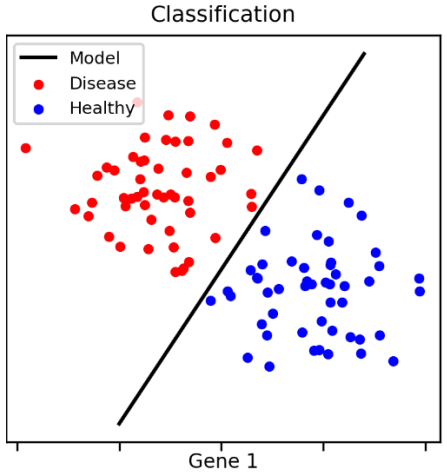
Basic Processes: Regression

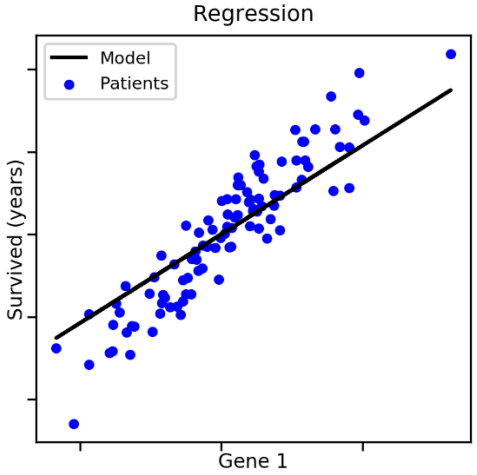
Regression: Prediction of numerical or continuous output variables



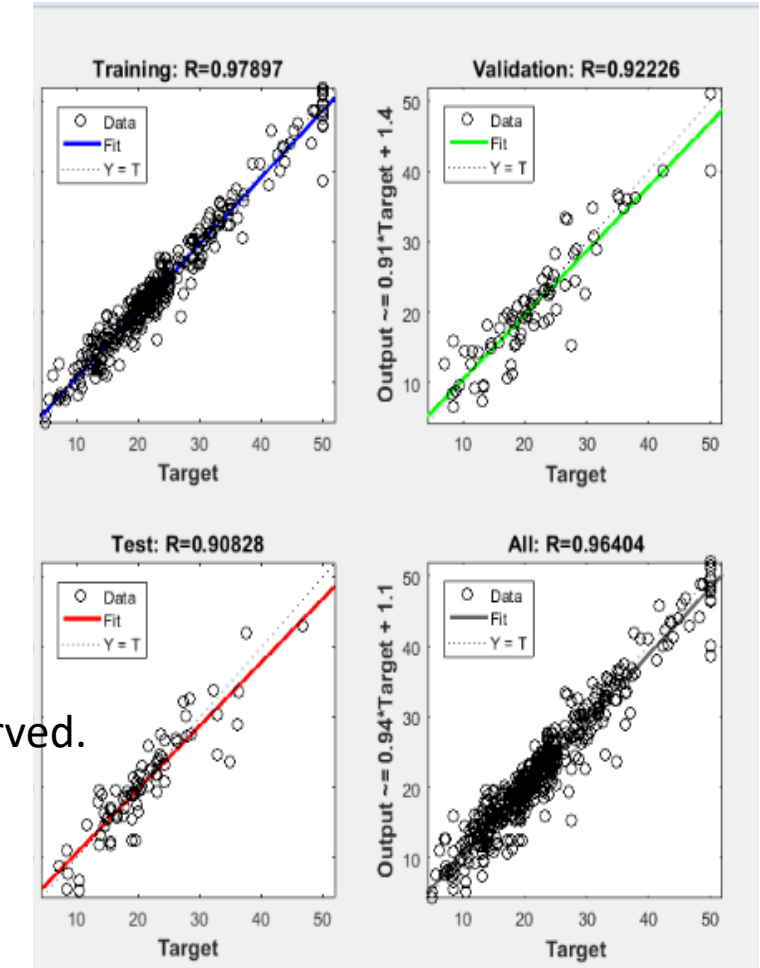








Source: Personal tutorials, also see: Park et al. 2015, Nature genomics



NETFLIX

- Forecasting of object based upon the past dynamics (behavior), historical trends observed.
- Sequence to sequence Model → next sequence prediction, long time prediction.

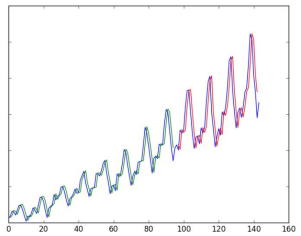
POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN

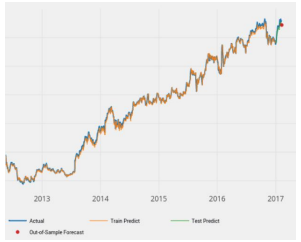
Basic Processes: Regression

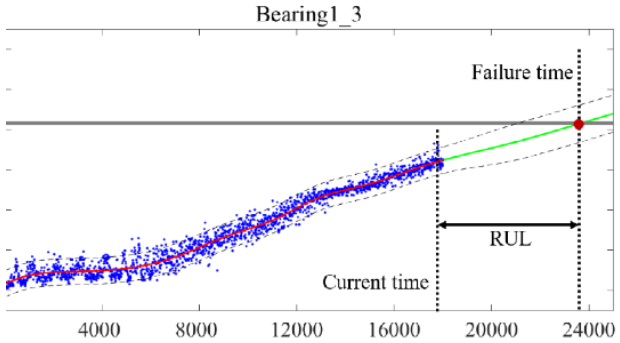
Regression: Prediction of numerical or continuous output variables

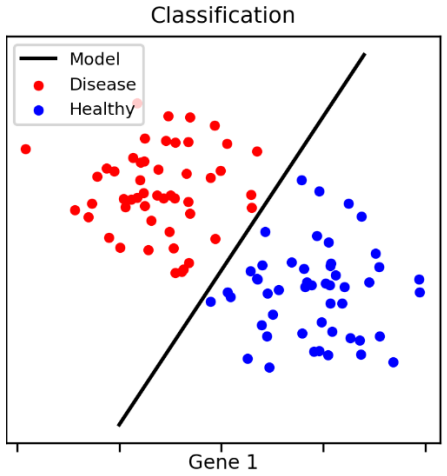
$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 \ldots\ldots + b$$

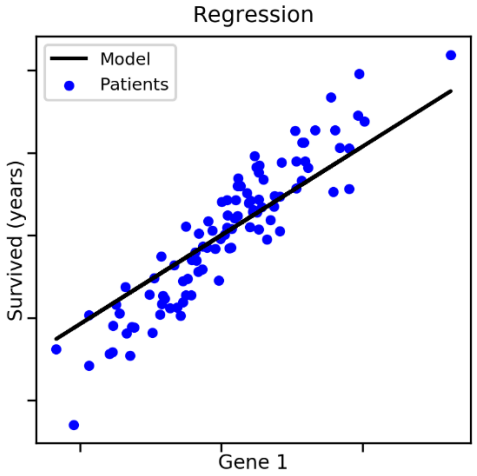
$$= \sum_{i=1}^{m} w_i x_i + b$$



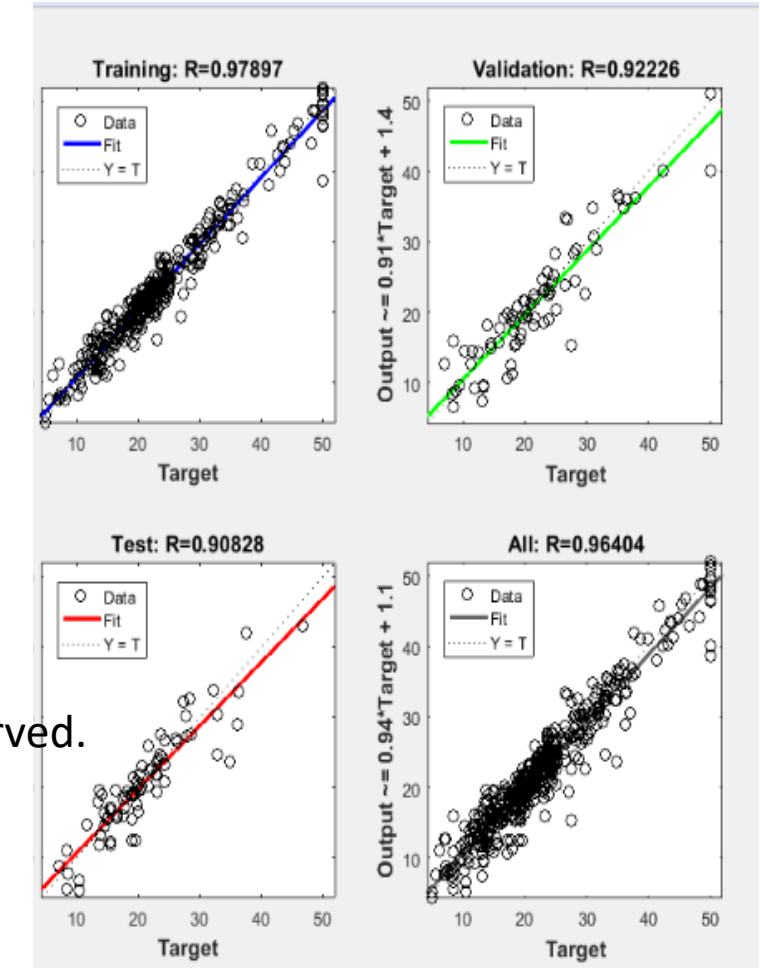








Source: Personal tutorials, also see: Park et al. 2015, Nature genomics



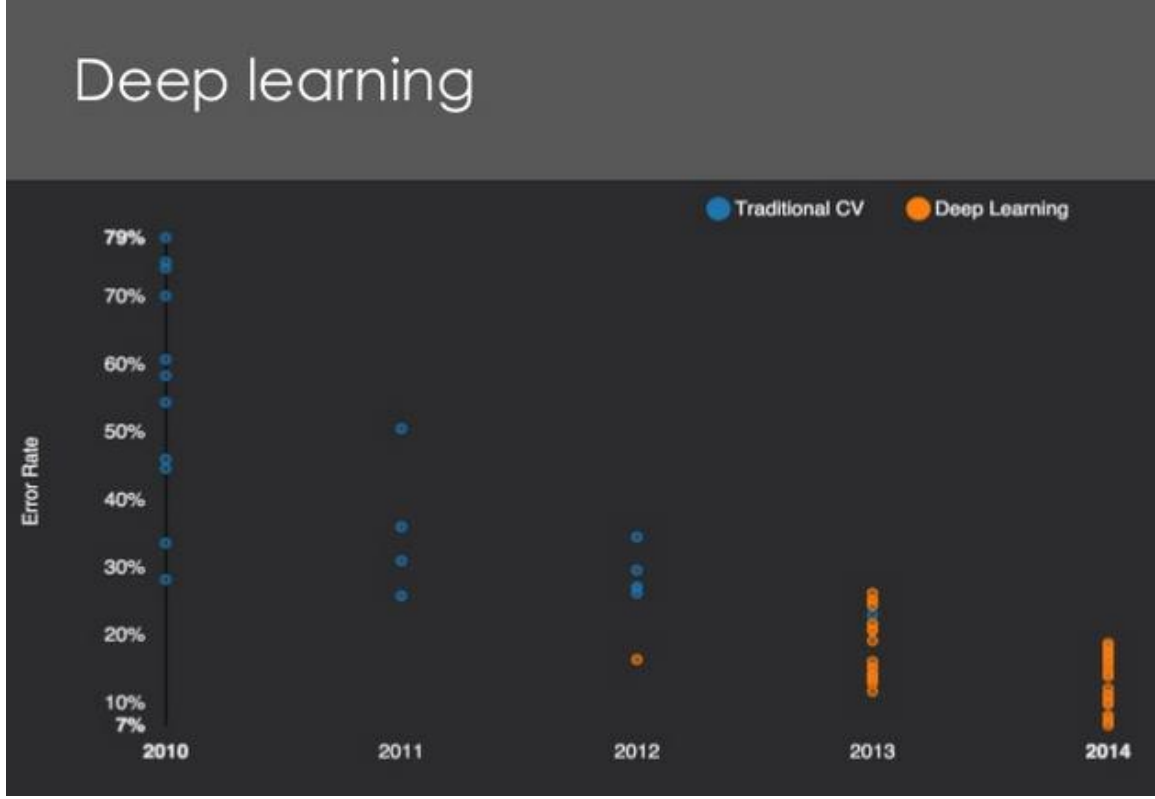
NETFLIX

- Forecasting of object based upon the past dynamics (behavior), historical trends observed.
- Sequence to sequence Model → next sequence prediction, long time prediction.

POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN

Machine Learning techniques for AI

Naïve Bayes,
Kernel Density Estimation
Rule Based,
Decision Trees,
Random Forests
Genetic Algorithms



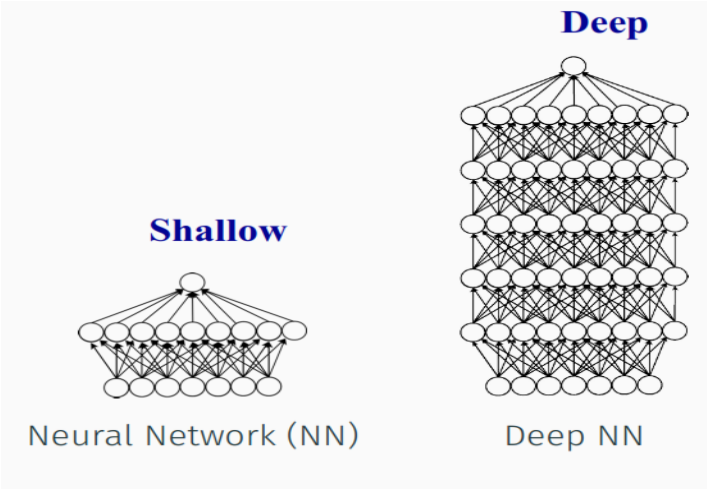
Support vector machines (1990-2007): very promising, better than NNs….till 1998.

Neural networks (NNs) (1960-1986, 1986-1998, 1998-2007)

Deep Neural Networks (1998,DNNs) : CNNs revolutionized NN based works,

Enter 2007,

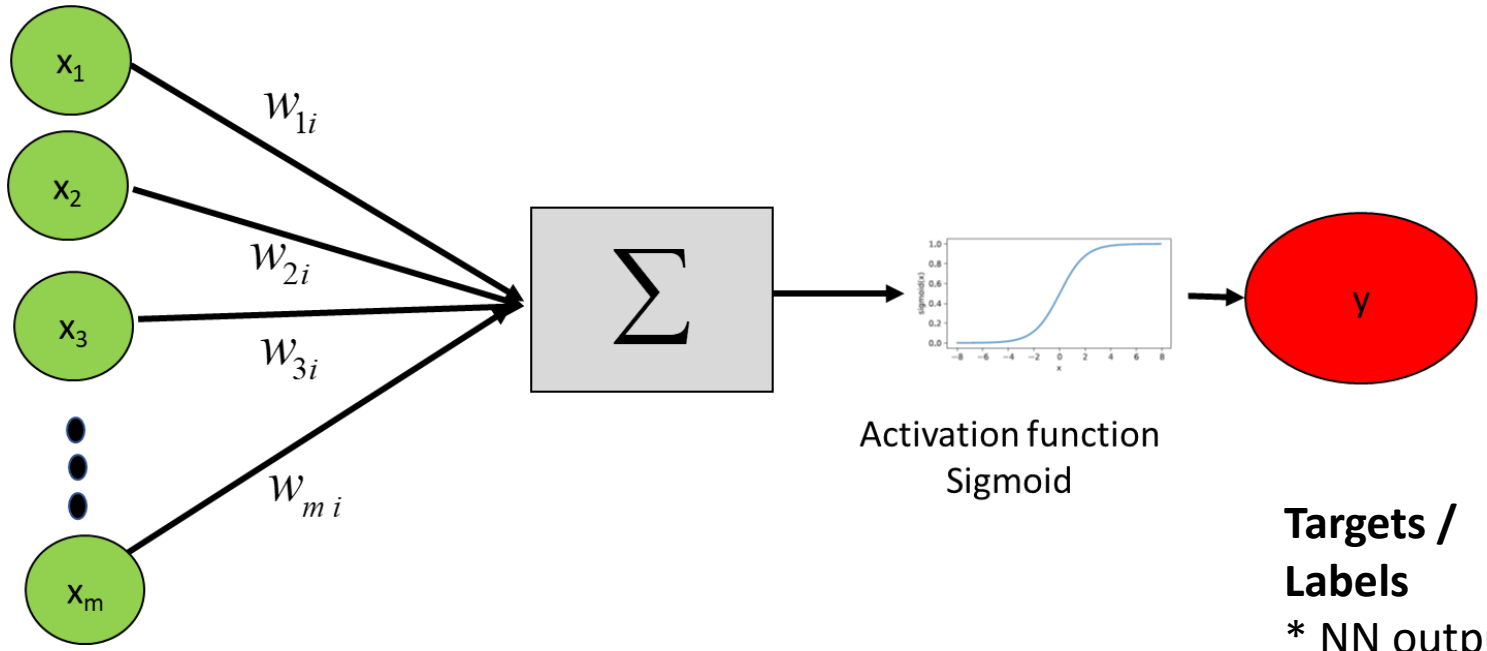
- Availability of data & data acquisition methods,
- GPU based distributed calculations
- Huge community of developers
- Surge in DNN



Learning Using Deep Neural networks : Supervised Learning

In this lecture, we look at **Neural Networks** and Mechanism of **Supervised type learning .**

Input Features/ Features Input data/ Inputs



Activation function
Sigmoid

Targets / Labels

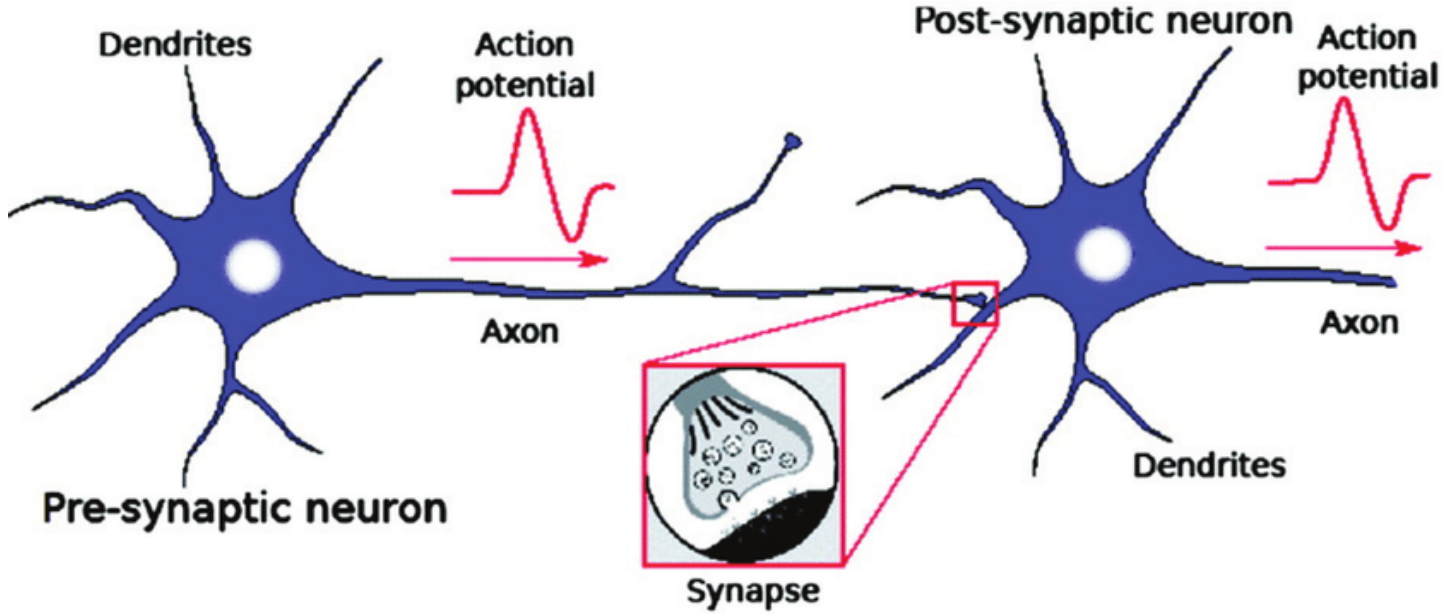
* NN output may or may not be equal to the target. Why?

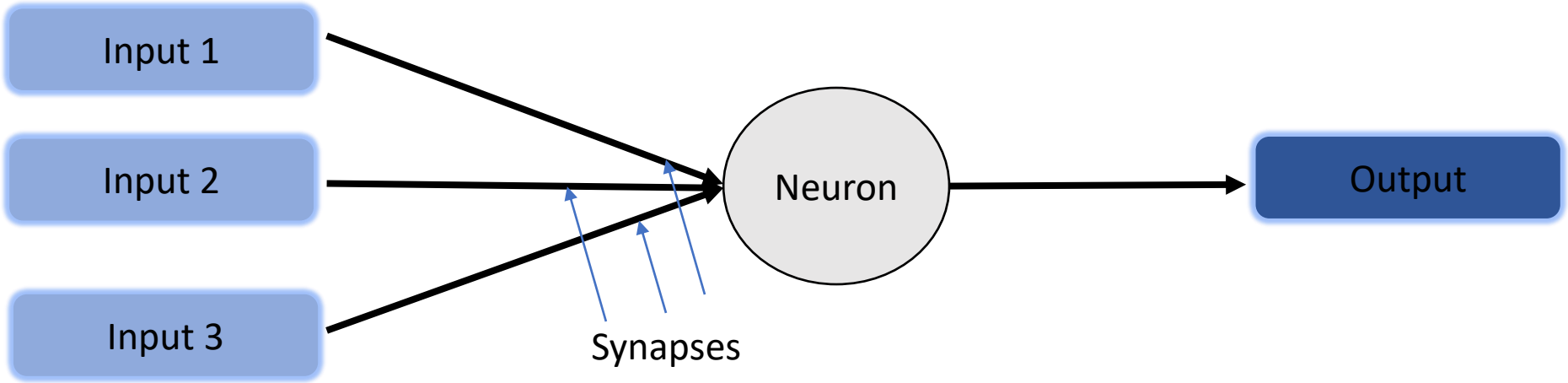
The Neuron

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

The Neuron

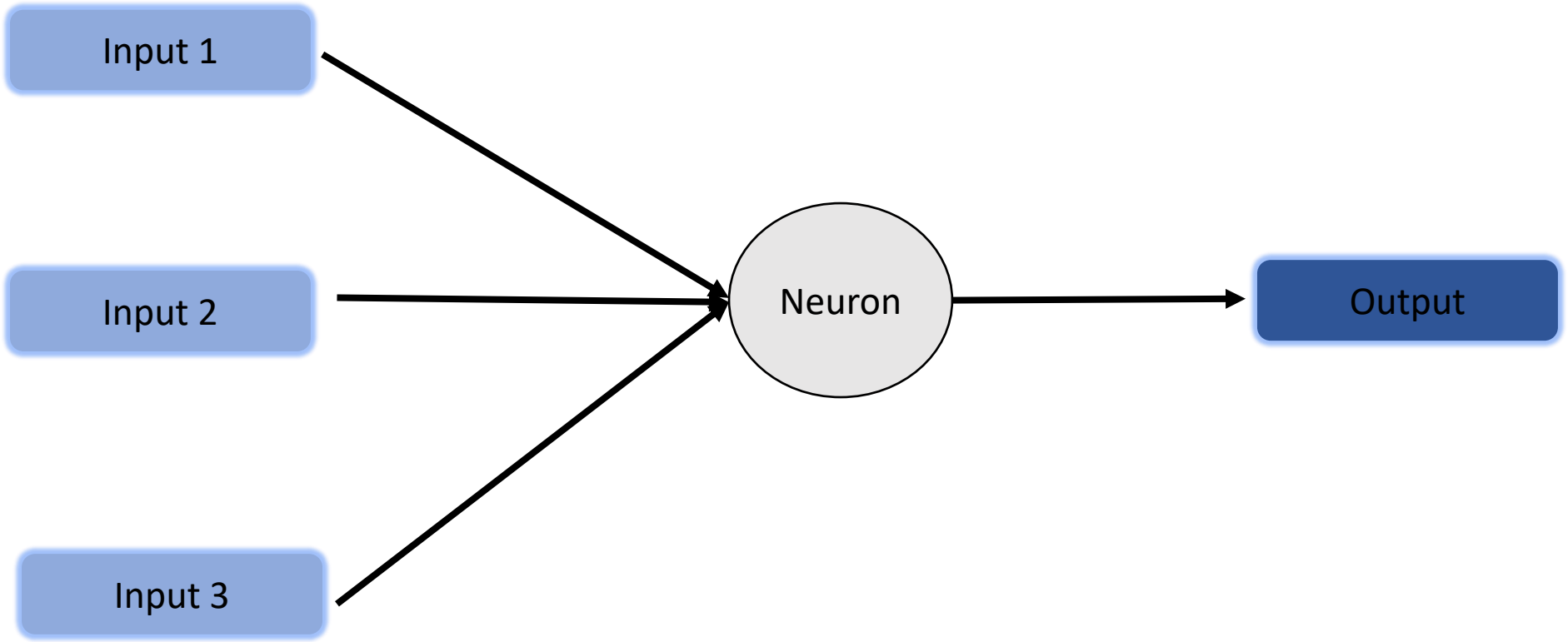
- A neuron only fires if its input signal exceeds a certain amount (the threshold) in a short time period.



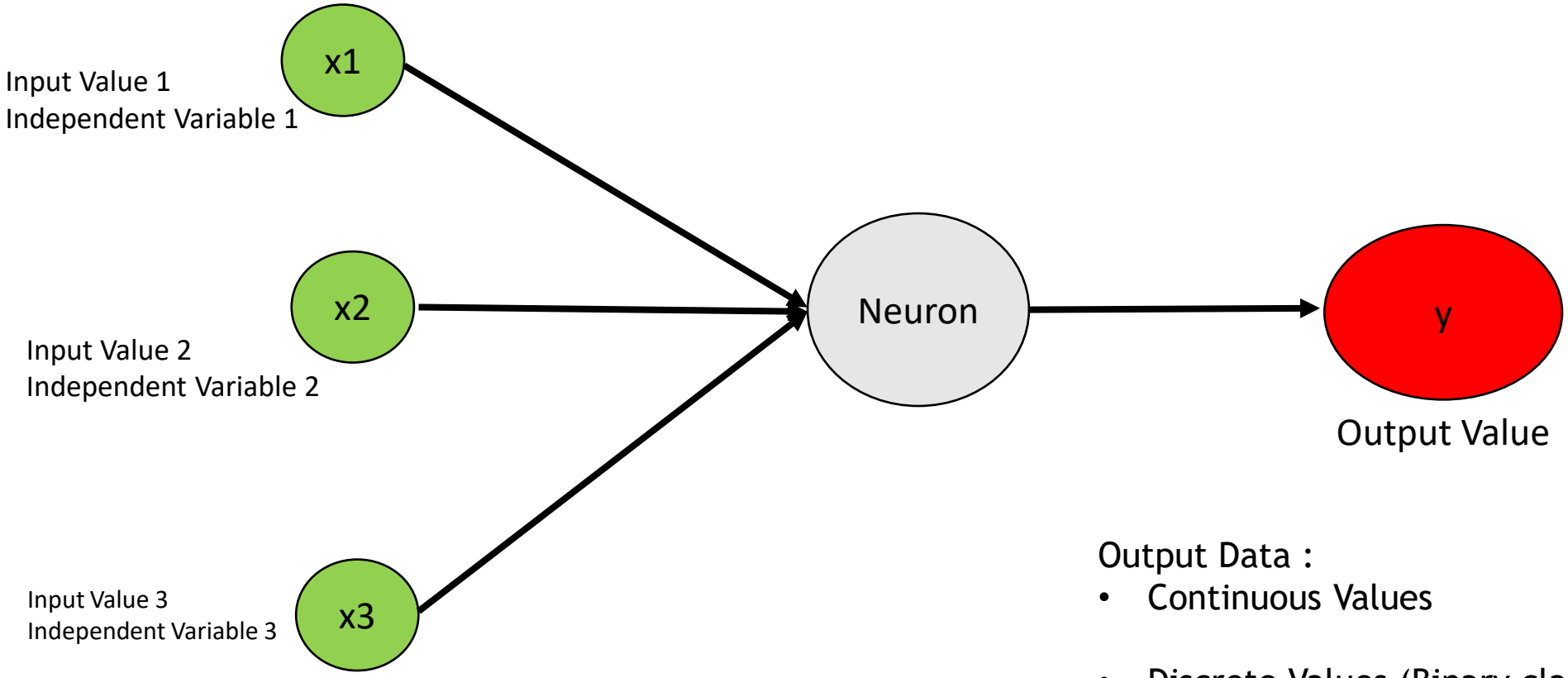


Huang, Anping, et al. 2017.

The Neuron



The Neuron

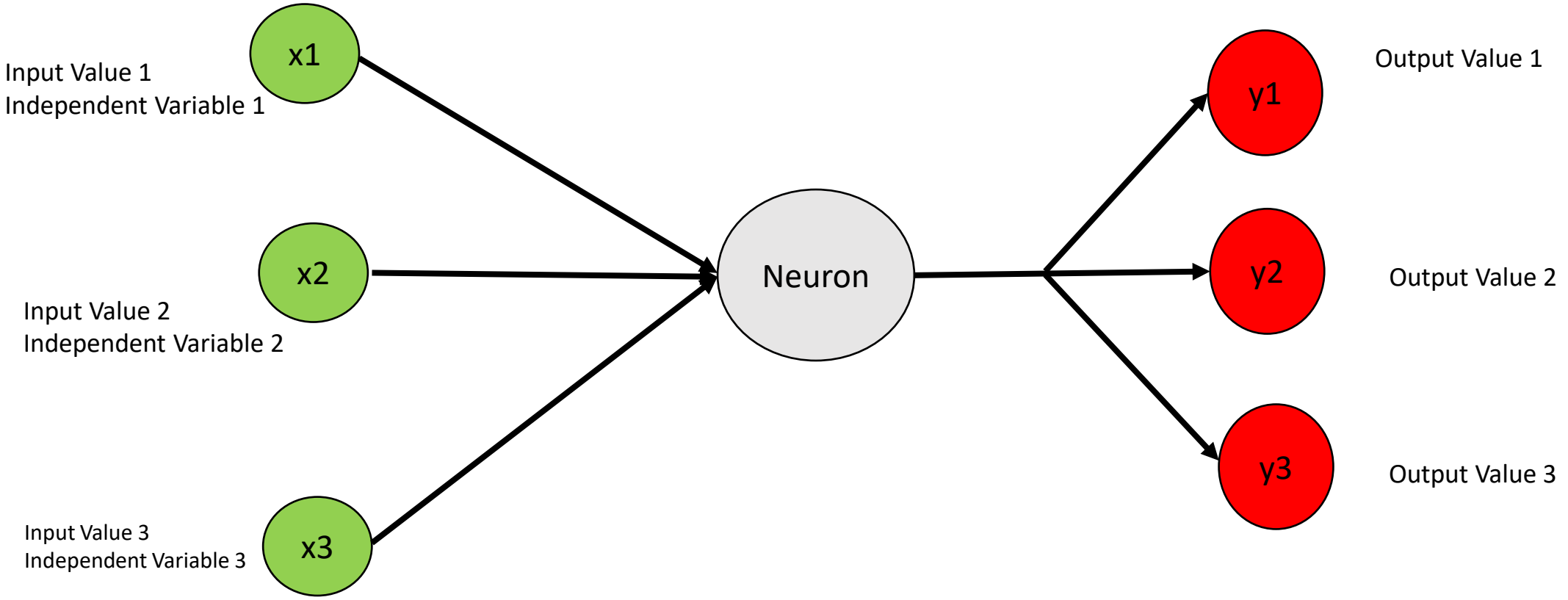


- Standardization of input data (Same Scale)
- Data preprocessing

Output Data :

- Continuous Values
- Discrete Values (Binary classes → Yes/No..)
- Categorical Variables (very small, small, large, very Large)

The Neuron



- Standardization of input data (Same Scale)
- Data preprocessing

Single Observation

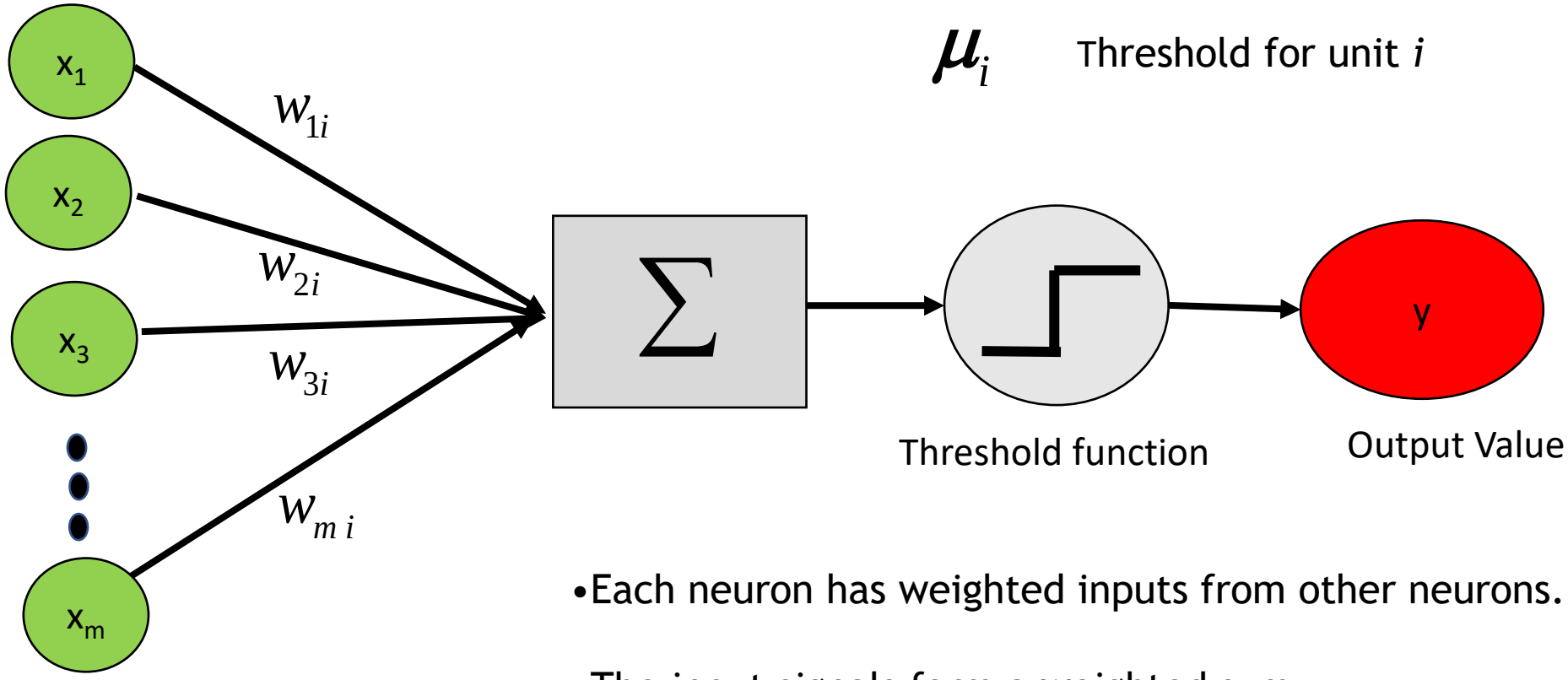
Same Observation
(Input data , Output Lable

Single Observation

The Neuron: Basic Perceptron

$w_{m\,i}$ Synaptic weight from unit *m* to unit *i*

μ_i Threshold for unit *i*

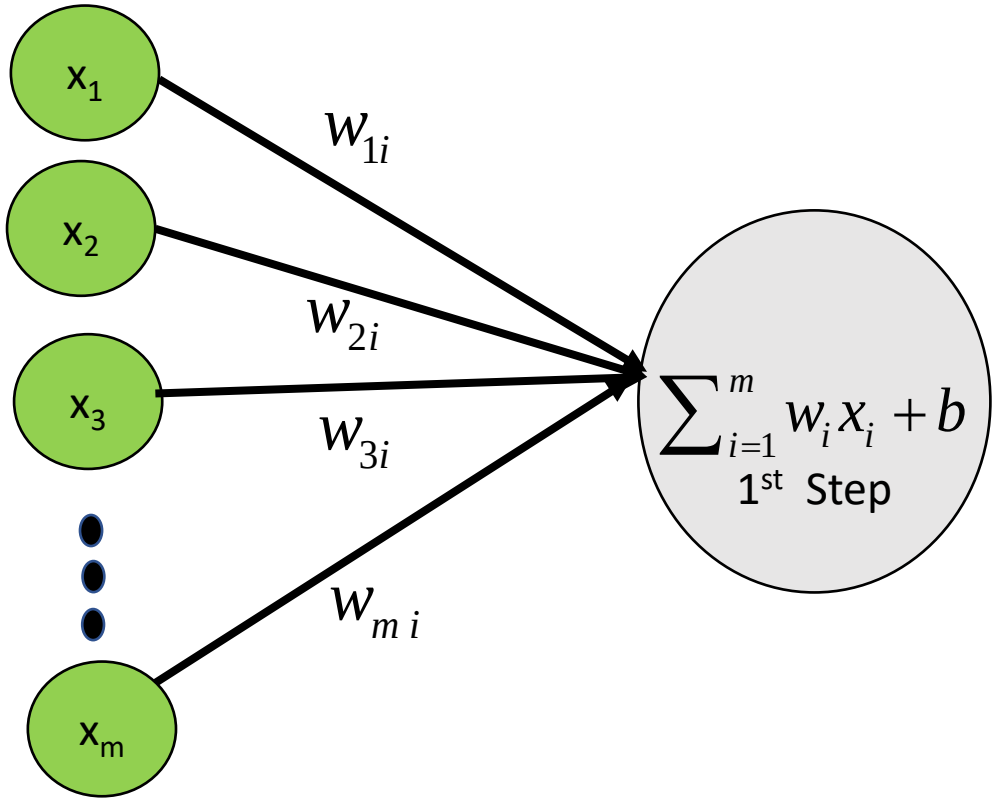


- Each neuron has weighted inputs from other neurons.
- The input signals form a weighted sum.
- If the activation level exceeds the threshold, the neuron “fires”.
- Each neuron has a threshold value.

