COMP/EECE 7/8740 Neural Networks

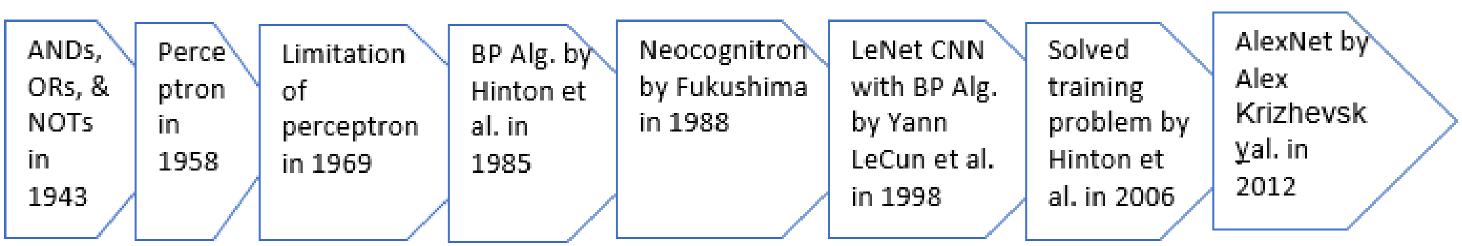
Topics:

- Neural Networks (NNs)
- Gradient descent with momentum
- Backpropagation Algorithm
- Batch learning and
- DL ecosystem

Md Zahangir Alom
Department of Computer Science
University of Memphis, TN

Neural Networks

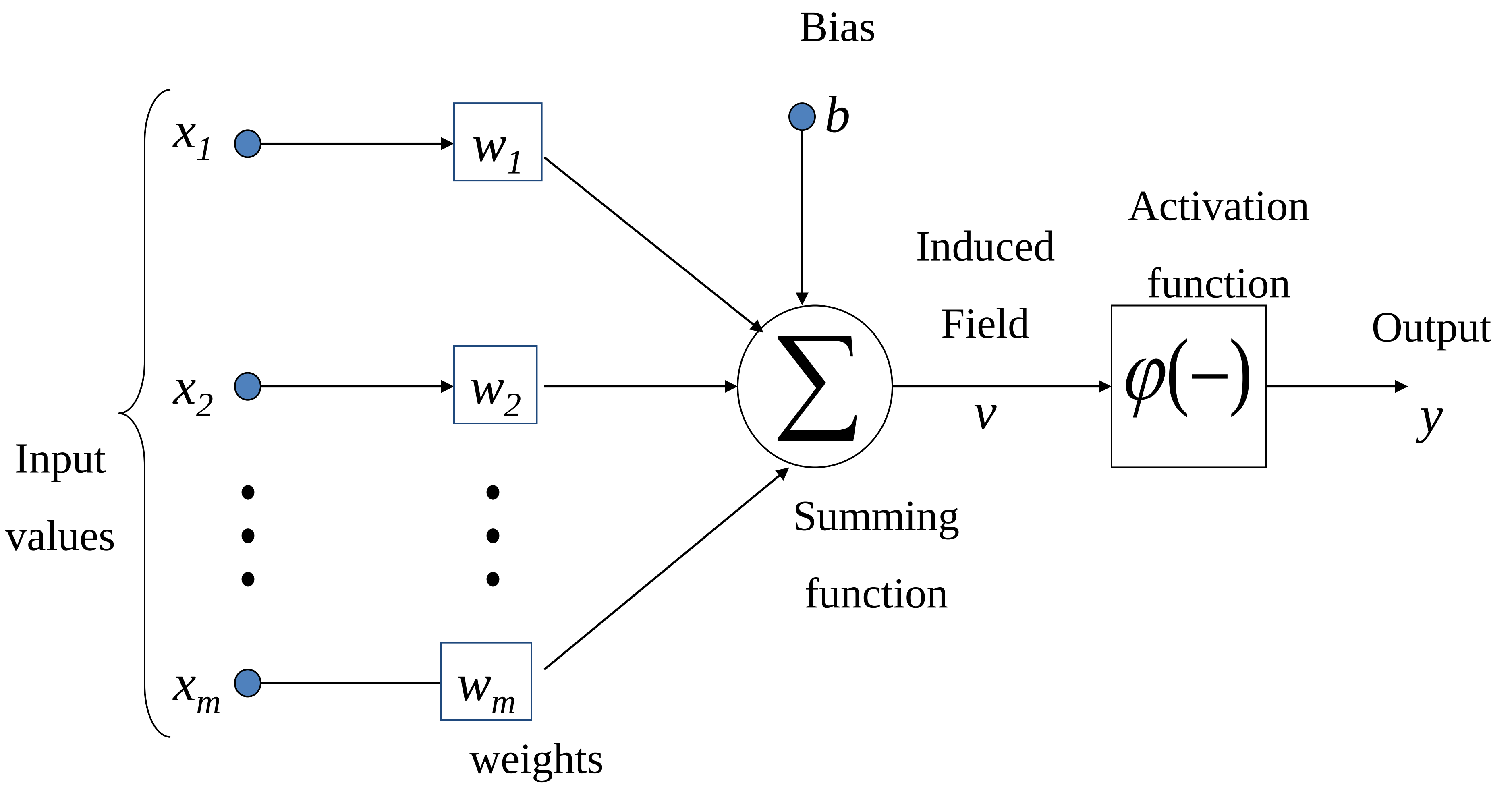
- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Neuron in ANNs tend to have fewer connections than biological neurons.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.



History of Neural Networks to Deep Learning

Alom, Md Zahangir, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul AS Awwal, and Vijayan K. Asari. "A state-of-the-art survey on deep learning theory and architectures." *Electronics* 8, no. 3 (2019): 292.

The Neuron Diagram



Mathematical model of a neuron

- The neuron is the basic information processing unit of a NN.
- It consists of:
  1. A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$
  2. An adder function (linear combiner) for computing the weighted sum of the inputs: (real numbers)

$$u = \sum_{j=1}^{m} W_j \, x_j$$

  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias.

$$\mathbf{y} = \varphi(\mathbf{u} + b)$$

Bias of a Neuron

- The bias (adjuster) ***b*** has the effect of applying a transformation to the weighted sum ***u***

$$v = u + b$$

- The bias is an external parameter of the neuron. It can be modeled by adding an extra input.

- ***v*** is called **induced field** of the neuron

$$u = \sum_{j=0}^{m} W_j \, x_j$$

$$W_0 = b$$

Neuron Models

- The choice of activation function φ determines the neuron model.

Examples:

- Step function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$

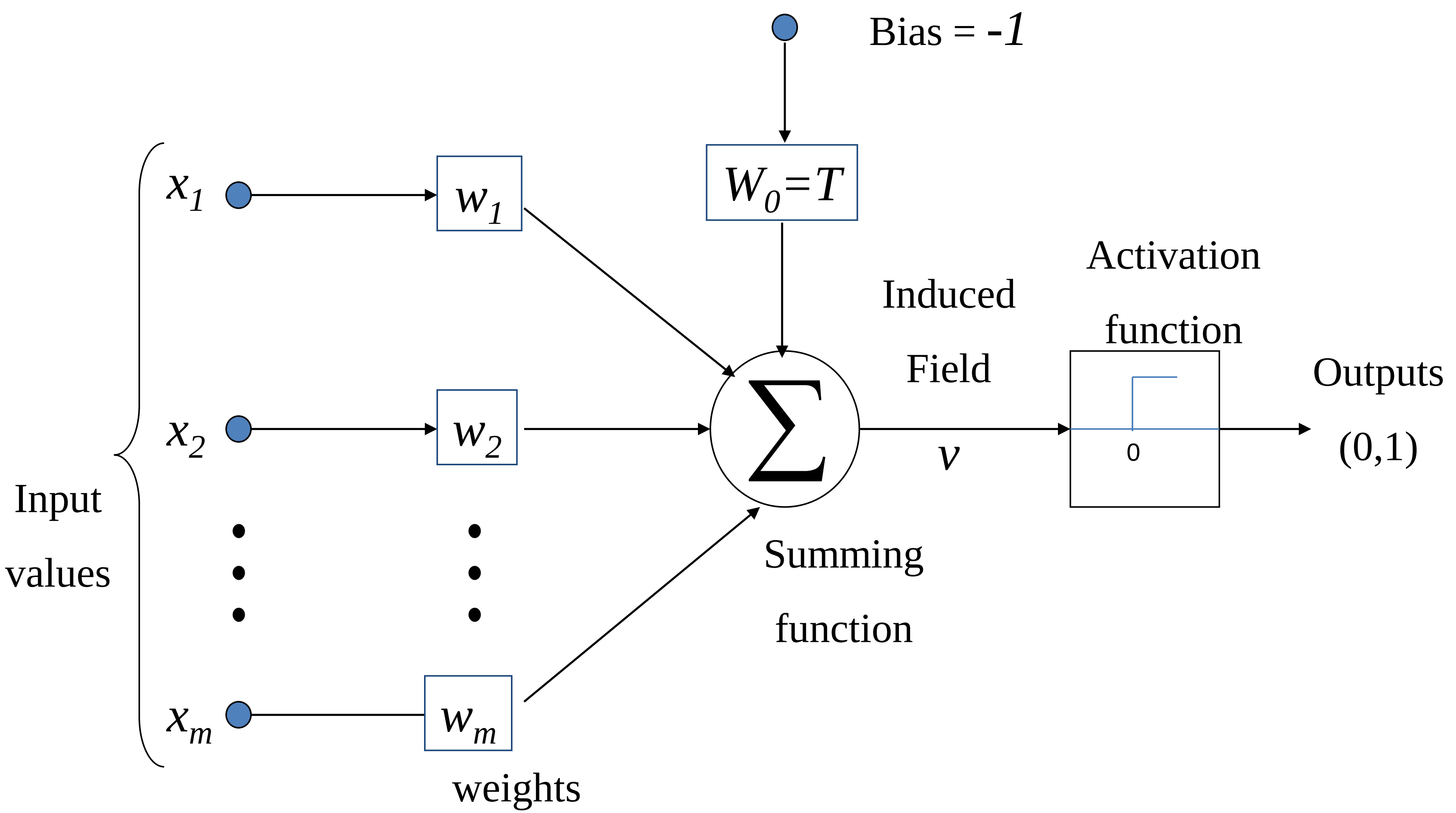
- Ramp function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v-c)(b-a)/(d-c)) & \text{otherwise} \end{cases}$$

- Sigmoid function with z, x, y parameters

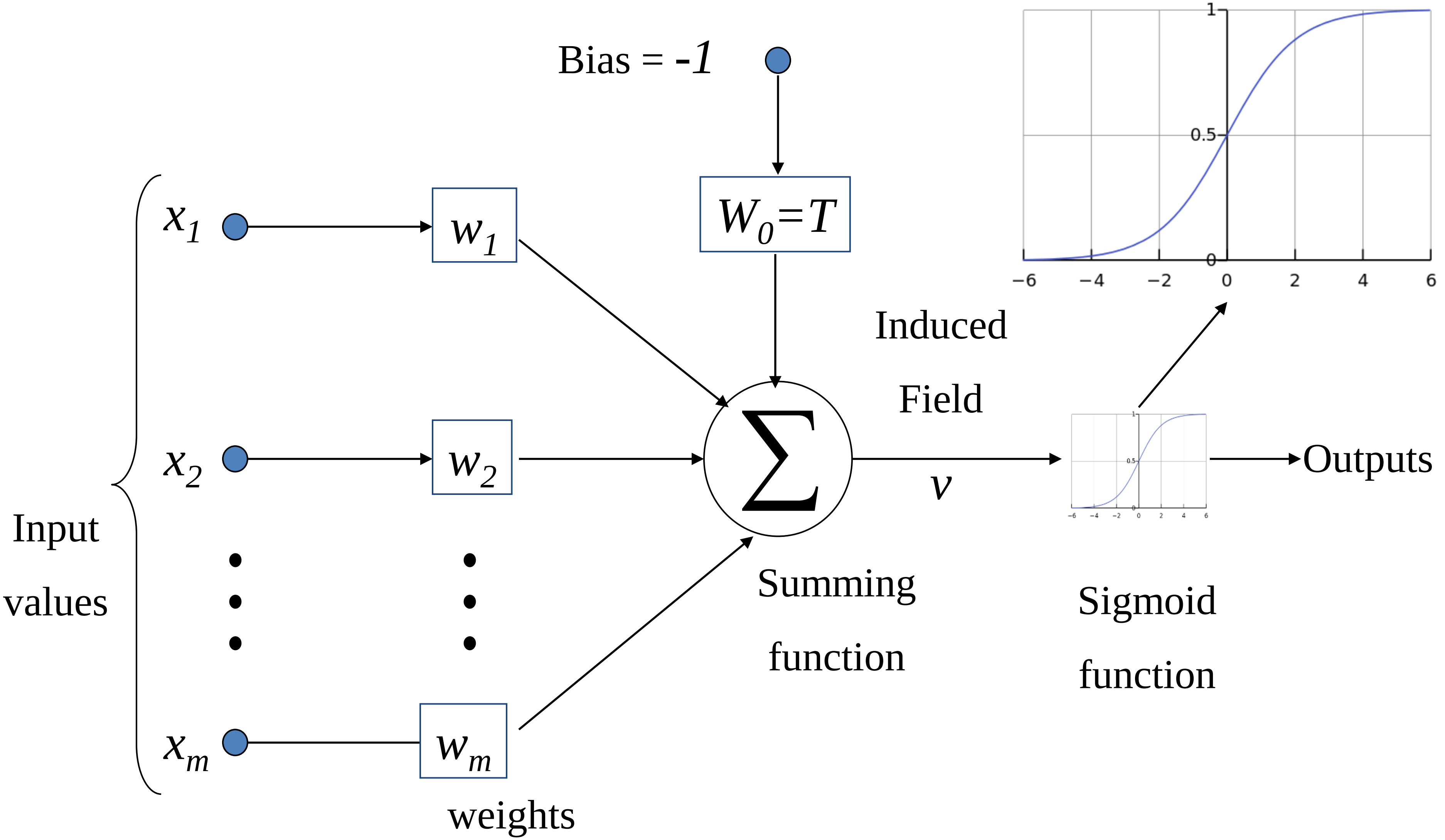
$$\varphi(v) = z + \frac{1}{1 + \exp(-xv + y)}$$