Artificial Neural Networks (ANNs)

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 ..... + b$$

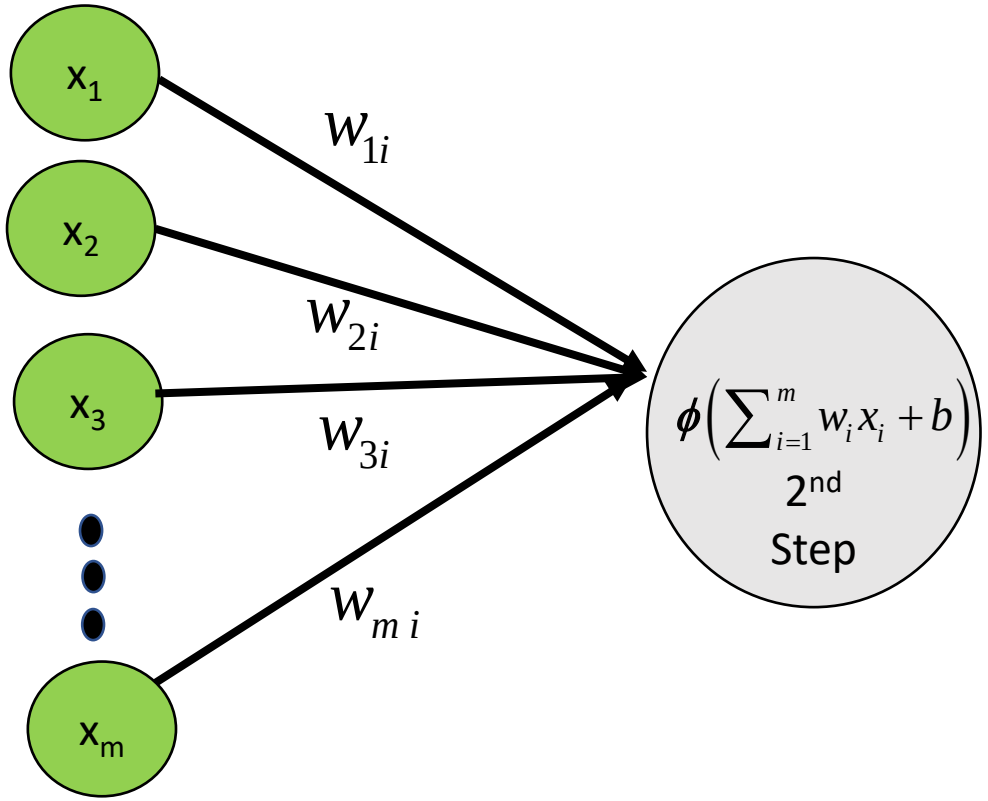
$$= \sum_{i=1}^{m} w_i x_i + b$$



Output Value

- Each hidden or output neuron has weighted input connections from each of the units in the preceding layer.
- The unit performs a weighted sum of its inputs, and subtracts its threshold value, to give its activation level

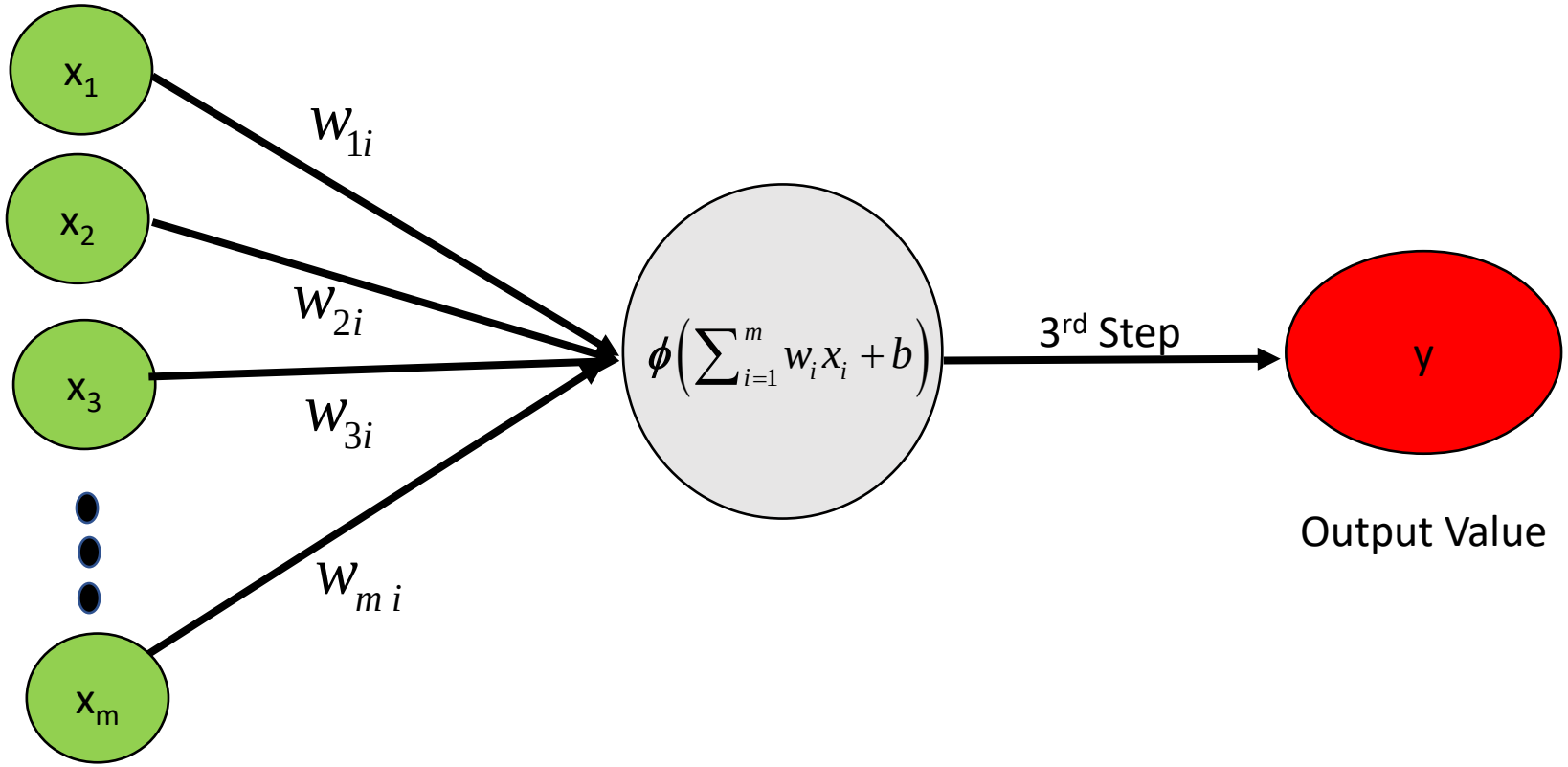
Artificial Neural Networks (ANNs)



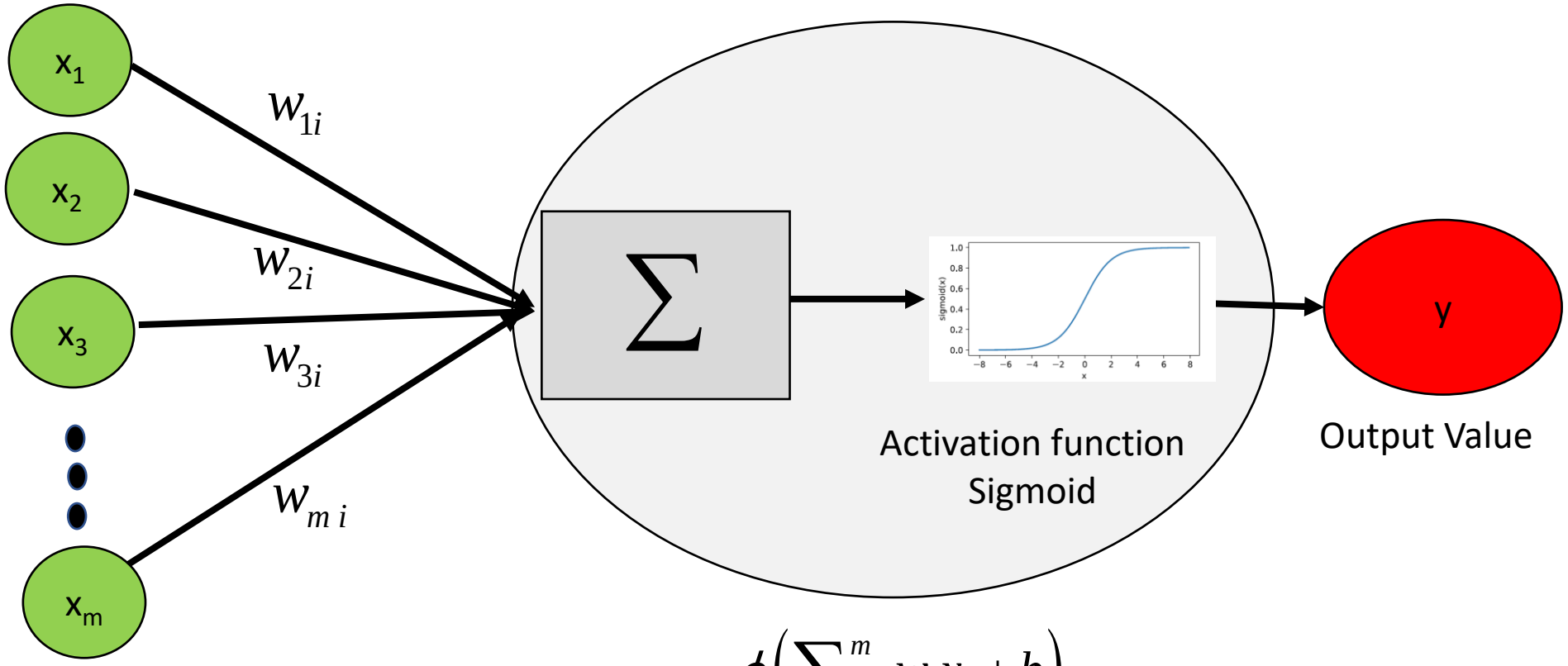
Output Value

- Activation level is passed through an activation function $\phi(x)$ to determine output

Artificial Neural Networks (ANNs)

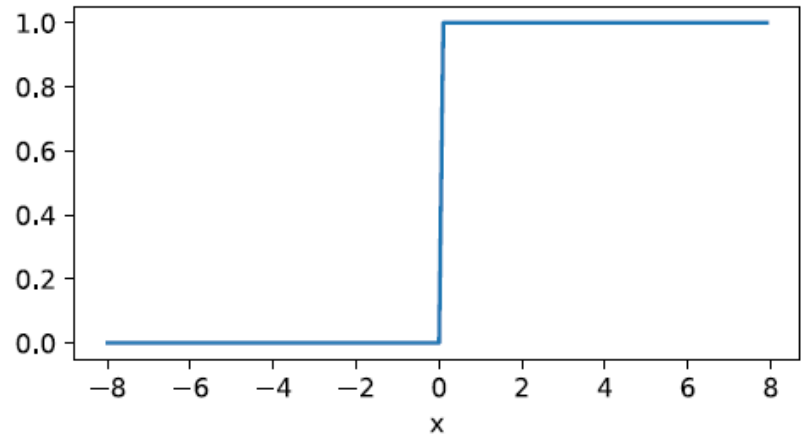


The Artificial Neural Network (ANN)

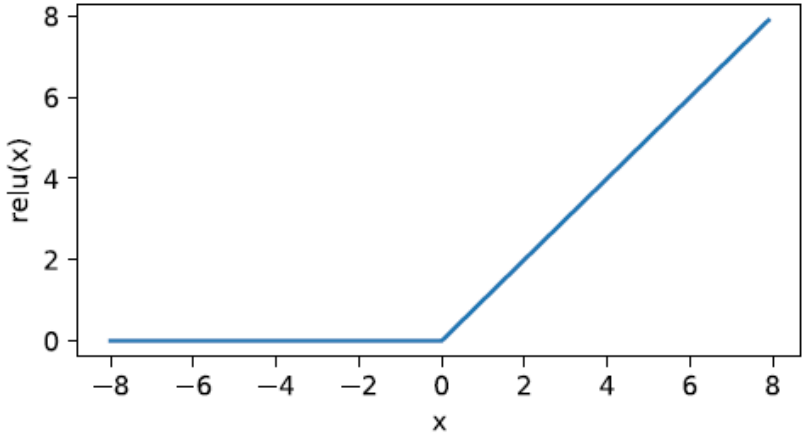


$$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

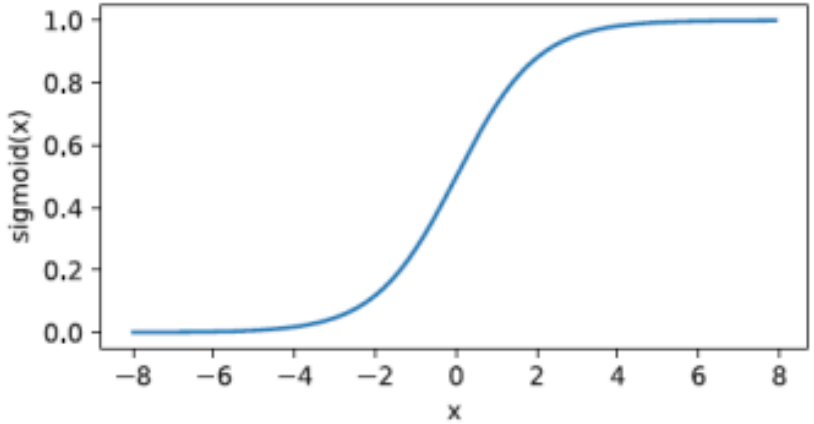
Activation functions (discussed later)



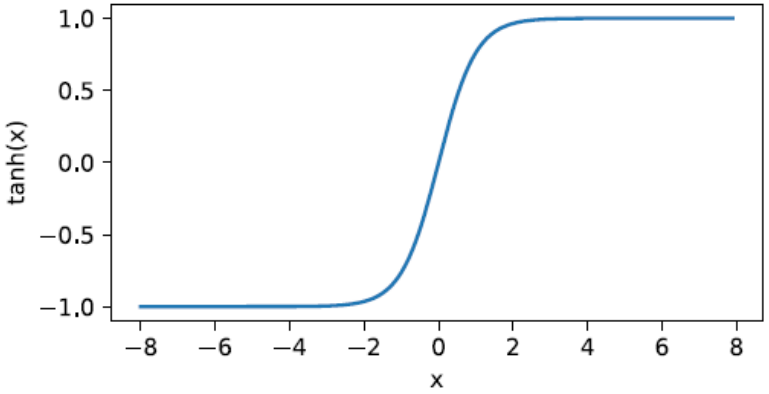
Threshold function (binary step function)



ReLu (Rectified Linear Unit)

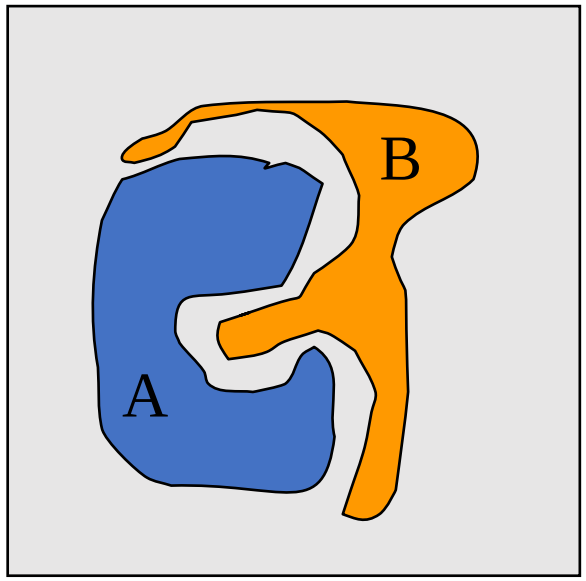


Sigmoid function

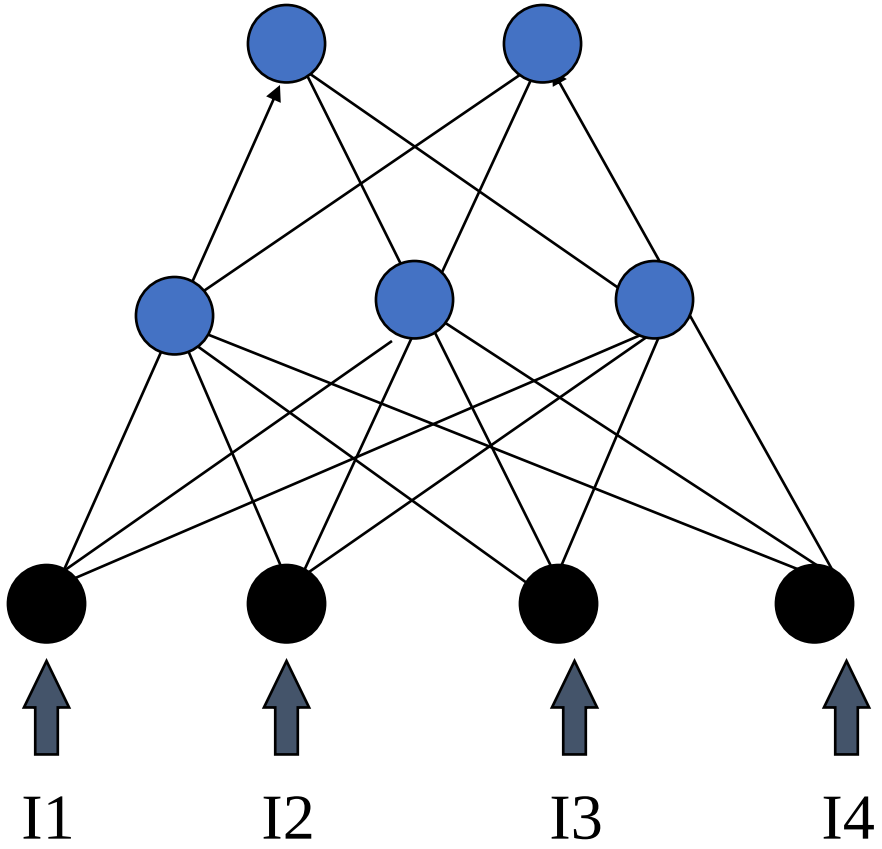


TanH / Hyperbolic Tangent

Multi Layered (Deep) Feed Foreword Neural Networks



3-Layer Network
has
2 active layers



Output Nodes

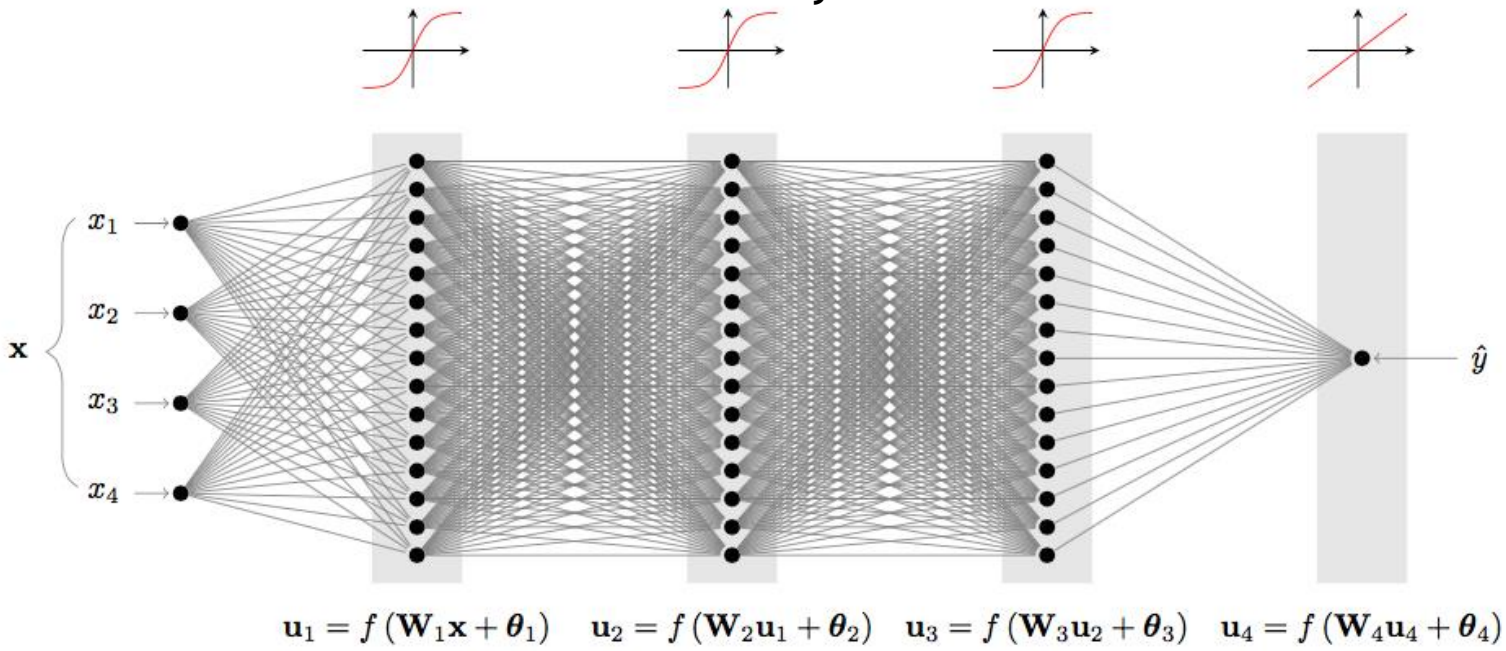
Hidden Nodes

Input Nodes

Multi Layered (Deep) Feed Foreword Neural Networks

Hidden Layers

Inputs



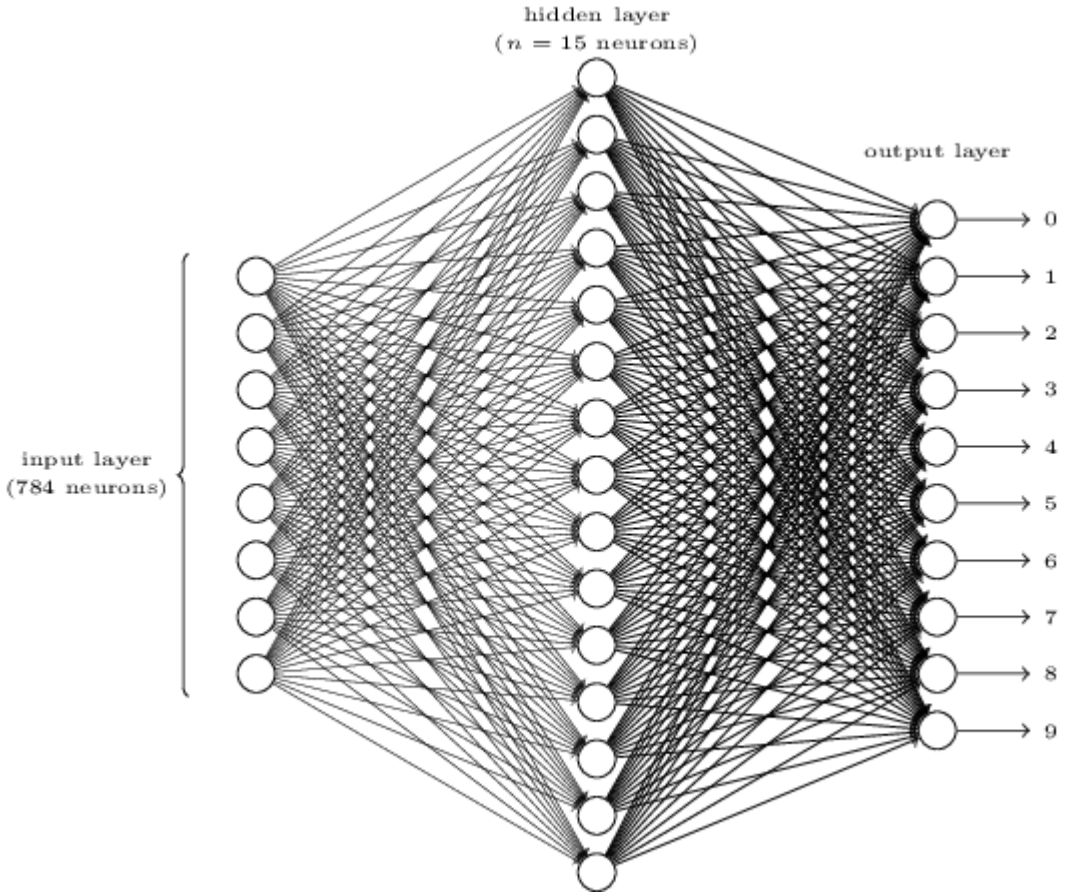
Outputs

- These are fully connected layers, **but need not be.**

Multi Layered (Deep) Feed Foreword Neural Networks

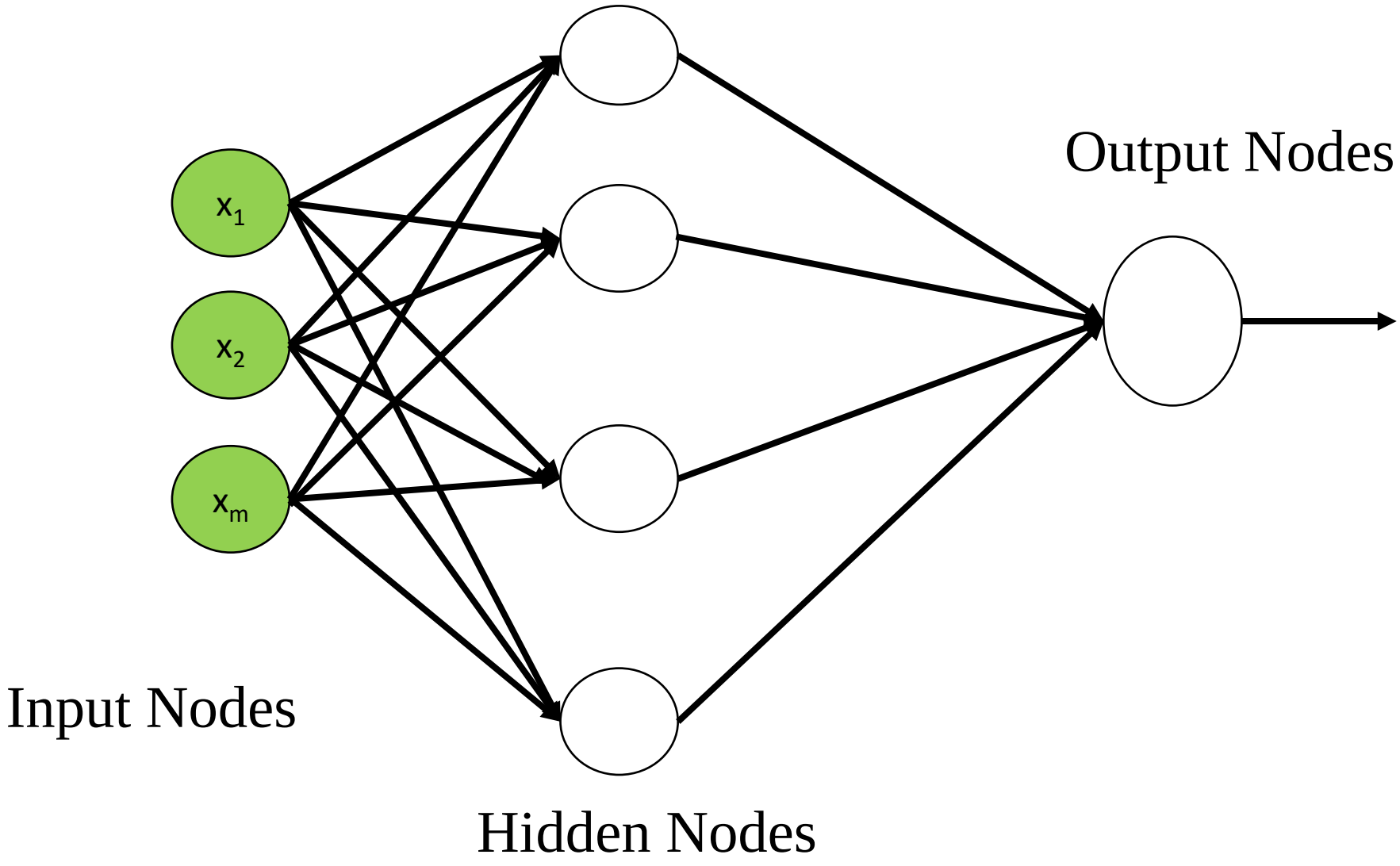
Hidden Layers

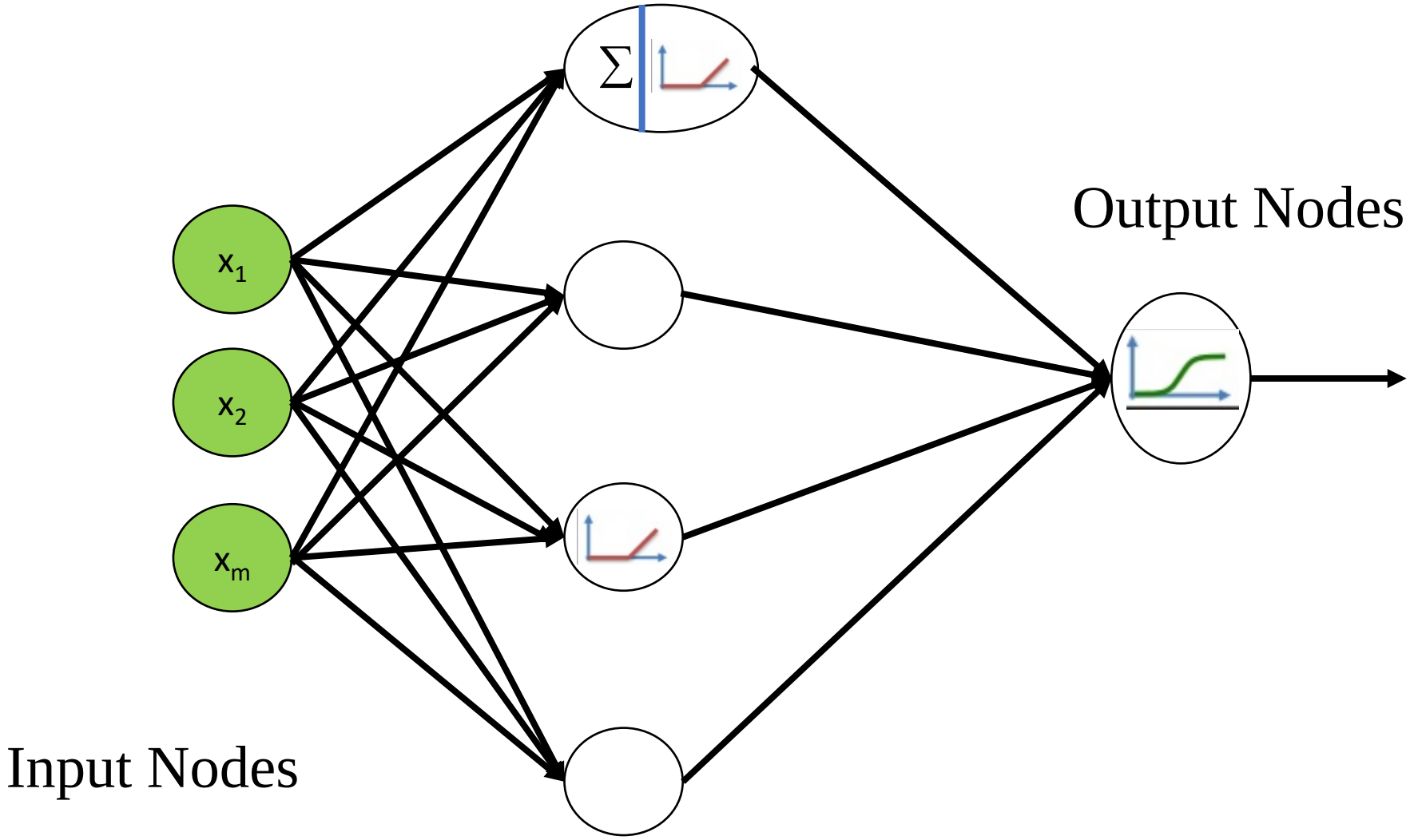
Inputs



Outputs

- Outputs can be multiple (multiple targets). See softmax activation later.

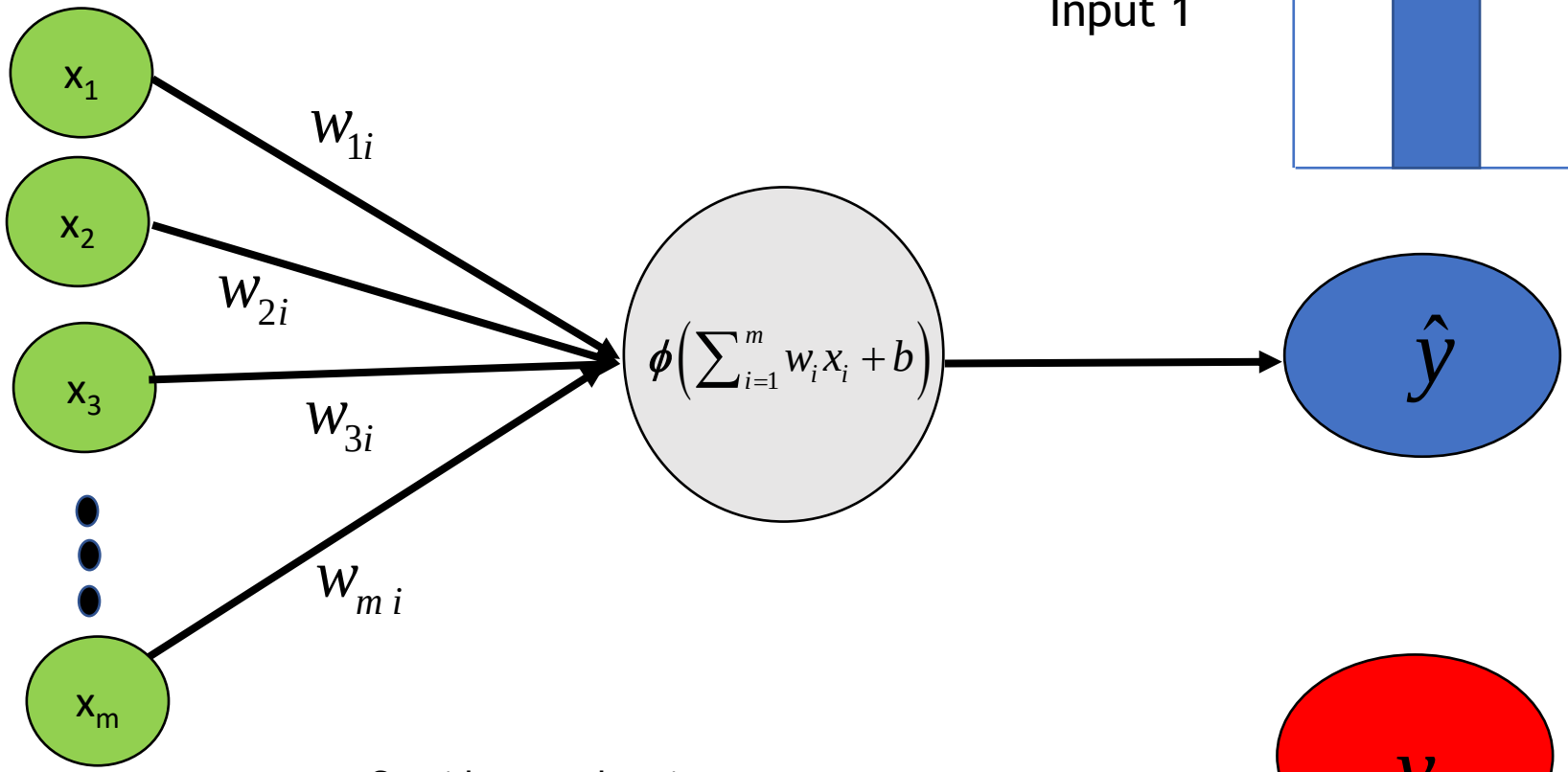

Output Nodes
x_1
x_2
x_m
Input Nodes
Hidden Nodes


Σ
Output Nodes
x_1
x_2
x_m
Input Nodes
Hidden Nodes

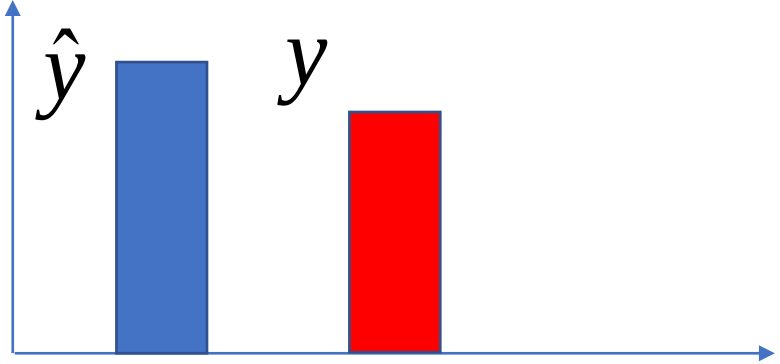
Learning in NNs

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

How do NNs learn?



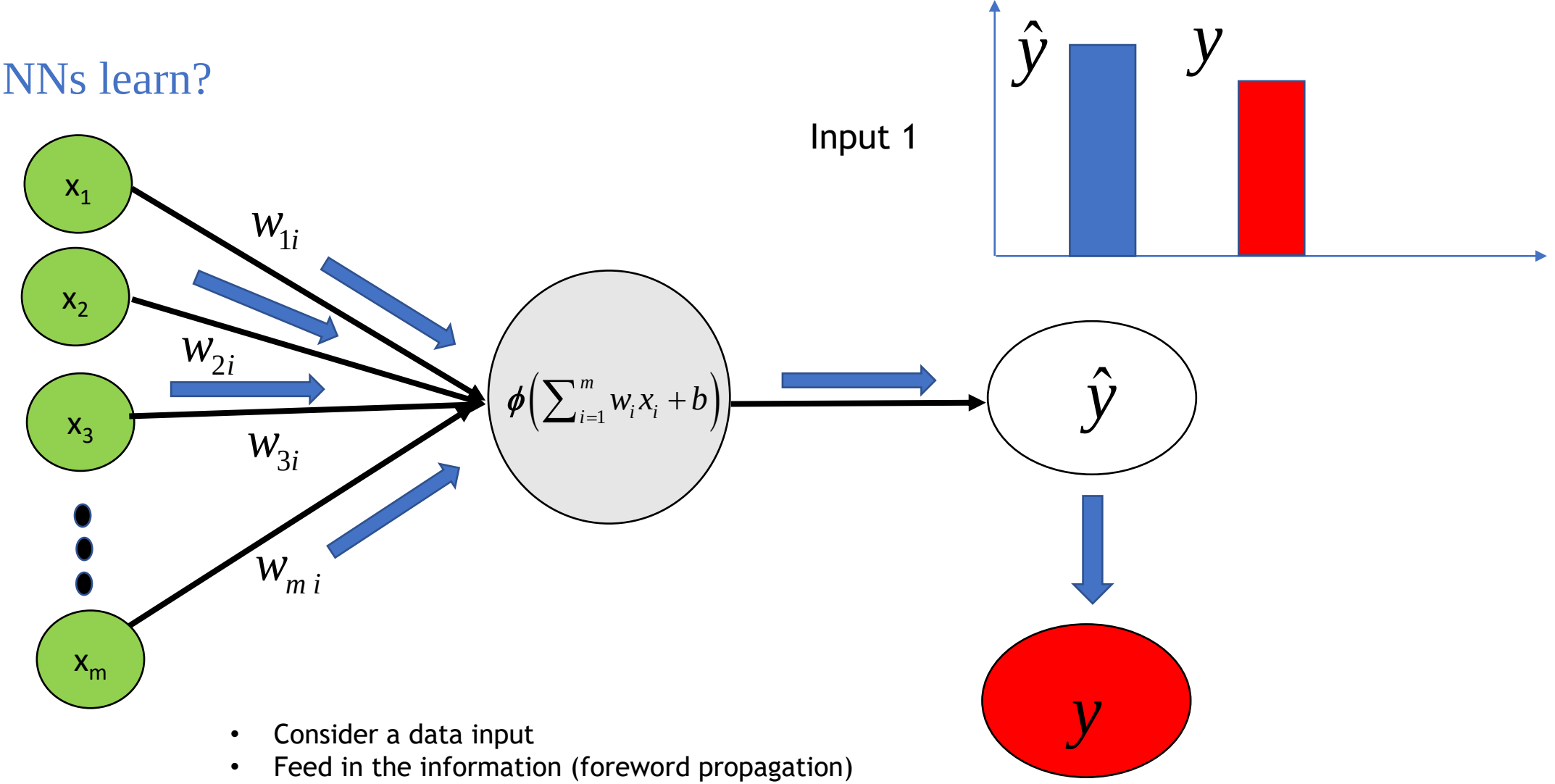
Input 1



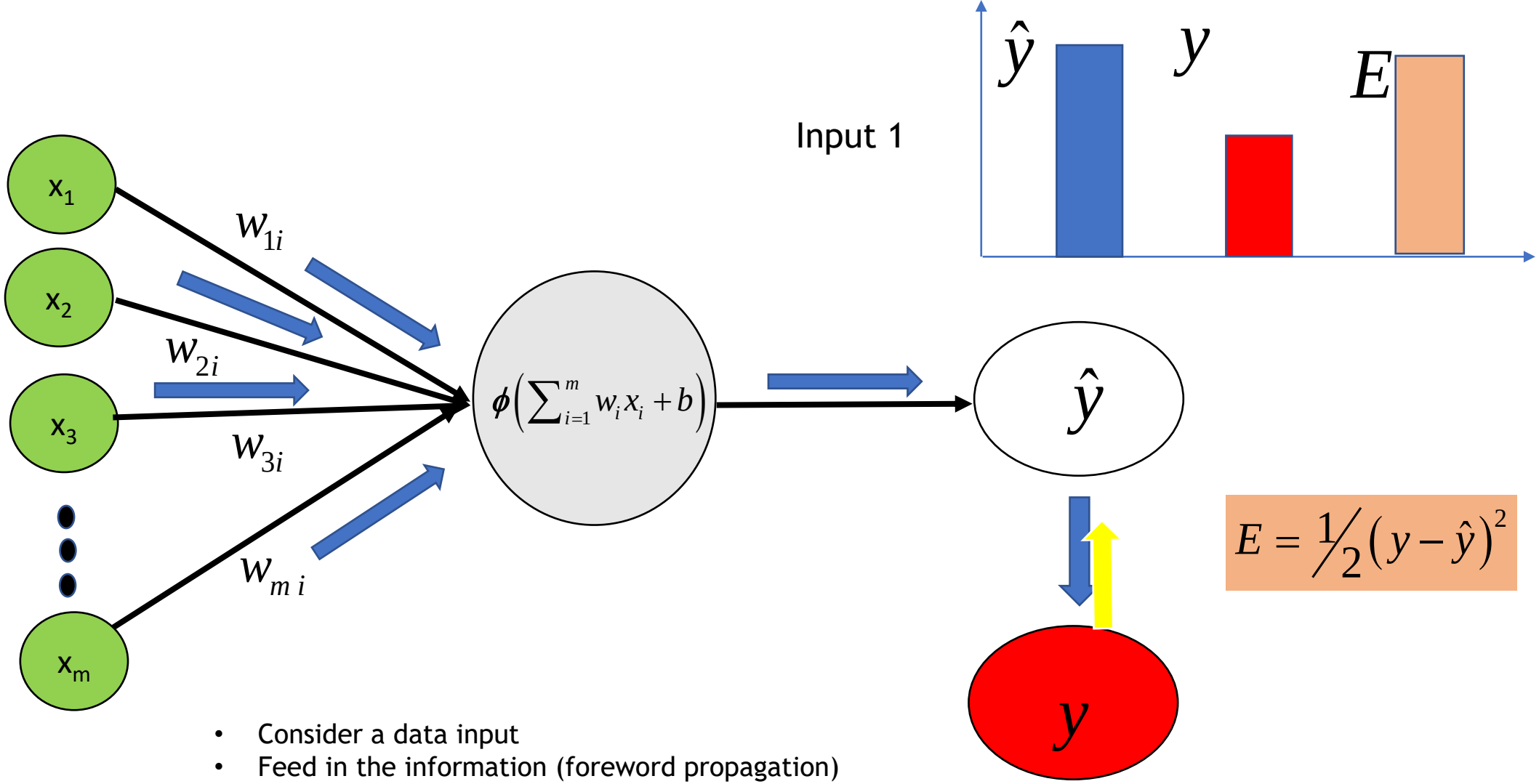
y

- Consider one data input

How do NNs learn?

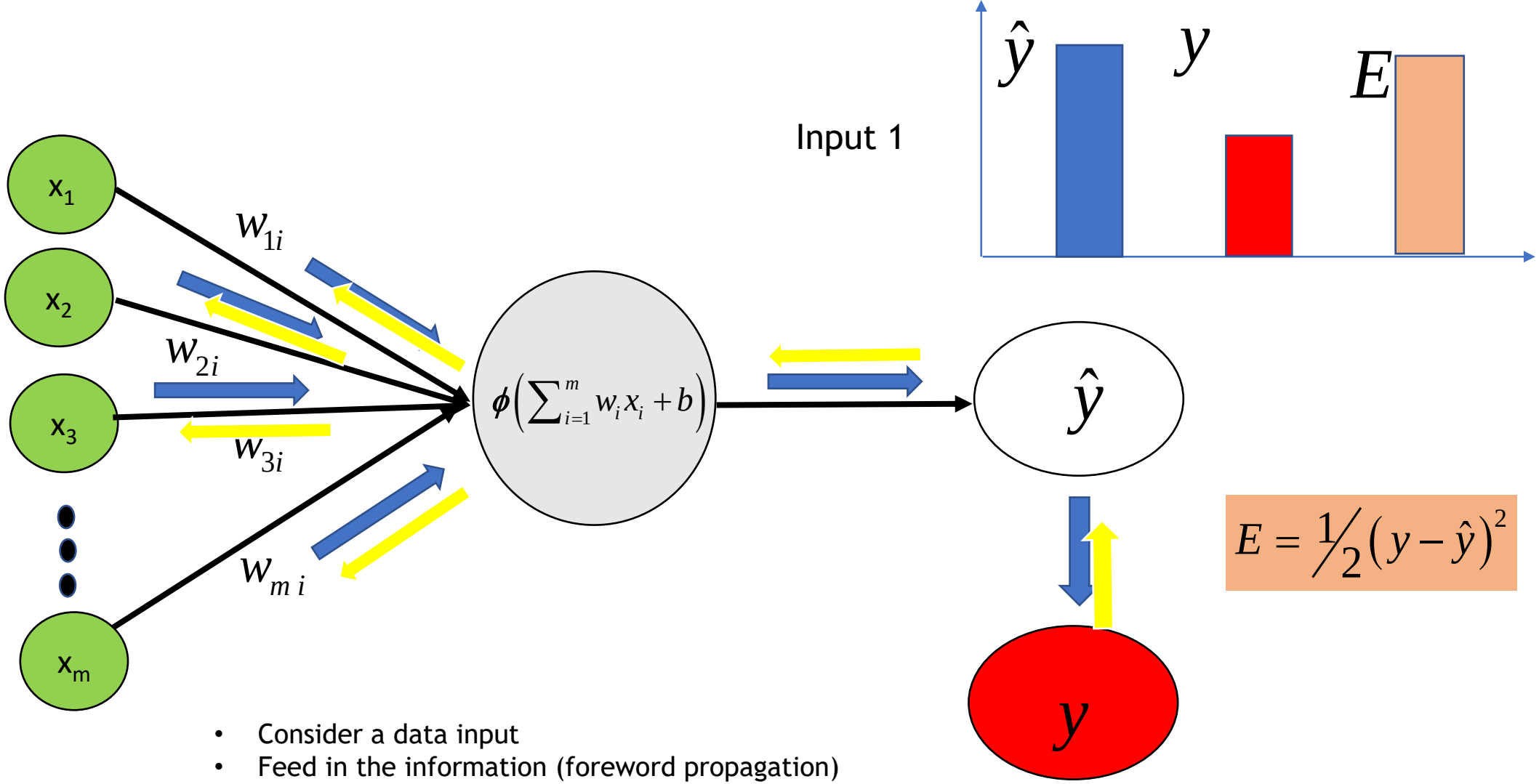


- Consider a data input
- Feed in the information (foreword propagation)



- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the individual loss wrt actual value.

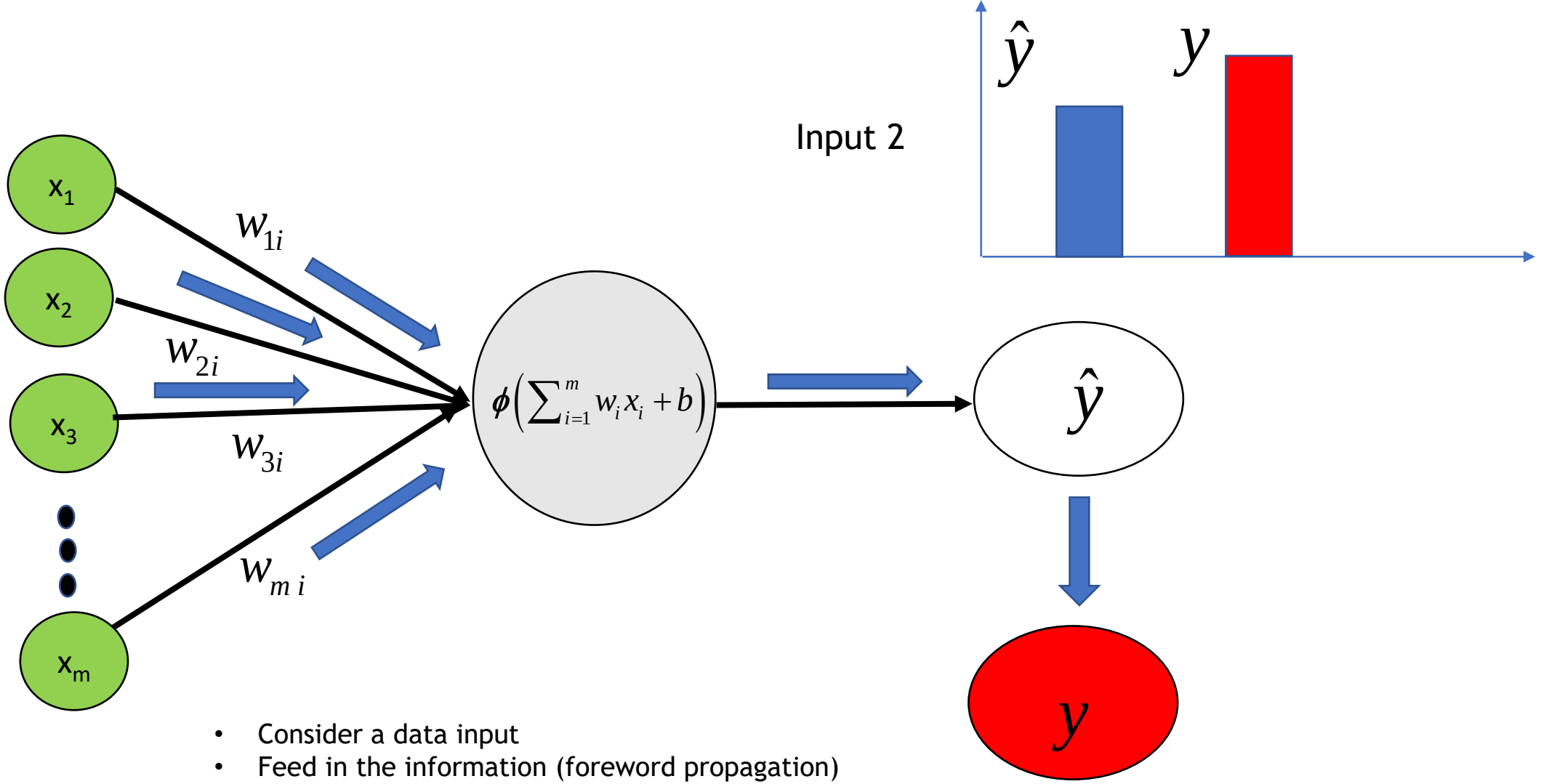
Note: Objective to minimise the cost function. Find optimal weights.



- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

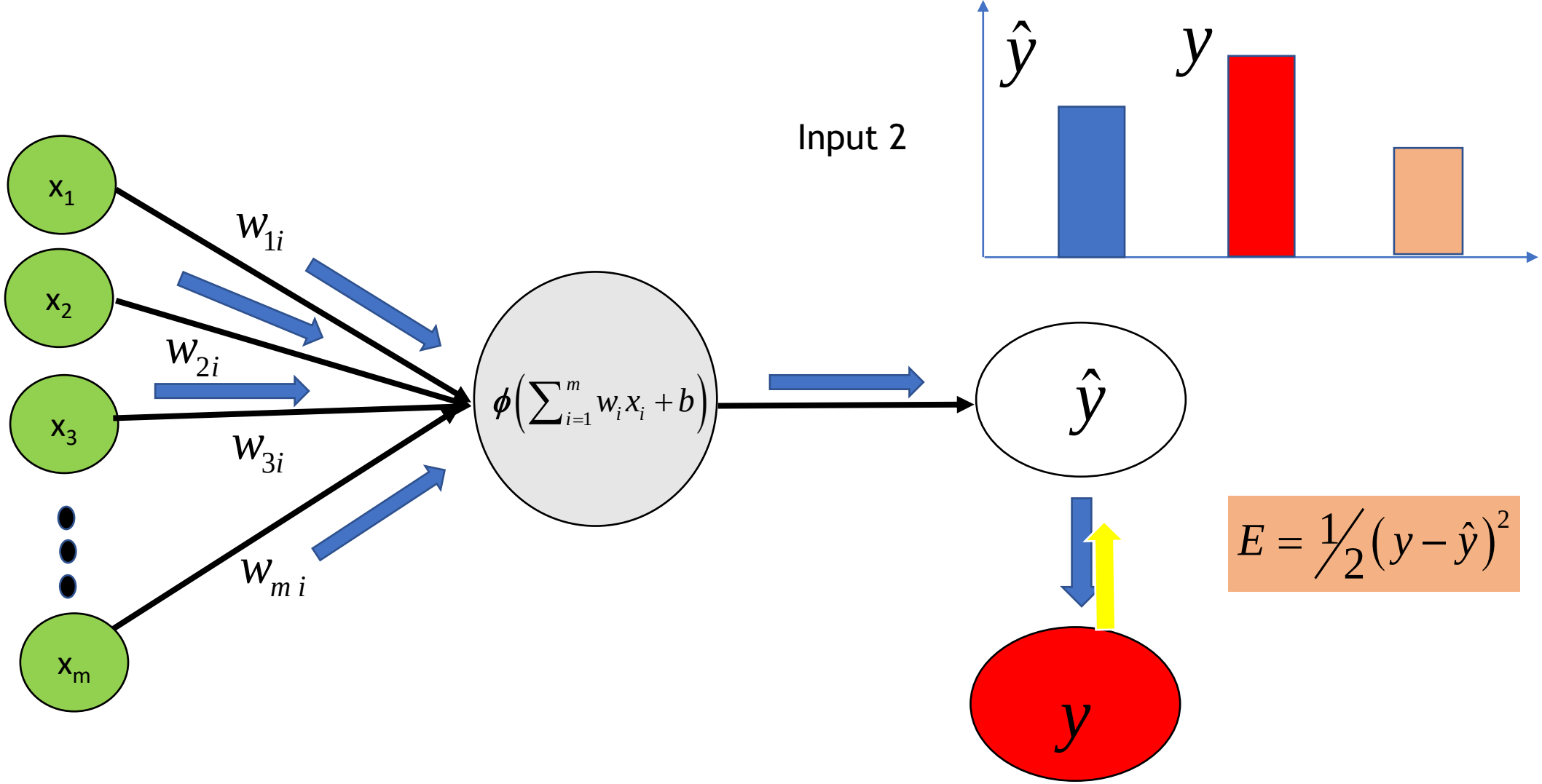
- Information can be fed back, to adjust the weights.

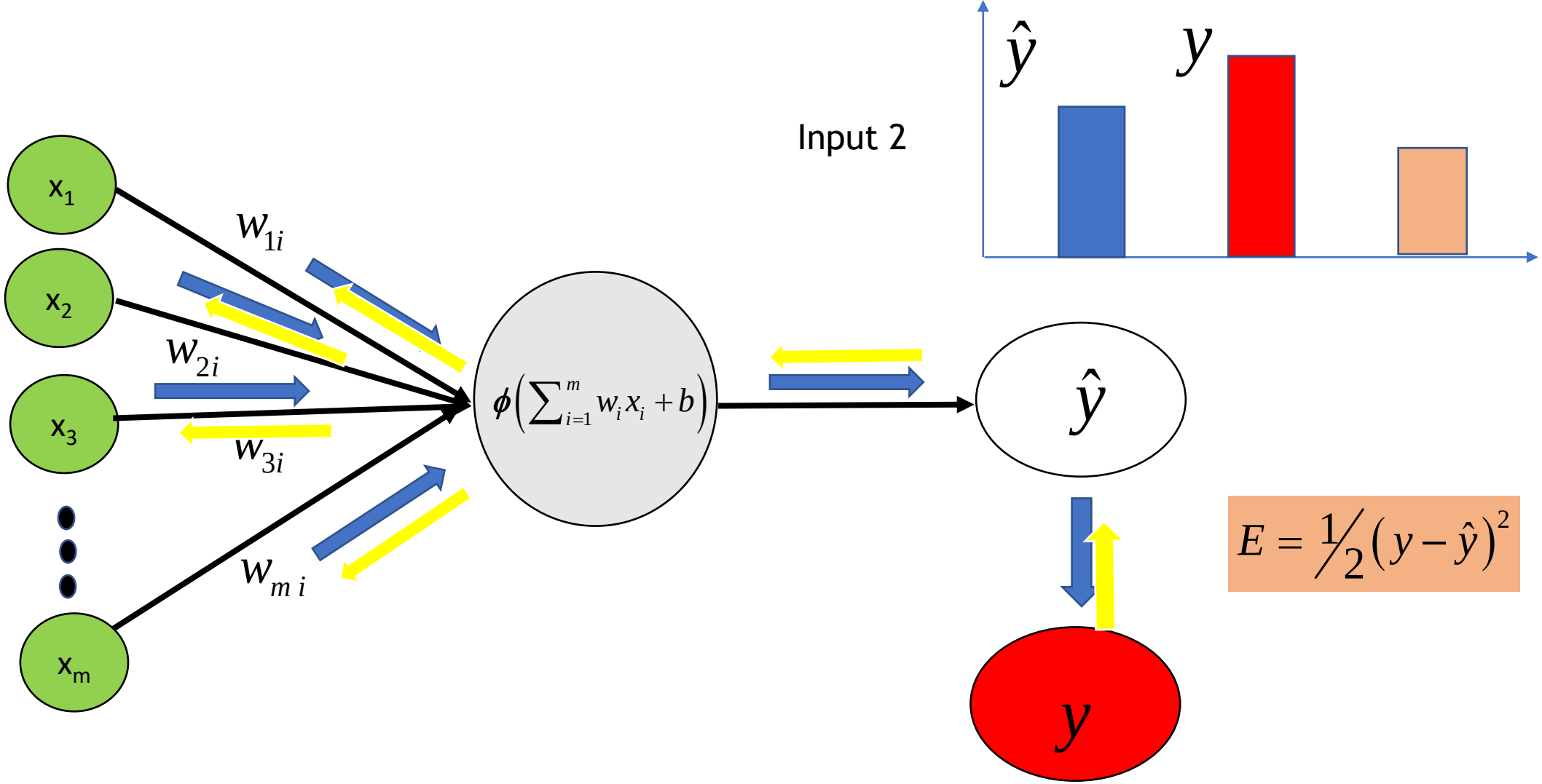


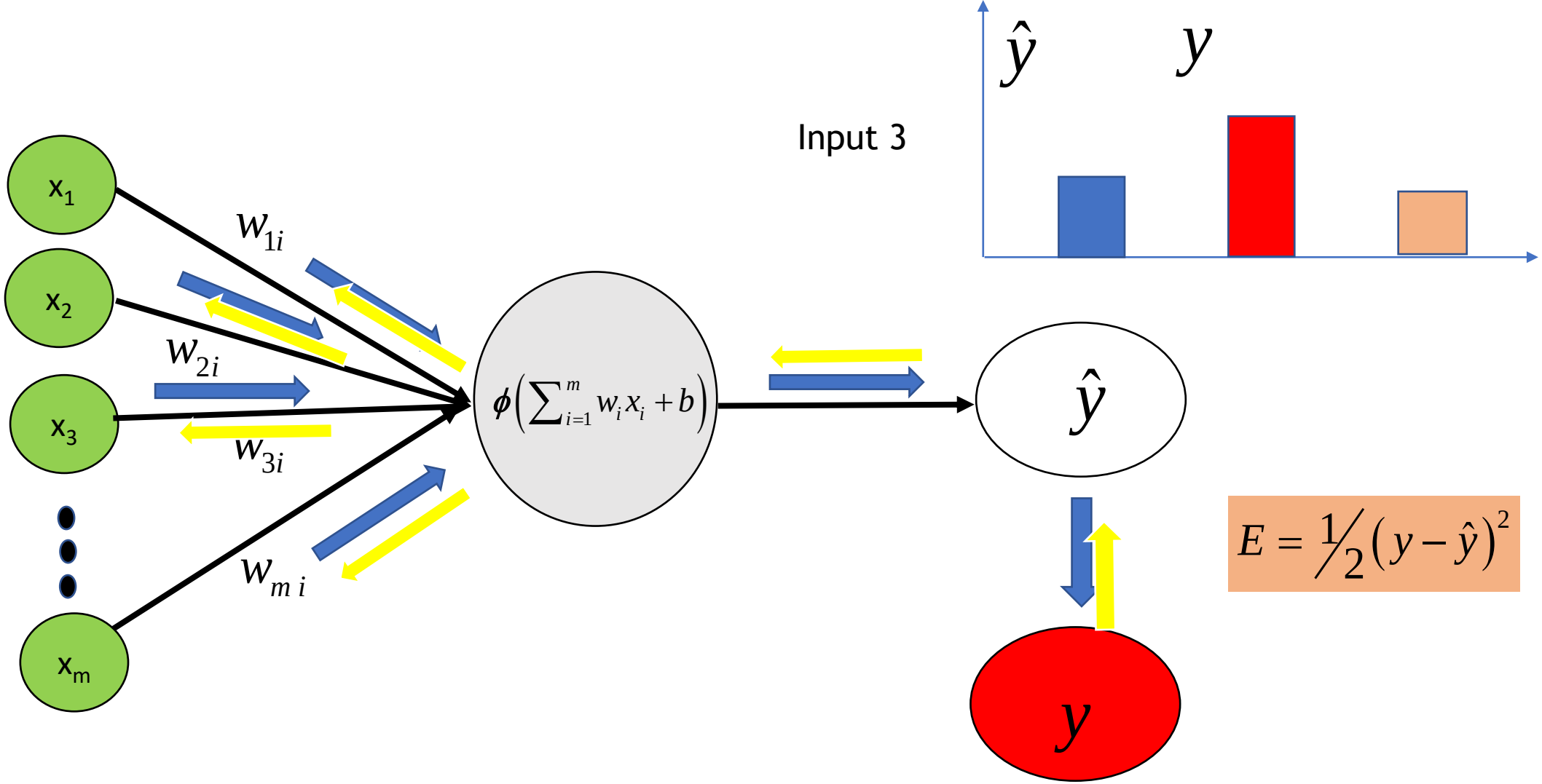
- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

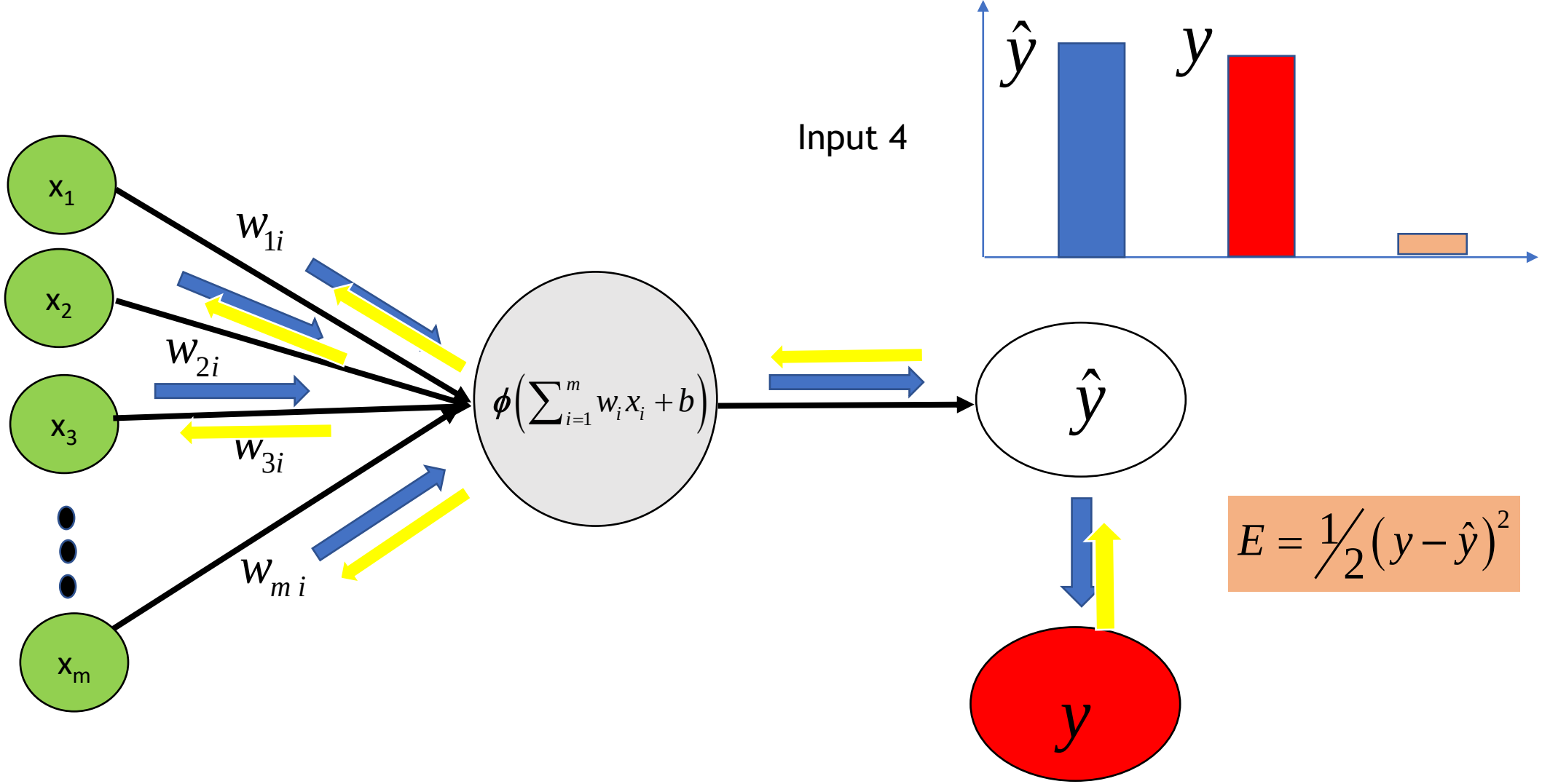
Note: Objective to minimise the cost function. Find optimal weights.

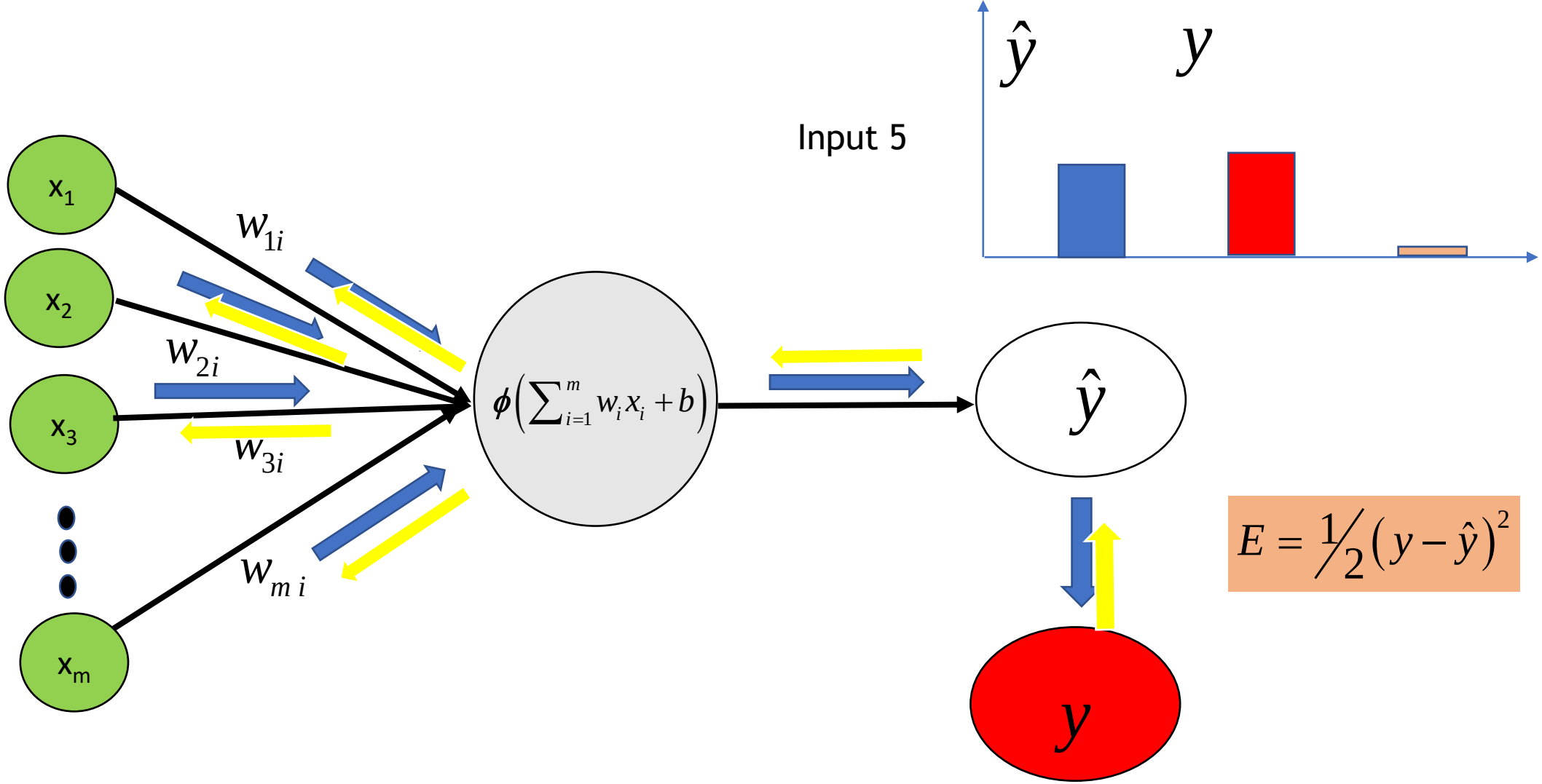
- Information can be fed back, to adjust the weights.
- Repeated with other data inputs.

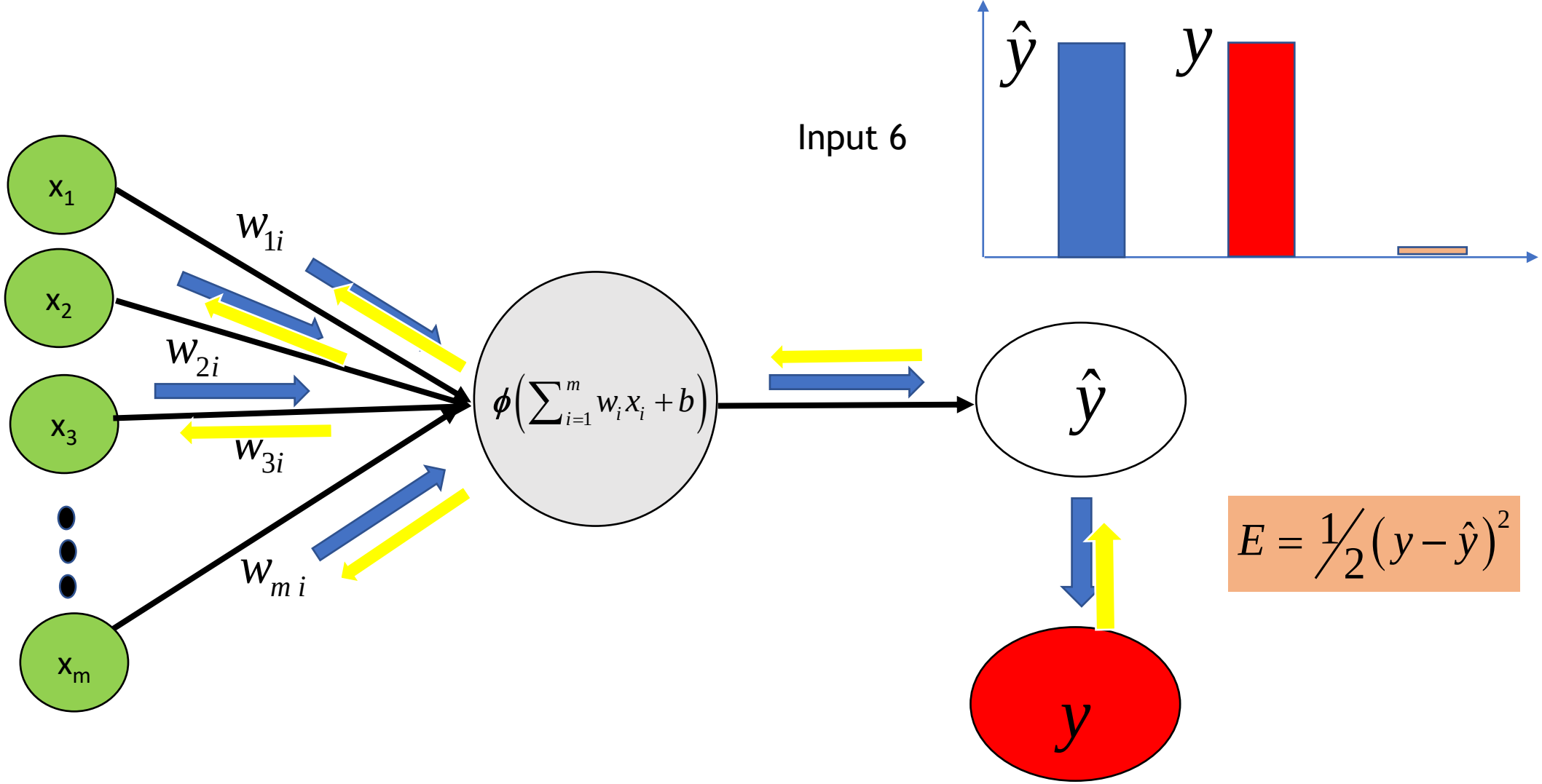

Input 2
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$

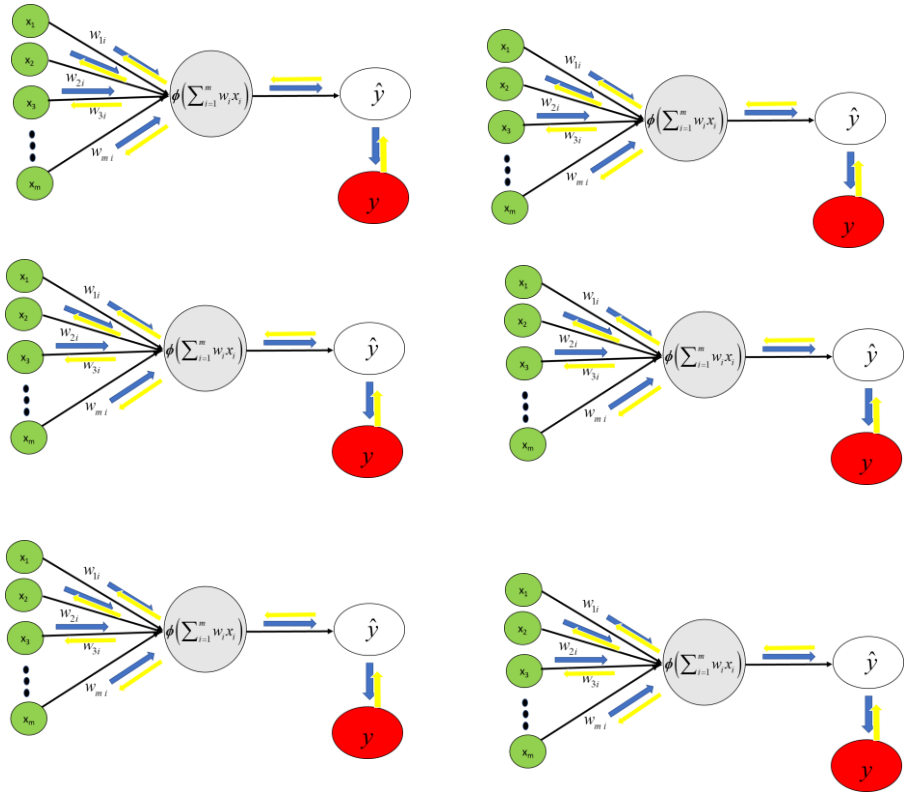

Input 2
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$


Input 3
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$


Input 4
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$


Input 5
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$


Input 6
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$

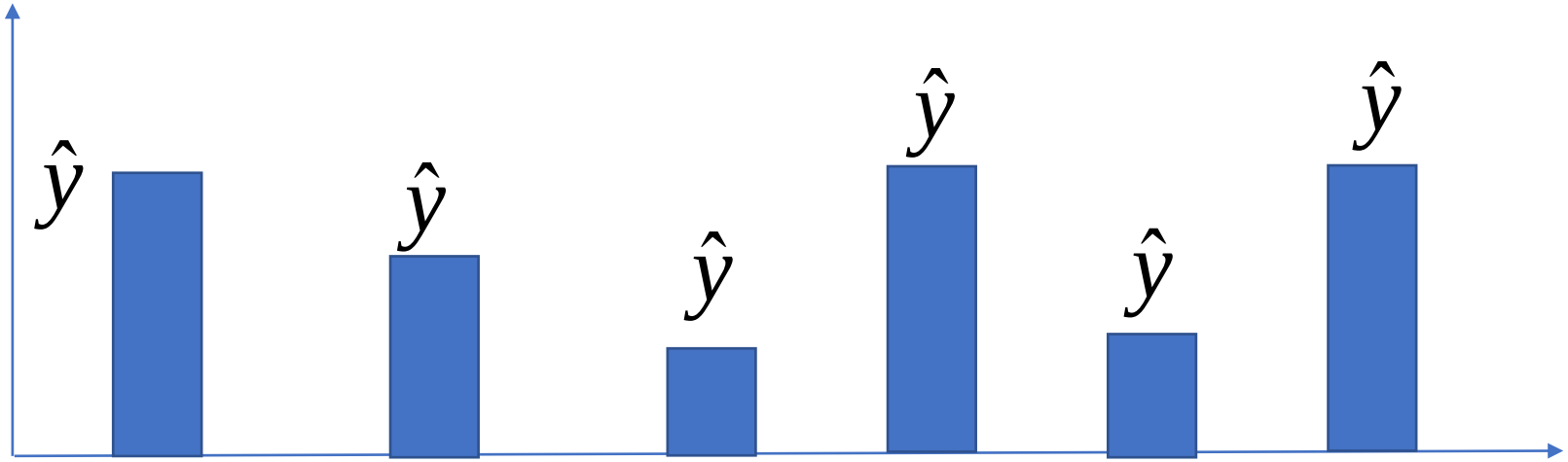


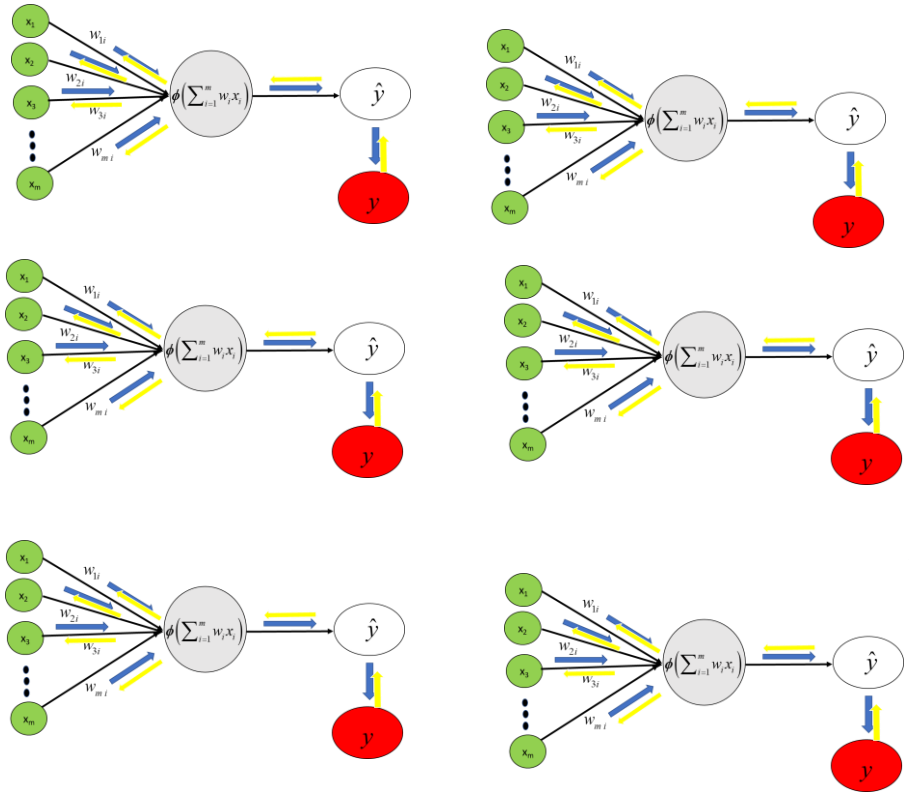
Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information can be fed back, to adjust the weights.
- Repeated with other data inputs.