- Gaussian function: $$\varphi(v) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{v-\mu}{\sigma}\right)^2\right)$$

The Neuron Diagram with activation



The Neuron Diagram with activation

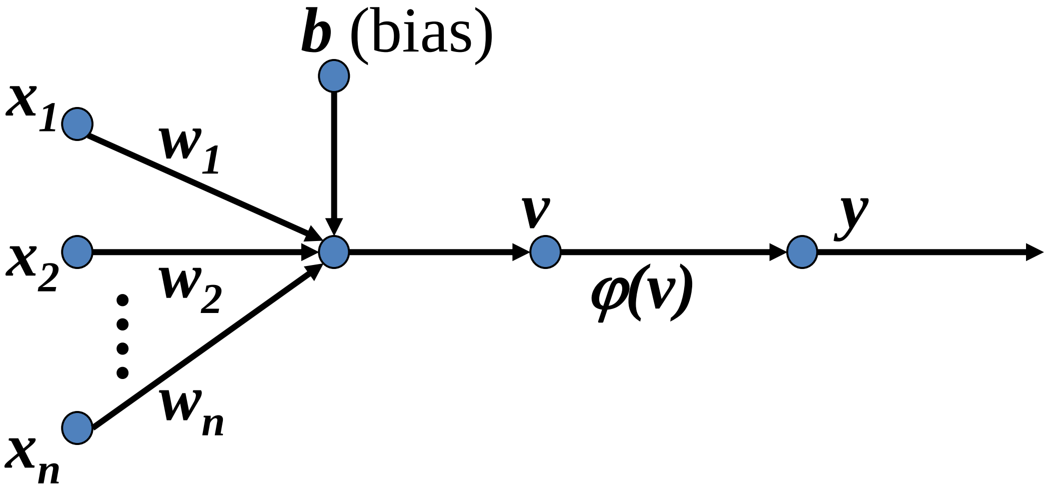


Perceptron: Neuron Model

(Special form of single layer feed forward)

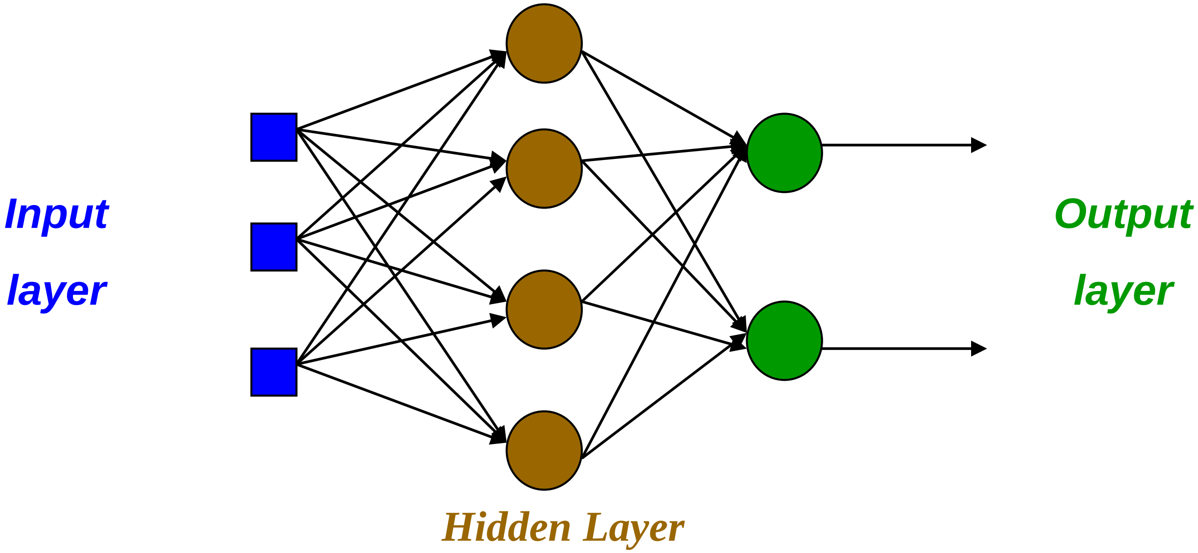
- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input $\geq$ 0 and -1 otherwise

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



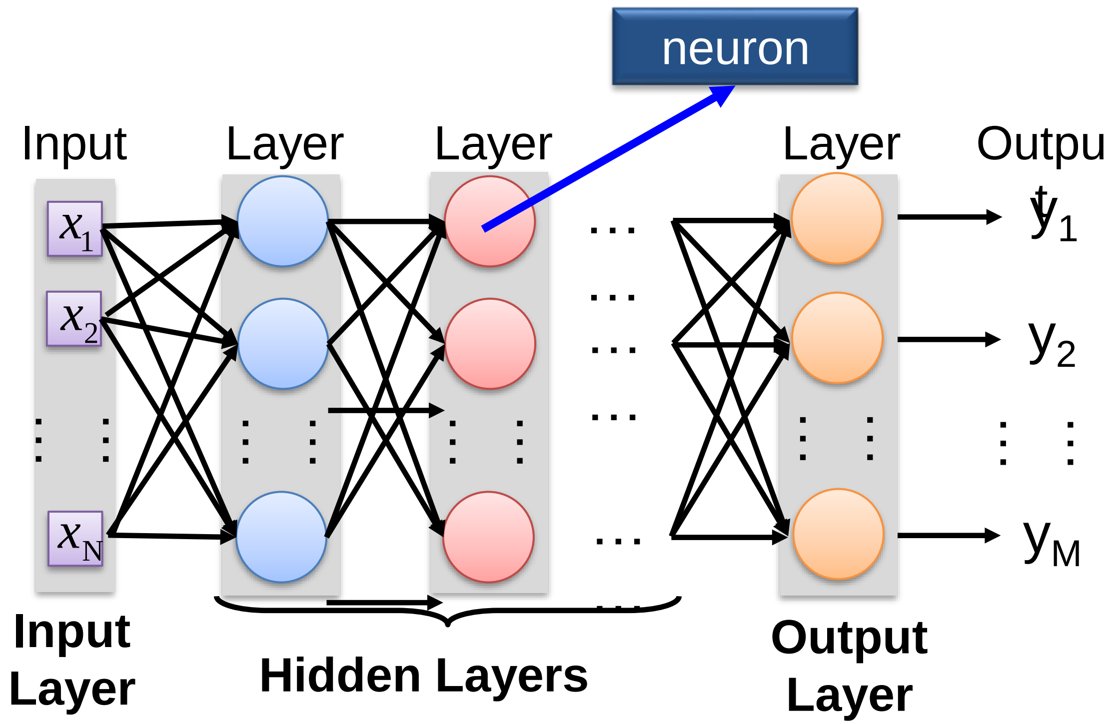
Multi Layer Perceptron (MLP)

- Feedforward Neural Network (FFNN) is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- Can **handle non-linearly separable learning tasks**.



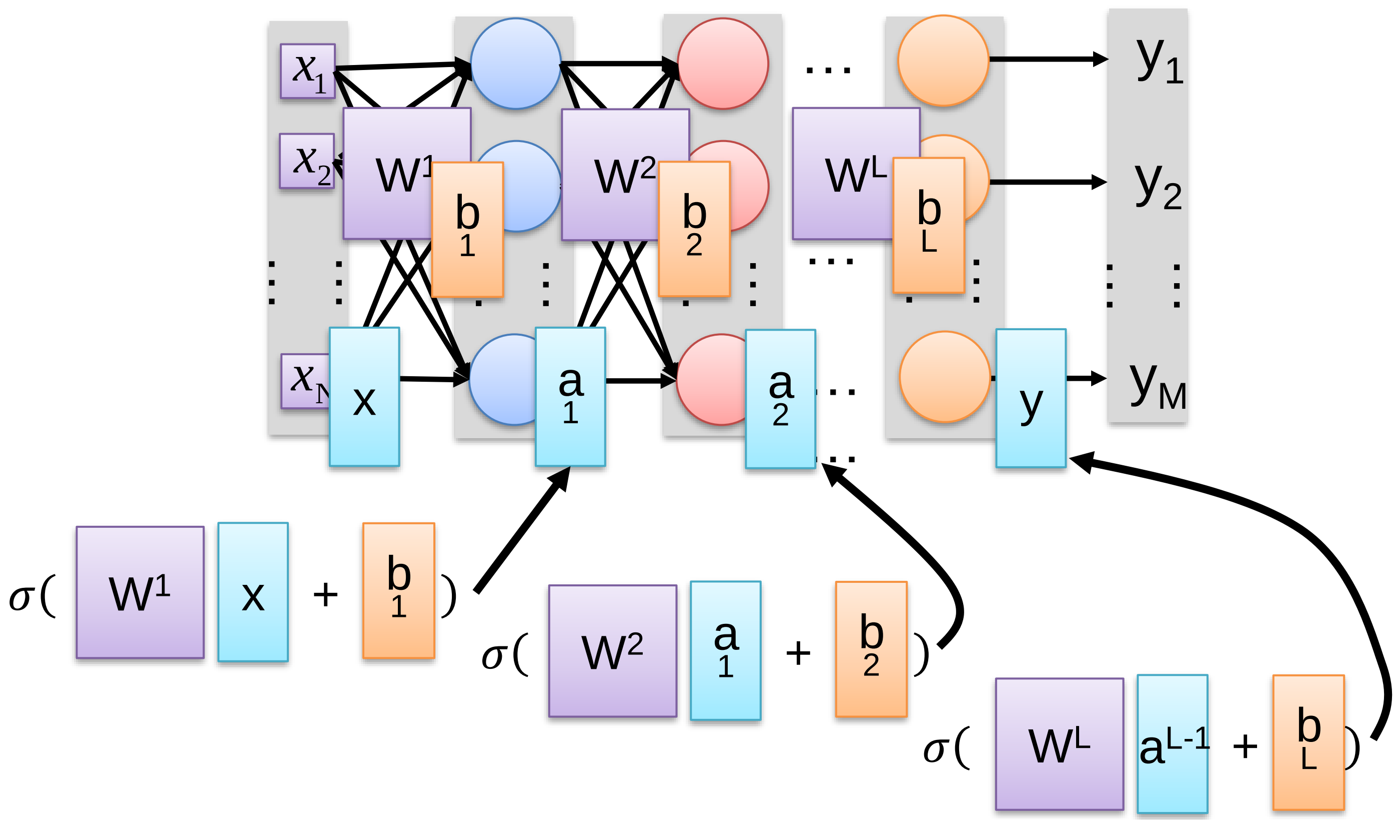
(3-4-2) Network

Neural Networks

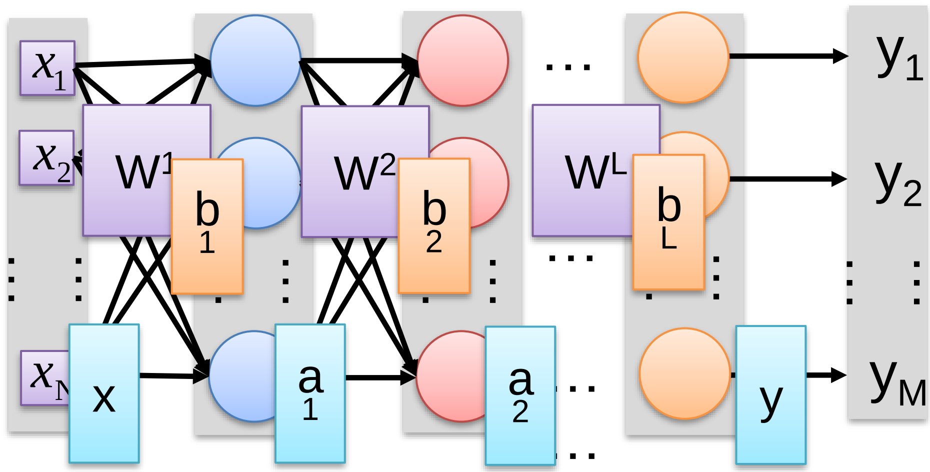


Deep means many hidden layers

Neural Networks



Neural Networks

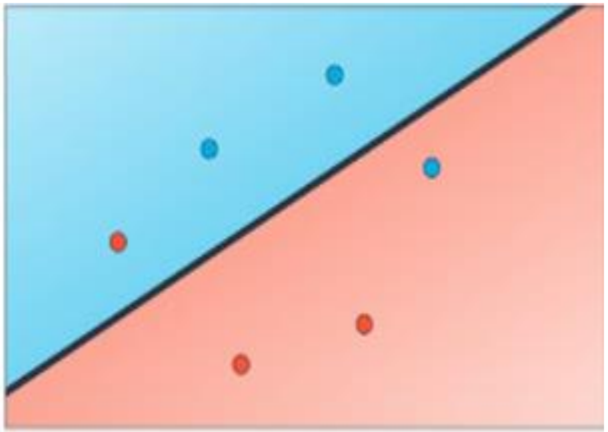


$$y = f(x)$$

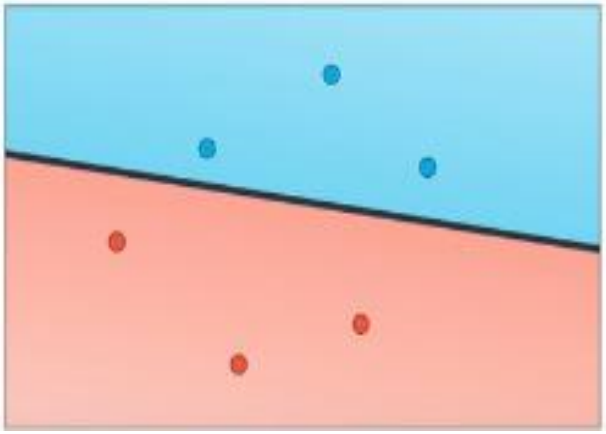
Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

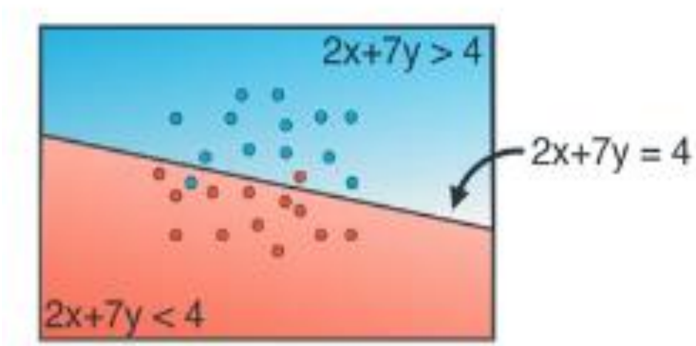
Visualization and Understanding of Neural Network

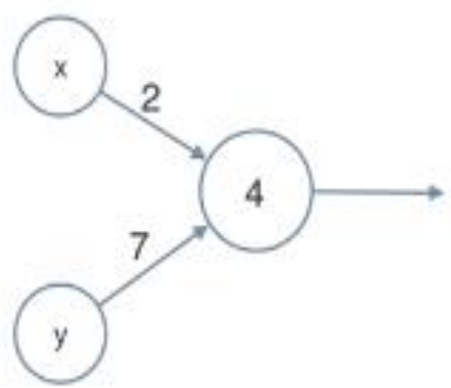


Goal: split data

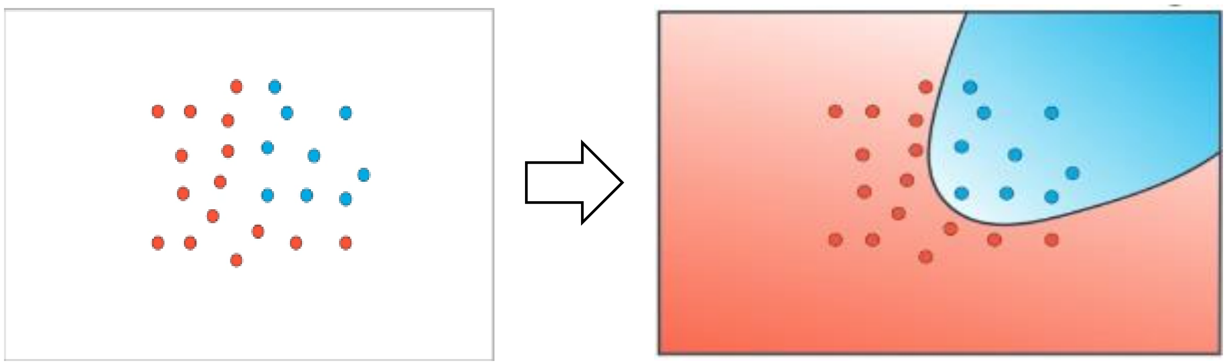


Split data with 0 errors HOT !!!

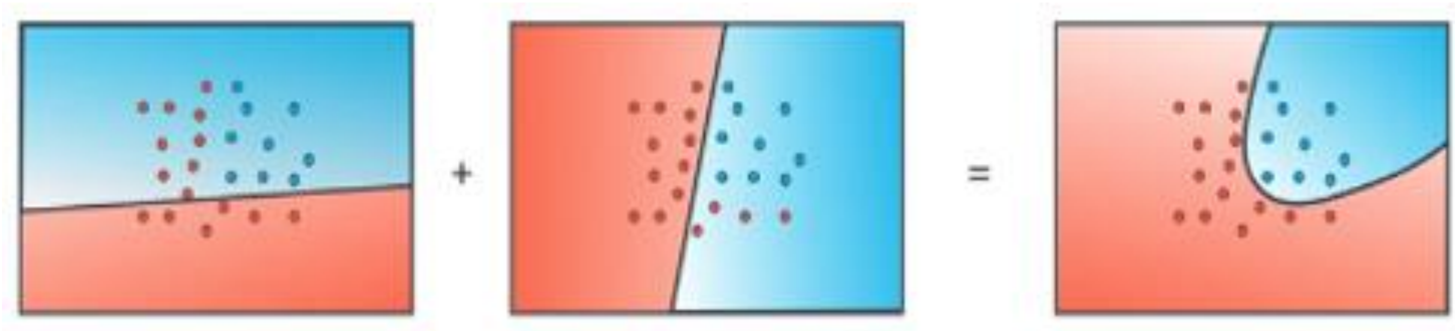




Visualization and Understanding of Neural Network



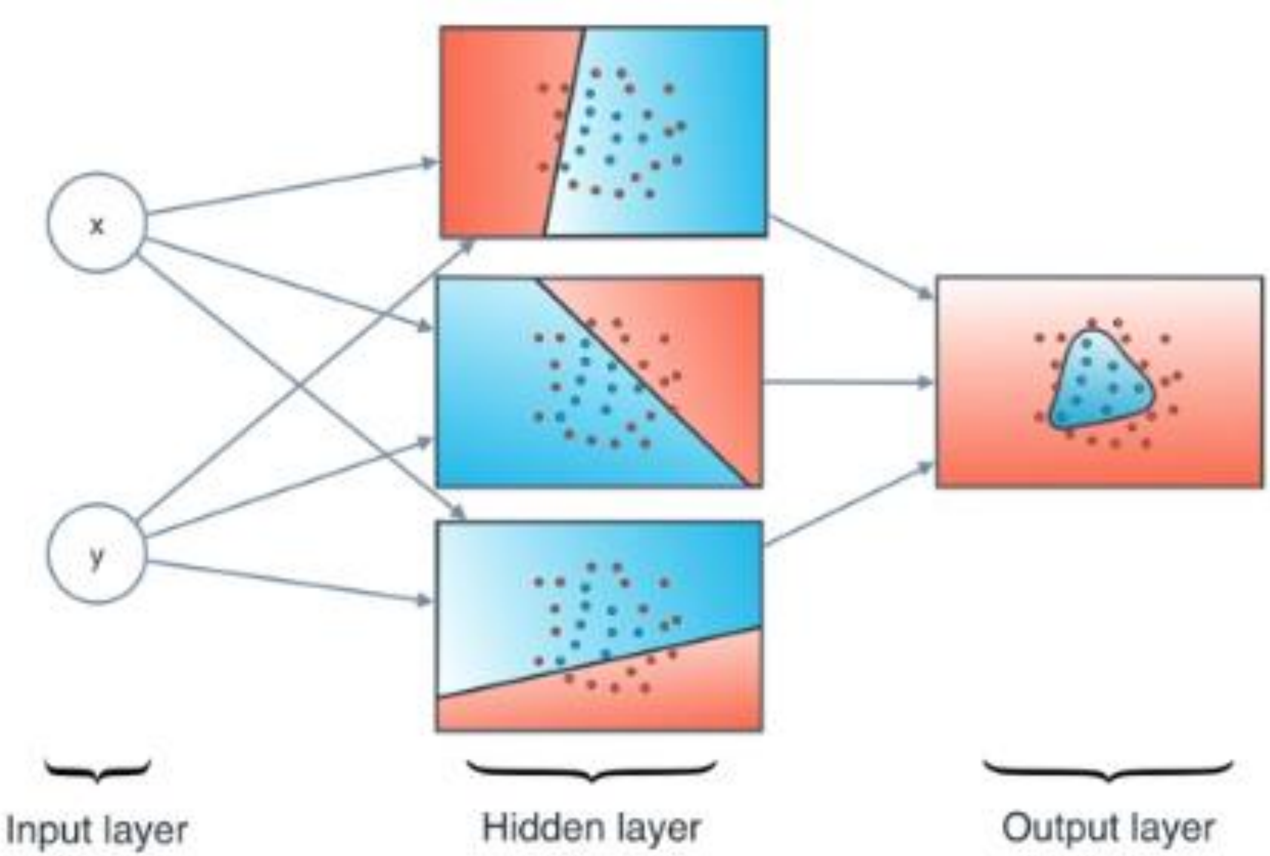
Non-linear regions



Combining regions

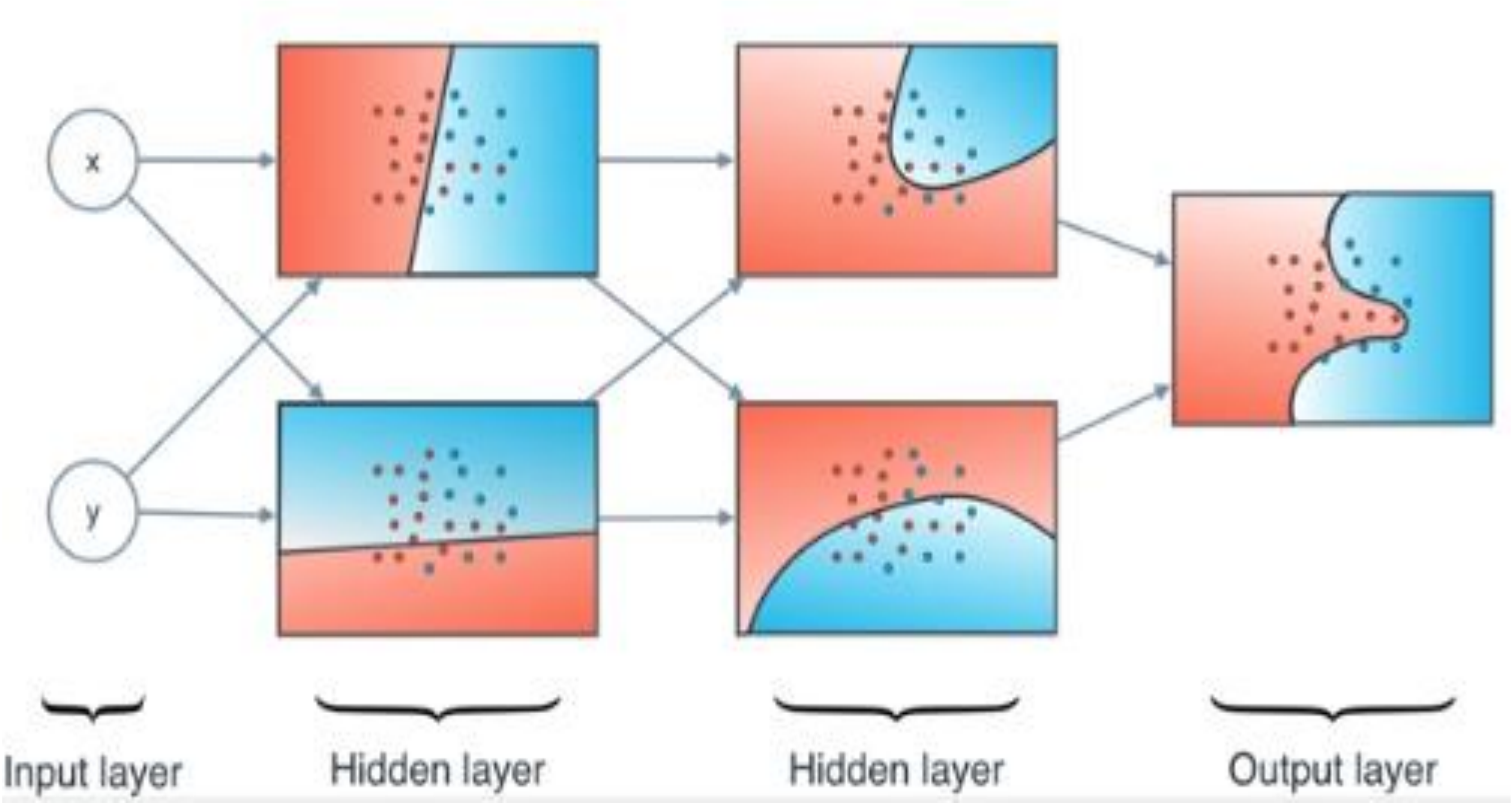
Visualization and Understanding of Neural Network