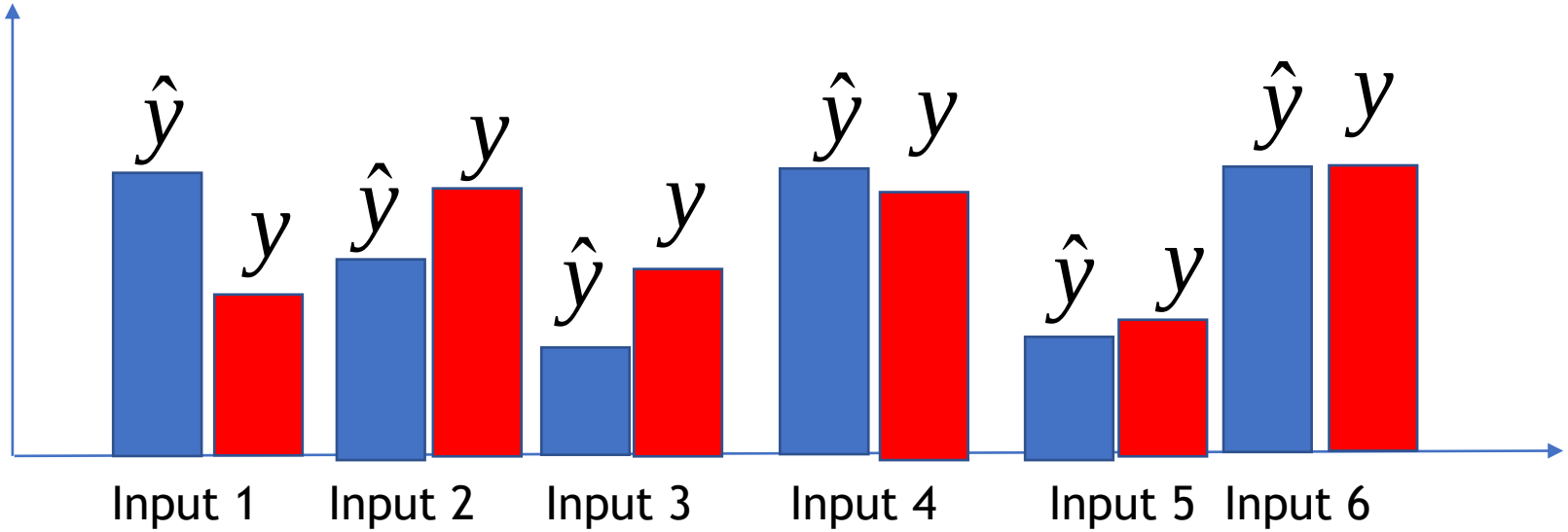


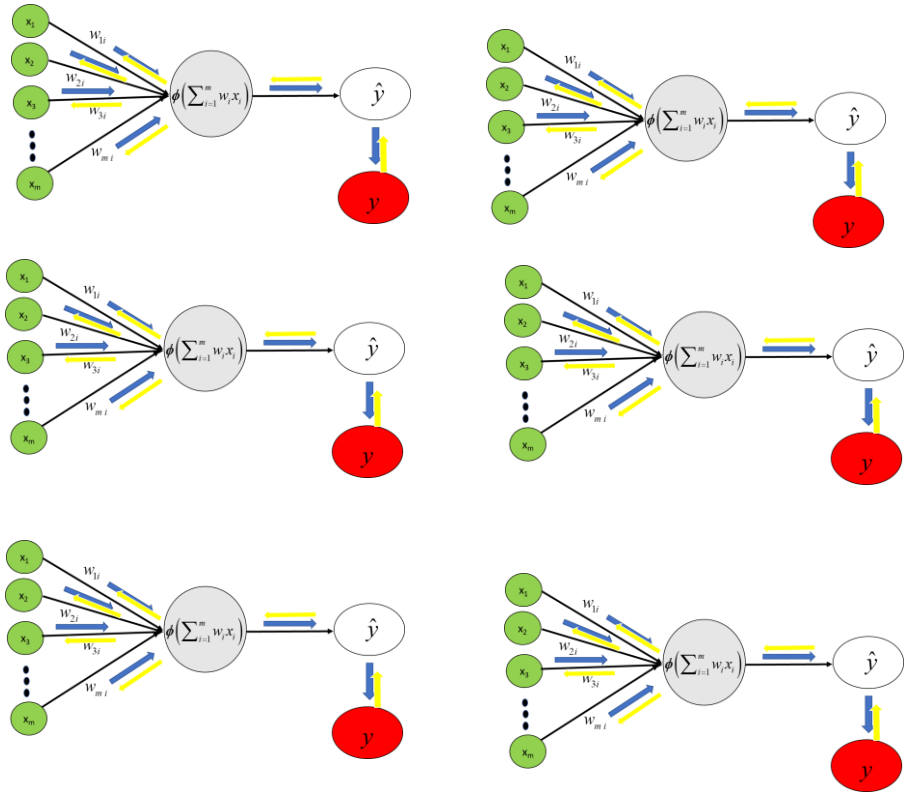
Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information can be fed back, to adjust the weights.





Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

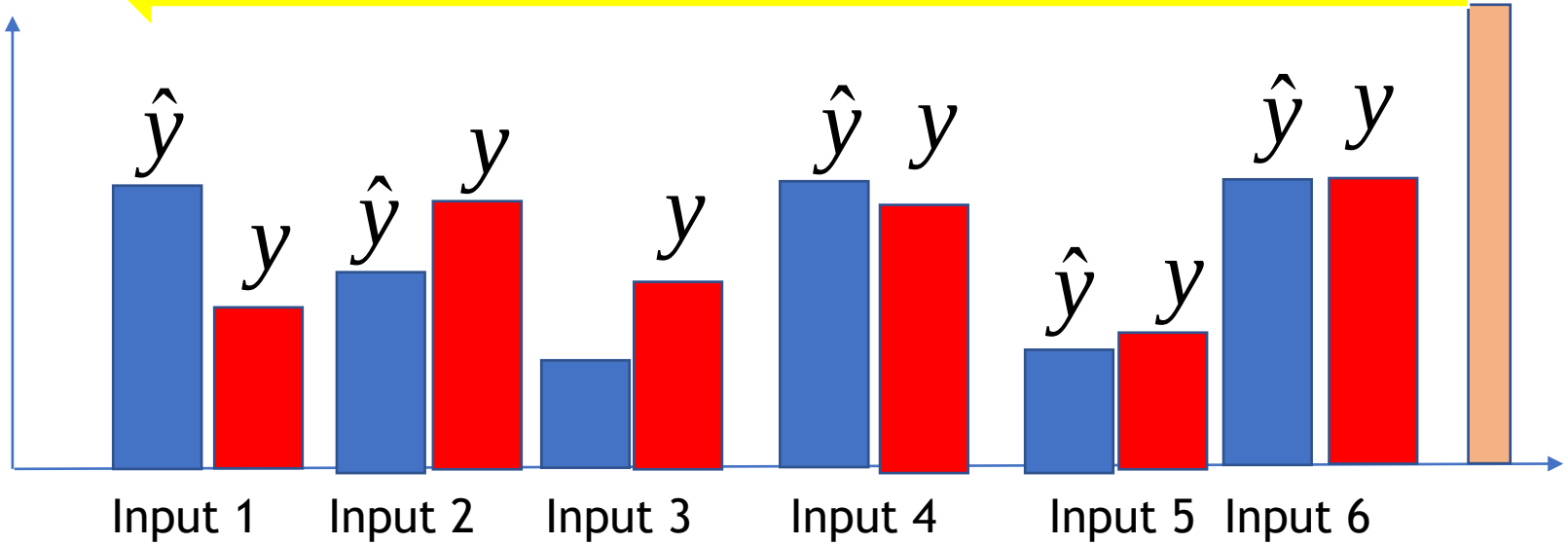
Note: Objective to minimise the cost function. Find optimal weights.

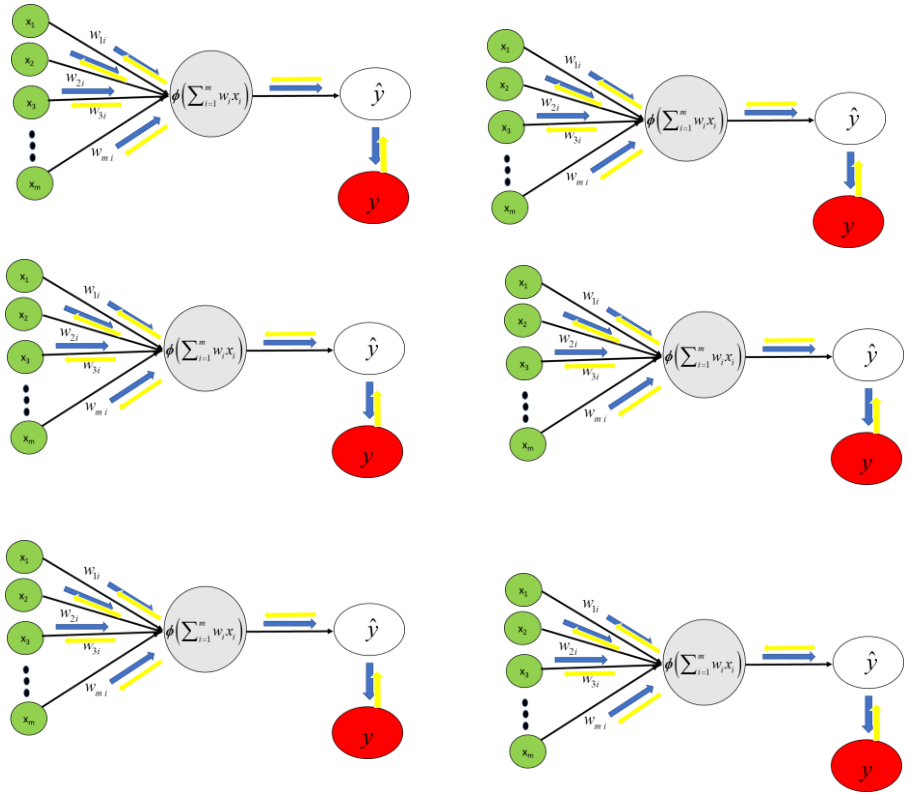
- Information can be fed back, to adjust the weights.

Rationale: The global error is backward propagated to network nodes, weights are modified proportional to their contribution.

$$E_{tot} = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Update all the weights





Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information **will be** fed back, to adjust the weights.
- Repeated with other data inputs.

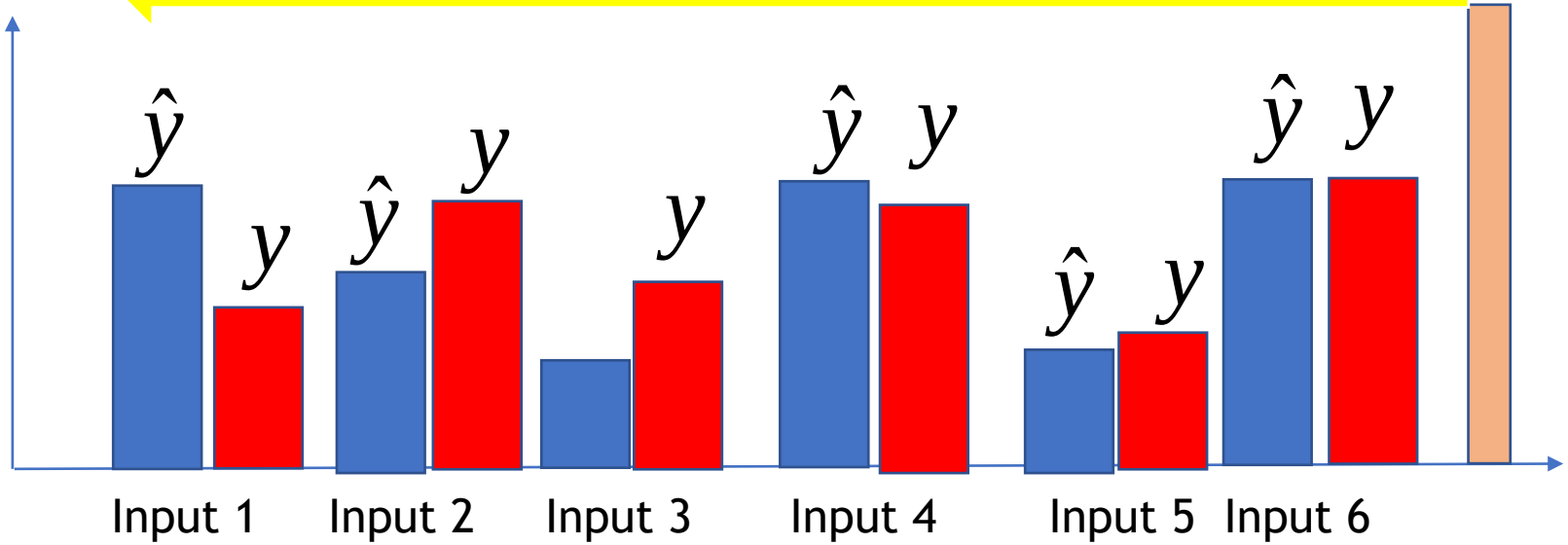
.

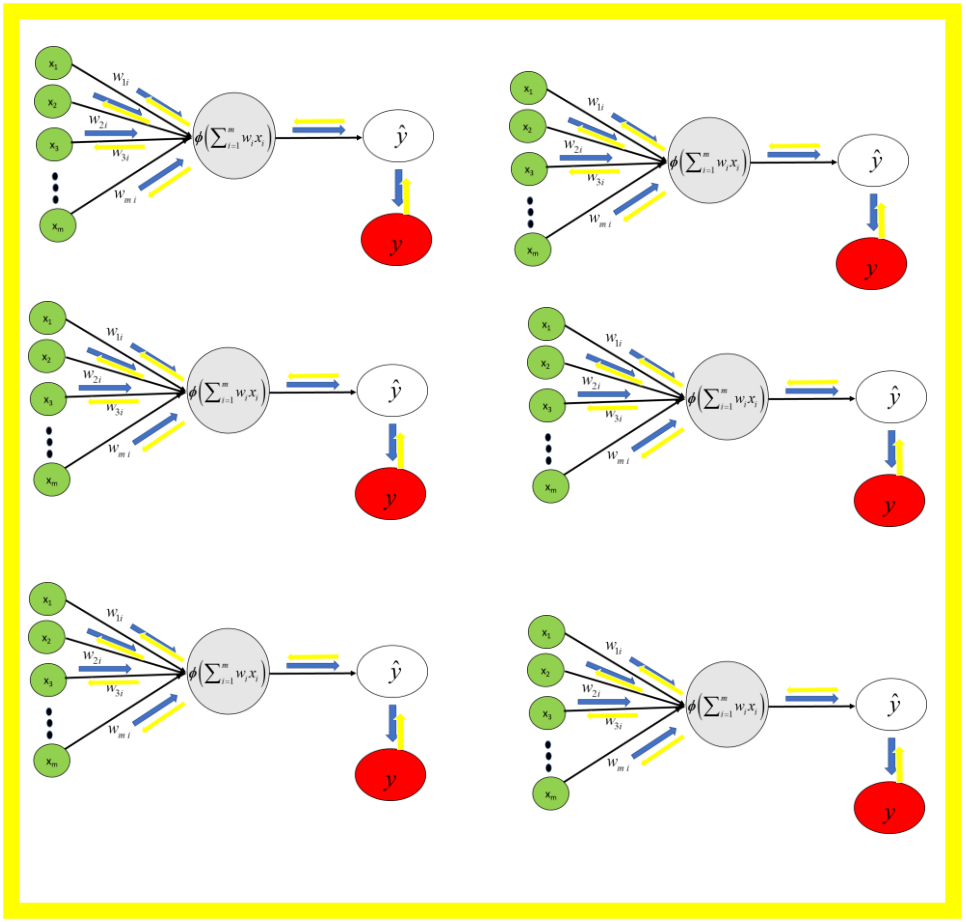
.

- Total loss → cost function
- The weights adjusted 'at the same time' using total loss.

$$E_{tot} = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Update all the weights





One epoch = training done on entire data set once.

Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information **will be** fed back, to adjust the weights.
- Repeated with other data inputs.

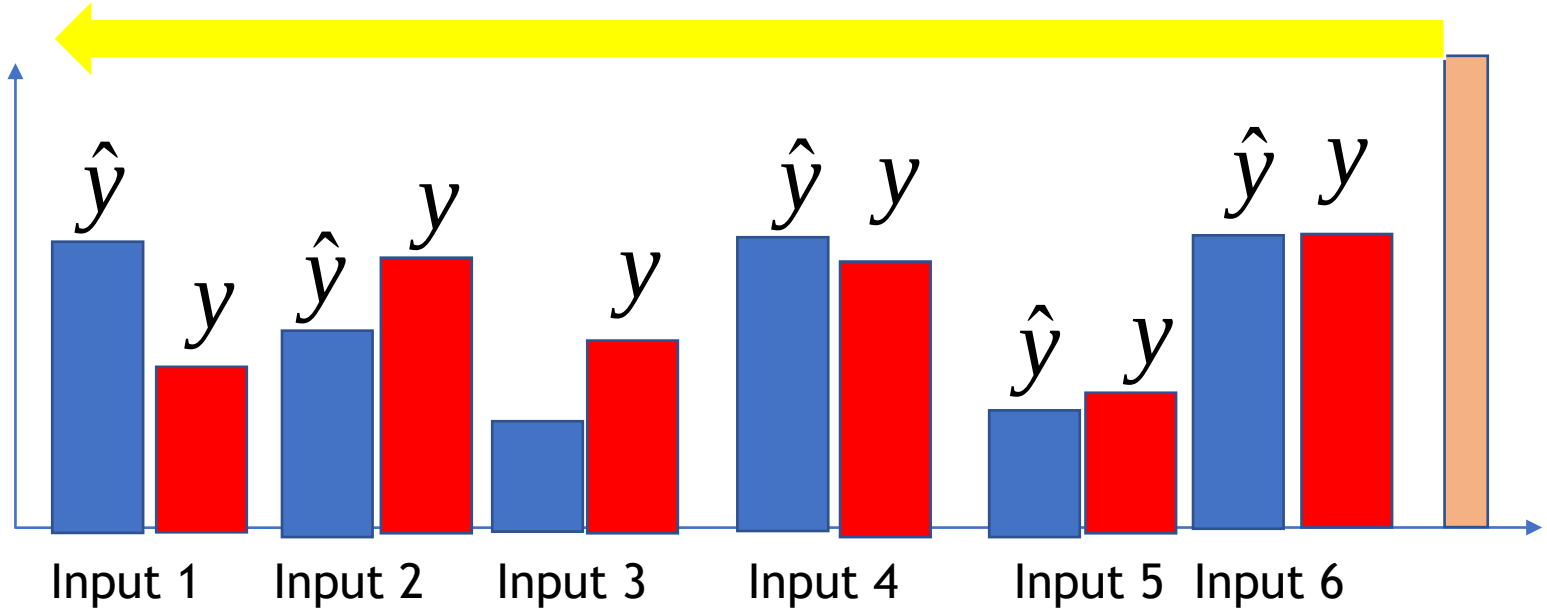
.

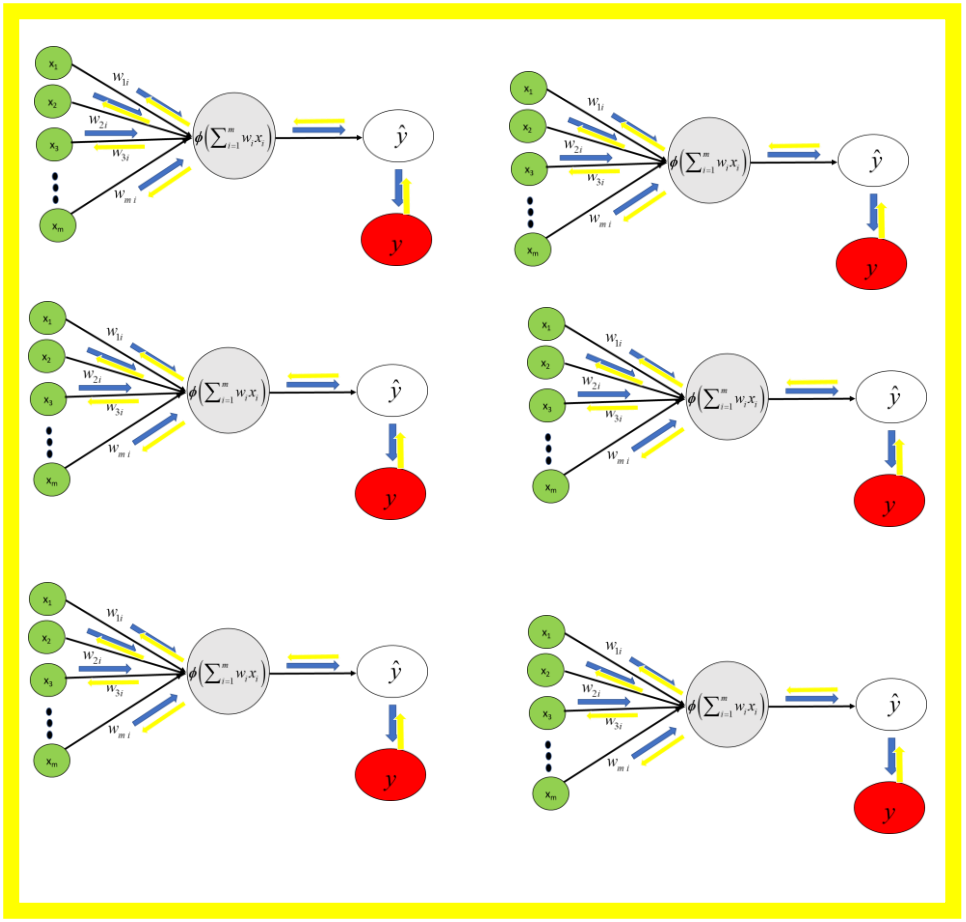
.

- Total loss → cost function
- The weights adjusted 'at the same time' using total loss.

$$E_{tot} = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Update all the weights

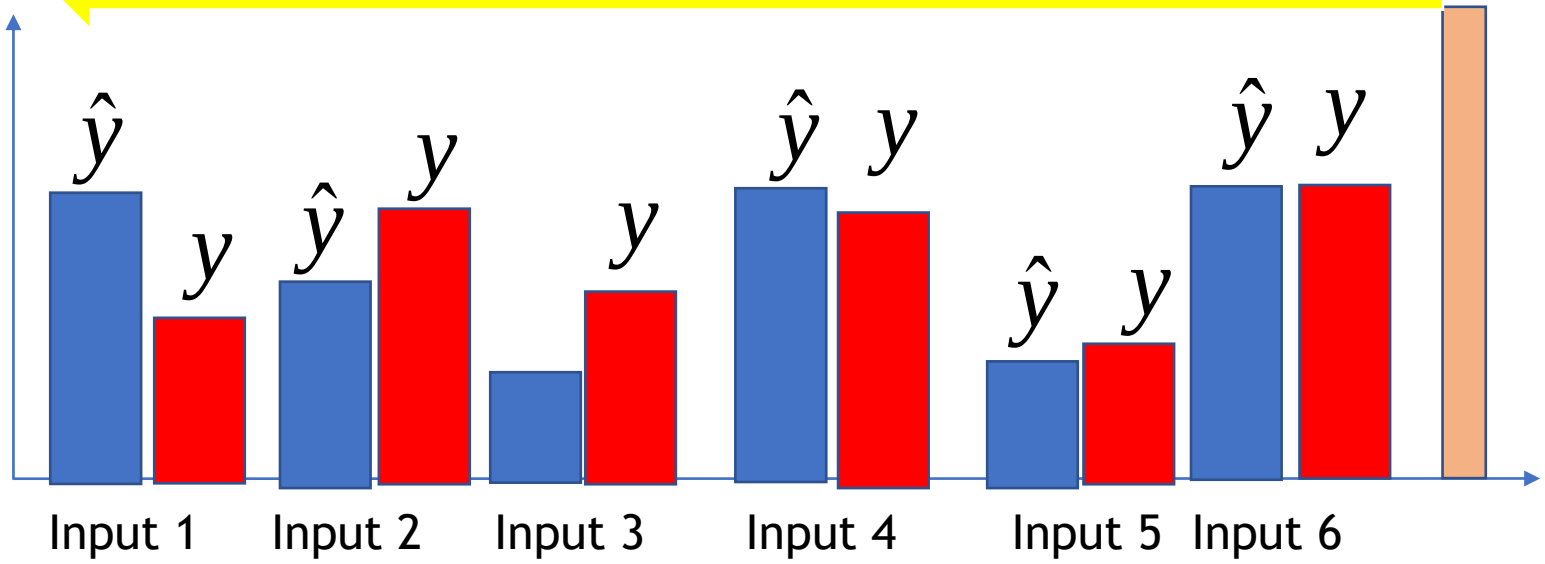




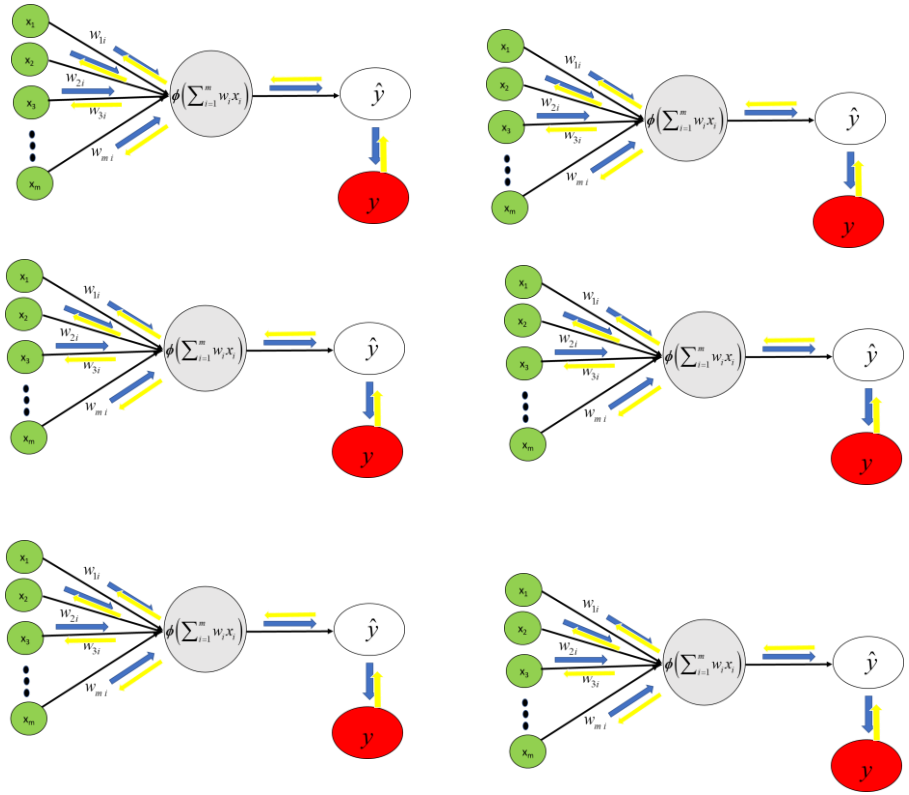
Objective : To minimize this loss,

$$E_{tot} = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



One epoch = training done on entire data set once.

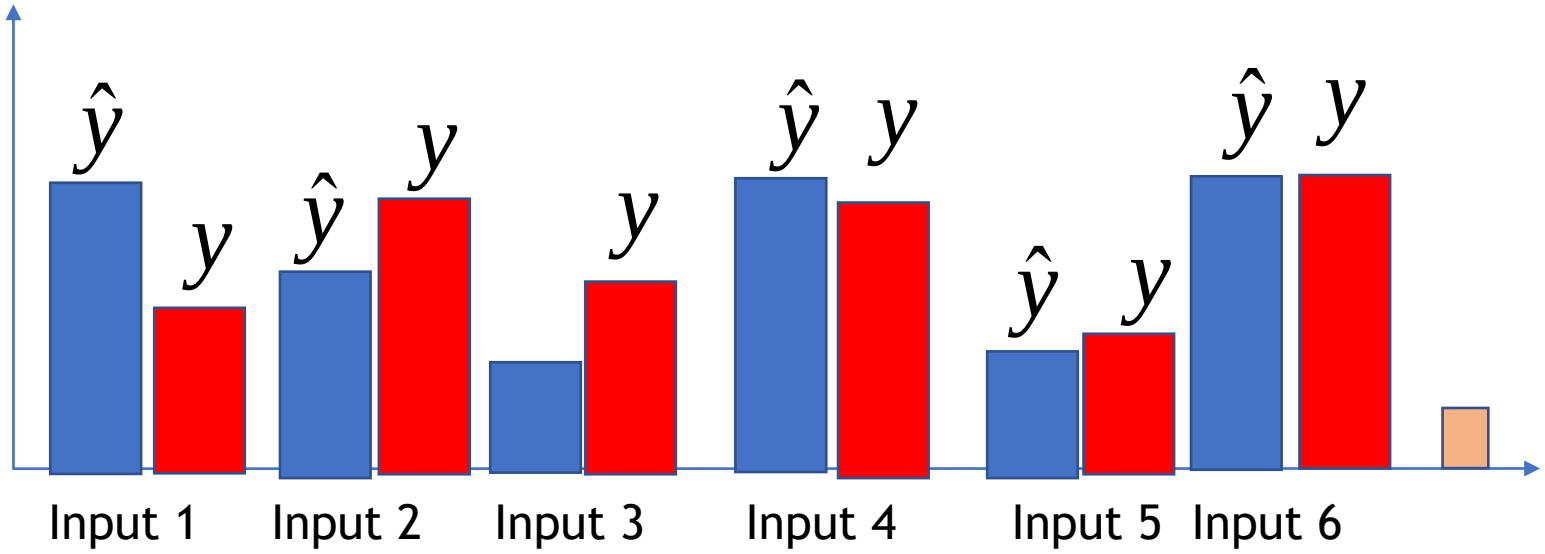


Objective : To minimize this loss, find optimal sets of weights.

How to minimise the loss and update the weights??

$$E_{tot} = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



One epoch = training done on entire data set once.

Gradient Descent

Gradient Descent

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function **(cost function)** or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

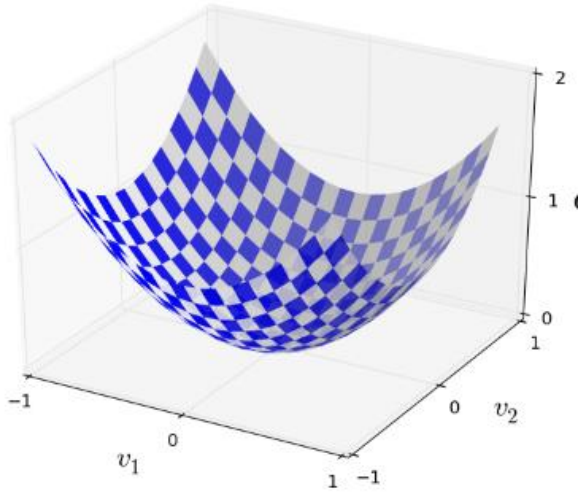
$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

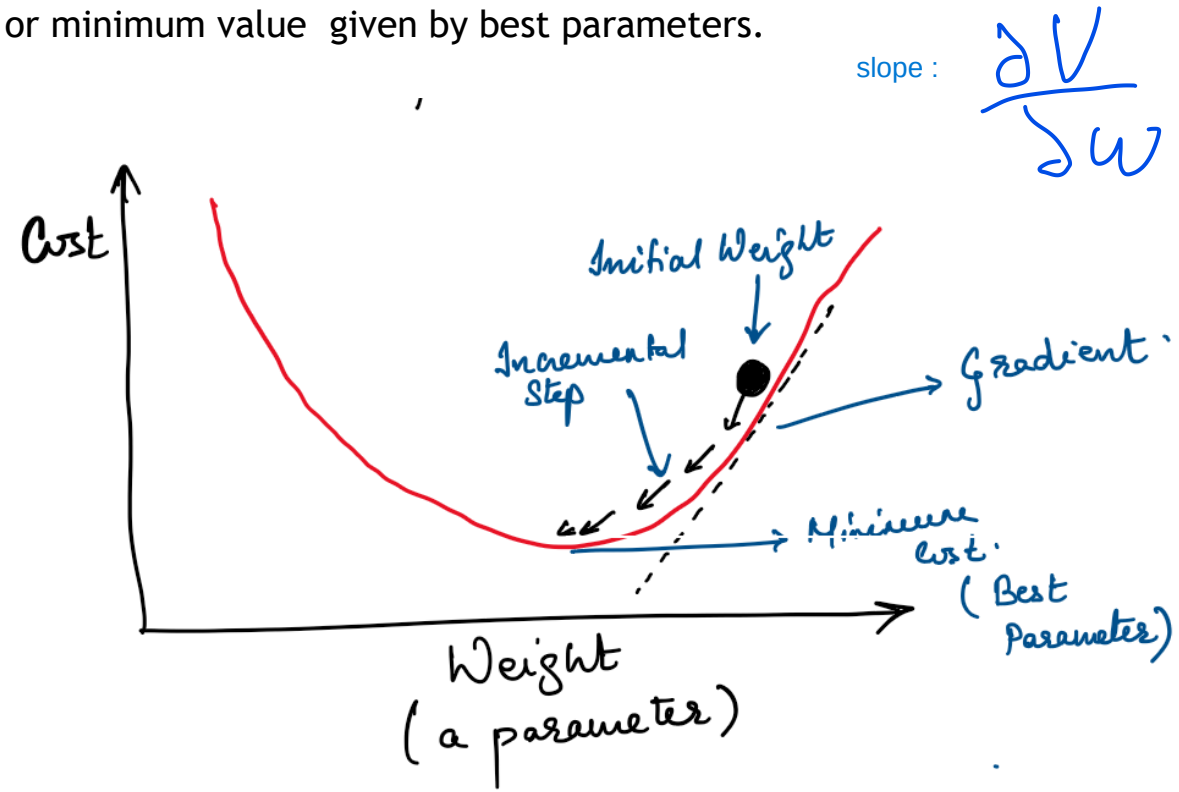
Gradient Descent

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

- Intuition behind GD: Climbing down the hill to find its bottom or minimum value given by best parameters.





Gradient Descent

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

- Intuition behind GD: Climbing down the hill to find its bottom or minimum value given by best parameters.

Basic steps:

Given the loss function $J(w,b)$

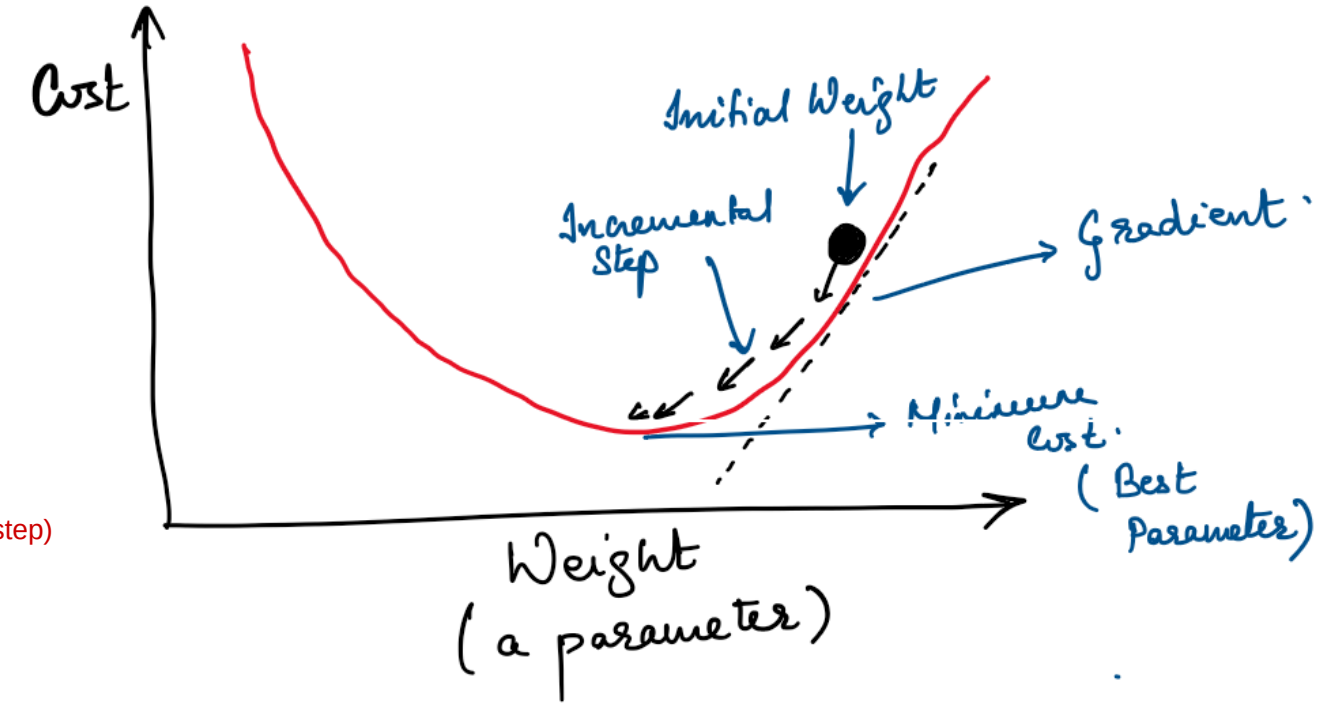
- Compute the slope (gradient) that is the first-order derivative of the function at the current point.
- Move-in the opposite direction of the slope increase from the current point by the computed amount.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial (J(\mathbf{w},b))}{\partial \mathbf{w}}$$

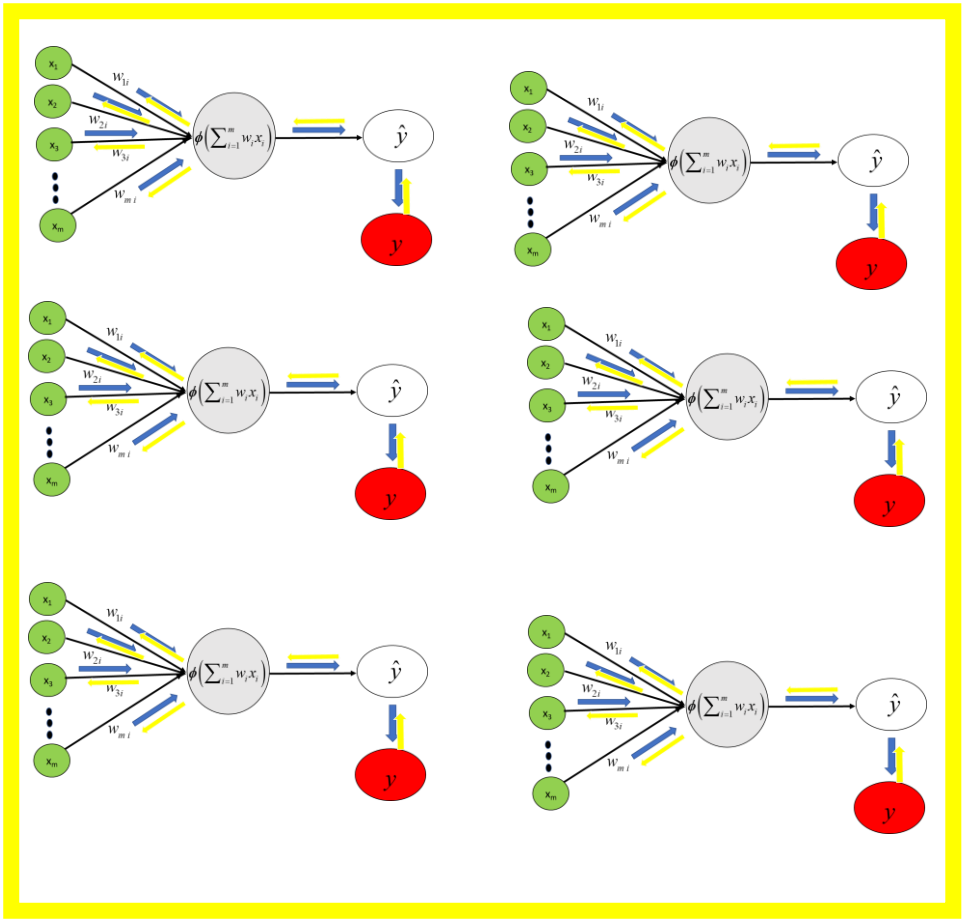
take a step in then negative direction

alpha is the learning weigh (how big is the step)

$$b \leftarrow b - \alpha \frac{\partial (J(w,b))}{\partial b}$$



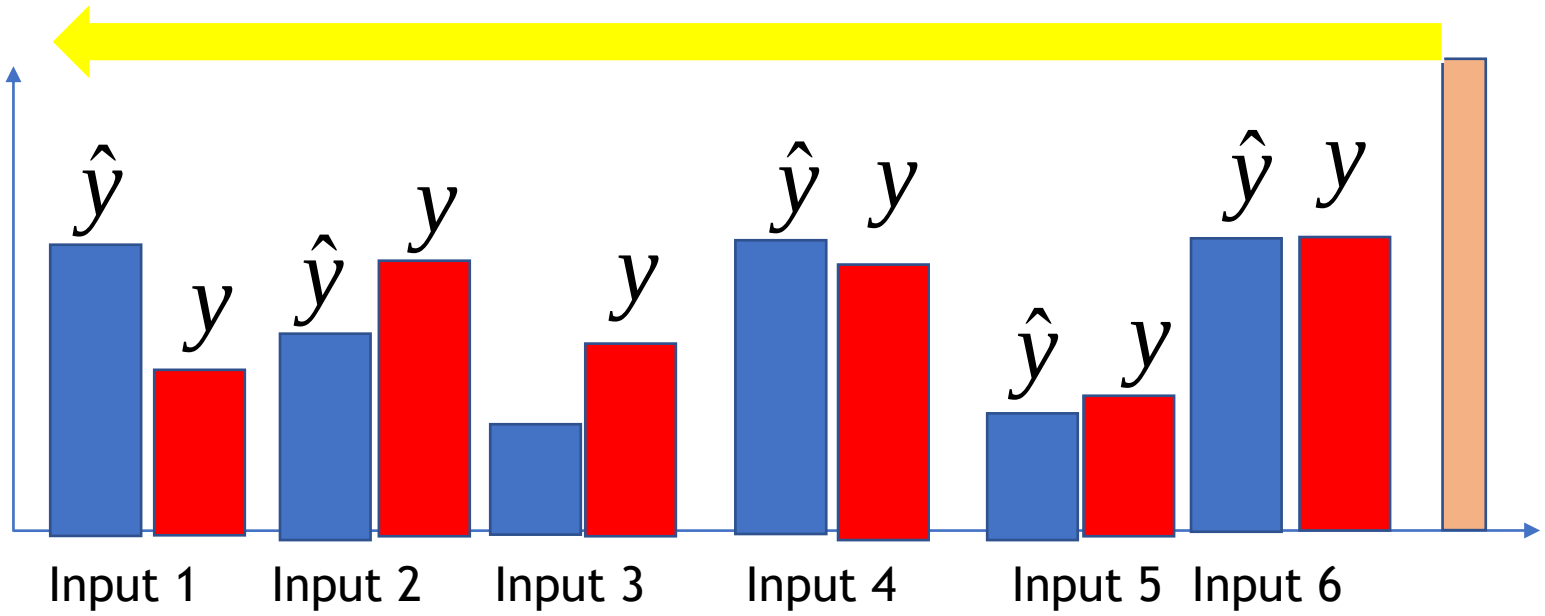
Batch Gradient Descent



Saw earlier: Weights were updated using total loss of a data batch ➔ Batch GD.

$$E_{tot} = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



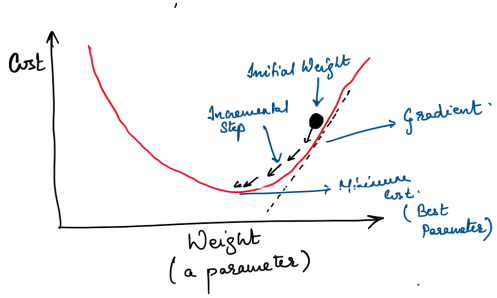
One epoch = training done on entire data set once.