- In case of two inputs



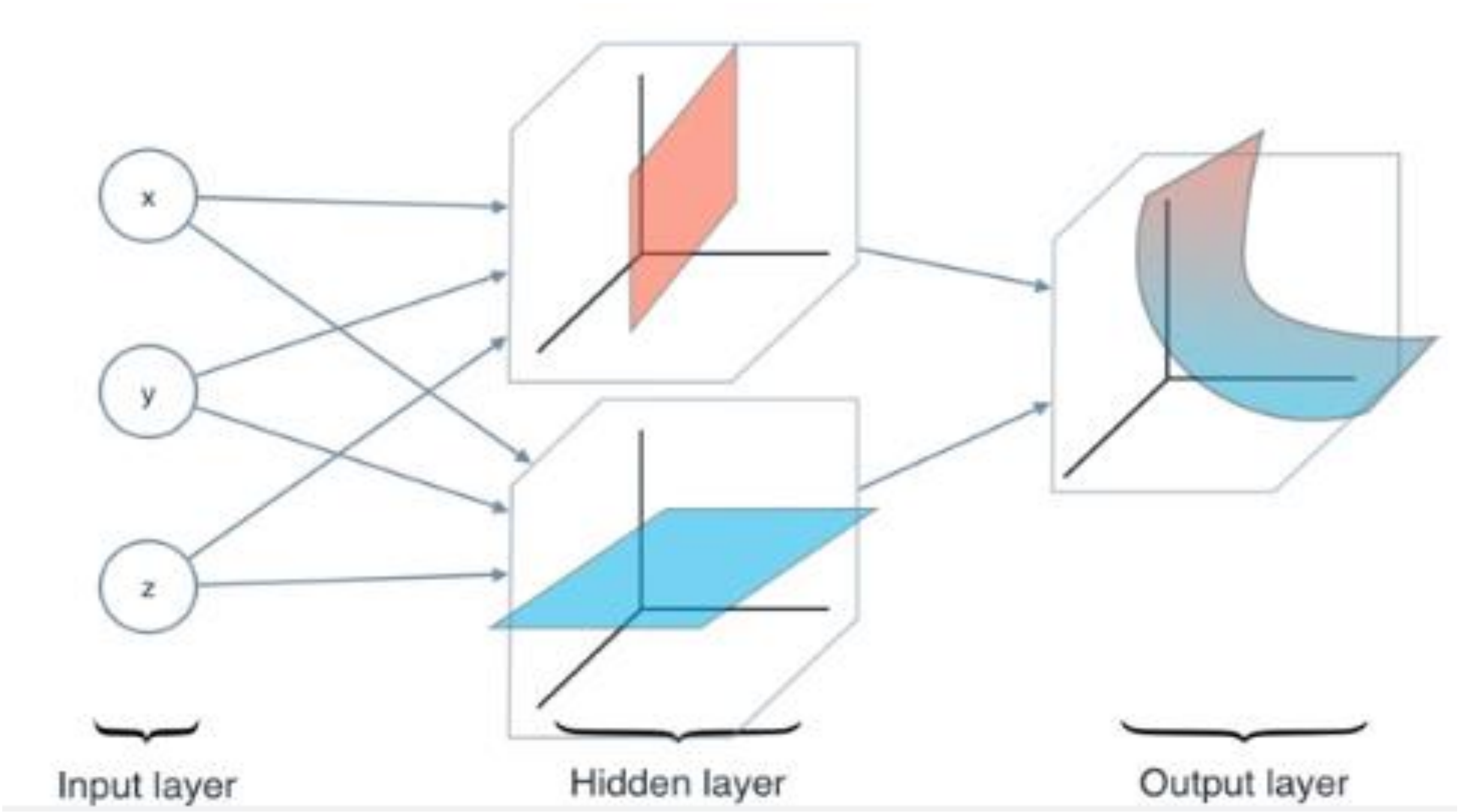
Visualization and Understanding of Neural Network

- Neural Network with more hidden layers

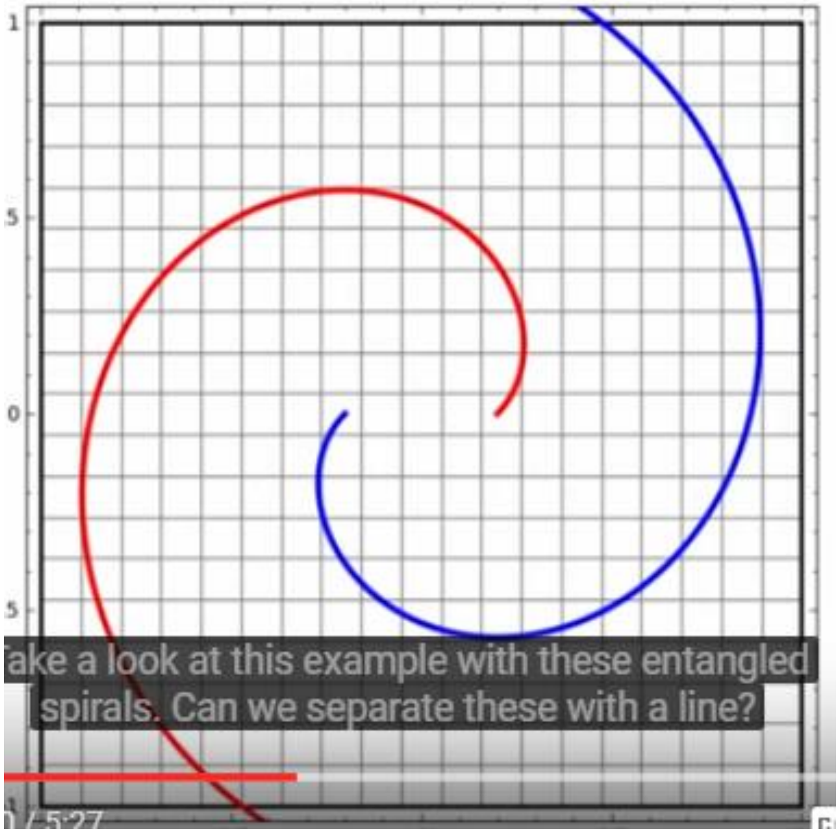


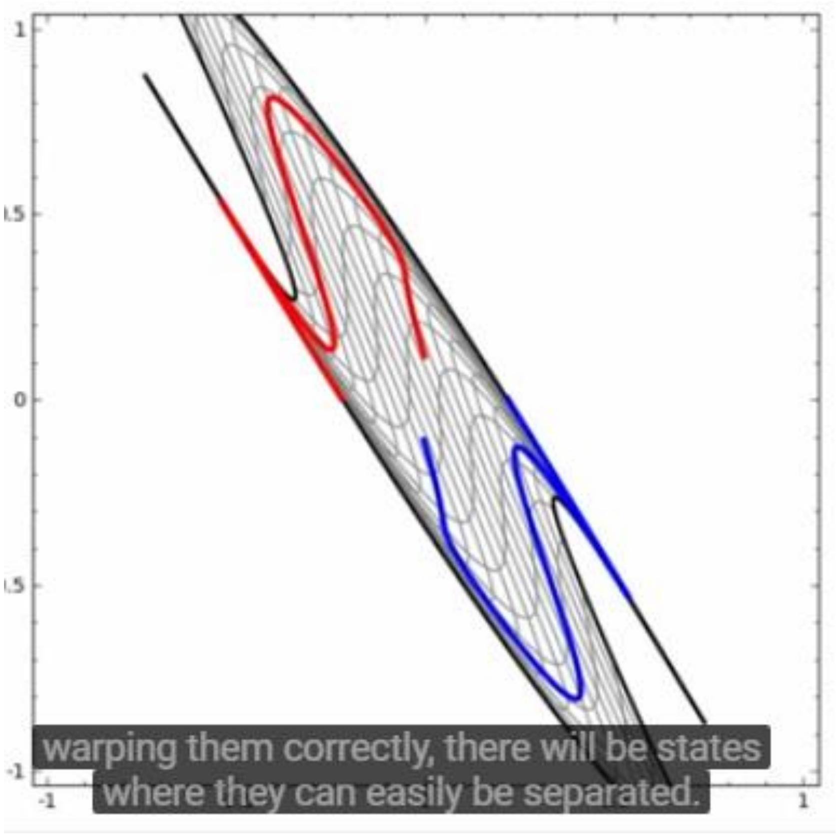
Visualization and Understanding of Neural Network

- In case of three inputs



Complex examples

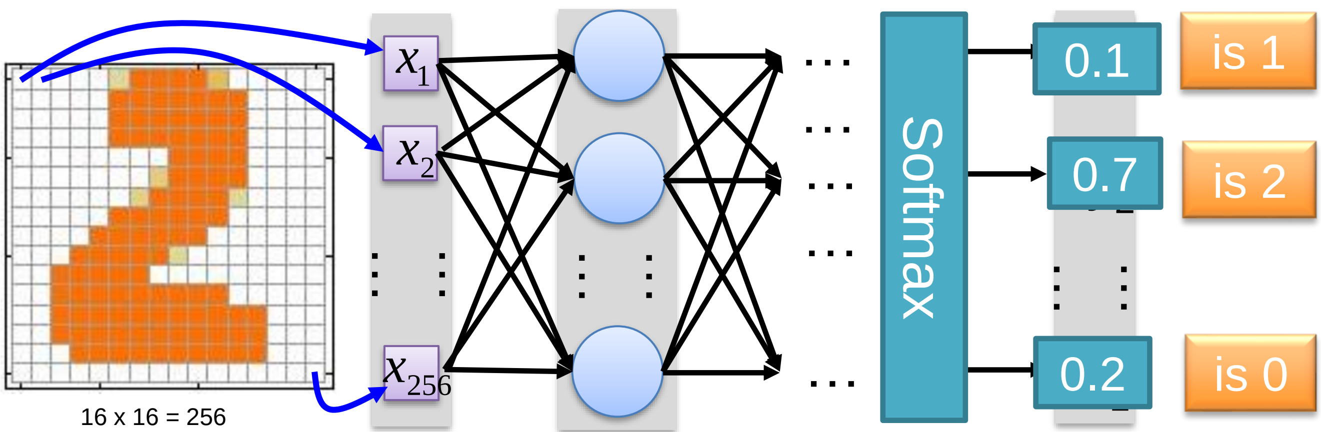




How to set network weights (parameters)

- Weight settings determine the behaviour of a network

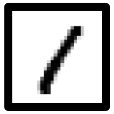
$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



Ink → 1

No ink → 0

Set the network parameters θ such that ……

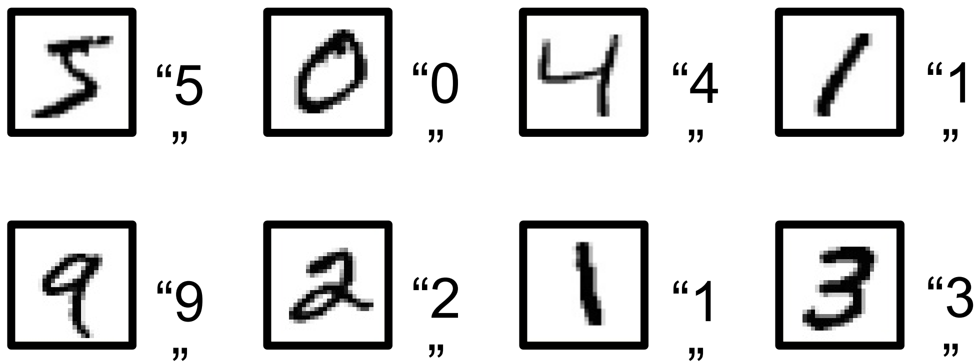
Input: 

Input: 