Gradient Descent

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
s
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

- Intuition behind GD: Climbing down the hill to find its bottom or minimum value given by best parameters.



Basic steps:

Given the loss function $J(w,b)$

- Compute the slope (gradient) that is the first-order derivative of the function at the current point.
- Move-in the opposite direction of the slope increase from the current point by the computed amount.

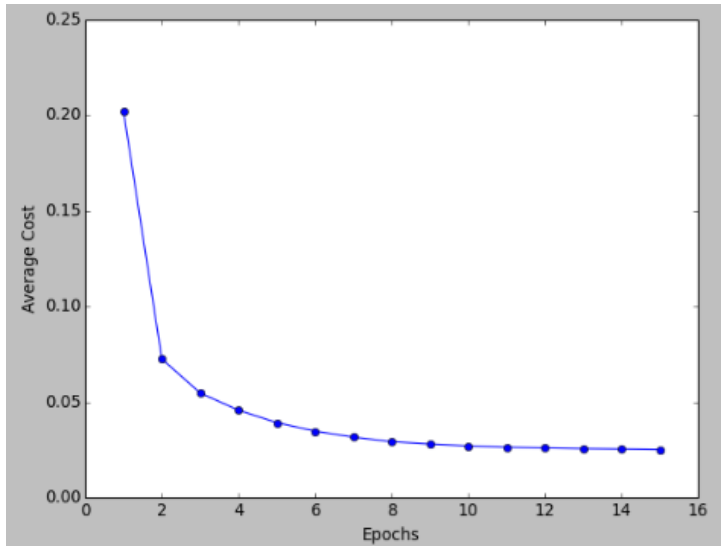
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial (J(\mathbf{w},b))}{\partial \mathbf{w}}$$

$$b \leftarrow b - \alpha \frac{\partial (J(w,b))}{\partial b}$$

α learning rate.

What happens when learning rate is very low?

What happens when learning rate is very high?

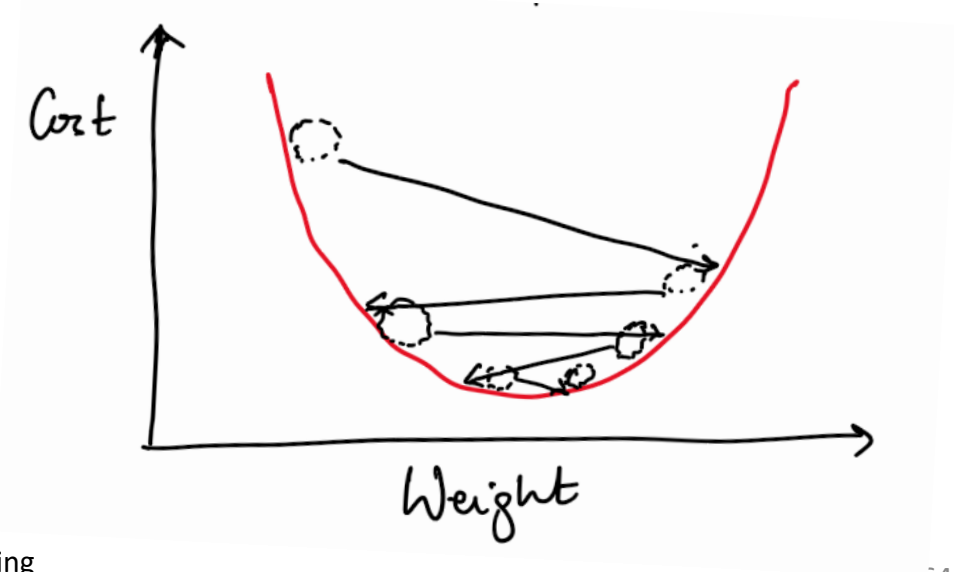


Gradient Descent

- When learning rate too low ➔ slow convergence.
- When learning rate too high ➔ minima will be overshot ➔ slow or no convergence.

- Learning rate is a ***Hyperparameter***.
- **It must be fine tuned. Neither too high, nor too low. We see hyperparameter tuning later.**

- GD works well when the total loss function is a convex function.



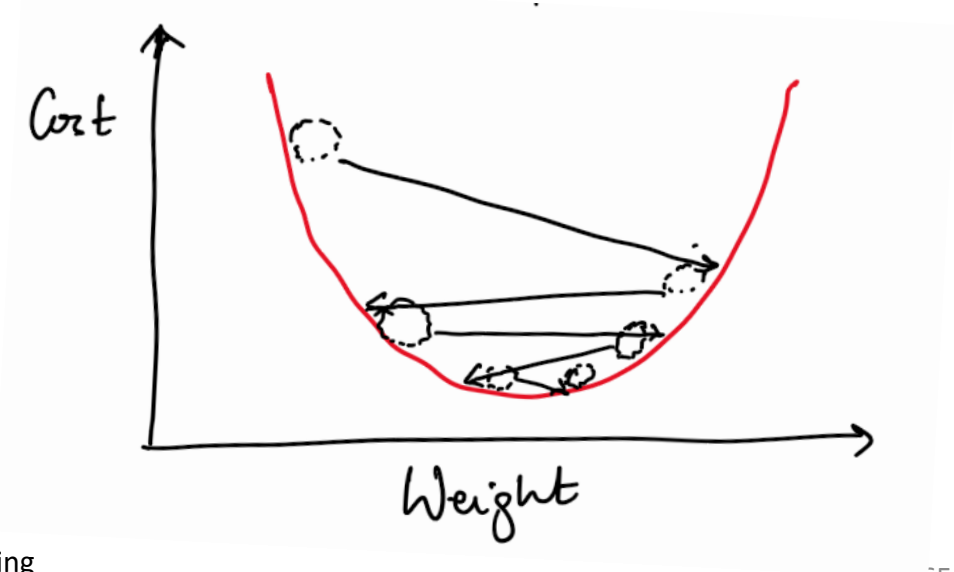
Gradient Descent

- When learning rate too low ➔ slow convergence.
- When learning rate too high ➔ minima will be overshot ➔ slow or no convergence.

- Learning rate is a ***Hyperparameter***.
- **It must be fine tuned. Neither too high, nor too low. We see hyperparameter tuning later.**

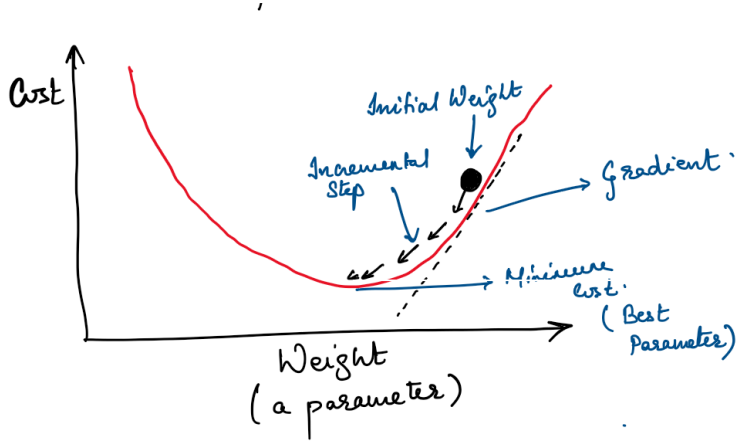
- GD works well when the total loss function is a convex function.

- What happens when function is non-convex?



Gradient Descent

- GD works well when the total loss function is a convex function.

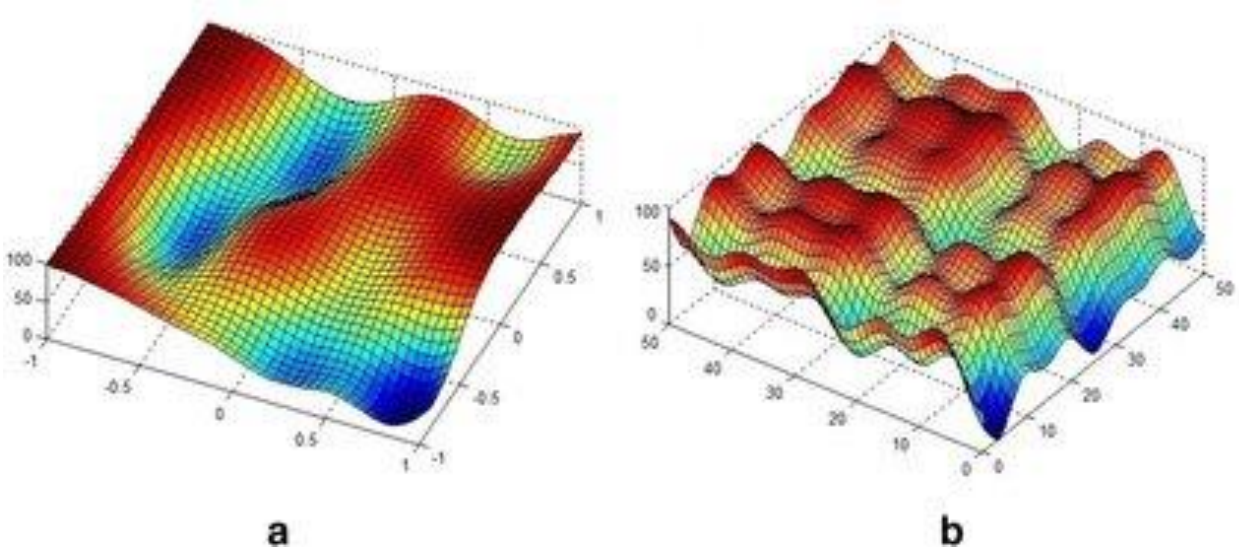


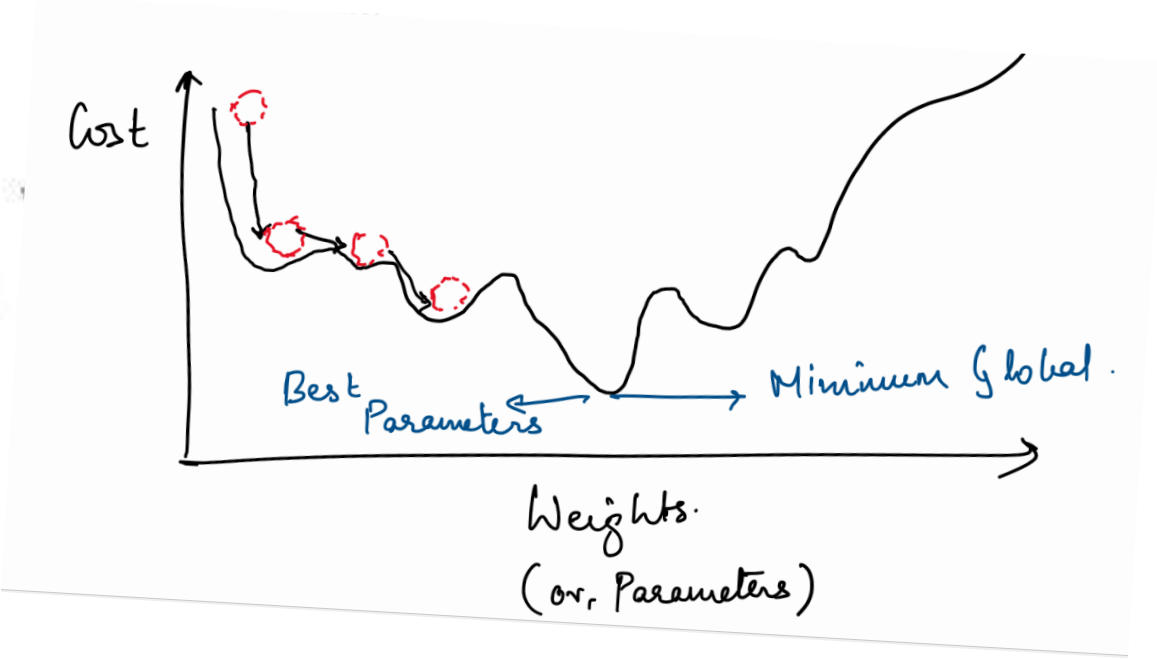
- What happens when function is non-convex?

exploration (random) - stocaticity : more powerfull strategie

Usually, the case, when millions of data are considered for training,

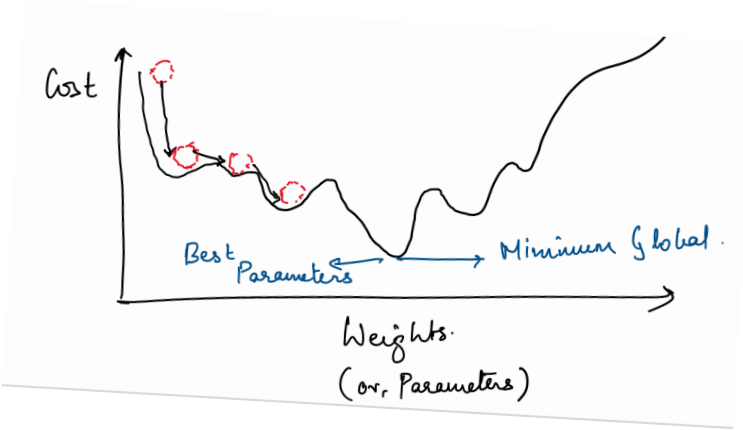
with millions of parameters (weights in many layers of NNs).





Source: Taig et al.

Stochastic Gradient Descent

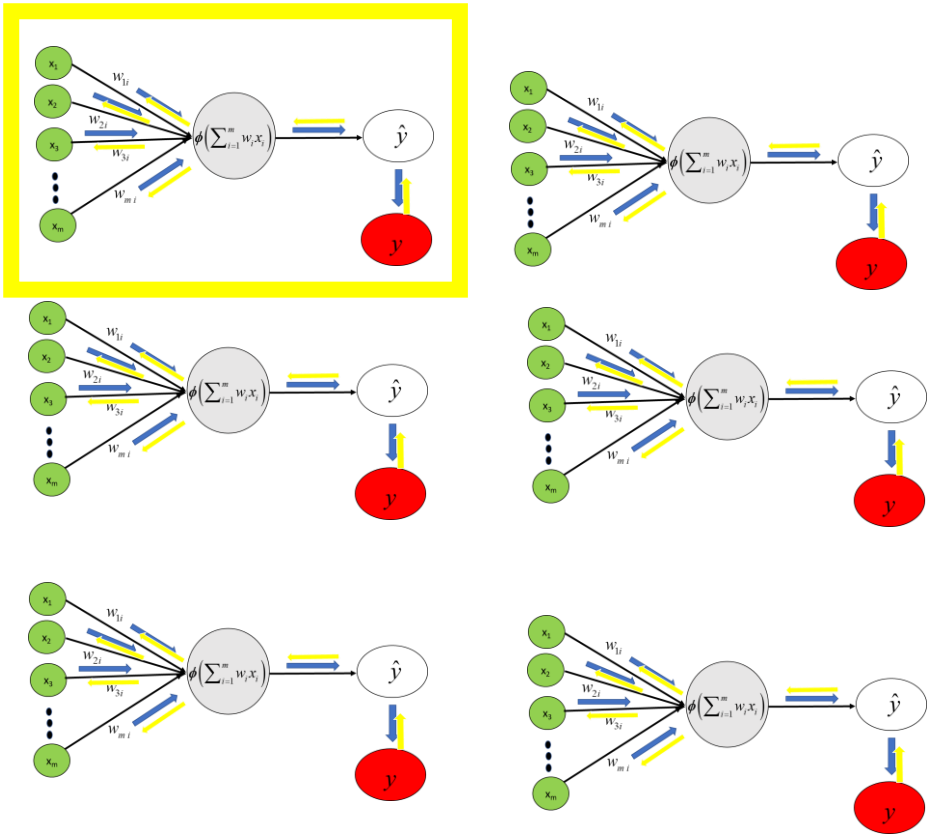


- GD : Consider a batch (set) of training data samples:
  - calculate loss
  - update weights based on total loss.

- Curse of dimensionality: Need more data for training, updating for whole set → extremely slow updates.

- To avoid getting stuck in local minima, a certain “jittering” or noise /exploration is needed.

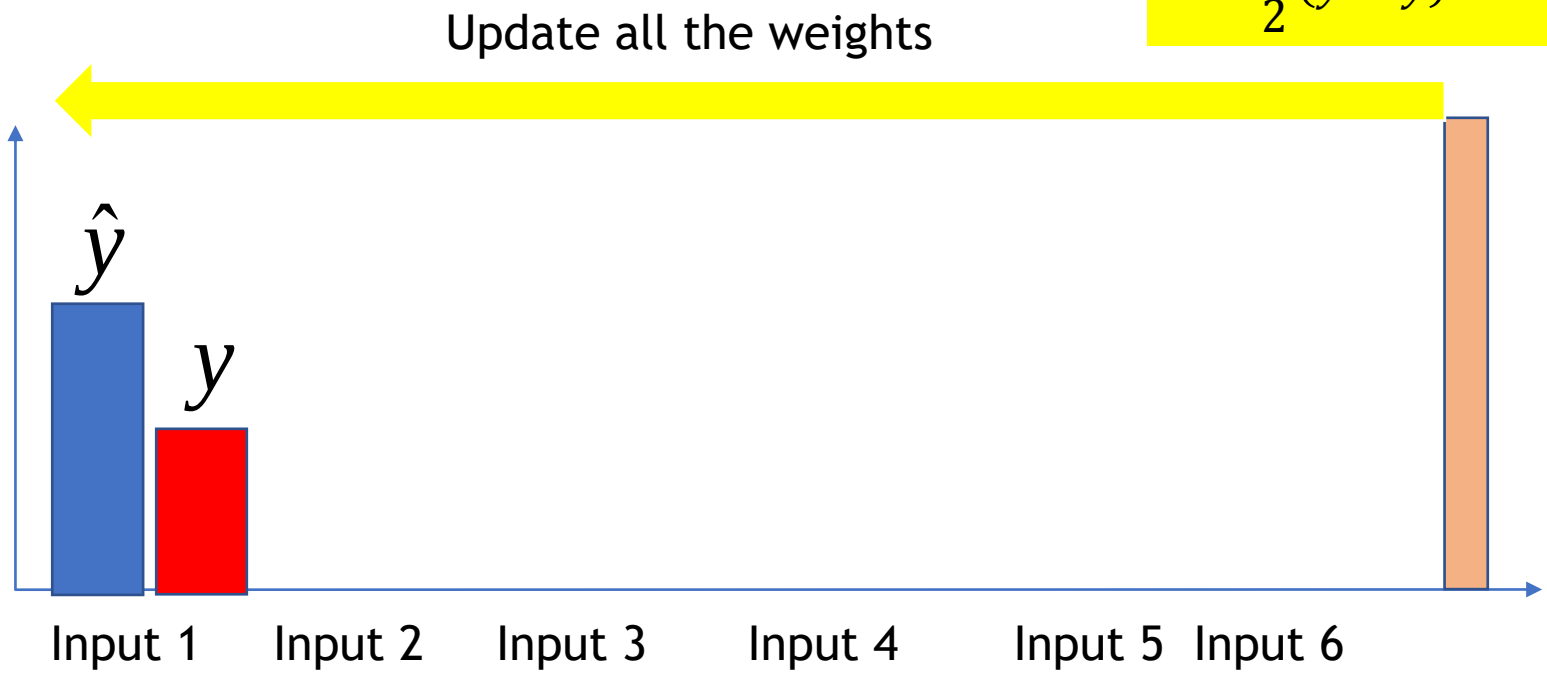
Stochastic Gradient Descent



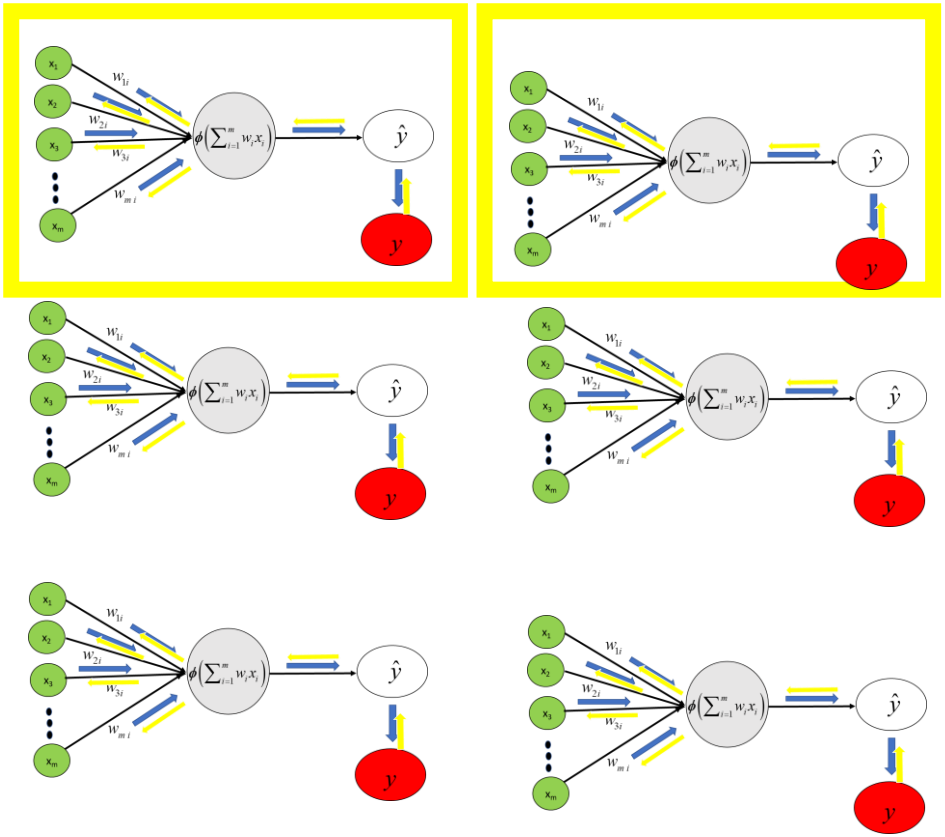
One epoch = training done on entire data set once.

- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

$$E = \frac{1}{2}(y - \hat{y})^2$$



Stochastic Gradient Descent

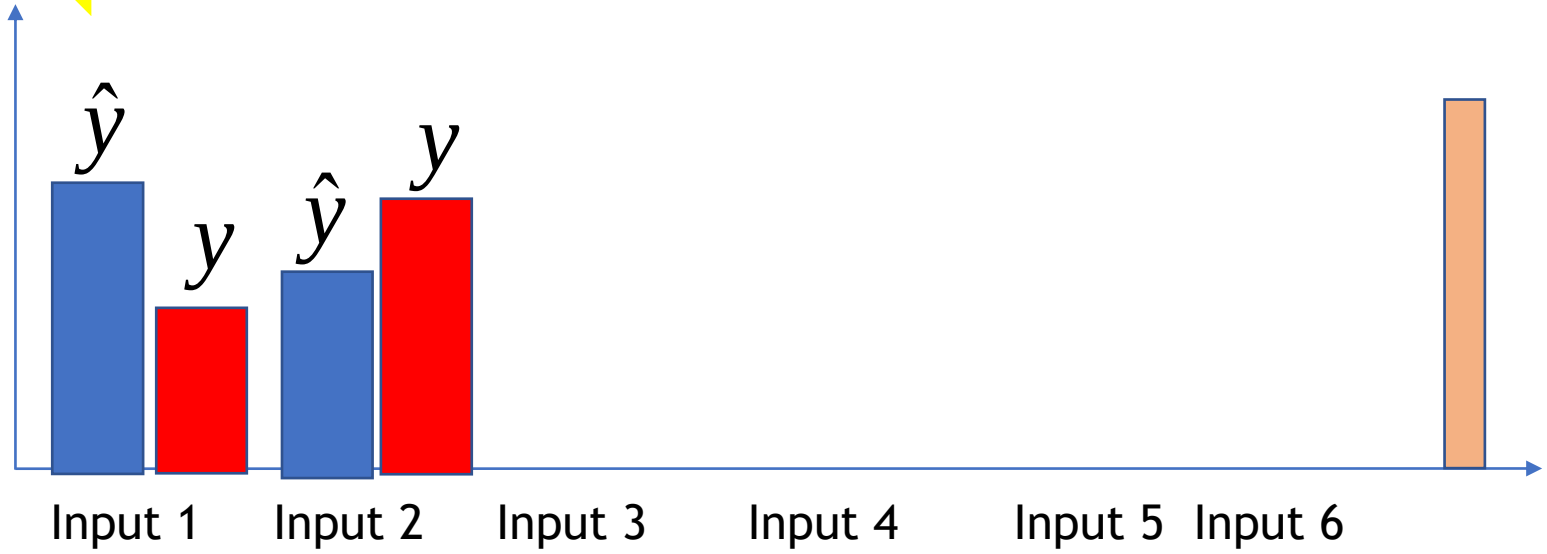


One epoch = training done on entire data set once.

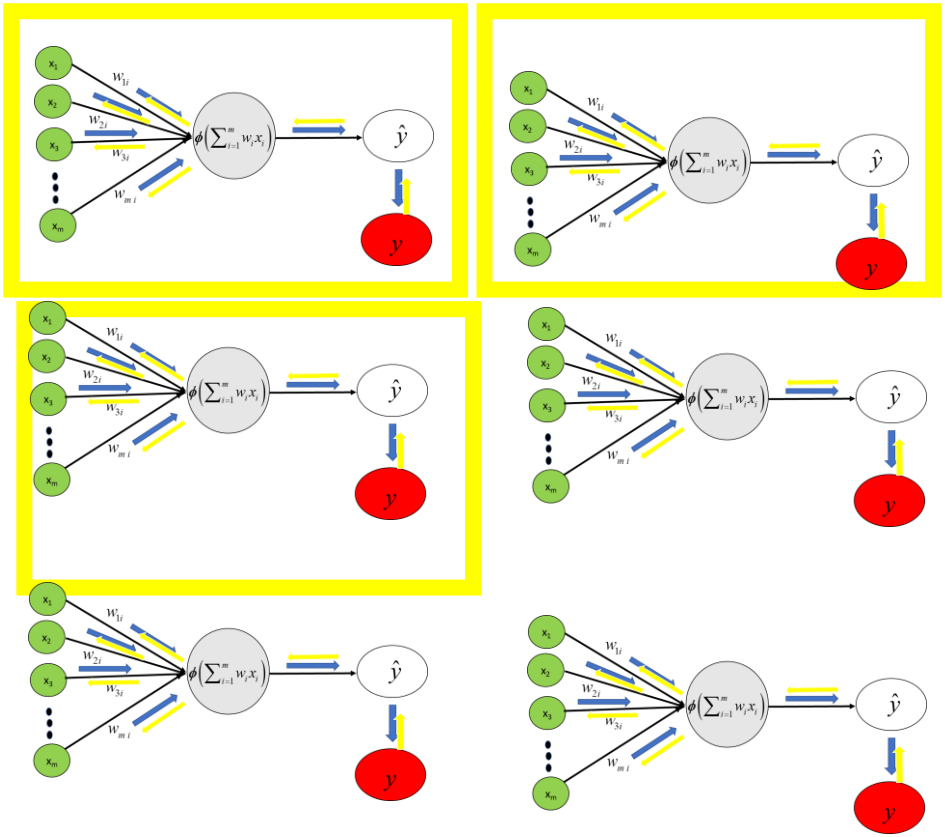
- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

$$E = \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



Stochastic Gradient Descent

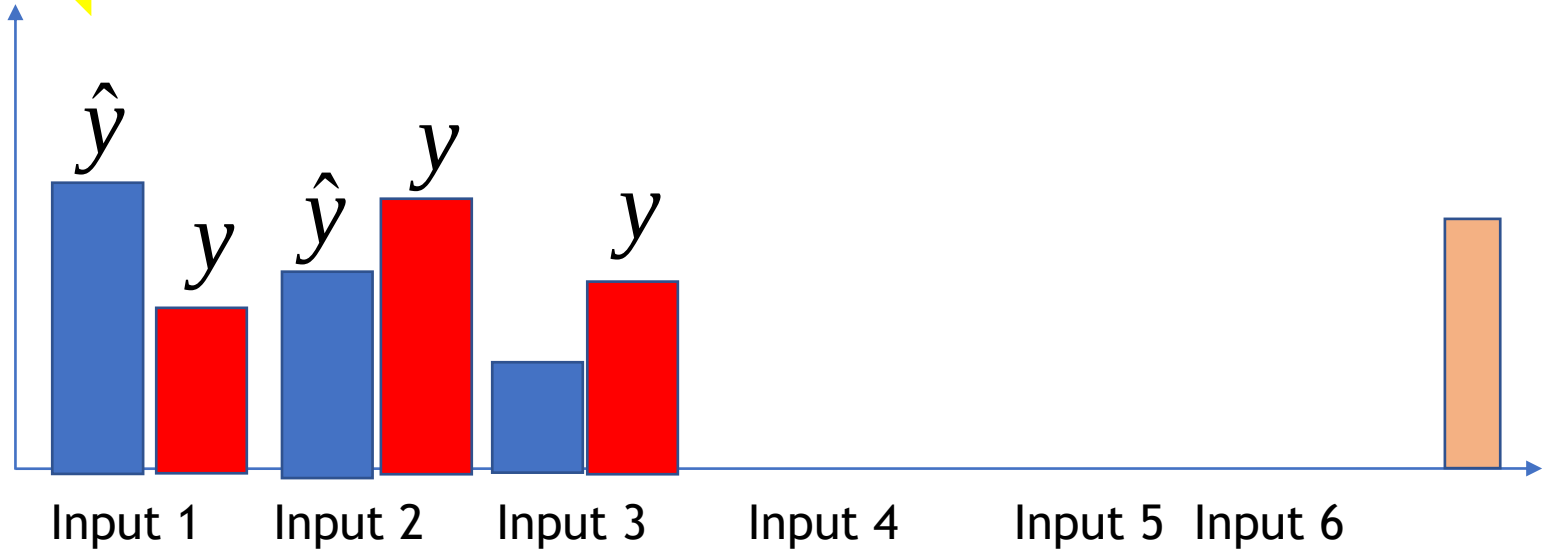


One epoch = training done on entire data set once.

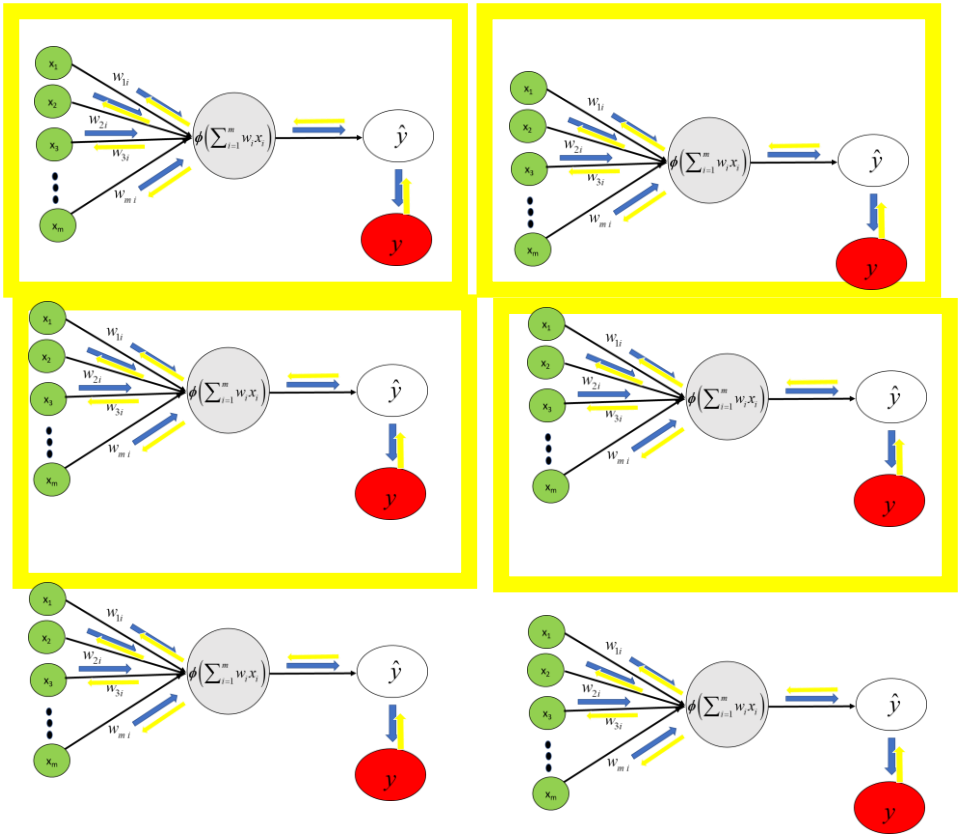
- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

$$E = \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



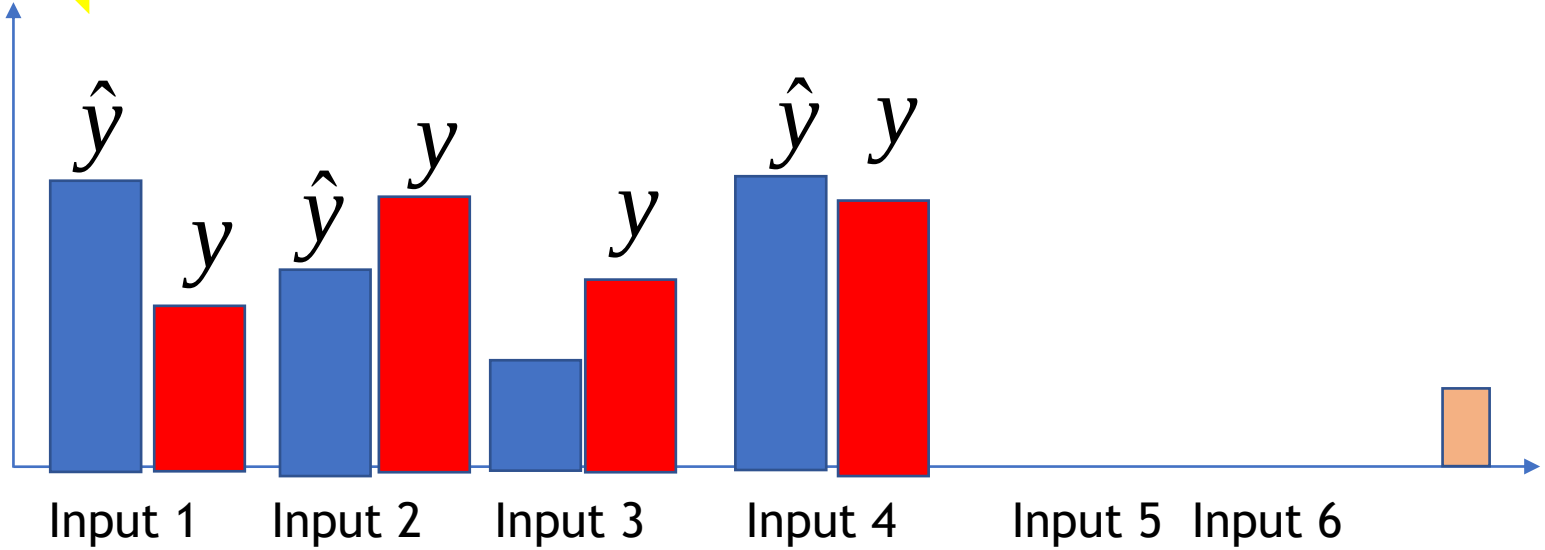
Stochastic Gradient Descent



- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

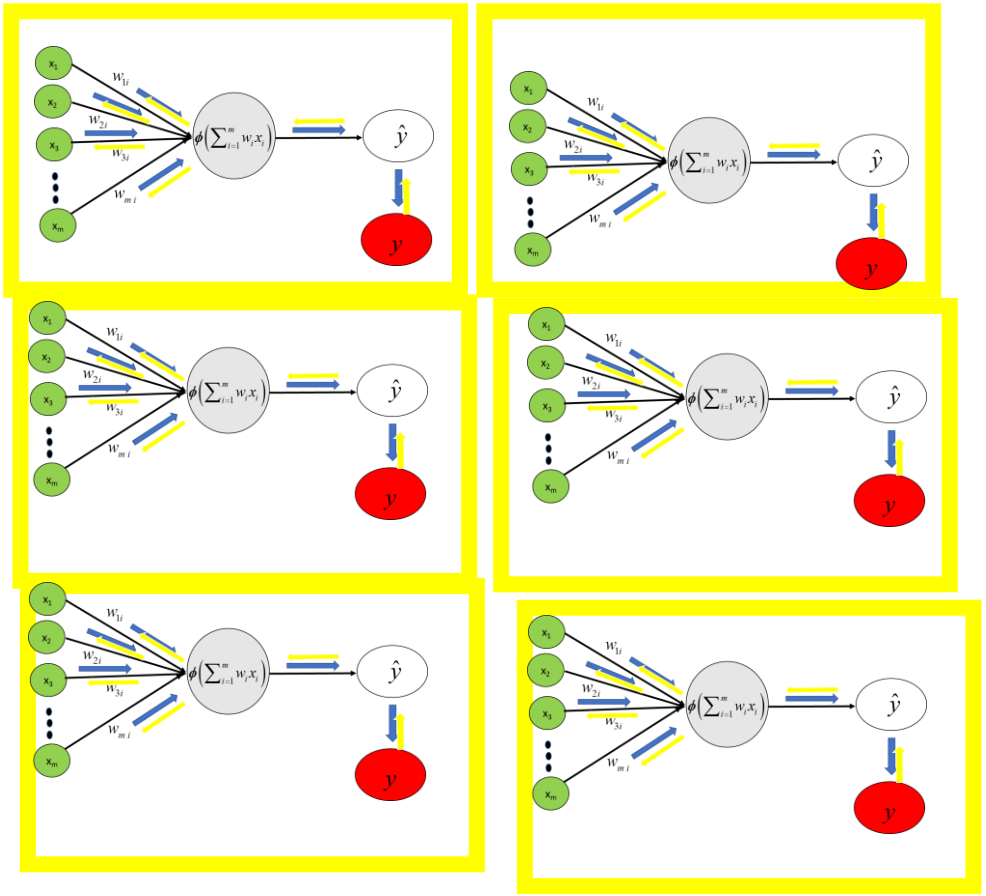
$$E = \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



One epoch = training done on entire data set once.