How to let the neural network achieve this

Training Data

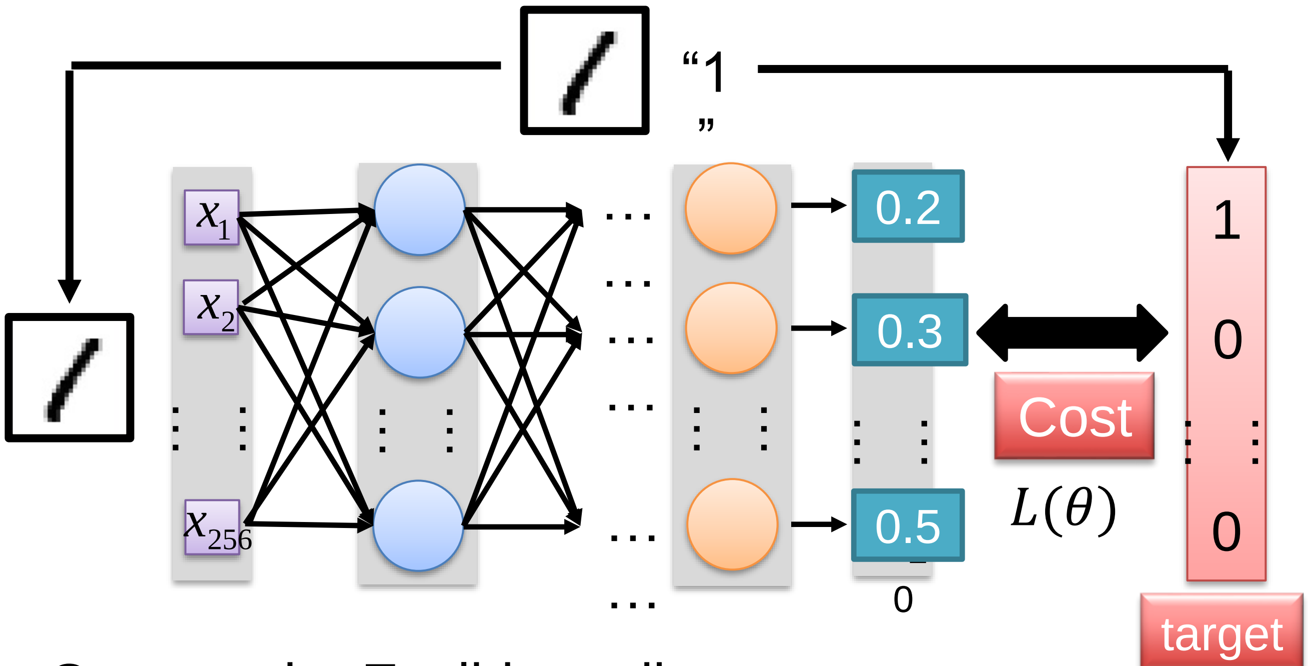
- Preparing training data: images and their labels



Using the training data to find the network parameters.

Cost

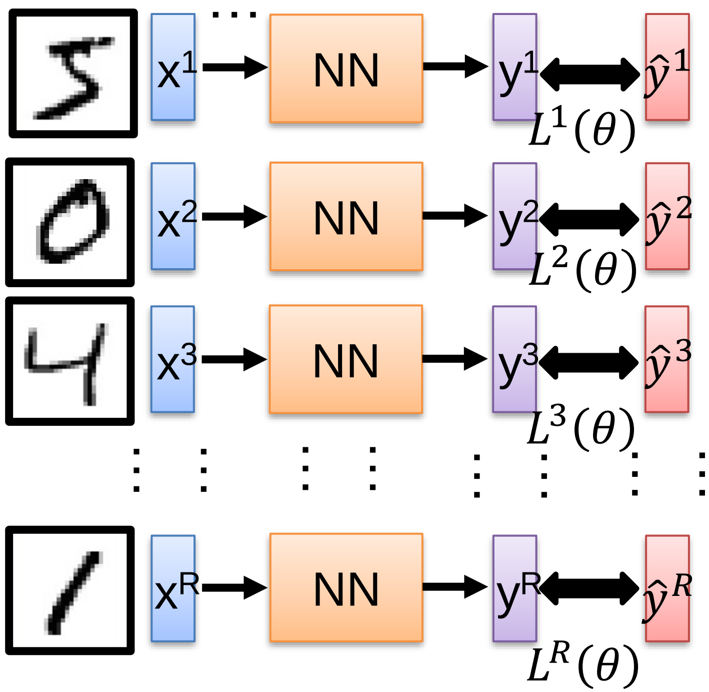
Given a set of network parameters θ, each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data



Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters θ is on this task

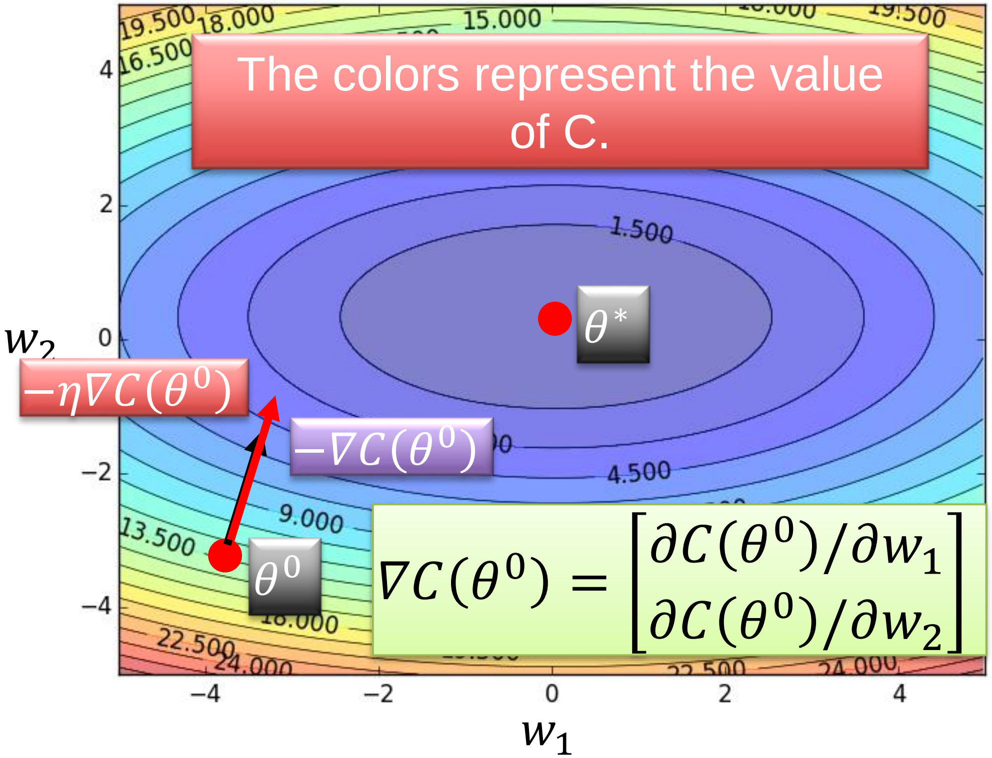
Find the network parameters θ^* that minimize this value

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$\theta = \{w_1, w_2\}$

Error Surface



$$\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta^0)/\partial w_1 \\ \partial C(\theta^0)/\partial w_2 \end{bmatrix}$$

Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

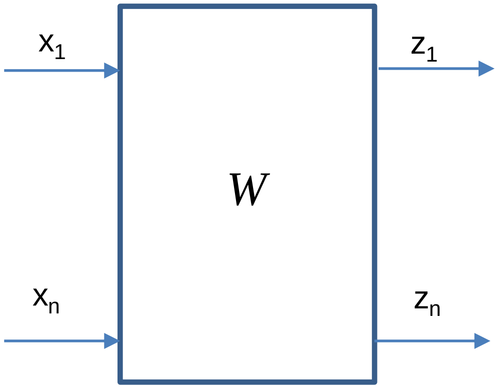
Times the learning rate η

$\Rightarrow -\eta\nabla C(\theta^0)$

Mathematical model

- Neural network: input / output transformation

$$z = F(x, W)$$



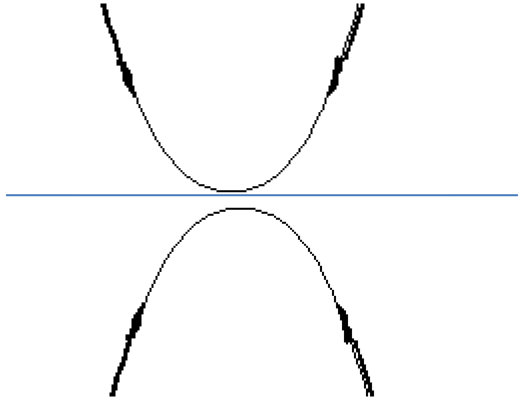
W is the matrix of all weight vectors, and d is the targets:

$$d = G(x)$$

Performance function:

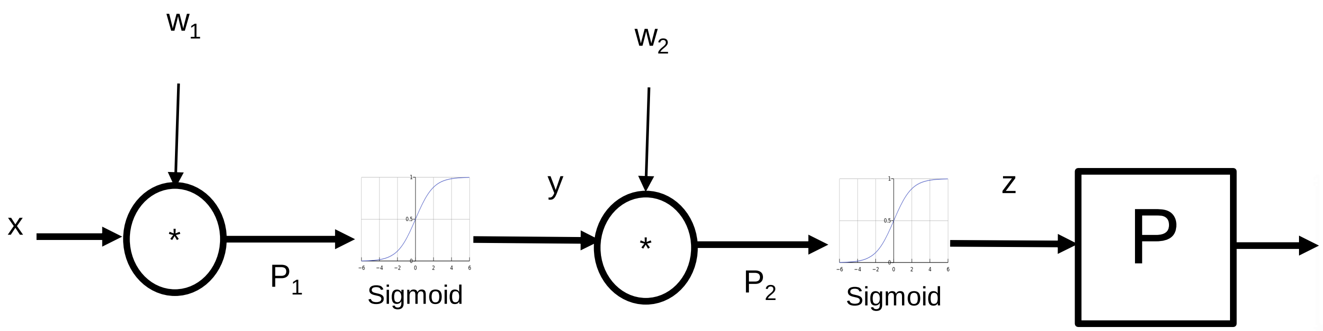
$$P = -\frac{1}{2}\left\| d - z \right\|^2$$