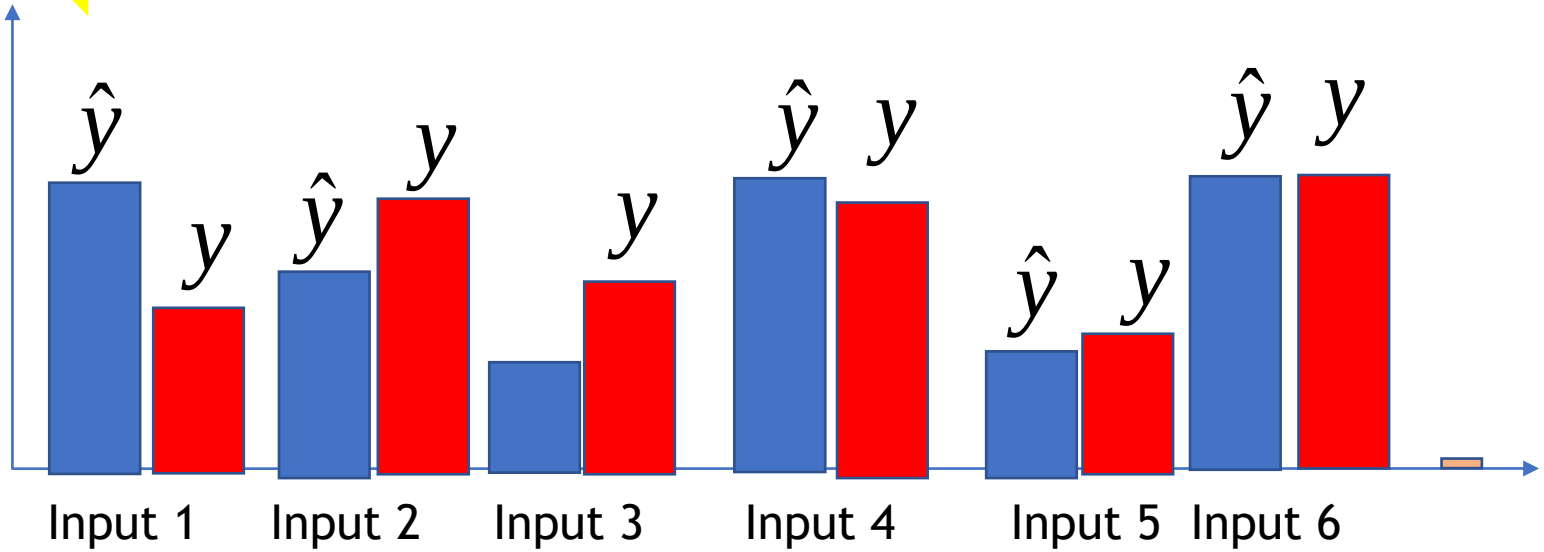
Stochastic Gradient Descent



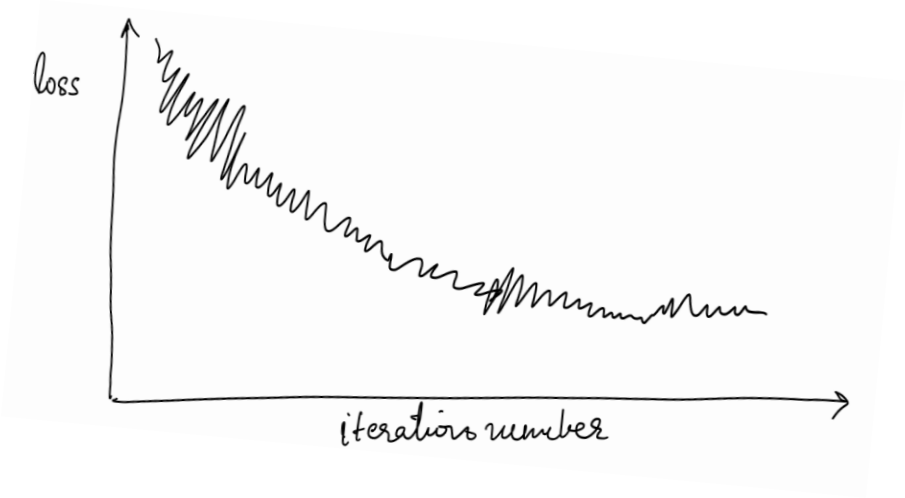
- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.
- Iterate until convergence detected.

$$E = \frac{1}{2}(y - \hat{y})^2$$

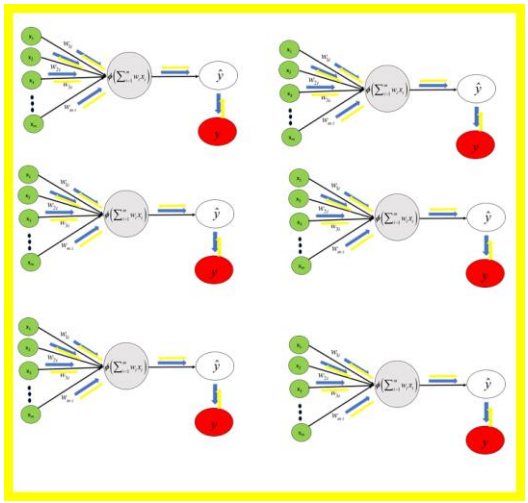
Update all the weights



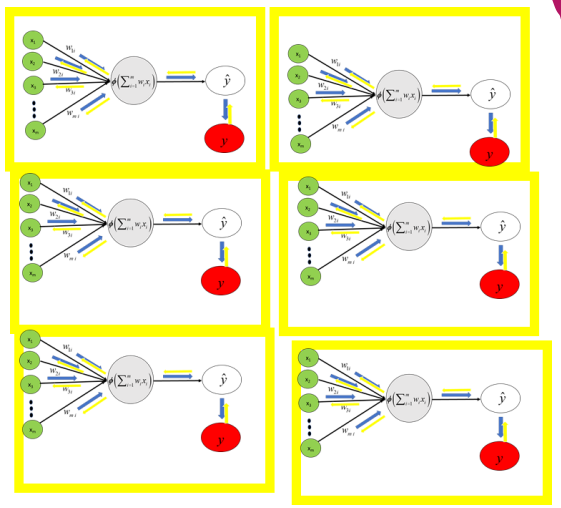
One epoch = training done on entire data set once.



- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.
- Iterate until convergence detected.



Batch GD



Stochastic GD

Batch GD : stores all data loss, updates after all data loss taken into account.

SGD : updates after each data sample.

- less time consuming
  - NN updated after each data,
  - memory not allocated to all data at once.
  - but, cannot vectorize the computations. (as only one data input treated once).

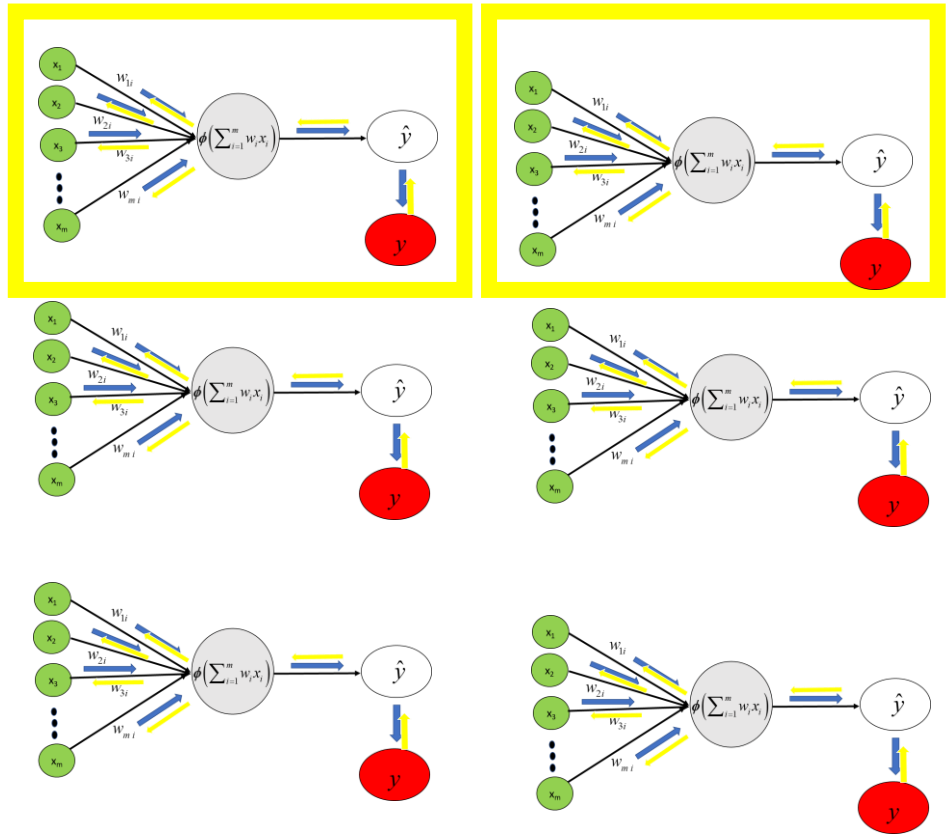
What happens when millions of data samples? but limited memory resources?

Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5.

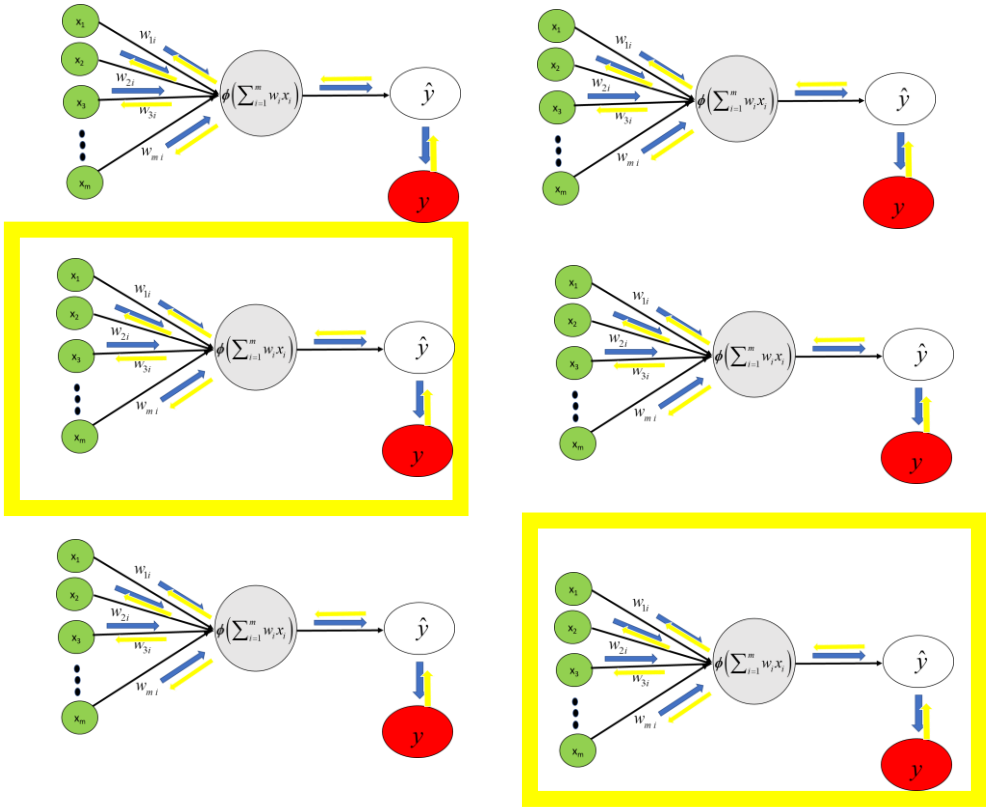


Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. **Repeat steps 1-4 for all the mini-batches**

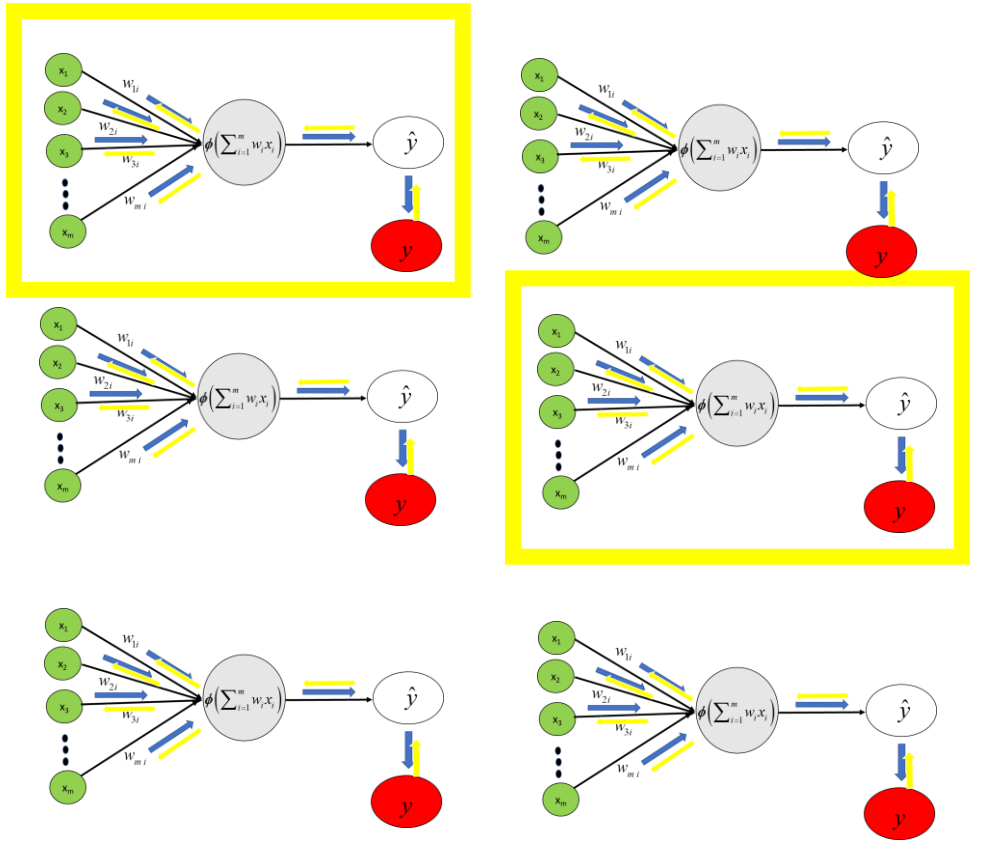


Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the mini-batches.

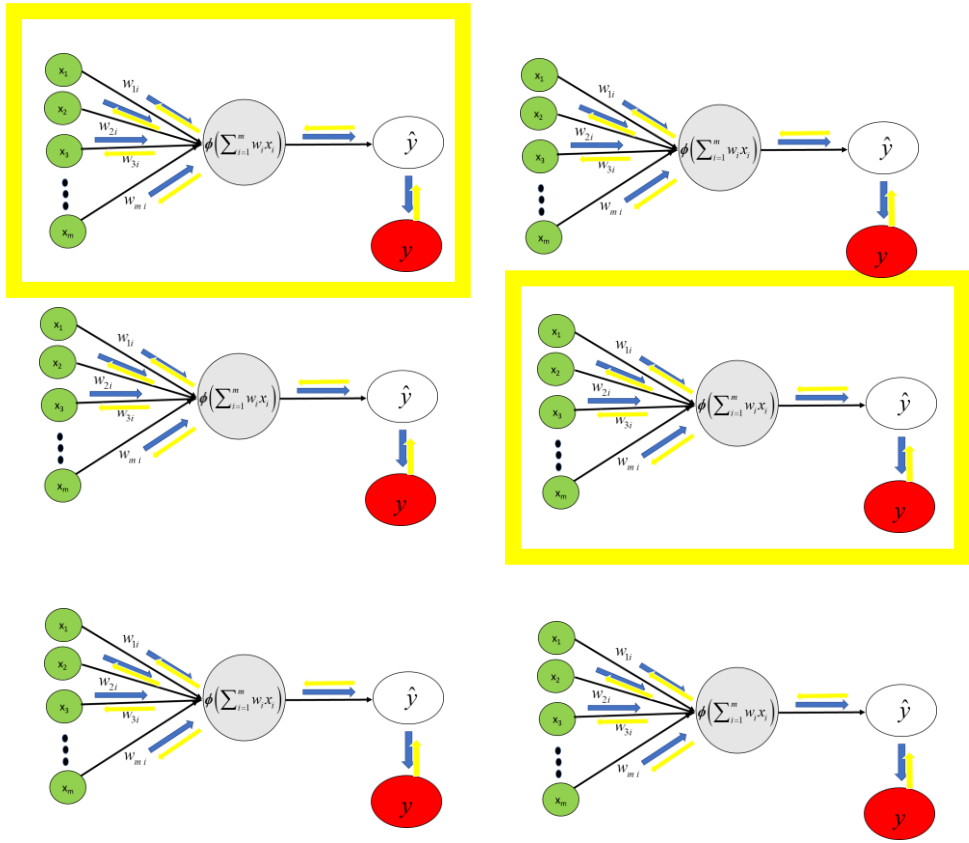


Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1-4 for the mini-batches we created.



Great!! We now know how NNs update weights …..using:

batch-GD, SGD or mini batch SGD….but…

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \left(J(\mathbf{w}, b)\right)}{\partial \mathbf{w}}$$

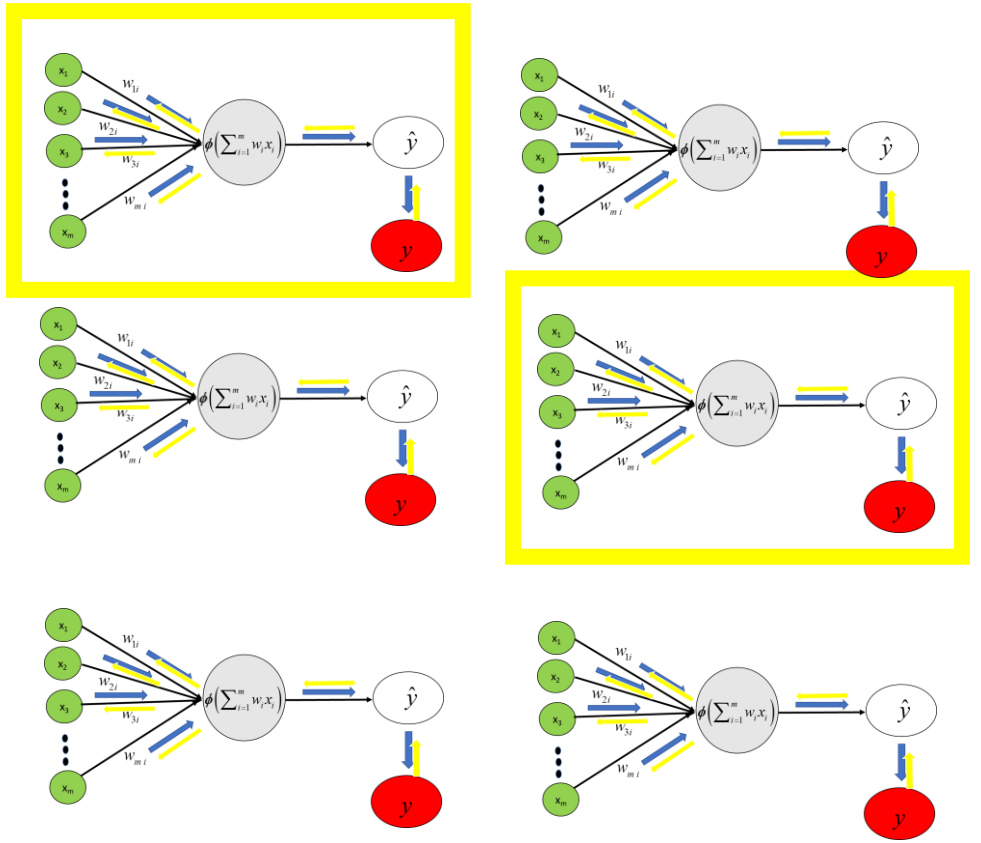
$$b \leftarrow b - \alpha \frac{\partial \left(J(w, b)\right)}{\partial b}$$

Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1-4 for the mini-batches we created.



Great!! We now know how NNs update weights …..using:

batch-GD, SGD or mini batch SGD….but…

how to calculate the gradient of the cost function!!

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial (J(\mathbf{w}, b))}{\partial \mathbf{w}}$$

$$b \leftarrow b - \alpha \frac{\partial (J(w, b))}{\partial b}$$

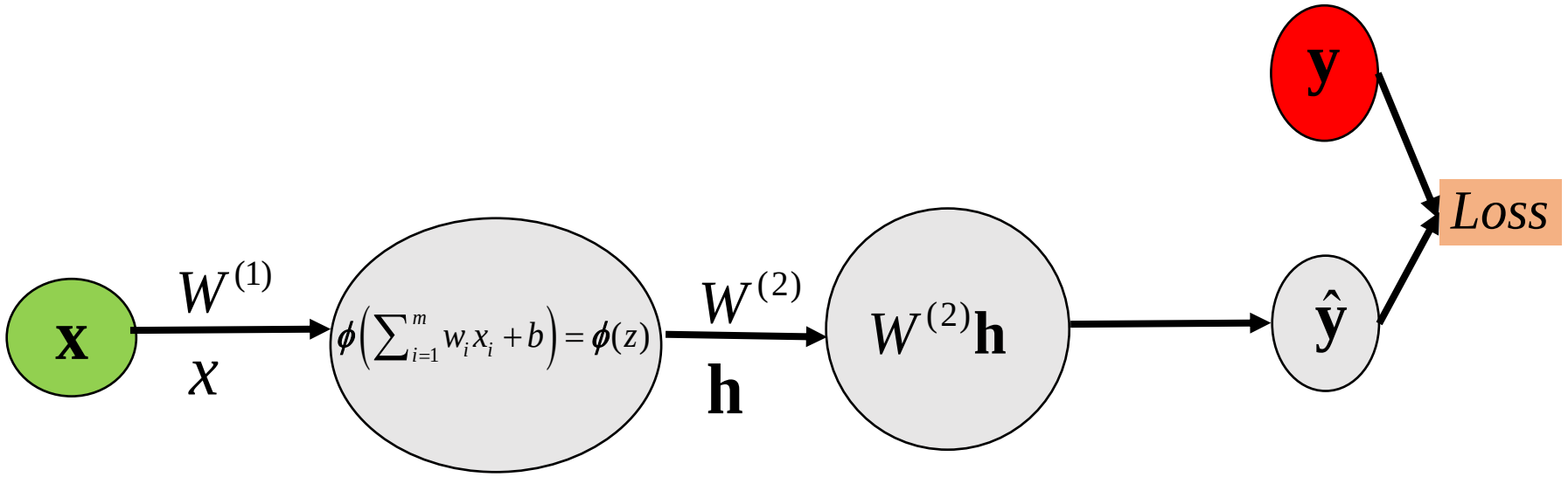
Backpropagation (Backprop)

Back propagation

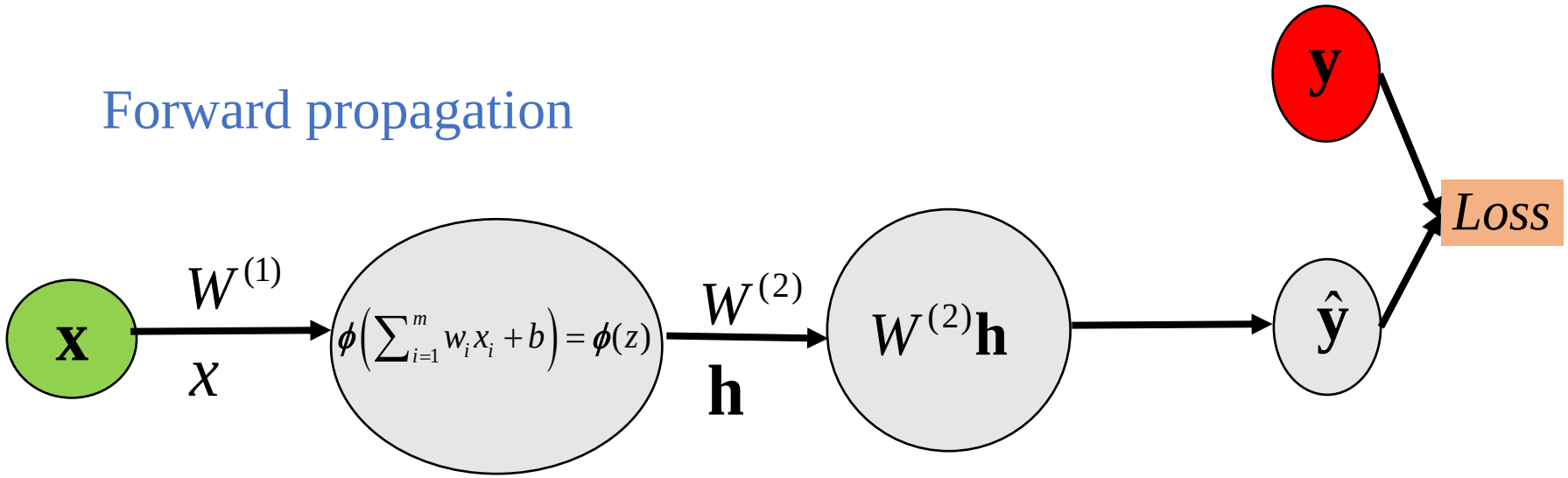
- Intuition: the global error is backward propagated to network nodes, weights are modified proportional to their contribution
- Objective: Calculate rate of change of Error with respect to each weights, to correct the weights.

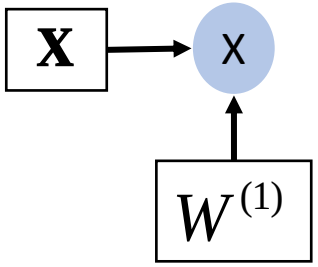
- Backpropagation rediscovered in 1986, efficient way of propagating backwards the error gradient and updating the weights.

but first, Forward Propagation : Illustration using **2 Hidden layer Deep NN**.

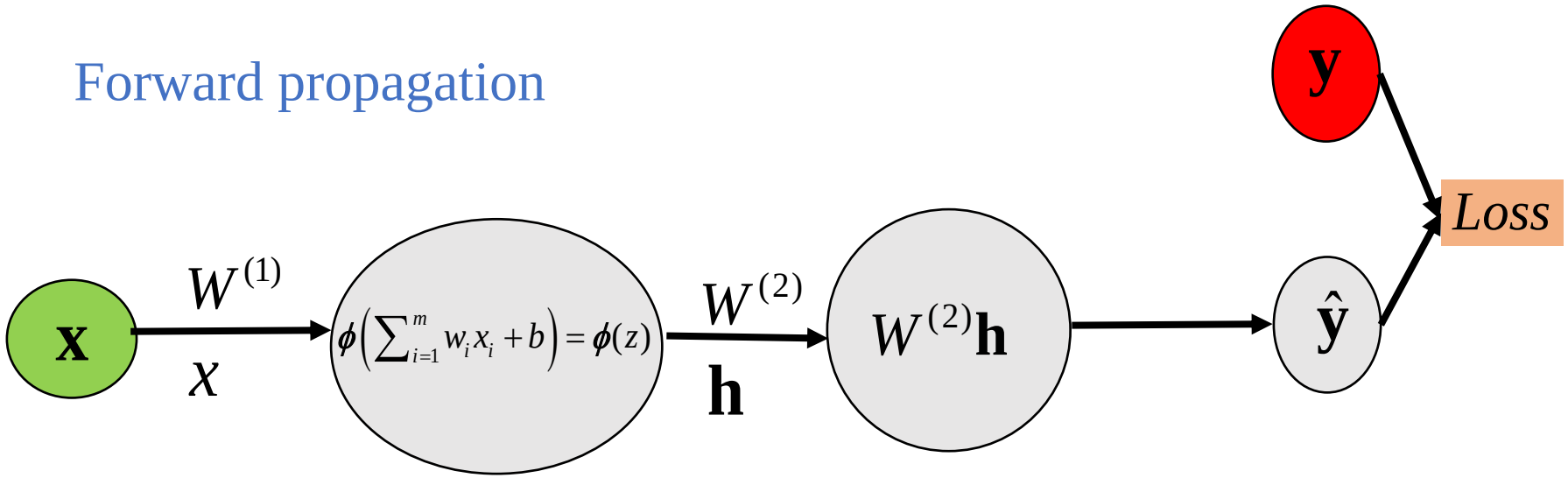


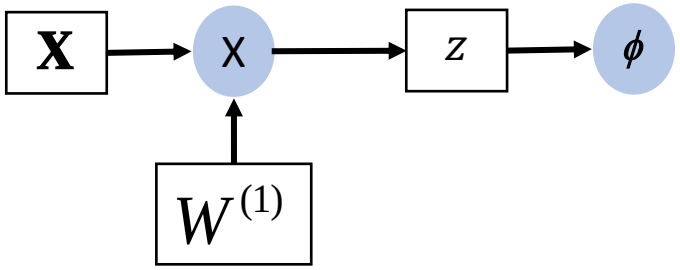
Forward propagation





Forward propagation





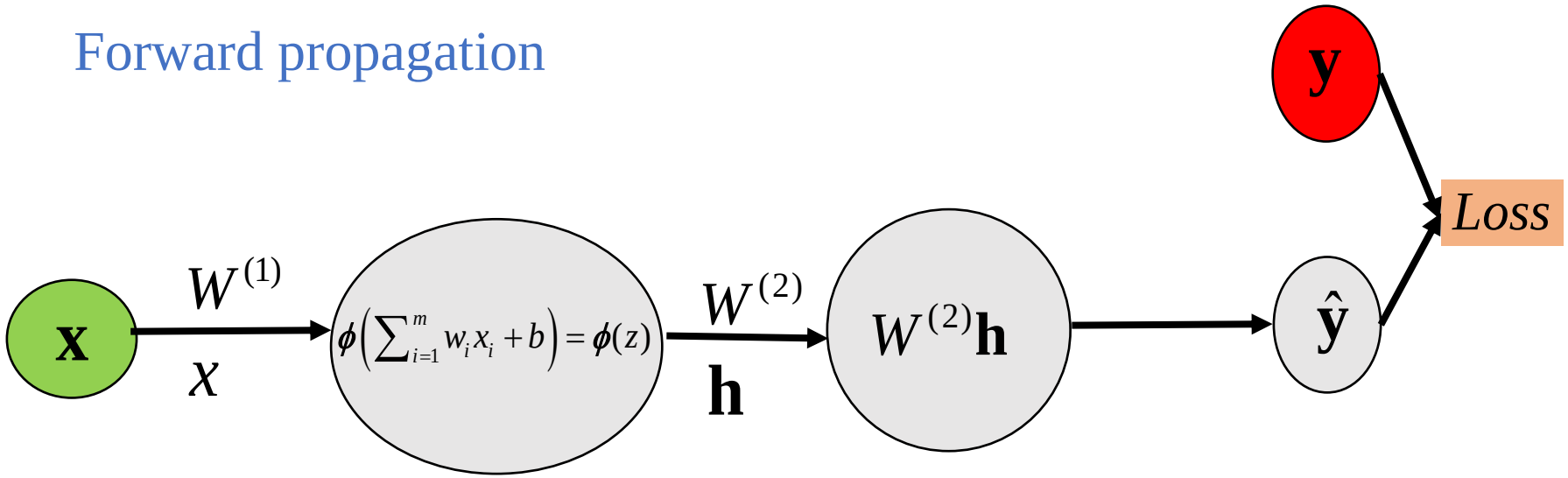
$z = W^{(1)} x$

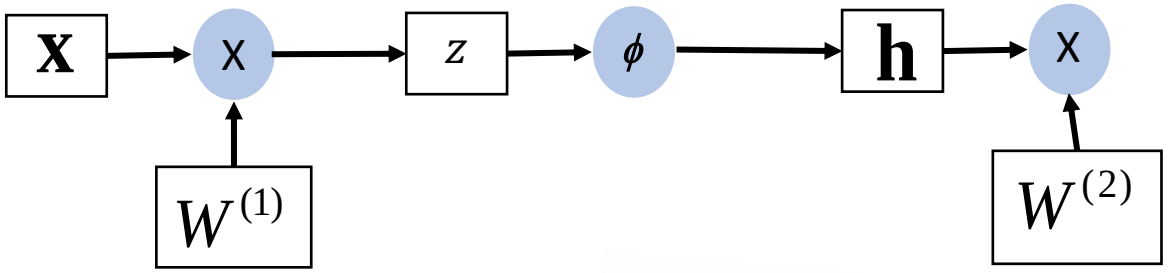
$x \in \mathbb{R}^d$

$W^{(1)} \in \mathbb{R}^{h \times d}$

$z \in \mathbb{R}^h$

Forward propagation





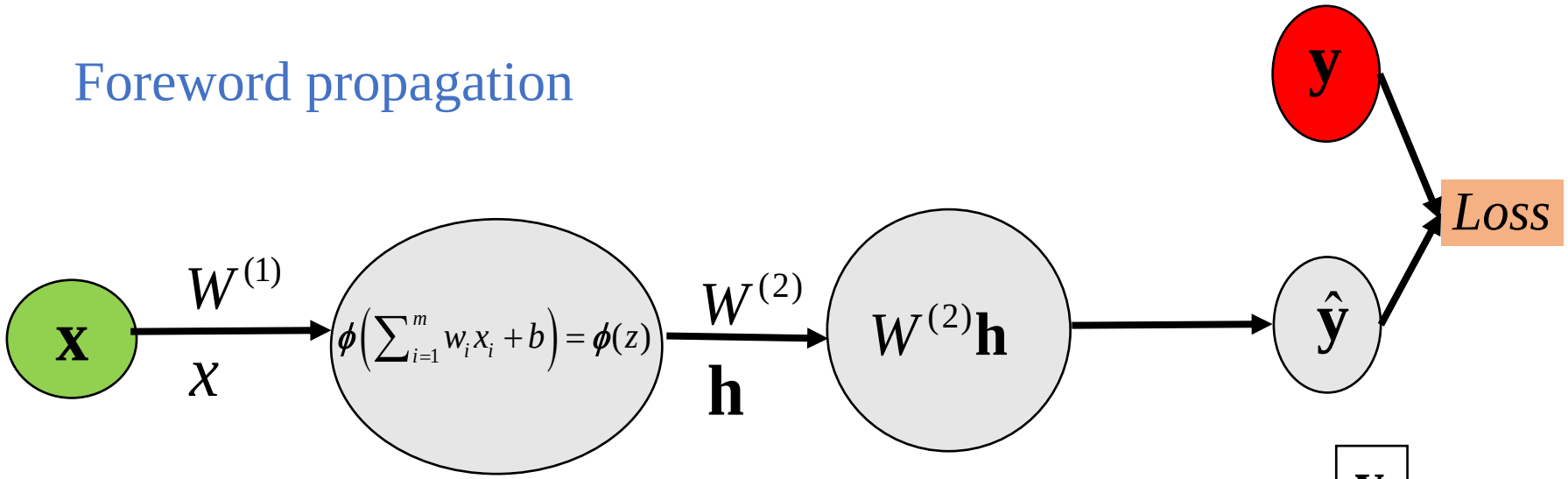
$z = W^{(1)} x$ $\quad h = \phi(z)$

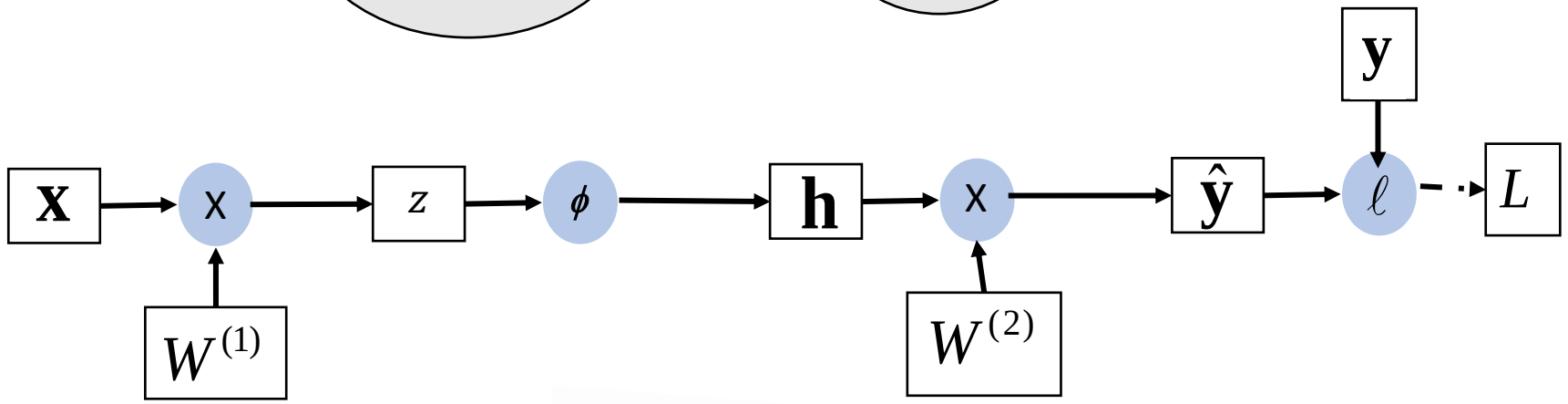
$x \in \mathbb{R}^d$

$W^{(1)} \in \mathbb{R}^{h \times d}$

$z \in \mathbb{R}^h$

Foreword propagation





$z = W^{(1)} x$

$h = \phi(z)$

$\hat{y} = W^{(2)} h$

$L = \ell(\hat{y}, y)$

$x \in \mathbb{R}^d$

$W^{(1)} \in \mathbb{R}^{h \times d}$

$z \in \mathbb{R}^h$

$W \in \mathbb{R}^{q \times h}$

$J = L$

Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(X)$$
$$Z = g(Y) = g \circ f(X)$$
$$\text{Then, } \frac{\partial Z}{\partial X} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right)$$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Back propagation