Gradient descent



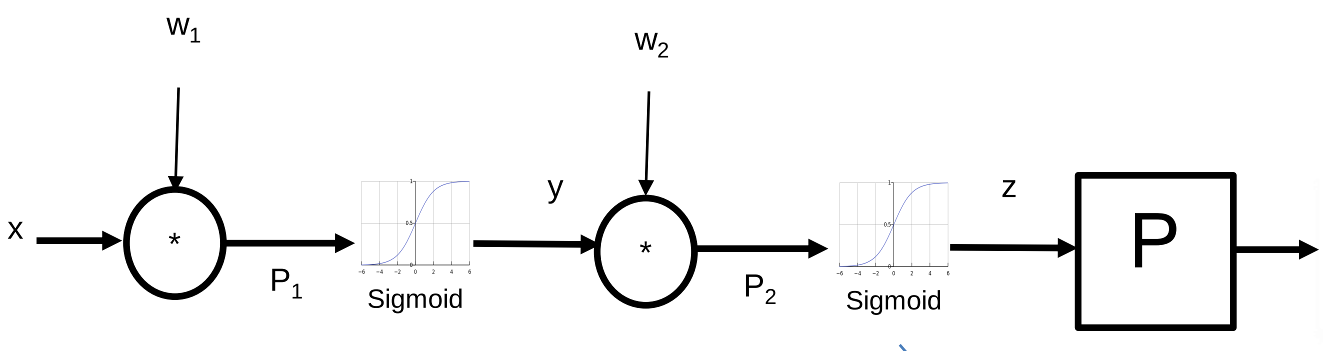
Gradient ascent

Simplest MLP



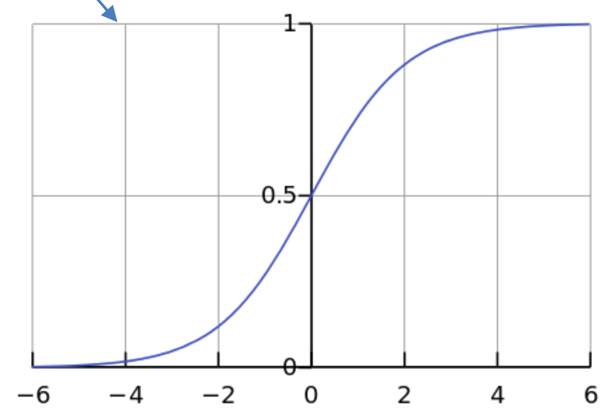
$$P = -\frac{1}{2}(d - z)^2$$

Simplest MLP

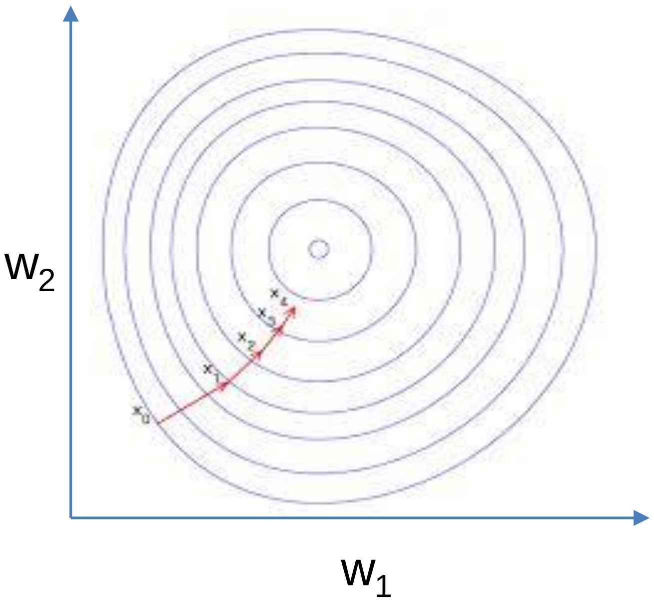


$$P = -\frac{1}{2}(d - z)^2$$

$$\frac{1}{1+e^{-\alpha}}$$

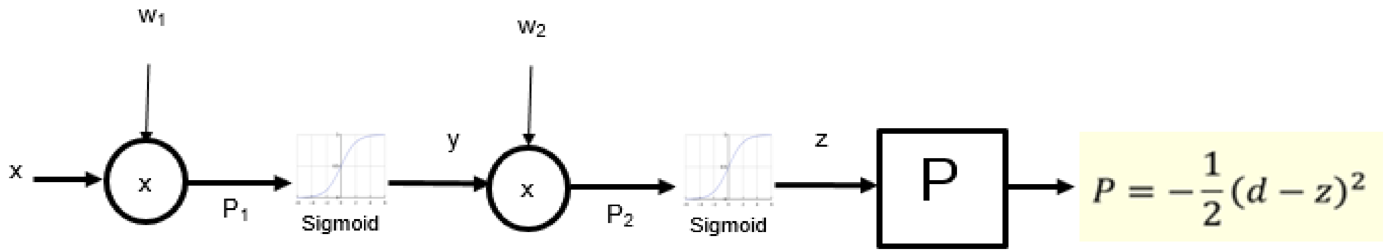


Gradient ascent (Hill Climbing Approach)



$$\Delta w = \eta\left(\frac{\partial P}{\partial x} i + \frac{\partial P}{\partial y} j\right) \quad = \eta\left(\frac{\partial P}{\partial w_1} + \frac{\partial P}{\partial w_2}\right)$$

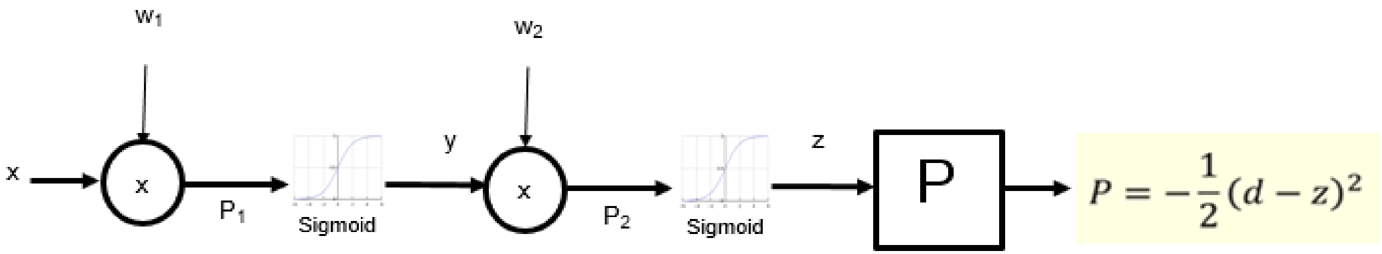
Weight update..



$$\frac{\partial P}{\partial w_2} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial w_2} = (d-z)\frac{\partial z}{\partial w_2}$$

$$= (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_2} = (d-z)\frac{\partial z}{\partial p_2} y$$

Weight update..



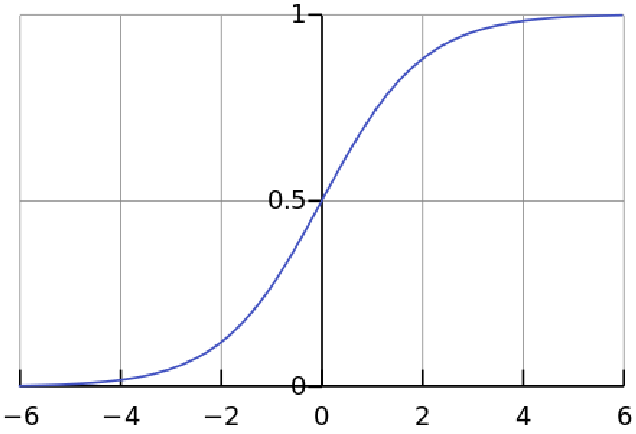
$$\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial w_1} = (d-z)\frac{\partial z}{\partial w_1}$$

$$= (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_1} = (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial y}\frac{\partial y}{\partial w_1}$$

$$= (d-z)\frac{\partial z}{\partial p_2} w_2 \frac{\partial y}{\partial p_1}\frac{\partial p_1}{\partial w_1} = (d-z)\frac{\partial z}{\partial p_2} w_2 \frac{\partial y}{\partial p_1} x$$

Weight update..

$$\beta = \frac{1}{1+e^{-\alpha}} = \left(1+e^{-\alpha}\right)^{-1}$$



$$\frac{\partial \beta}{\partial \alpha} = \frac{e^{-\alpha}}{\left(1+e^{-\alpha}\right)^2} = \frac{1+e^{-\alpha}-1}{\left(1+e^{-\alpha}\right)^2}$$

$$= \frac{1}{\left(1+e^{-\alpha}\right)}\left[\frac{\left(1+e^{-\alpha}\right)}{\left(1+e^{-\alpha}\right)} - \frac{1}{\left(1+e^{-\alpha}\right)}\right] = \beta\left(1-\beta\right)$$

Weight update…

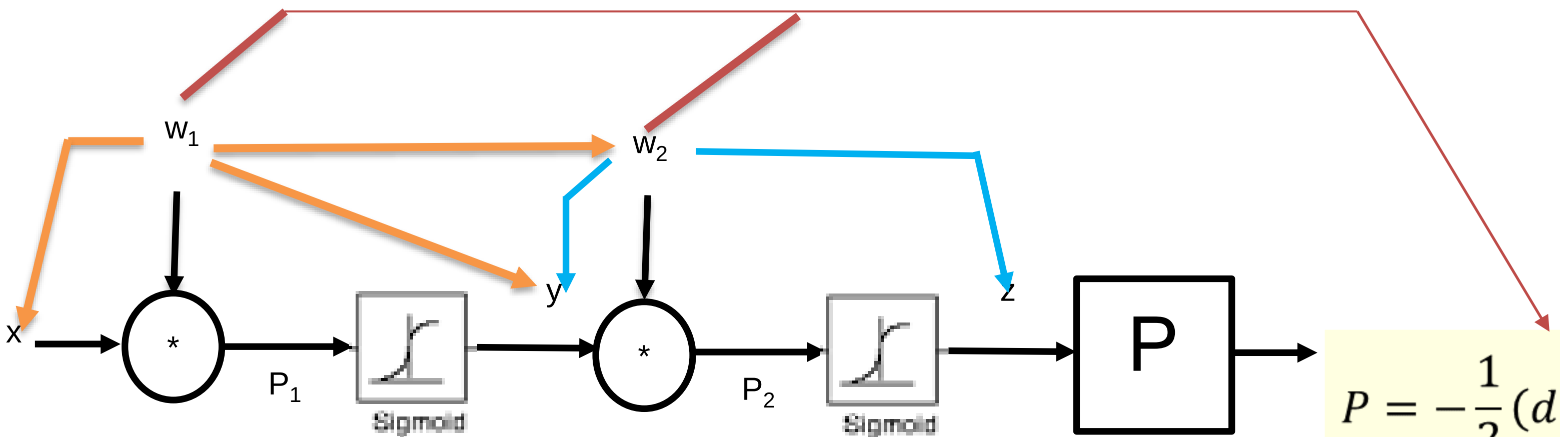
$$\boxed{\frac{\partial z}{\partial p_2} = z(1-z)} \qquad \boxed{\frac{\partial y}{\partial p_1} = y(1-y)}$$

$$\frac{\partial P}{\partial w_2} = (d-z)\frac{\partial z}{\partial p_2}y = (d-z)z(1-z)y$$

$$\frac{\partial P}{\partial w_1} = (d-z)\frac{\partial z}{\partial p_2}w_2\frac{\partial y}{\partial p_1}x$$

$$= (d-z)z(1-z)w_2y(1-y)x$$

Weight update…

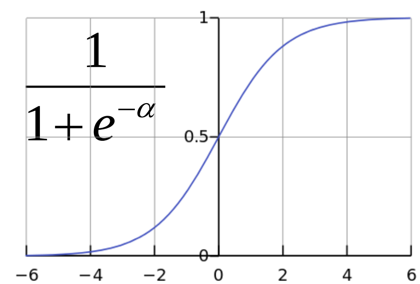


$$P = -\frac{1}{2}(d-z)^2$$

$$\frac{\partial P}{\partial w_2} = (d-z)\frac{\partial z}{\partial p_2}y = (d-z)z(1-z)y$$

$$\frac{\partial P}{\partial w_1} = (d-z)\frac{\partial z}{\partial p_2}w_2\frac{\partial y}{\partial p_1}x$$

$$= (d-z)z(1-z)w_2 y(1-y)x$$



Weight update…

- Finally, the Δw is

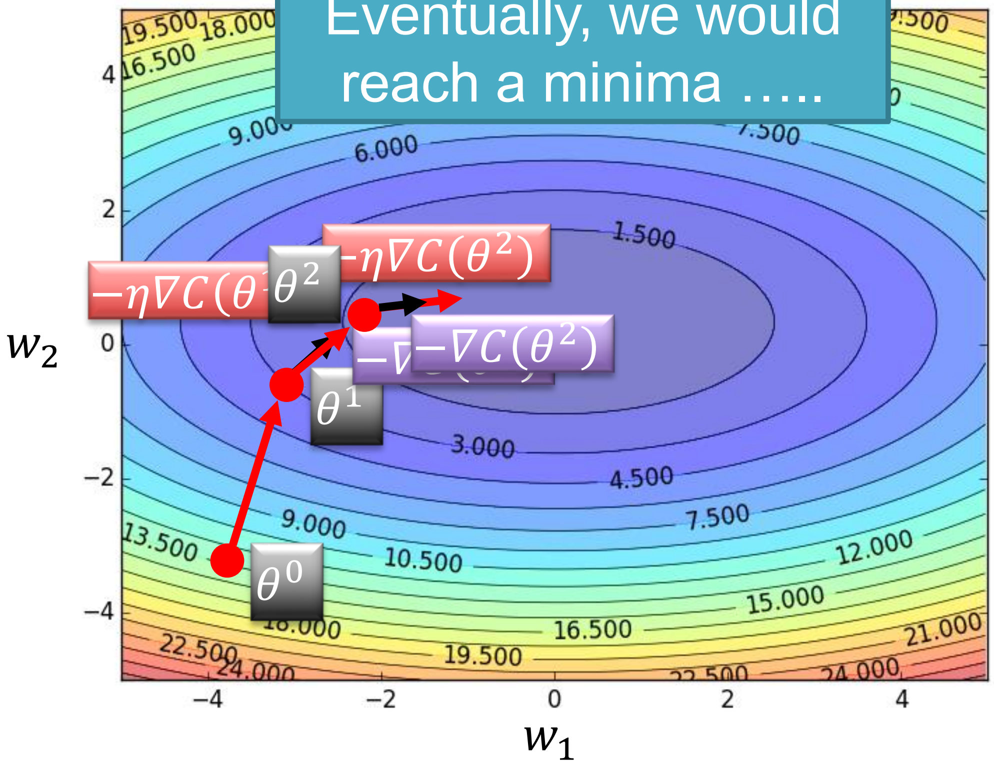
$$\Delta w = \eta\left(\frac{\partial P}{\partial x} i + \frac{\partial P}{\partial y} j\right) = \eta\left(\frac{\partial P}{\partial w_1} + \frac{\partial P}{\partial w_2}\right)$$

Δw = η ((d-z)z(1-z)$w_2 y(1-y)x$) + (d−z)z(1−z))

Then updates the weight with Δw

Gradient Descent

Eventually, we would reach a minima …..