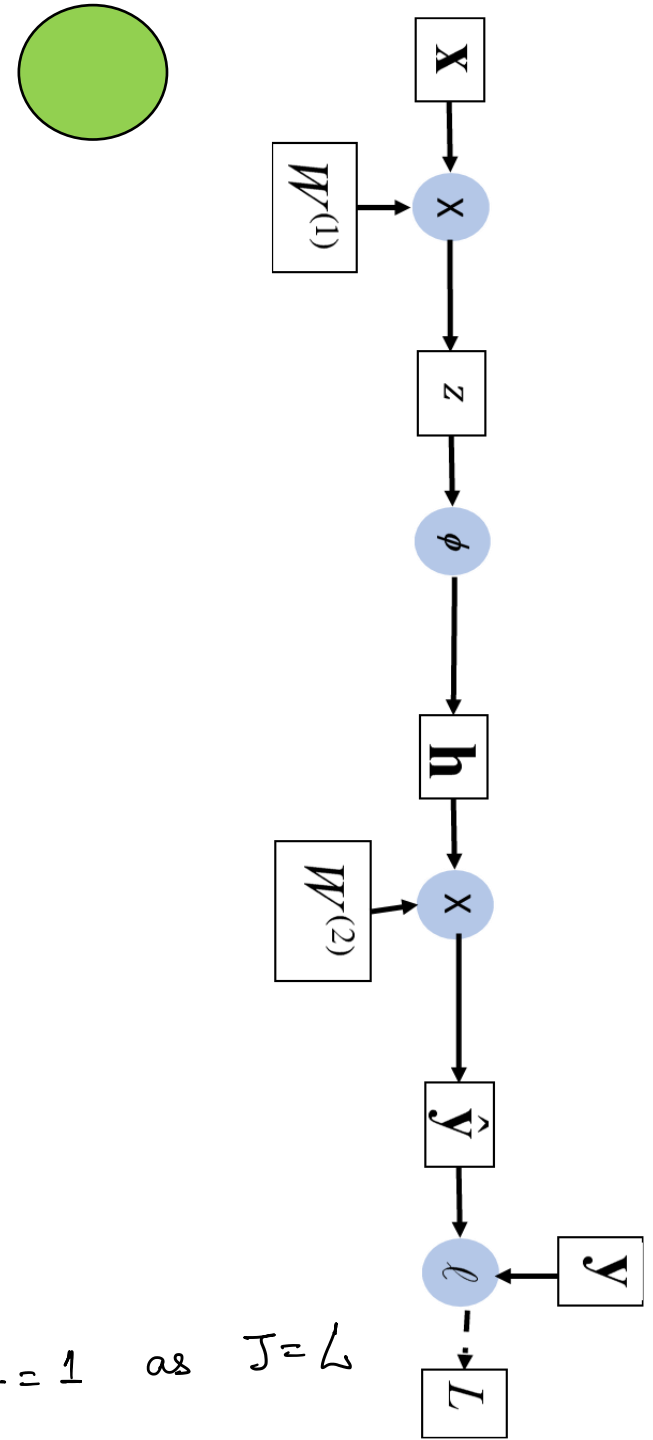
- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$y = f(x)$$
$$z = g(y) = g \circ f(x)$$
$$\text{Then, } \frac{\partial z}{\partial x} = \text{prod}\left(\frac{\partial z}{\partial y}, \frac{\partial y}{\partial x}\right)$$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$



$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$

Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

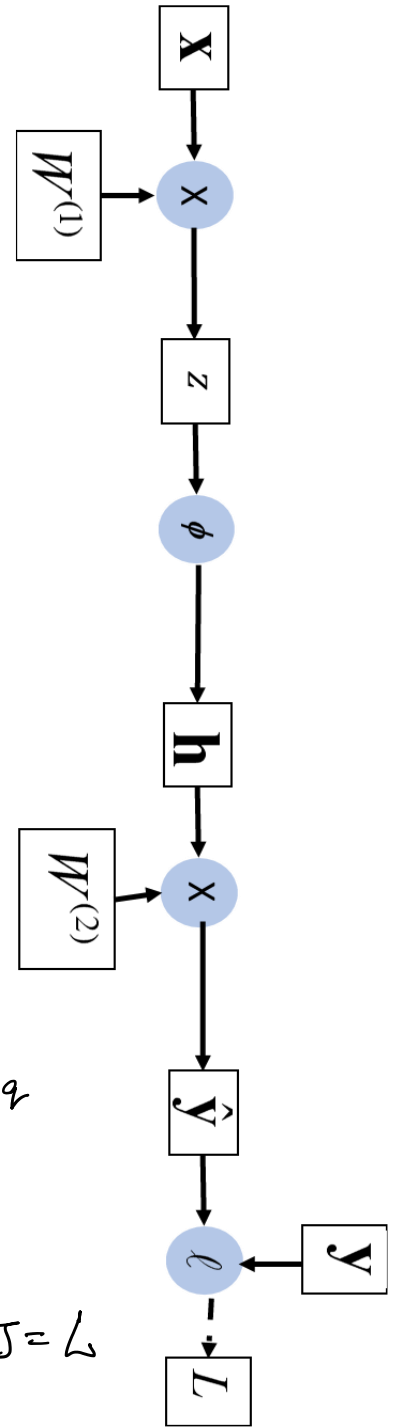
$$Y = f(X)$$
$$Z = g(Y) = g \circ f(X)$$
$$\text{Then, } \frac{\partial Z}{\partial X} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right)$$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$



$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$

Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(X)$$
$$Z = g(Y) = g \circ f(X)$$
$$\text{Then, } \frac{\partial Z}{\partial X} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right)$$

Objective of Backprop:

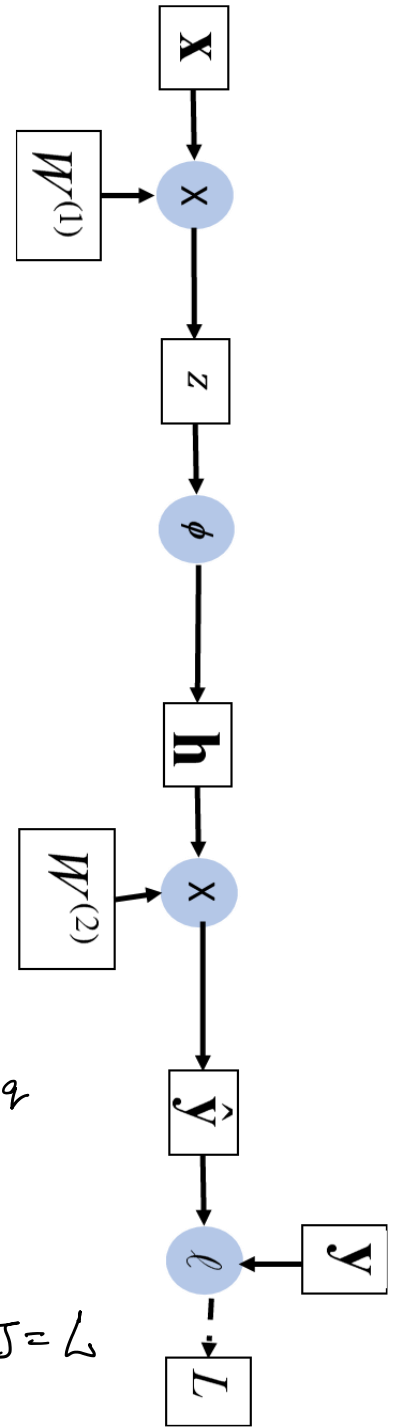
$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Gradient wrt $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$



$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$

Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$y = f(x)$$
$$z = g(y) = g \circ f(x)$$
$$\text{Then, } \frac{\partial z}{\partial x} = \text{prod}\left(\frac{\partial z}{\partial y}, \frac{\partial y}{\partial x}\right)$$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Gradient wrt $W^{(2)}$

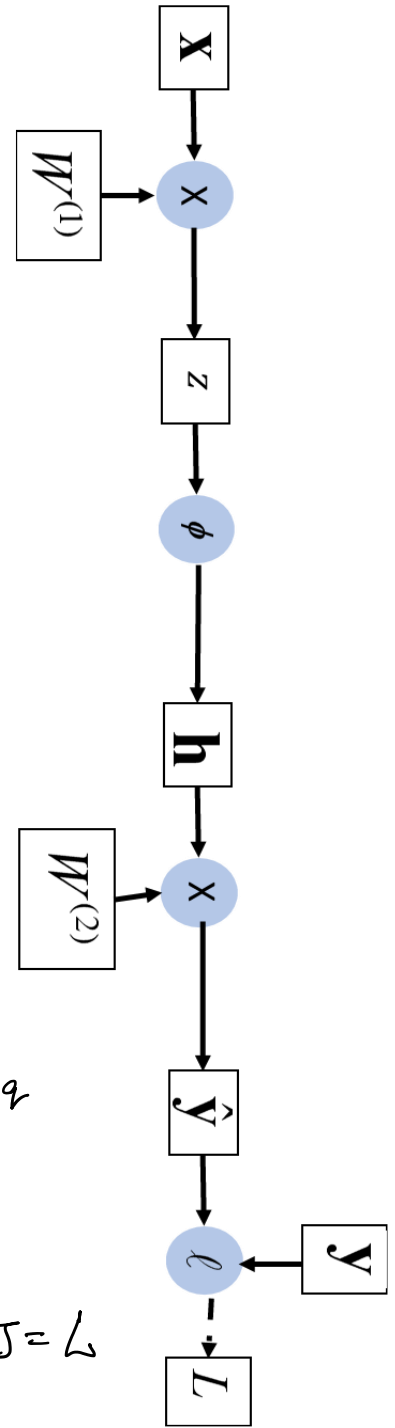
Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)^T} \frac{\partial J}{\partial \hat{y}}$$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$



Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$y = f(x)$

$z = g(y) = g \circ f(x)$

Then, $\frac{\partial z}{\partial x} = \text{prod}\left(\frac{\partial z}{\partial y}, \frac{\partial y}{\partial x}\right)$

$$\frac{\partial J}{\partial z} = \text{prod}\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)^T} \frac{\partial J}{\partial \hat{y}}$$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

Gradient wrt $W^{(2)}$

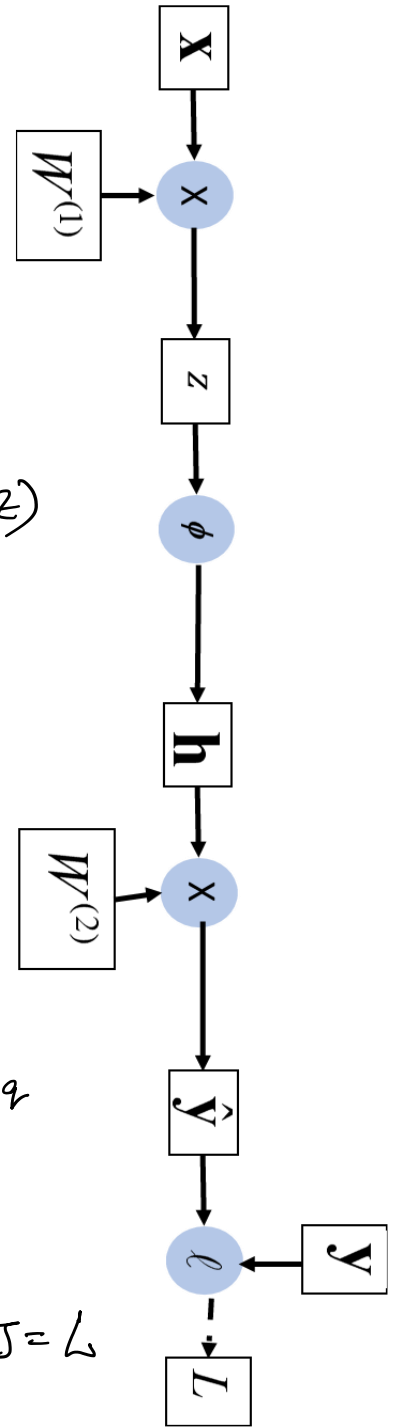
Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$



Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

$$\frac{\partial J}{\partial W^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial z}, \frac{\partial z}{\partial W^{(1)}}\right) = \frac{\partial J}{\partial z} x^T$$

- Reminder: Chain rule

$$y = f(x)$$
$$z = g(y) = g \circ f(x)$$
$$\text{Then, } \frac{\partial z}{\partial x} = \text{prod}\left(\frac{\partial z}{\partial y}, \frac{\partial y}{\partial x}\right)$$

$$\frac{\partial J}{\partial z} = \text{prod}\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)T} \frac{\partial J}{\partial \hat{y}}$$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

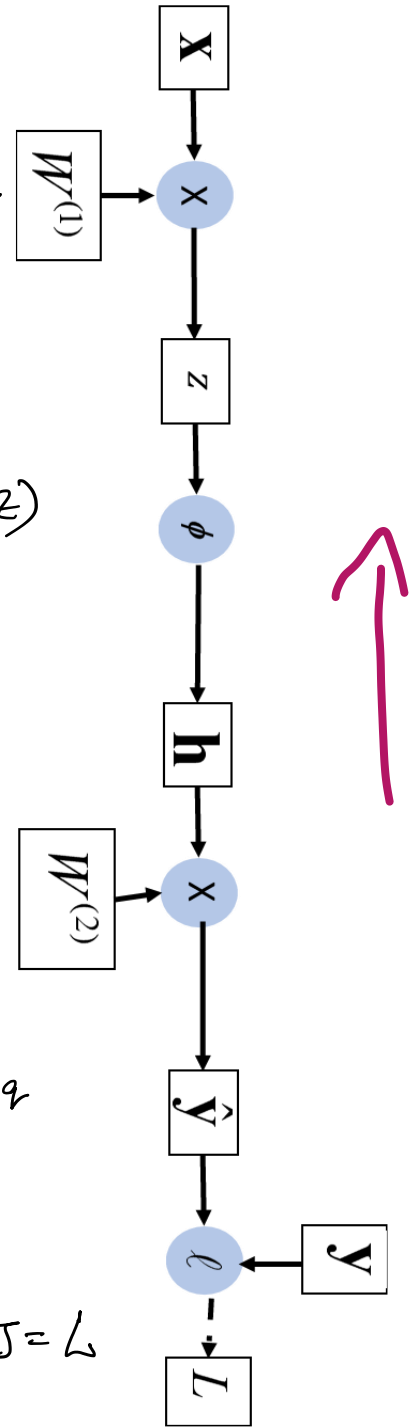
Gradient wrt $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$



Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights

Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights
2. Present the data input and targets for the deep NN

Forward propagation: Traverse the computational graph in the direction of dependencies and compute all the variables on its path.

Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights
2. Present the data input and targets for the deep NN

Forward propagation: Traverse the computational graph in the direction of dependencies and compute all the variables on its path.

3. Compute Deep NN output
4. Back propagation of errors

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\text{where, } \Delta w_{ij} = -\alpha \frac{\partial J}{\partial w_{ij}}$$

5. Update all the weights using Gradient descent:

Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights
2. Present the data input and targets for the deep NN

Forward propagation: Traverse the compute graph in the direction of dependencies and compute all the variables on its path.

3. Compute Deep NN output
4. Back propagation of errors
5. Update all the weights using Gradient descent:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\text{where, } \Delta w_{ij} = -\alpha \frac{\partial J}{\partial w_{ij}}$$

6. Repeat the steps from 2 , until acceptable error levels observed .

Remarks:

- intermediate values must be stored until backpropagation
- backpropagation requires significantly more memory than plain inference.
- Gradients as tensors variables must be stored to invoke the chain rule.
- Minibatches → GD on several data inputs together ➔ more intermediate activations need to be stored.

Deep NN Training Algorithm

On-Line algorithm:

1. **Initialize weights.** **How? what is the best way?** randomly !
2. Present the data input and targets for the deep NN

Forward propagation: Traverse the compute graph in the direction of dependencies and compute all the variables on its path.

3. Compute Deep NN output
4. Back propagation of errors
5. Update all the weights using Gradient descent:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\text{where, } \Delta w_{ij} = -\alpha \frac{\partial J}{\partial w_{ij}}$$

6. Repeat the steps from 2 , until **acceptable error** levels observed.

How to access? What is the best model? When is training over?

Remarks:

- intermediate values must be stored until backpropagation
- backpropagation requires significantly more memory than plain inference.
- Gradients as tensors variables must be stored to invoke the chain rule.
- Minibatches → GD on several data inputs together ➔ more intermediate activations need to be stored.

Summary

- Forward propagation sequentially calculates and stores intermediate variables within the compute graph defined by the neural network. It proceeds from input to output layer.

- Back propagation sequentially calculates and stores the gradients of intermediate variables and parameters within the neural network in the reversed order.

- When training deep learning models, forward propagation and back propagation are interdependent.

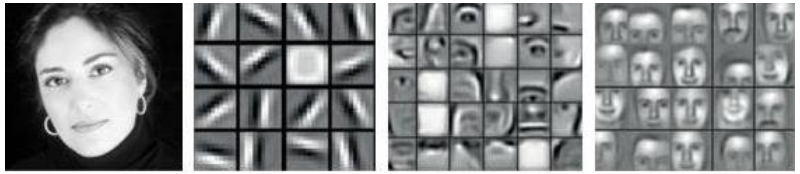
- Training requires significantly more memory and storage.

Generalization and Training

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Generalization

- Goal is to discover general pattern (underlying data distribution),
  achieve good generalization to **New** cases.



- but, generalized performance over unseen, new data (assumed same distribution)

depends upon:

  - rich, diversity and quantity of input data for training
  - complexity of model trained.
  - underfitting and overfitting of model

Example: Consider training data input represent only 10% of general distribution.

- trained model likely to perform well over training data,
- likely to perform poorly over test data .

Generalization

- Training Error: Error exhibited by model during training

- Generalization Error: Error expected when model applied over-imaginary (unseen) data sampled from underlying data distribution.

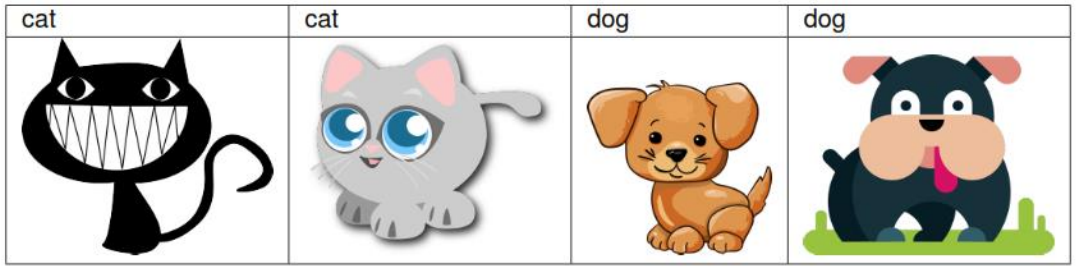
- Generalization error → Test error when that data is Test data

Remark: Assume that training set and test set drawn independently and identically from same underlying distribution (possibly hidden).

When **such is NOT** the case:

Training

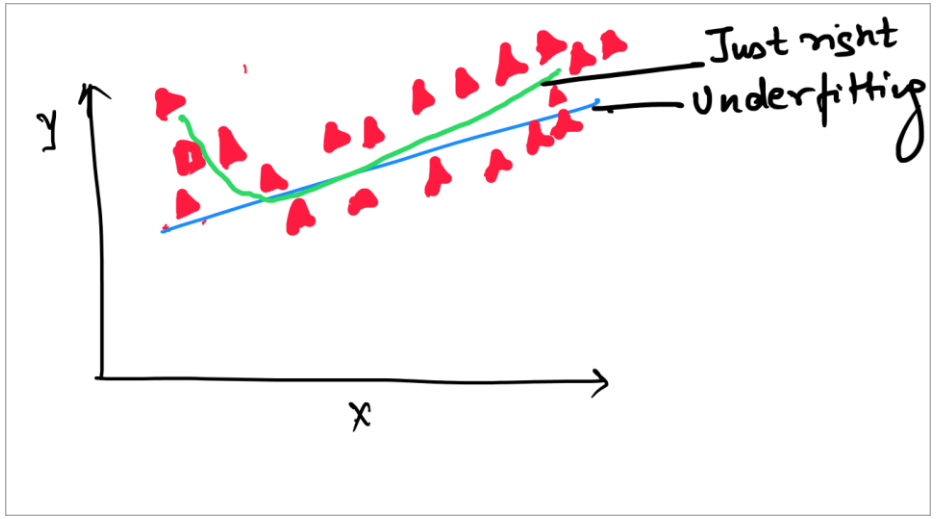
Test

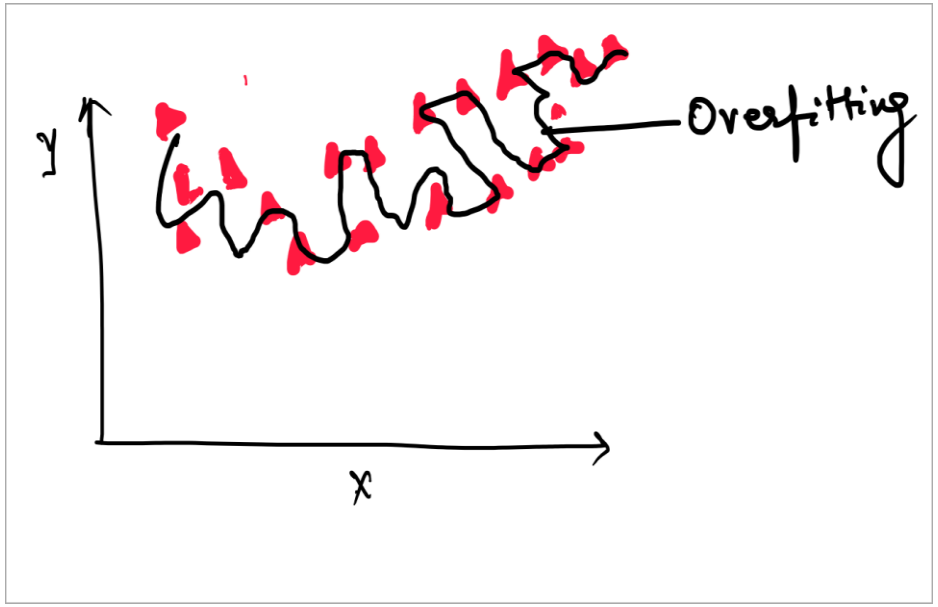


Generalization: Underfitting and Overfitting

- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.



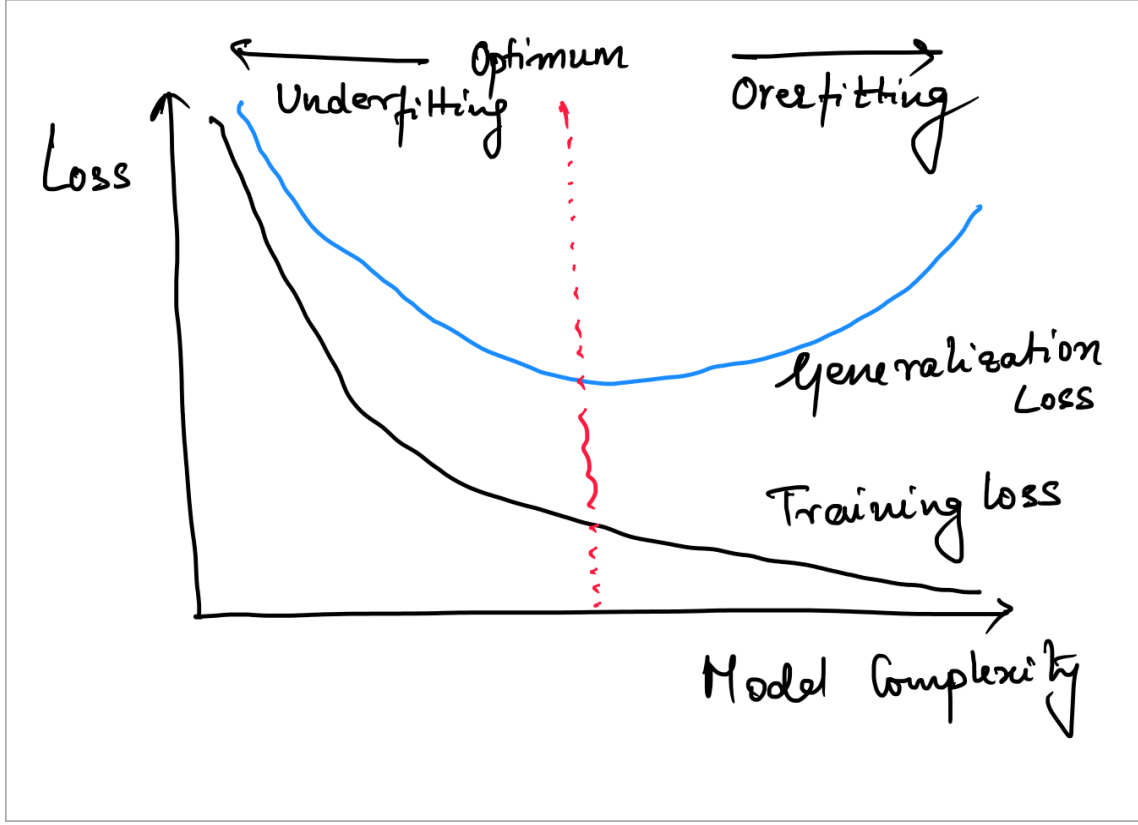


Generalization: Underfitting and Overfitting

- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.

How does it depend on Model complexity?

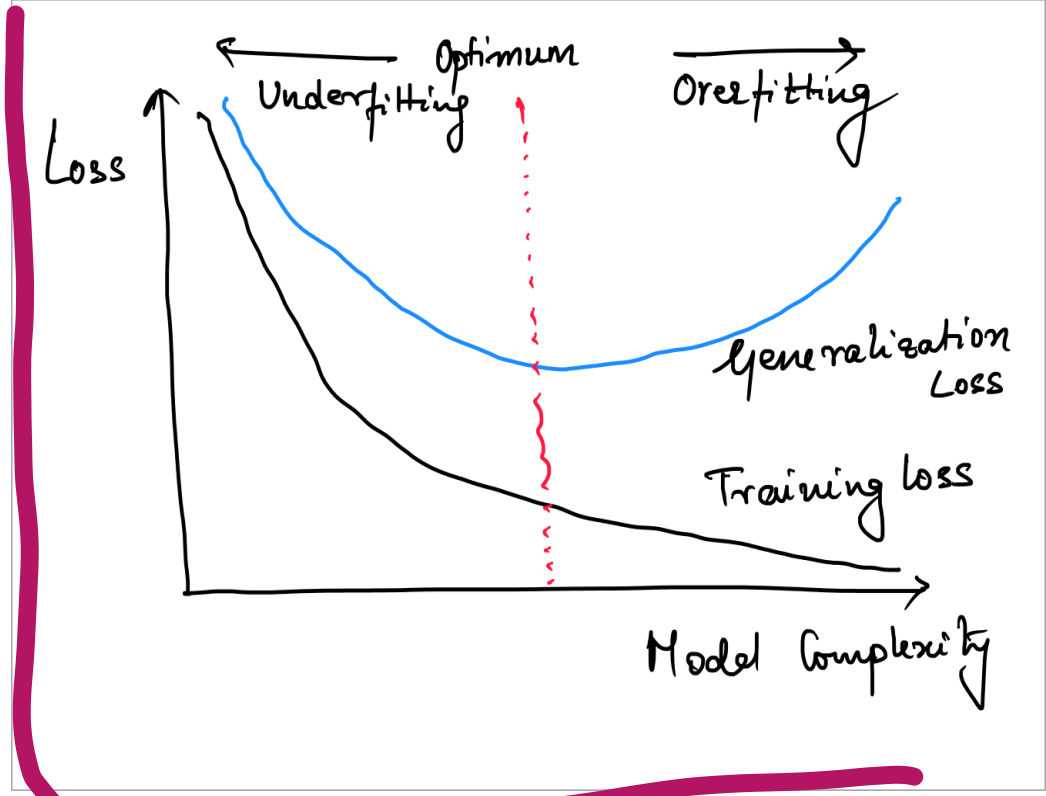


Underfitting and Overfitting

- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.

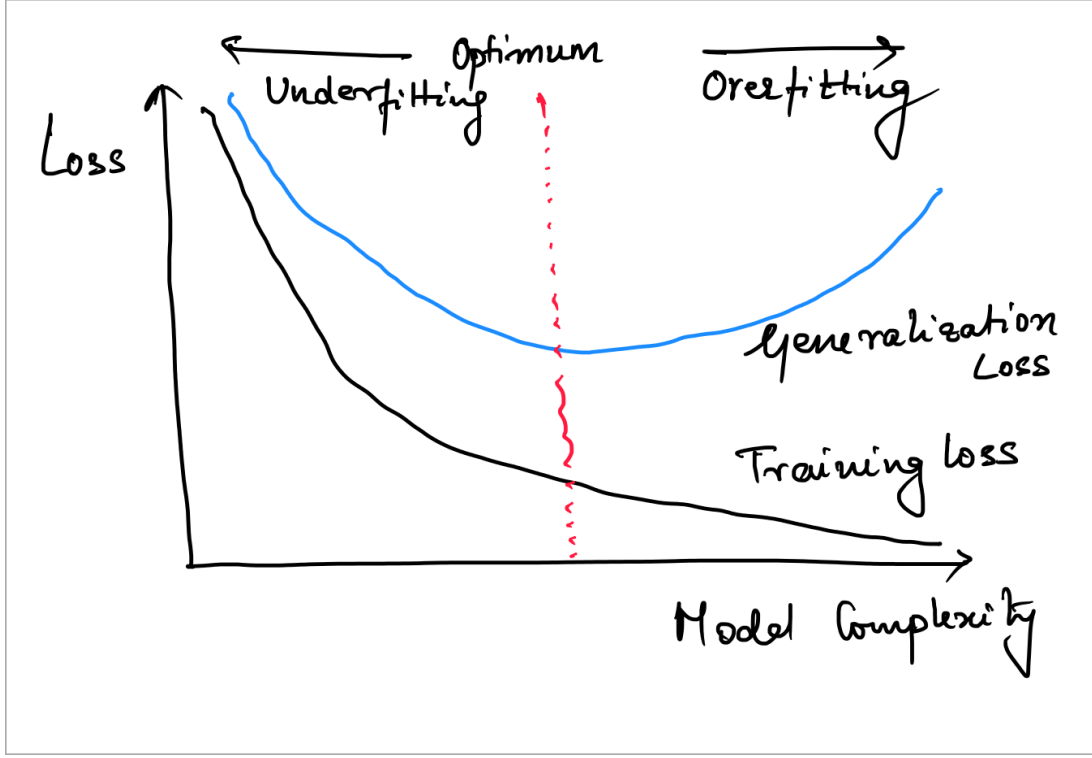
How does it depend on Model complexity?



What is model complexity?

- depends on context,
- number of hyper-parameters (tunable parameters),
- number of layers, hidden nodes in each layer,
- number of weights, range of values taken by weights,
- activation functions, choice of activation functions….

Underfitting and Overfitting



- Under fitting: model is unable to reduce training errors.

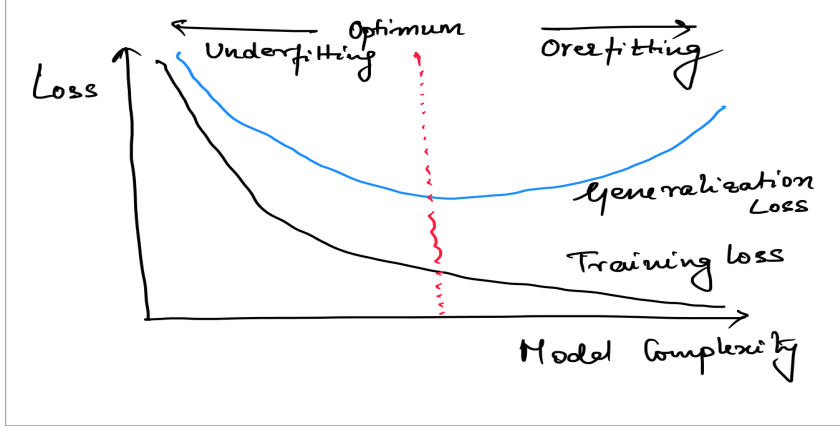
- Overfitting: model test error is significantly higher than training error.

How does it depend on Model complexity?

Over/under fitting depends on:

- Model complexity:
  - Model too simple→ underfitting (large rich dataset, not enough weights)
  - Model too complex→ overfitting (large amount of weights, range of weights and training data small/ less rich.
- quality and quantity of training data set:
  - large and rich training data set→ higher probability of estimating underlying distribution.

Underfitting and Overfitting



- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.

How does it depend on Model complexity?

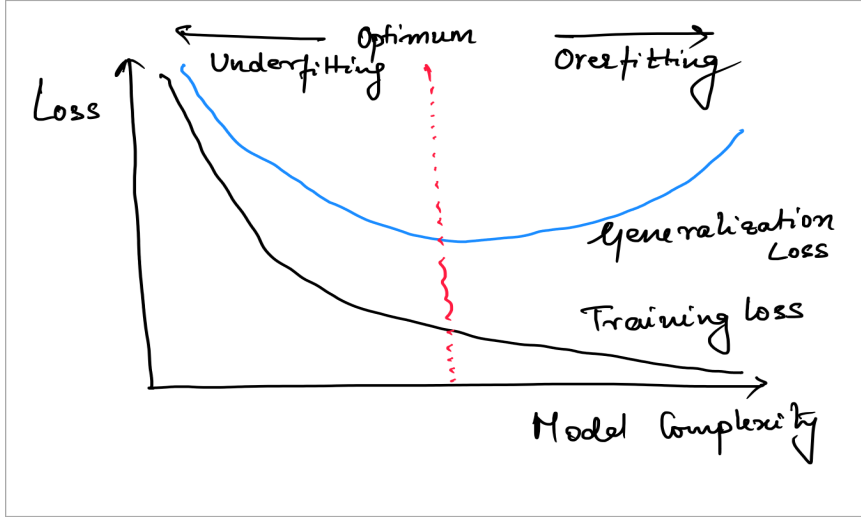
Over/under fitting depends on:

- Model complexity:
  - Model too simple→ underfitting (large rich dataset, not enough weights)
  - Model too complex→ overfitting (large amount of weights, range of weights and training data small/ less rich.
- quality and quantity of training data set:
  - large and rich training data set→ higher probability of estimating underlying distribution.

Best practice:

- consider large and diverse training data set + sufficiently complex deep NN model
- observe the training loss and test loss
- fine tune the model (tune hyper parameters, add/remove weights) to achieve minimal under/overfitting.

Underfitting and Overfitting



- Under fitting: model is unable to reduce training errors.
- Overfitting: model test error is significantly higher than training error.

How does it depend on Model complexity?

Over/under fitting depends on:

- Model complexity:
  - Model too simple→ underfitting (large rich dataset, not enough weights)
  - Model too complex→ overfitting (large amount of weights, range of weights and training data small/ less rich.
- quality and quantity of training data set:
  - large and rich training data set→ higher probability of estimating underlying distribution.

Best practice:

- consider large and diverse training data set + sufficiently complex deep NN model
- observe the training loss and test loss
- fine tune the model (tune hyper parameters, add/remove weights) to achieve minimal under/overfitting.

Rule of thumb:

The simplest model which explains the majority of the data is usually the best

Generalization : Preventing over-fitting (over-training)

Goal: To achieve good generalization accuracy on new examples/cases

How to ensure that a network has been well trained??

1. **Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.**
   - Monitor error on the test set as network trains.
   - Stop network training just prior to over-fit error occurring - *early stopping* or tuning

2. **Number of effective weights is reduced : Number of weights and value range.**

Generalization : Preventing over-fitting (over-training)

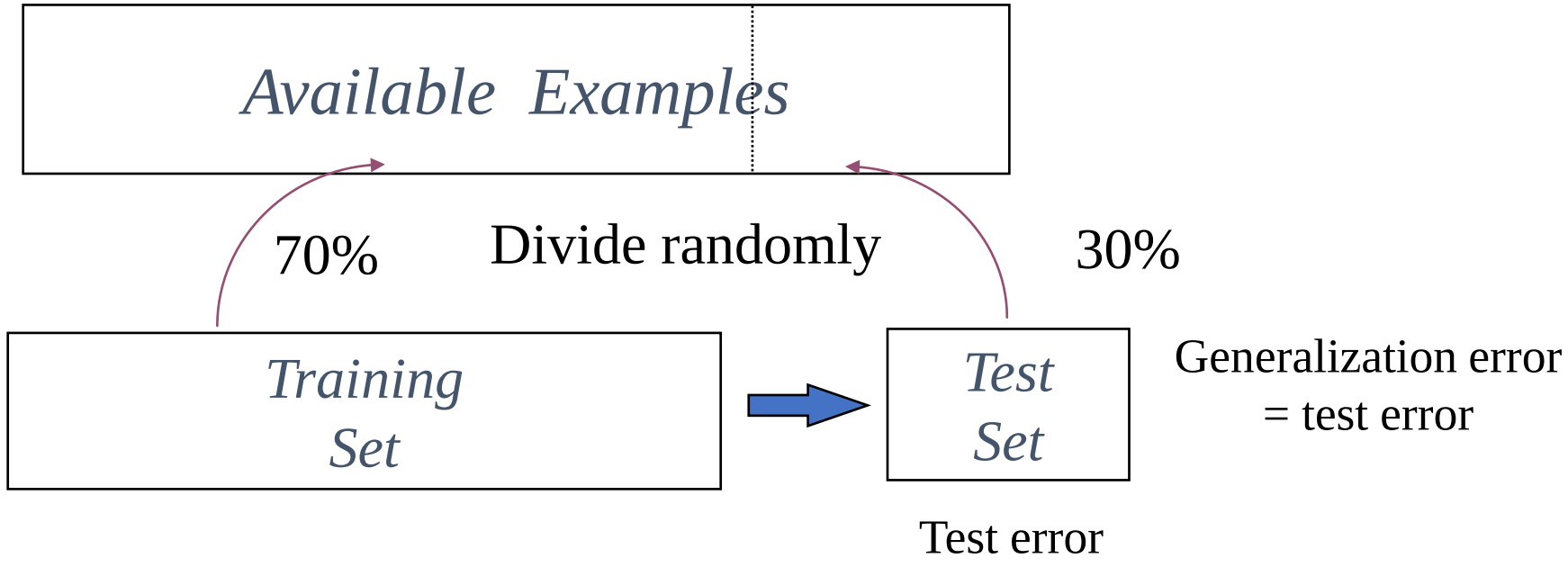
Goal: To achieve good generalization accuracy on new examples/cases

How to ensure that a network has been well trained??

1. **Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.**

When amount of data is large

Training Set 70% | Validation Set 15% | Test Set 15%



Generalization : Preventing over-fitting (over-training)

Goal: To achieve good generalization accuracy on new examples/cases

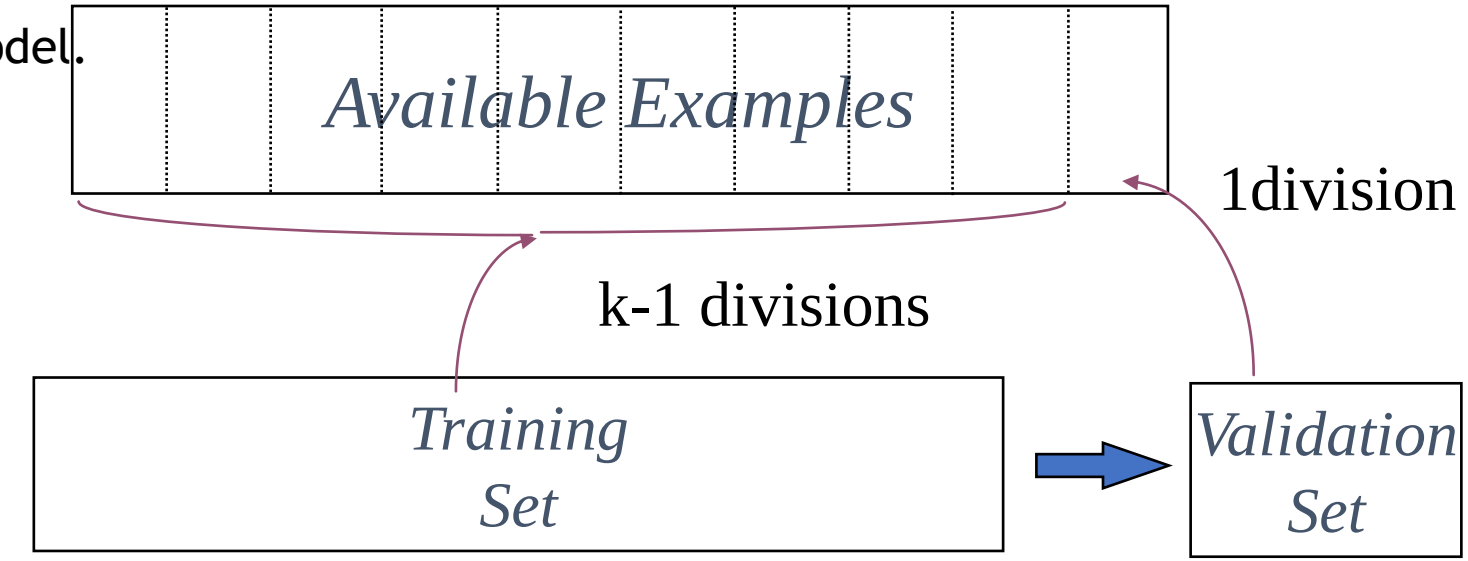
How to ensure that a network has been well trained??

1.**Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.**

When amount of data is small: Cross-Validation (K-fold)

- original training data set is split into K noncoincident sub-data sets
- use the K -1 sub-data set to train the model.
- validate the model using a sub-data set
- Repeat model training and validation

process *k* times.



Generalization : Preventing over-fitting (over-training)

2. **How to control number of effective weights?**

- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods
  - Adjust the bp error function to penalize the growth of unnecessary weights
  - Keep the weight vector small magnitude ➔add its value as a penalty to the problem of minimizing the loss.

Generalization : Preventing over-fitting (over-training)

2. **How to control number of effective weights?**

- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods
  - Adjust the bp error function to penalize the growth of unnecessary weights
  - Keep the weight vector small magnitude ➔add its value as a penalty to the problem of minimizing the loss.
    - Weight vector becomes too large, ➔ the learning algorithm prioritizes minimizing ***w*** over minimizing the training error.

$$\ell(w, b) + \frac{\lambda}{2} \|w\|^2$$

Generalization : Preventing over-fitting (over-training)

2. How to control number of effective weights?

- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods
  - Adjust the bp error function to penalize the growth of unnecessary weights
  - Keep the weight vector small magnitude ➔add its value as a penalty to the problem of minimizing the loss.
    - Weight vector becomes too large, ➔ the learning algorithm prioritizes minimizing w over minimizing the training error.

$$\ell(w, b) + \frac{\lambda}{2} \|w\|^2$$

  - Squared Norm Regularization:

$$\|x\|^2 = \sum_{i=1}^{n} x_i^2$$

  - Gradient Descent update becomes :

$$w \leftarrow w(1 - \alpha\lambda) - \alpha \frac{\partial J}{\partial w}$$

Weights decay by an amount proportional to its magnitude

λ weight-cost parameter

another *Hyperparameter*

Generalization : Preventing over-fitting (over-training)

2. **How to control number of effective weights?**

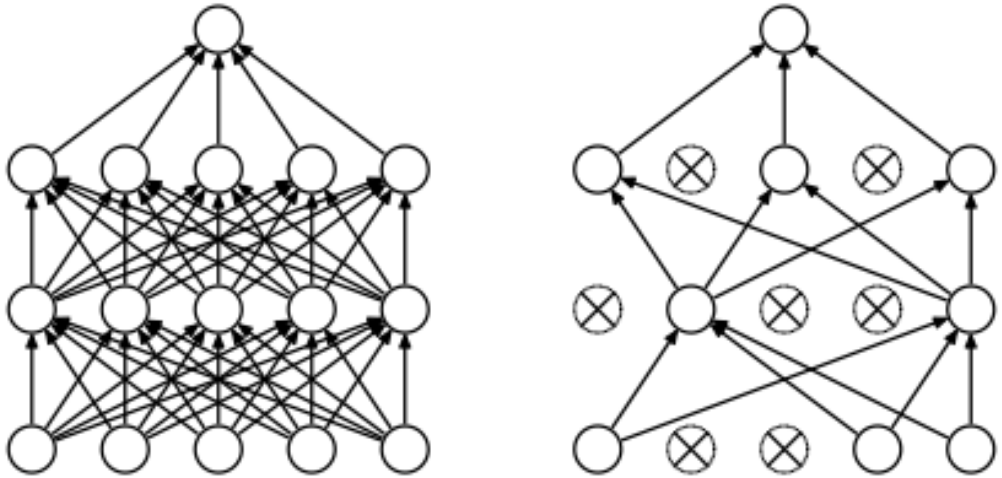
- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods

- Dropout

Srivastava et al. 2014.

Shown : Dropout = Regularized NN.



(a) Standard Neural Net (b) After applying dropout.

Training

1. Network Design (Architecture of NN networks.) #layers, #hidden nodes, activation functions, model ..
2. Initialize model parameters.
3. Choose Loss function
4. Training and Backpropagation : Mini batch, batch, or stochastic GD.
5. Monitor the loss function and error .

When no overfitting observed (epochs of training)

- Stop if the error fails to improve (has reached a minimum)
- Stop if the rate of improvement drops below a certain level
- Stop if the error reaches an acceptable level
- Stop when a certain number of epochs have passed

When overfitting observed: fine tune the NN network

(initialize parameters, prune or regularize the weights, …)

Training

1. Network Design (Architecture of NN networks.) #layers, #hidden nodes, activation functions, model ..
2. Initialize model parameters. **How?**
3. Choose Loss function
4. Training and Backpropagation : Mini batch, batch, or stochastic GD.
5. Monitor the loss function and error .

When no overfitting observed (epochs of training)

- Stop if the error fails to improve (has reached a minimum)
- Stop if the rate of improvement drops below a certain level
- Stop if the error reaches an acceptable level
- Stop when a certain number of epochs have passed

When overfitting observed: fine tune the NN network

(initialize parameters, prune or regularize the weights, …)

How to initialize the parameters

- Backprop involves successive multiplication of weight gradients of each layer➔ multiplication of gradient matrices.
- They might be small, they might be large, their product (millions of layers) ➔ very large or very small.
- Optimization is bound to fail: either very very large or excessively small!

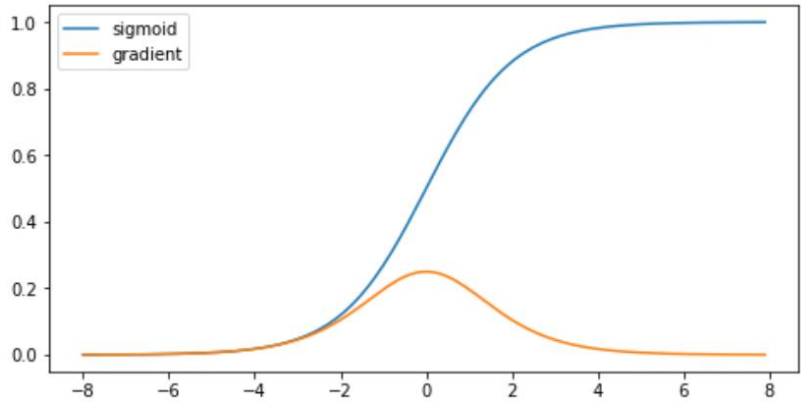
Exploding Gradient problem: when gradients matrix product becomes excessively large.

Vanishing gradient problem: when chain product becomes infinitesimally small.

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

Intuition:

- As we can see, the gradient of the sigmoid vanishes for very large or very small arguments.
- the gradients of the overall product may vanish!!



How to initialize the parameters

Rule of thumb ➔Initialization :

Random initialization from Gaussian distribution

zero mean and standard deviation $= \frac{2}{\sqrt{n}}$

n= number of input layers.

Intuition: consider simplistic Neural network

Remark: Therefore, best practice is to choose number of layers = power of 2 or, 2^n works for several hundreds of layers!!

$$H^{t+1} = f_t(H^t)$$

$$O = f_d \circ f_{d-1} \circ \cdots f_1(X)$$

$$\partial W_t O = \partial_{H^{d-1}} H^d \cdots \partial_{H^t} H^{t+1} \partial_{W_t} H^t$$

$$H_i = \sum_{j=1}^{n_{in}} W_{ij} x_{ij}$$

Mean and Variance of H_i ?

How to initialize the parameters

Xavier Initialization(Xavier et Benjio 2010):

Thus, this factor should be controlled: $n_{in}\,\sigma^2 = 1$

$$\mathbb{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2 x_j^2]$$

$$= \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2]\,\mathbb{E}[x_j^2]$$

$$= n_{in} \cdot \sigma^2 \gamma^2$$

How to initialize the parameters

Xavier Initialization(Xavier et Benjio 2010):

Thus, this factor should be controlled: $n_{in}\,\sigma^2 = 1$

But consider backprop too: $n_{out}\,\sigma^2 = 1$.

Can not be done simultaneously

so,

$$\mathbb{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2 x_j^2]$$

$$= \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2]\,\mathbb{E}[x_j^2]$$

$$= n_{in} \cdot \sigma^2 \gamma^2$$

How to initialize the parameters

Xavier Initialization(Xavier et Benjio 2010):

$$\mathbb{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2 x_j^2]$$

$$= \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2]\,\mathbb{E}[x_j^2]$$

$$= n_{in} \cdot \sigma^2 \gamma^2$$

Thus, this factor should be controlled: $n_{in}\,\sigma^2 = 1$

But consider backprop too: $n_{out}\,\sigma^2 = 1$.

Can not be done simultaneously so,

$$\frac{1}{2}(n_{in} + n_{out})\,\sigma^2 = 1$$

⟸ Drawn from Gaussian

If drawn from uniform distribution: For random variables $U[-a, a]$

$$Var = a^2/3$$

Thus, initialization

Initialization of weights of a layer :

$$U\left[-\sqrt{6/(n_{in}+n_{out})},\ \sqrt{6/(n_{in}+n_{out})}\right)$$

How to initialize the parameters (scratching the surface)

Xavier Initialization(Xavier et Benjio 2010):

$$\mathbb{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2 x_j^2]$$

$$= \sum_{j=1}^{n_{in}} \mathbb{E}[w_{ij}^2]\,\mathbb{E}[x_j^2]$$

$$= n_{in} \cdot \sigma^2 \gamma^2$$

Thus, this factor should be controlled: $n_{in}\,\sigma^2 = 1$

But consider backprop too: $n_{out}\,\sigma^2 = 1$.

Can not be done simultaneously so,

$$\frac{1}{2}(n_{in} + n_{out})\,\sigma^2 = 1$$

⟸ Drawn from Gaussian

If drawn from uniform distribution: For random variables $U[-a, a]$

$$Var = a^2/3$$

Thus, initialization

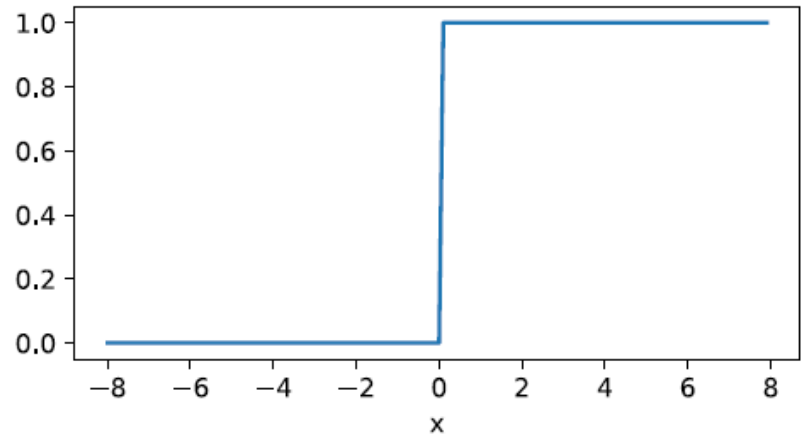
Initialization of weights of a layer :

$$U\left[-\sqrt{6/(n_{in}+n_{out})},\ \sqrt{6/(n_{in}+n_{out})}\right)$$

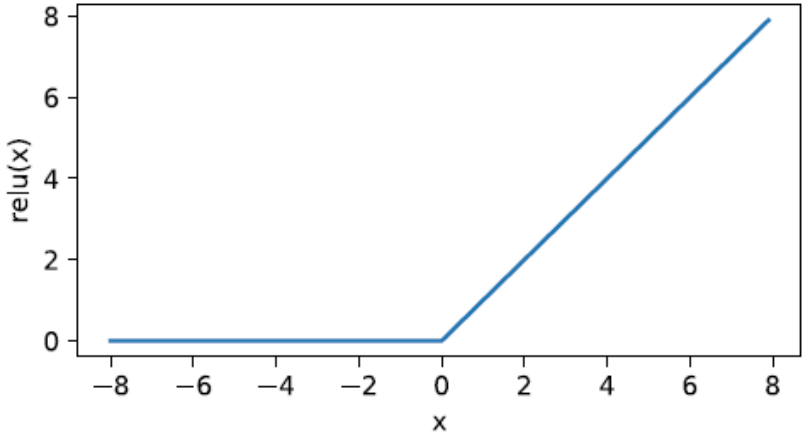
Types of Activation functions

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

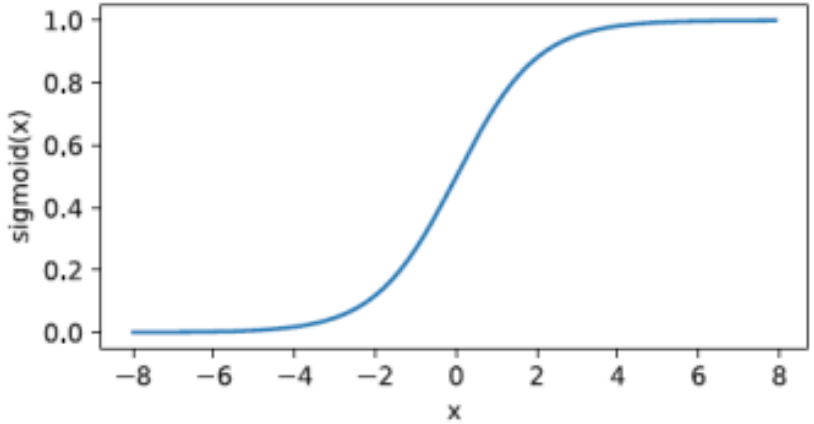
Activation functions



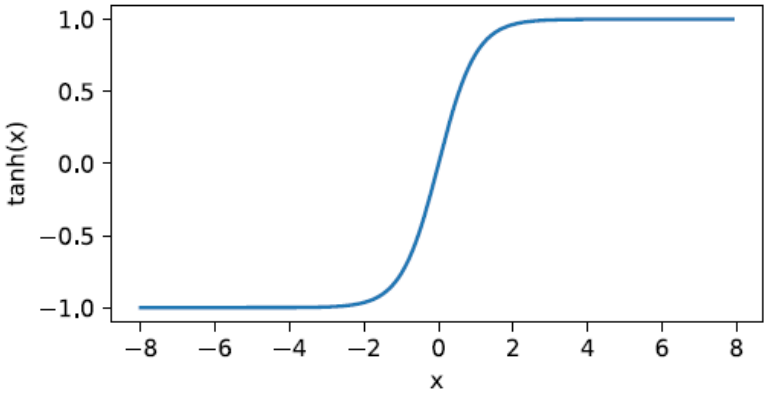
Threshold function (binary step function)



ReLu (Rectified Linear Unit)

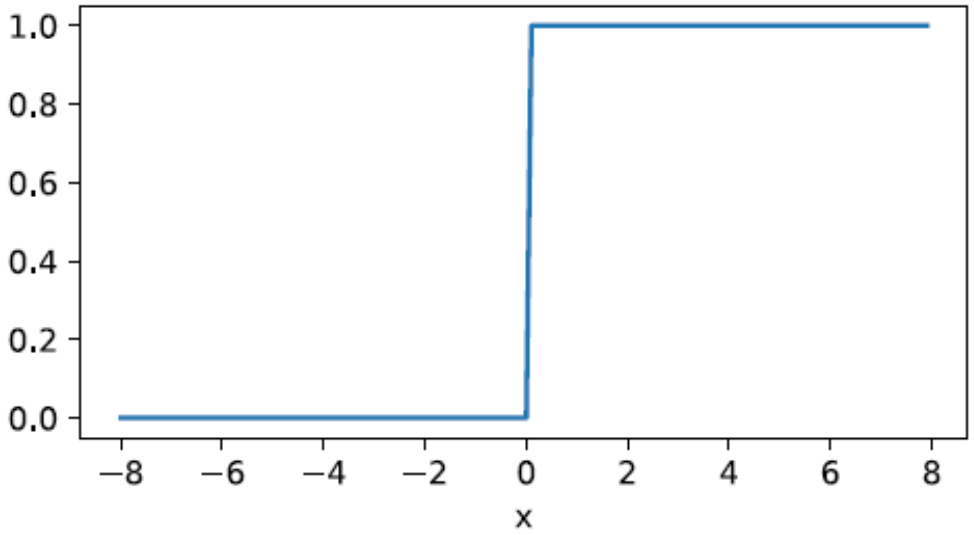


Sigmoid function



TanH / Hyperbolic Tangent

Activation Functions



$$\phi(x) = \begin{cases} 1 \ if & x \geq 0 \\ 0 \ if & x < 0 \end{cases}$$

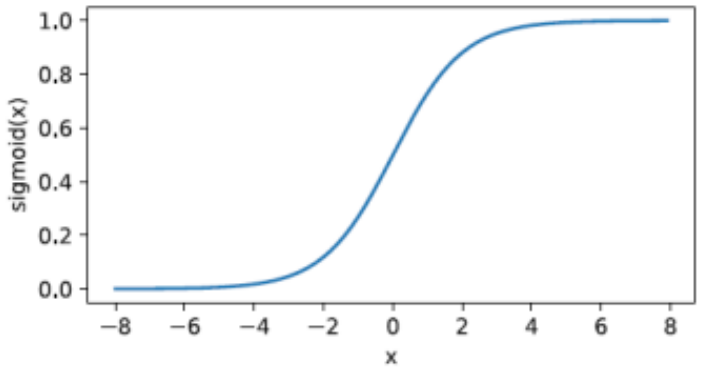
Threshold function (binary step function)

- If the input value is above or below a certain threshold,

the neuron is activated and sends the same signal to the next layer.

- Good for Binary outputs → 2 class classifications.
- Does NOT allow multi value outputs → does not support classification of input into multiple categories.

Activation Functions : Non-linear functions (why linear functions not preferred?



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

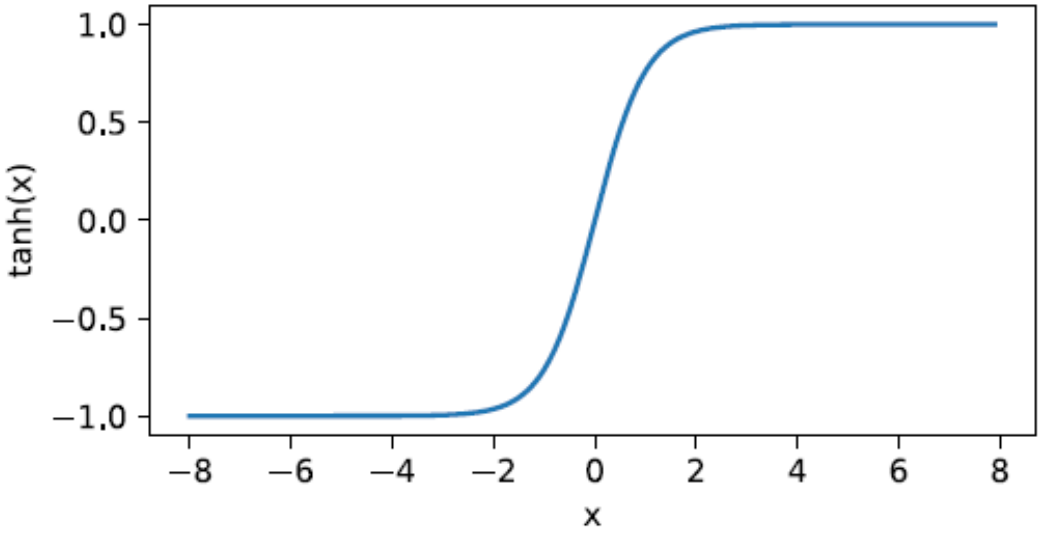
Sigmoid function

- Smooth gradient, preventing “jumps” in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.
- The Sigmoid function used for **binary classification** in logistic regression model.
- While creating artificial neurons sigmoid function used as the **activation function**.

Disadvantages

- Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem.
- This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
- Computationally expensive
- Not Zero centered !!

Activation Functions



$$\phi(x) = \begin{cases} \dfrac{1-e^{-2x}}{1+e^{-2x}} \end{cases}$$

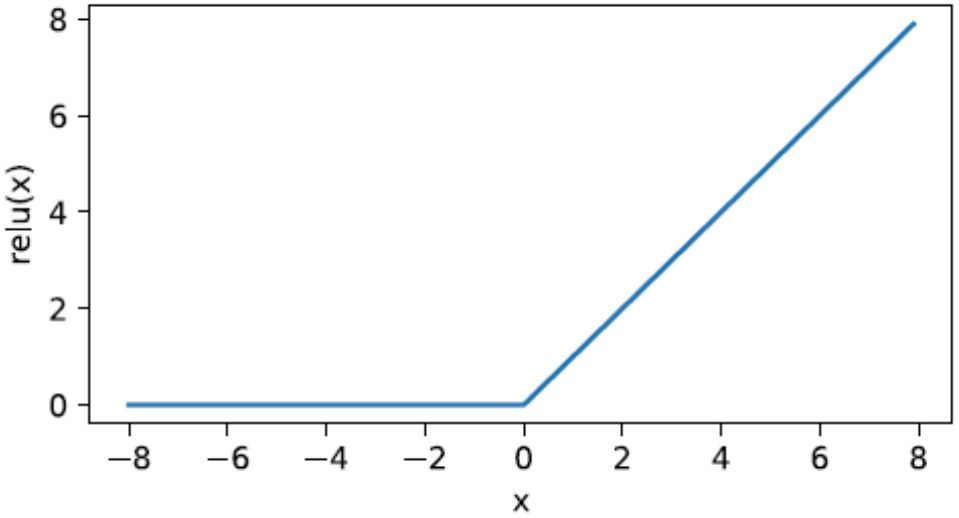
TanH / Hyperbolic Tangent

Zero centred ➔making it easier to model inputs that have strongly negative, neutral, and strongly positive values.

All advantages of Sigmoid function preserved.

Computationally expensive.

Activation Functions



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

ReLu (Rectified Linear Unit)

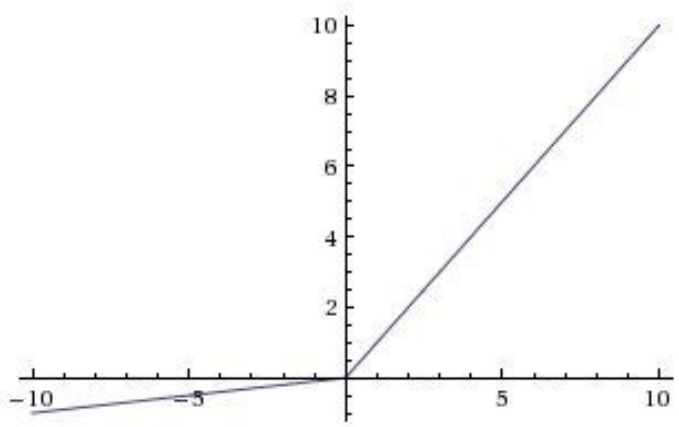
- **Computationally efficient**—allows the network to converge very quickly
- **Non-linear—**although it looks like a linear function, ReLU has a derivative function and allows for backpropagation.
- Avoids vanishing or exploding gradient problems unless…

Disadvantages:

The Dying ReLU problem—when inputs approach zero, or negative,

the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

Activation Functions



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

Leaky ReLu

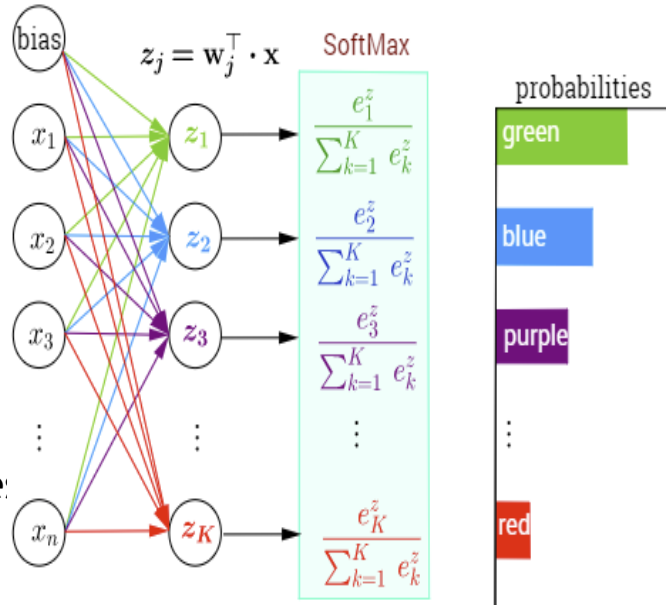
- **Computationally efficient**—allows the network to converge very quickly (faster than Sigmoid/tanh)
- **Does not Saturate/**
- **Does not "die"**

[Mass et al., 2013] [He et al., 2015]

Activation function

Softmax function

$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3....k$$



- Calculates the probabilities distribution of the event over 'n' different events.
- In general, calculates the probabilities of each target class over all possible target classe
- Later the calculated probabilities will be helpful for determining the target class for the given inputs.
- The range will **0 to 1**, and the sum of all the probabilities will be **equal to one.**

Remark: Useful for output neurons—typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

- Very often used for multi-class classification.

softmax : à la derniere etape uniquement (pour class classification)
tanh, thresow, sigm : partout
relu : jamais à la dernière etape

Loss functions

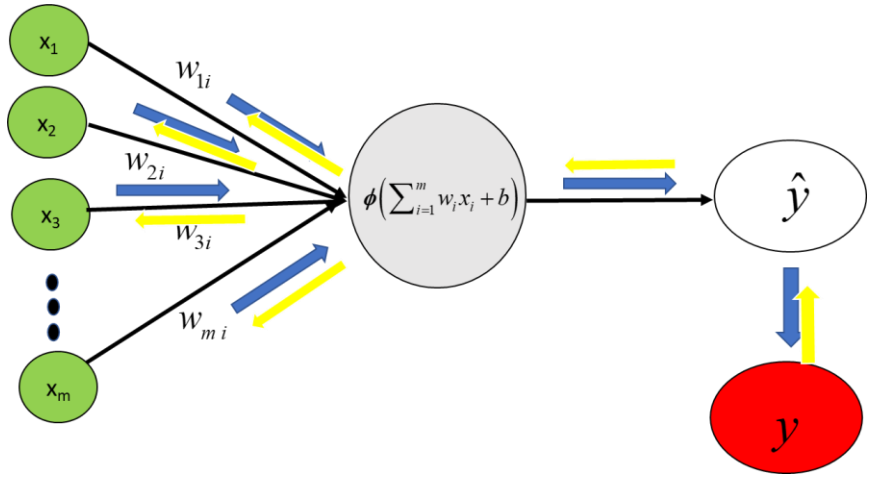
JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Common Loss functions

Regression

Mean Square Error (MSE) Loss: measured as the average of squared difference between predictions and actual observations.

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$$



Also known as: L2 loss, Quadratic loss, MSE loss, ..

Remarks:

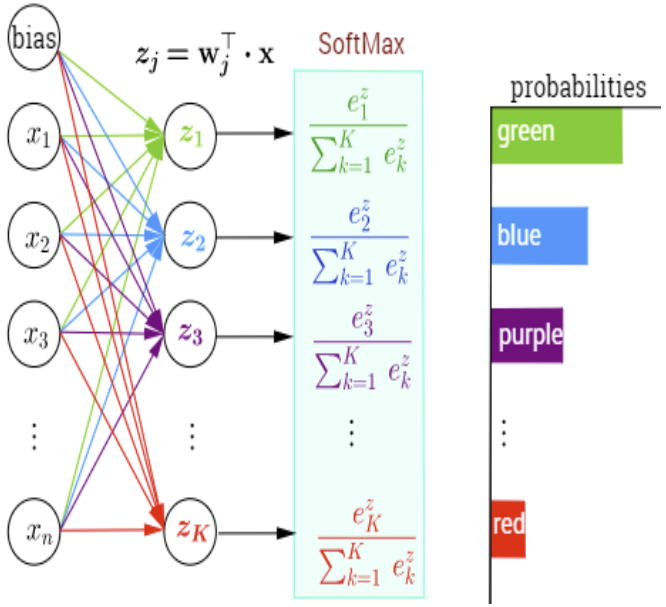
- Predicted values that are far from actual values are penalized heavily.
- Squaring : positivity, quadratic function→ nice properties helpful in finding gradients.

Common Loss functions

Classification (recall: binary classification and multi class classification

Softmax function)

- Often, for classification: outputs are probabilities of belonging to each class.
- Thus, loss must be calculated based on assessment of probabilities.



$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3....k$$

Common Loss functions

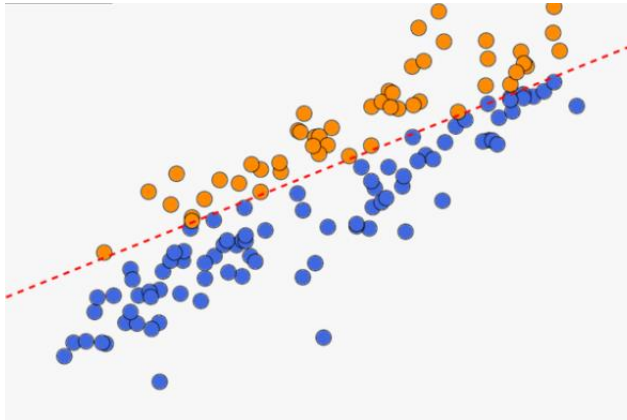
Classification Loss (recall: binary classification and multi class classification

Softmax function)

Cross Entropy Loss (log loss, logistic loss, logarithmic loss, negative log loss..)

(Binary Class , or 2 classes)

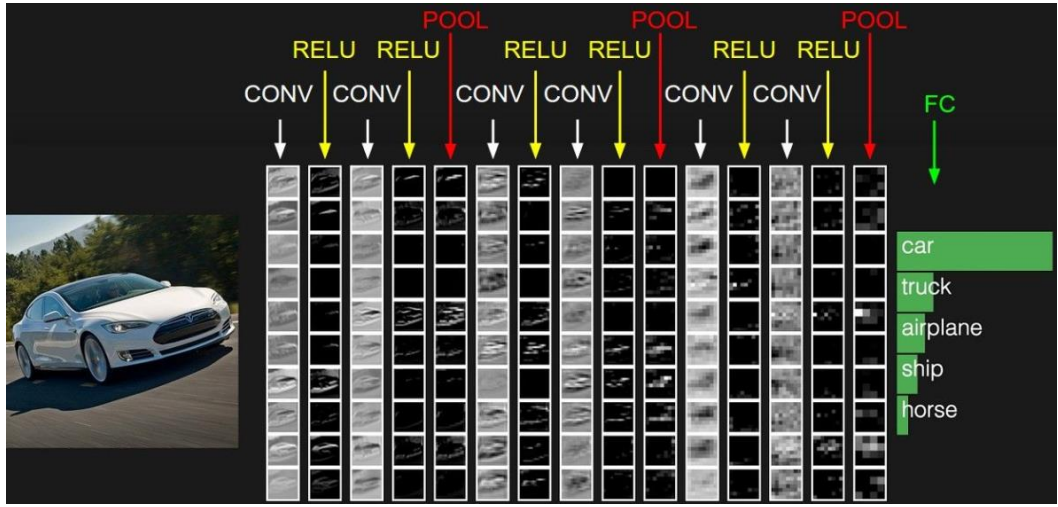
$$L_{CE} = -\left(y \log(\hat{p}) + (1-y) \log(1-\hat{p}) \right)$$



- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
- Cross-entropy loss increases as the predicted probability diverges from the actual label.
- Notice that when actual label is 1 (y = 1), second half of function disappears whereas in case actual label is 0 (y = 0) first half is dropped off.
- A perfect model would have a log loss of 0.

Common Loss functions

Classification Loss (multi class classification, Softmax function)



Cross Entropy Loss

(Multi Class)

$$L_{CE} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

M : Number of classes

$y_{i,c}$ *:* true probability of belonging to that class

$p_{i,c}$: predicted probability of belonging to that class.

- Cross-entropy can be calculated for multiple-class classification.
- The classes have been **one hot encoded**, meaning that there is a binary feature for each class value.

Common Loss functions

Classification Loss (multi class classification, Softmax function)

Cross Entropy Loss

(Multi Class)

$$L_{CE} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

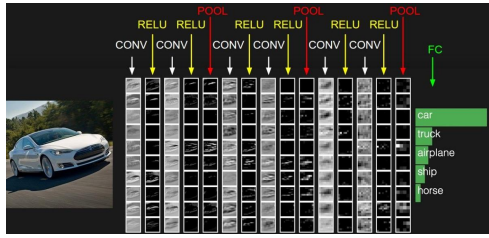
Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Source: DeepAI

- Cross-entropy can be calculated for multiple-class classification.
- The classes have been **one hot encoded**, meaning that there is a binary feature for each class value.

One hot Coding: Converting Labels (categorical, non numeric feautres) into ➔ numeric features without any cardinal (ordering) values.

Common Loss functions



Classification Loss (multi class classification, Softmax function)

Cross Entropy Loss

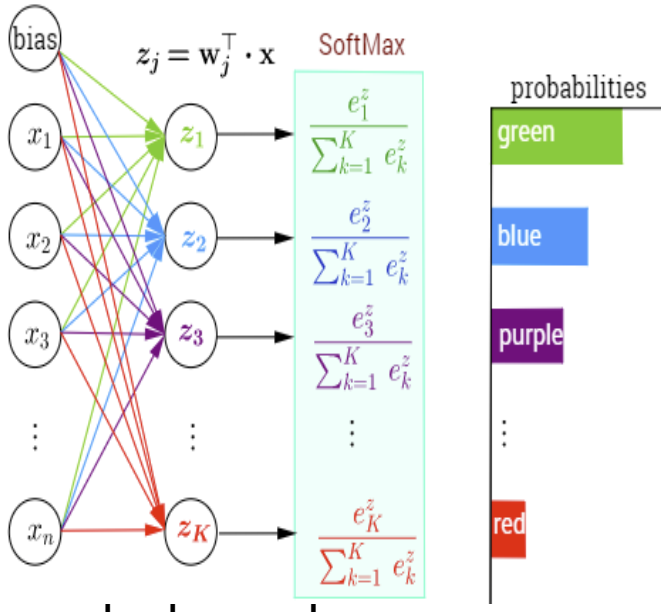
(Multi Class)

$$L_{CE} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

M : Number of classes

$y_{i,c}$ *:* true probability of belonging to that class

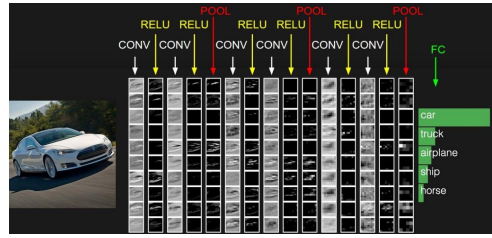
$p_{i,c}$: predicted probability of belonging to that class.



- Cross-entropy can be calculated for multiple-class classification.
- The classes have been **one hot encoded**, meaning that there is a binary feature for each class value.
- The predictions must have predicted probabilities for each of the classes (Example: Softmax).
- The cross-entropy is **then summed across each binary feature** and **averaged across all examples** in the dataset.

$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3....k$$

Common Loss functions



Classification Loss (multi class classification, Softmax function)

Cross Entropy Loss

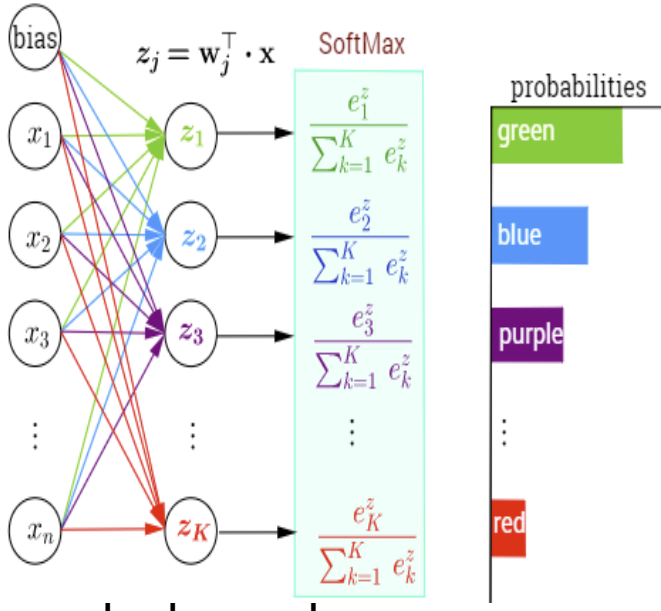
(Multi Class)

$$L_{CE} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

M : Number of classes

$y_{i,c}$: true probability of belonging to that class

$p_{i,c}$: predicted probability of belonging to that class.



- Cross-entropy can be calculated for multiple-class classification.
- The classes have been **one hot encoded**, meaning that there is a binary feature for each class value.
- The predictions must have predicted probabilities for each of the classes (Example: Softmax).
- The cross-entropy is **then summed across each binary feature** and averaged across all examples in the dataset.

$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3....k$$

Suggestion: Read this thread of discussion on forum on using Cross entropy in practice.

Loss functions: Best practices

Regression Problem

- A problem where you predict a real-value quantity.
- **Output Layer Configuration**: One node with a linear activation unit.
- **Loss Function**: Mean Squared Error (MSE).

Binary Classification Problem

- A problem where you classify an example as belonging to one of two classes.
- The problem is framed as predicting the likelihood of an example belonging to class one, e.g. the class that you assign the integer value 1, whereas the other class is assigned the value 0.
- **Output Layer Configuration**: One node with a sigmoid activation unit.
- **Loss Function**: Cross-Entropy

Multi-Class Classification Problem

- A problem where you classify an example as belonging to one of more than two classes.
- The problem is framed as predicting the likelihood of an example belonging to each class.
- **Output Layer Configuration**: One node for each class using the softmax activation function.
- **Loss Function**: Cross-Entropy.

Summary

- Simple NN functioning, analogy with linear regressions
- Feed foreword Deep NN functioning
- Weight updates through backprop and gradient descent (batch, mini batch and stochastic GD)
- Generalization :Training /validation/test set
- Generalization and Training issues: overfitting, underfitting, finding the right tradeoff.
- Weights initializations: Exploding and Vanishing gradients, Xavier initilisations.
- Note on Activation functions.

References

Kattan, Ali, and Rosni Abdullah. "Training of feed-forward neural networks for pattern-classification applications using music inspired algorithm." *International Journal of Computer Science and Information Security* 9.11 (2011): 44.

Taig, Efrat, and Ohad Ben-Shahar. "Gradient Surfing: A New Deterministic Approach for Low-Dimensional Global Optimization." *Journal of Optimization Theory and Applications* 180.3 (2019): 855-878.

Guest, D., Cranmer, K., & Whiteson, D. (2018). Deep learning and its application to LHC physics. *Annual Review of Nuclear and Particle Science*, *68*, 161-181.

Sirunyan, A. M., Tumasyan, A., Adam, W., Ambrogi, F., Asilar, E., Bergauer, T., ... & Del Valle, A. E. (2019). Search for the Higgs boson decaying to two muons in proton-proton collisions at s= 13 TeV. *Physical review letters*, *122*(2), 021801.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). **Human-level control through deep reinforcement learning**. *Nature*, *518*(7540), 529.

Cully, Antoine, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. “Robots That Can Adapt like Animals.” ***Nature*** 521 (7553): 503.

Park, Y., & Kellis, M. (2015). Deep learning for regulatory genomics. *Nature biotechnology*, *33*(8), 825.

Gugulothu, N., TV, V., Malhotra, P., Vig, L., Agarwal, P., & Shroff, G. (2017). Predicting remaining useful life using time series embeddings based on recurrent neural networks. *arXiv preprint arXiv:1709.01073*.

Sutton, Richard S., and Andrew G. Barto. *Introduction to reinforcement learning*. Vol. 135. Cambridge: MIT press, 1998.