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

$\rightarrow -\nabla C(\theta^0)$

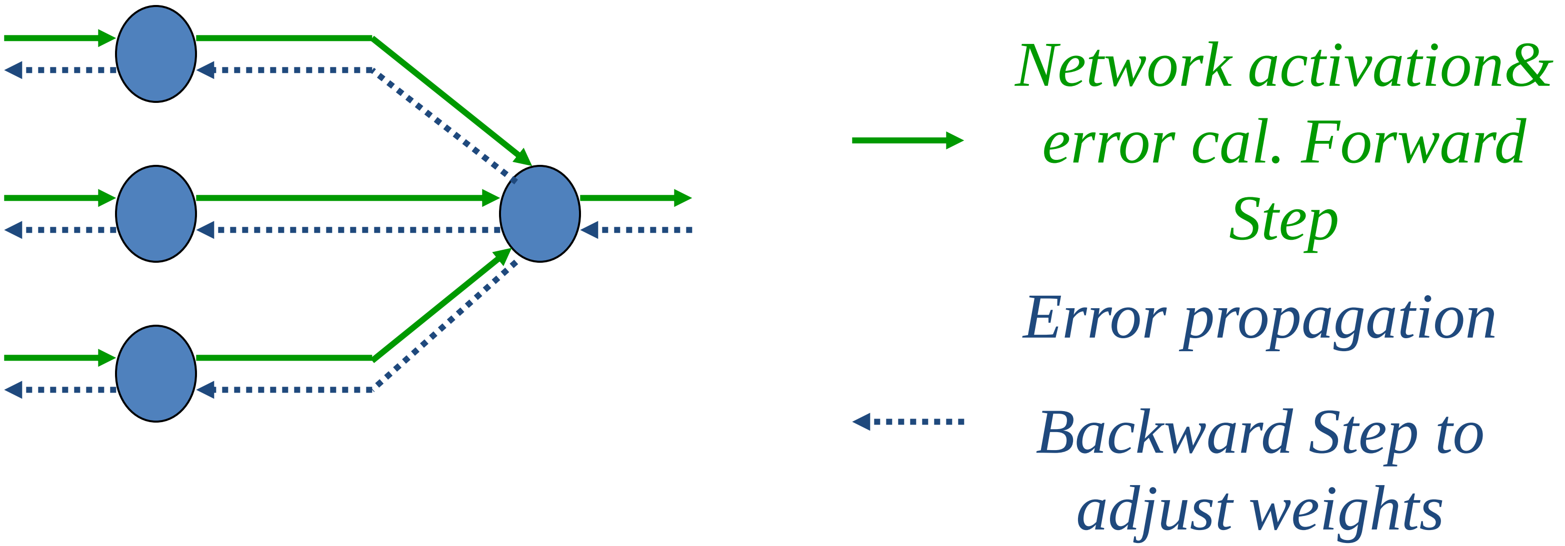
Times the learning rate η

$\rightarrow -\eta\nabla C(\theta^0)$

Gradient descent is an optimization algorithm for minimizing the loss of a predictive model with regard to a training dataset

Training Algorithm: Back-propagation

- Back-propagation algorithm learns in the same way as single perceptron.
- Back propagation adjusts the weights of the NN in order to minimize the network total errors.



Gradient decent with Back-prop algorithm

1. Initialize all weights to small random values
2. REPEAT until done:

   a) For each with w_{ij} set $\Delta w_{ij} := 0$

   b) For each data point point $(\mathbf{x}, \mathbf{t})^p$

      1. set input units to **x**
      2. compute value of output units
      3. For each weight w_{ij} set $\Delta w_{ij} := \Delta w_{ij} + (t_i - y_i) y_j$

   c) For each weight w_{ij} set $w_{ij} := w_{ij} + \Delta w_{ij}$

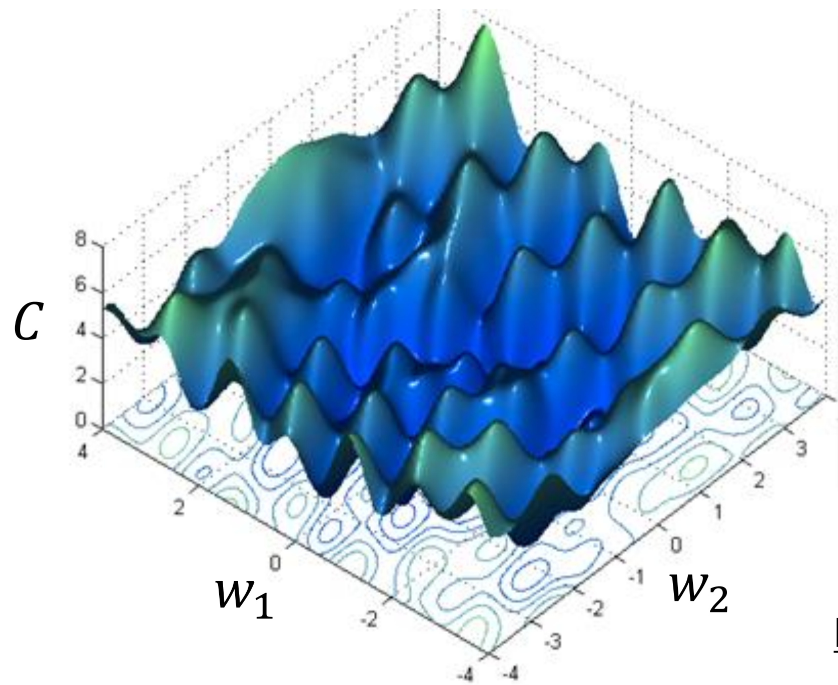
   or $w_{ji}(t+1) = w_{ji}(t) + \alpha \Delta w_{ji}(t)$

3. The algorithm terminates once we are at, or sufficiently near to, the minimum of the error function, where G = 0.

4. We say then that the **network has converged.**

Local Minima

- Gradient descent never guarantee global minima



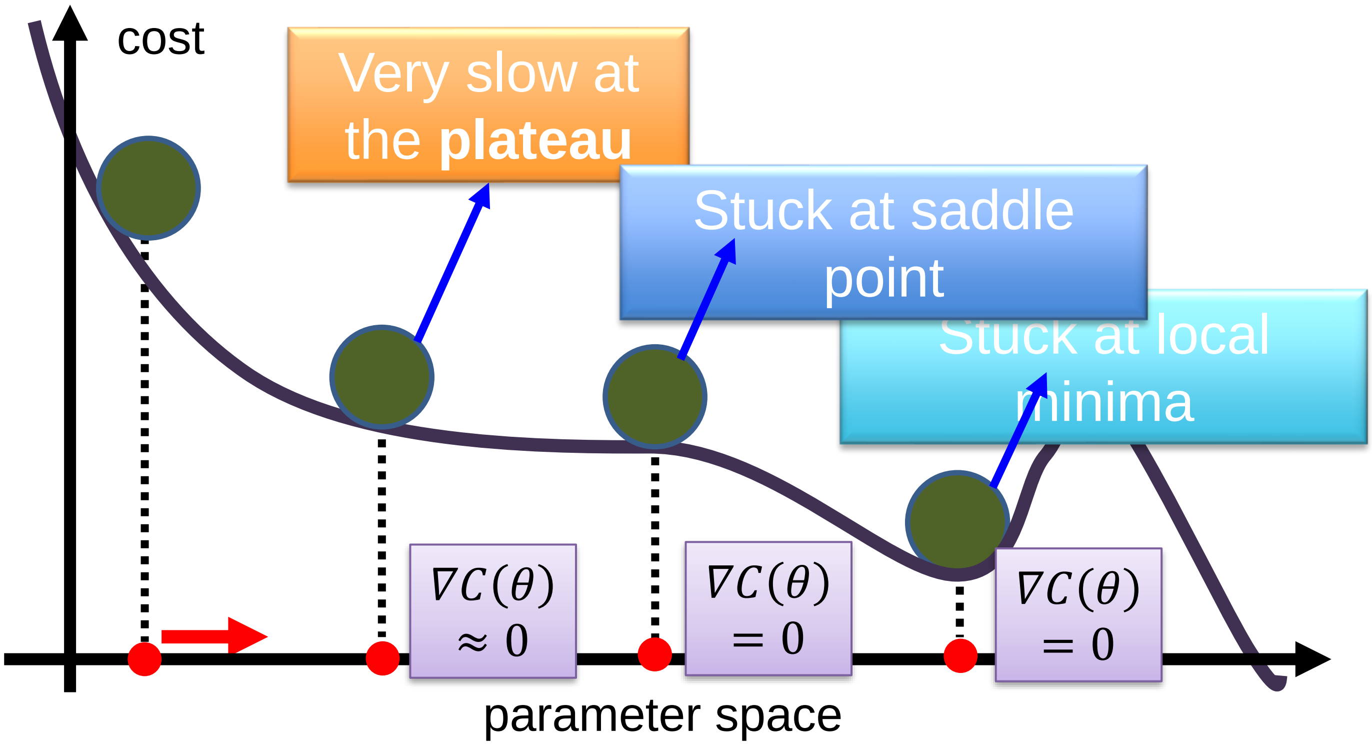
Different initial point θ^0

Reach different minima, so different results

Who is Afraid of Non-Convex Loss Functions?

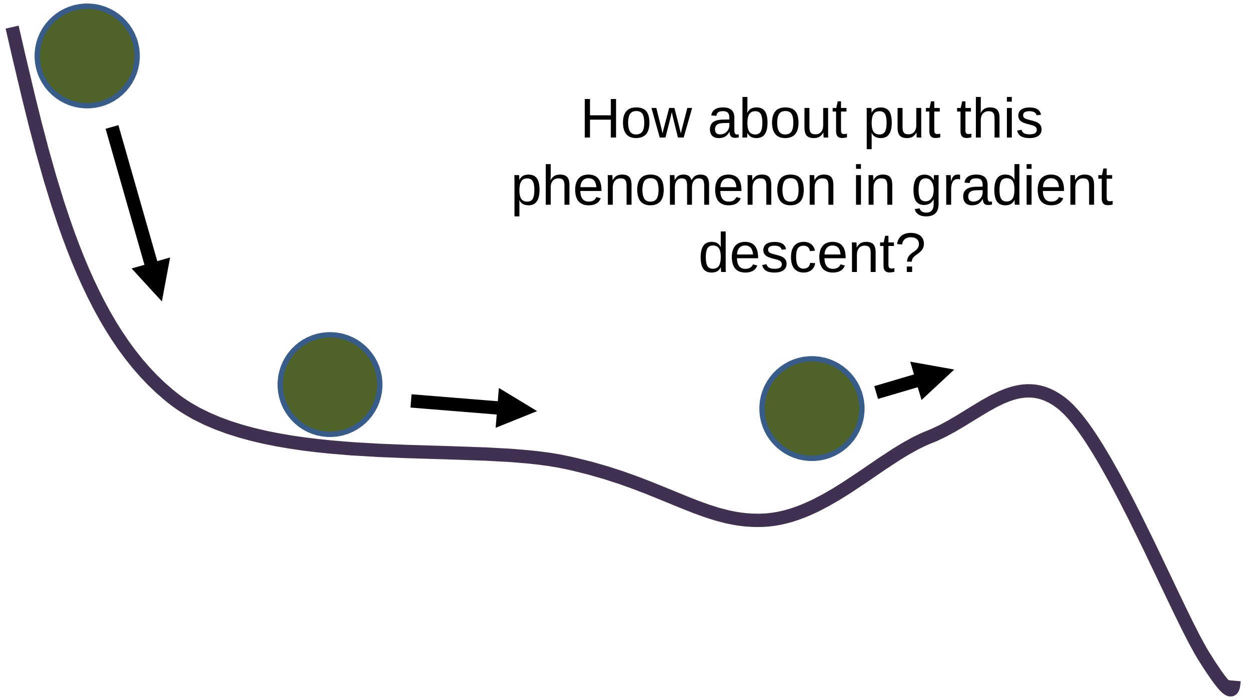
http://videolectures.net/eml07_lecun_wia/

Besides local minima ……



In physical world ……

- The momentum method by (Polyak, 1964)



How about put this phenomenon in gradient descent?

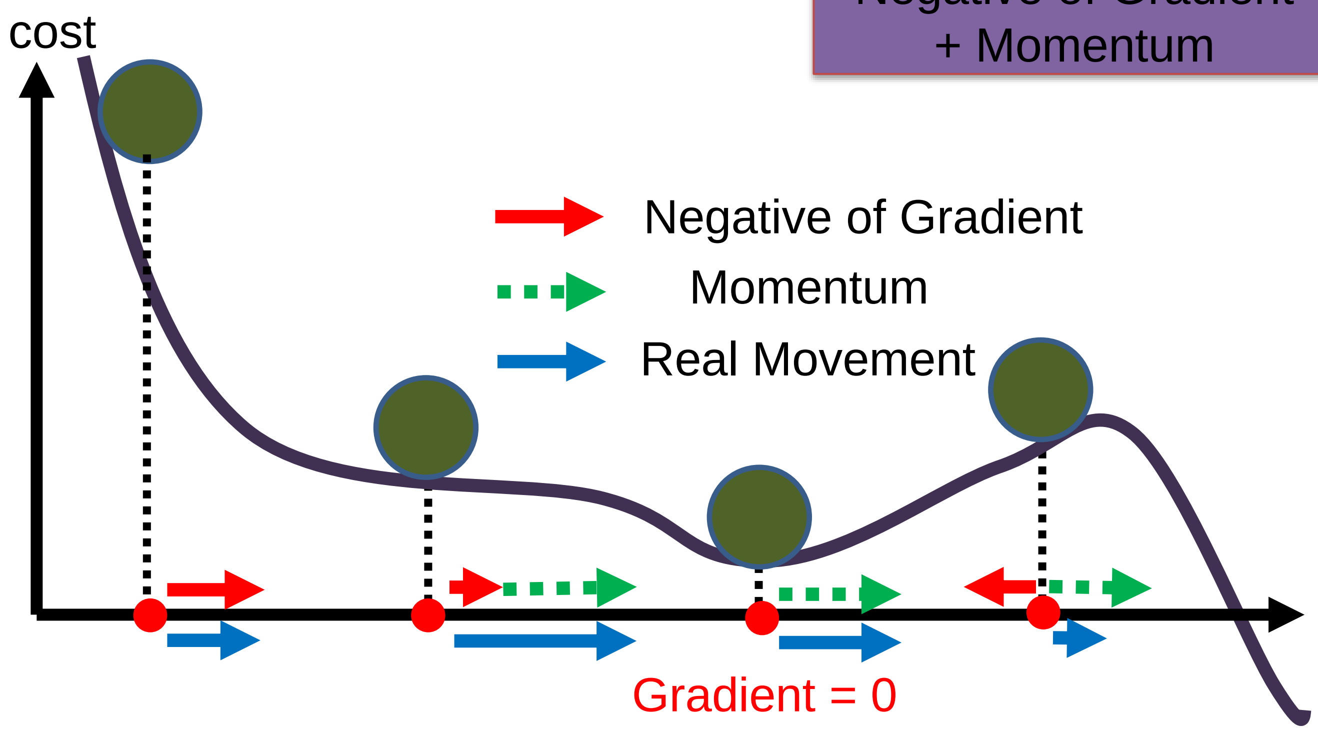
Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning*. 2013.

Momentum

Still not guarantee reaching global minima, but give some hope ……

Movement =

Negative of Gradient + Momentum



Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning*. 2013.

Why Momentum?

- Accelerate the training process
- Prevent to stuck in local minima
- Momentum values: $\gamma \in (0,1]$
  - Generally, γ is set to 0.5 until the initial learning stabilizes and then is increased to 0.9 or higher

NN learning approach: sample by sample

- Problem with individual sample learning
  - Learning with individual sample over the training sets converge very well to local optima (minima).
  - In practice computing cost and gradient computation for the entire training set is very slow.
  - Not suitable for a single machine if the **dataset is too big to fit to the memory**
- Solution: Batch learning

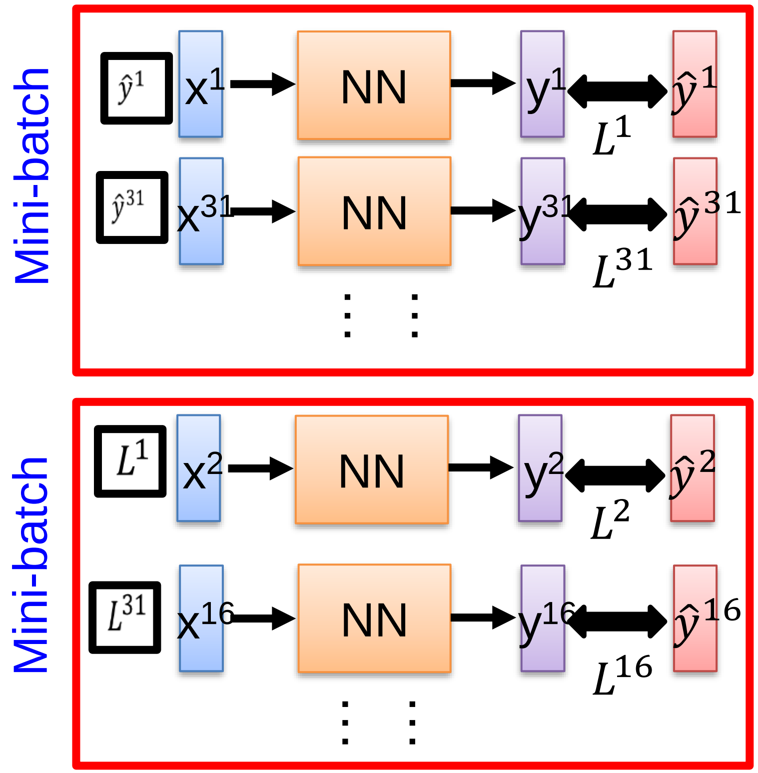
Batch learning

- Stochastic Gradient Descent (SGD) can **overcome computational cost and still lead to fast convergence.**
- Instead of learning whole training dataset at a time by sample, take a batch of images for one iteration.

number of batch= Total training samples / Number of samples per batch

- We define batch-size during the model training
- Batch learning user for **data parallelism**

Mini-batch



- Randomly initialize θ^0
  - Pick the 1st batch

    $C = L^1 + L^{31} + \cdots$

    $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
  - Pick the 2nd batch

    $C = L^2 + L^{16} + \cdots$

    $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

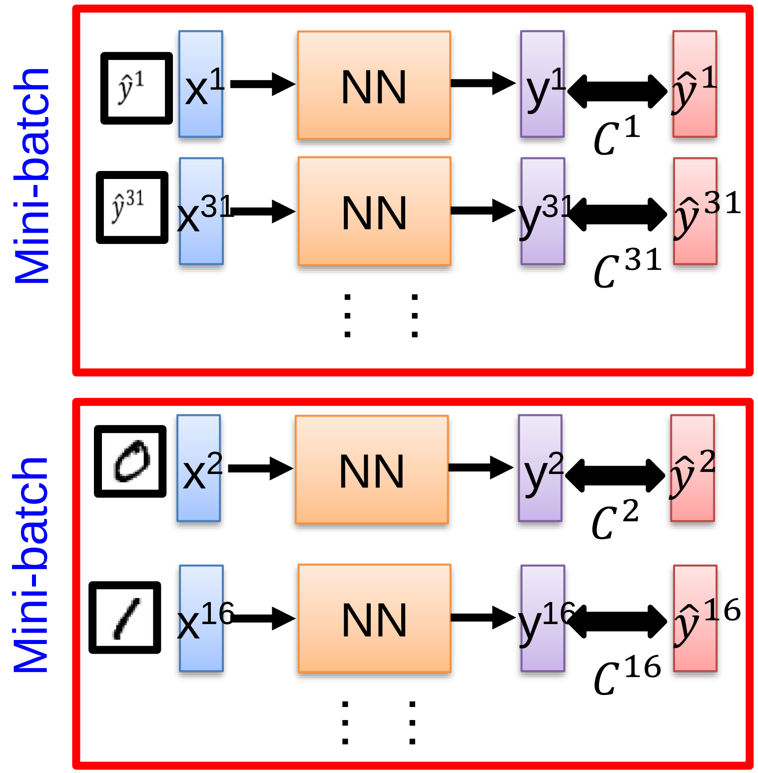
    ⋮

C is different each time when we update parameters!

Mini-batch

Faster

Better!



- Randomly initialize θ^0
  - Pick the 1st batch

$$C = C^1 + C^{31} + \cdots$$
$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

  - Pick the 2nd batch

$$C = C^2 + C^{16} + \cdots$$
$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

$\vdots$

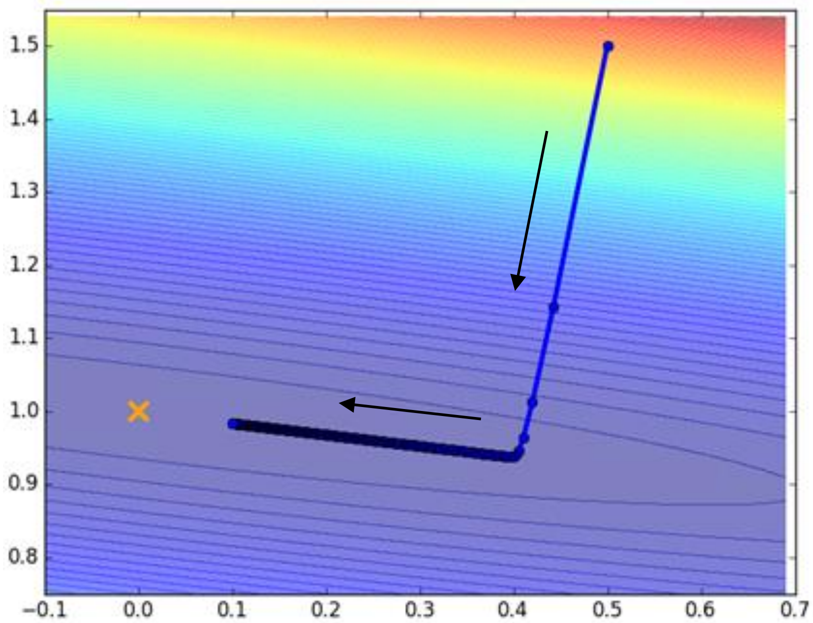
  - Until all mini-batches have been picked

one epoch

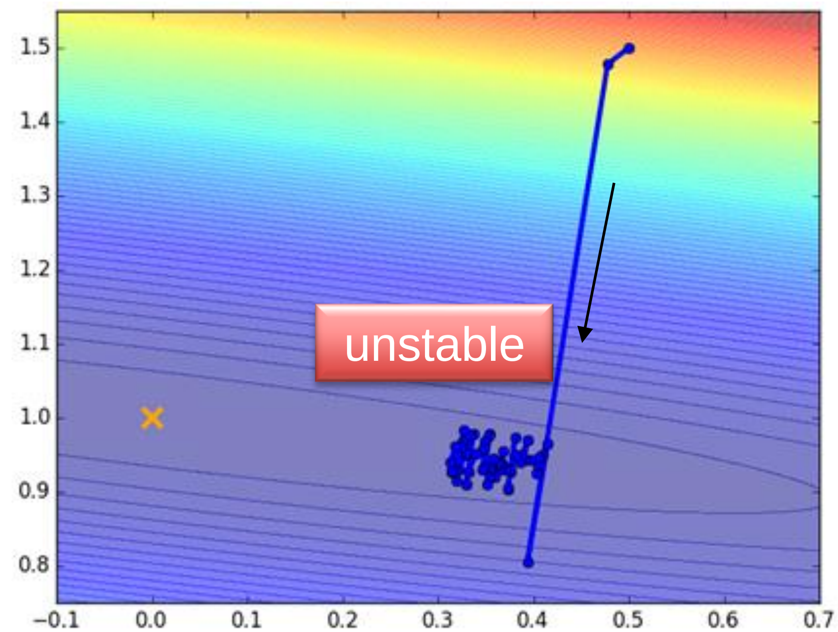
Repeat the above process

Mini-batch

Original Gradient Descent



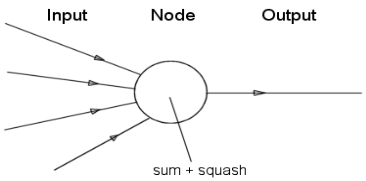
With Mini-batch: Stochastic Gradient Descent (SGD)



The colors represent the total C on all training data.

Learning rate for mini-batch

- All the neurons in the MLP should learn ideally of the same rate
- Last layers gradient is larger than the layers in the front end of the network
- η should be assign with smaller value in the last layers than in the front layers
- Neurons with more inputs should have smaller learning rate compare to few inputs
- LeCunn (1993) : $\eta \infty \frac{1}{\sqrt{\# \, syp.connections \; to \; a \; neuron}}$

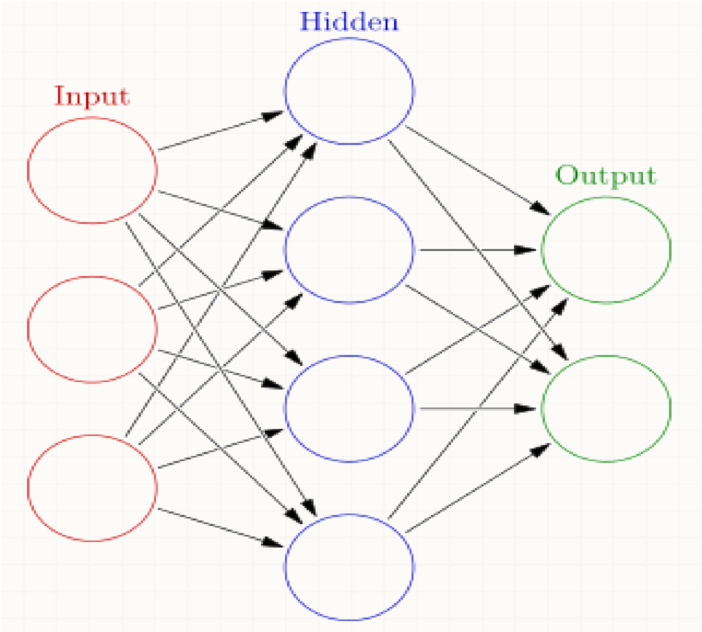


Stopping criterions

- Total mean squared error change or loss:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
  - Maximum number of iterations or epochs (N)
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used, the part of the training set used for testing the network generalization will not used for updating the weights.

Computational parameters for DNN

- Every connection that is learned in a feedforward neural network is called parameter.



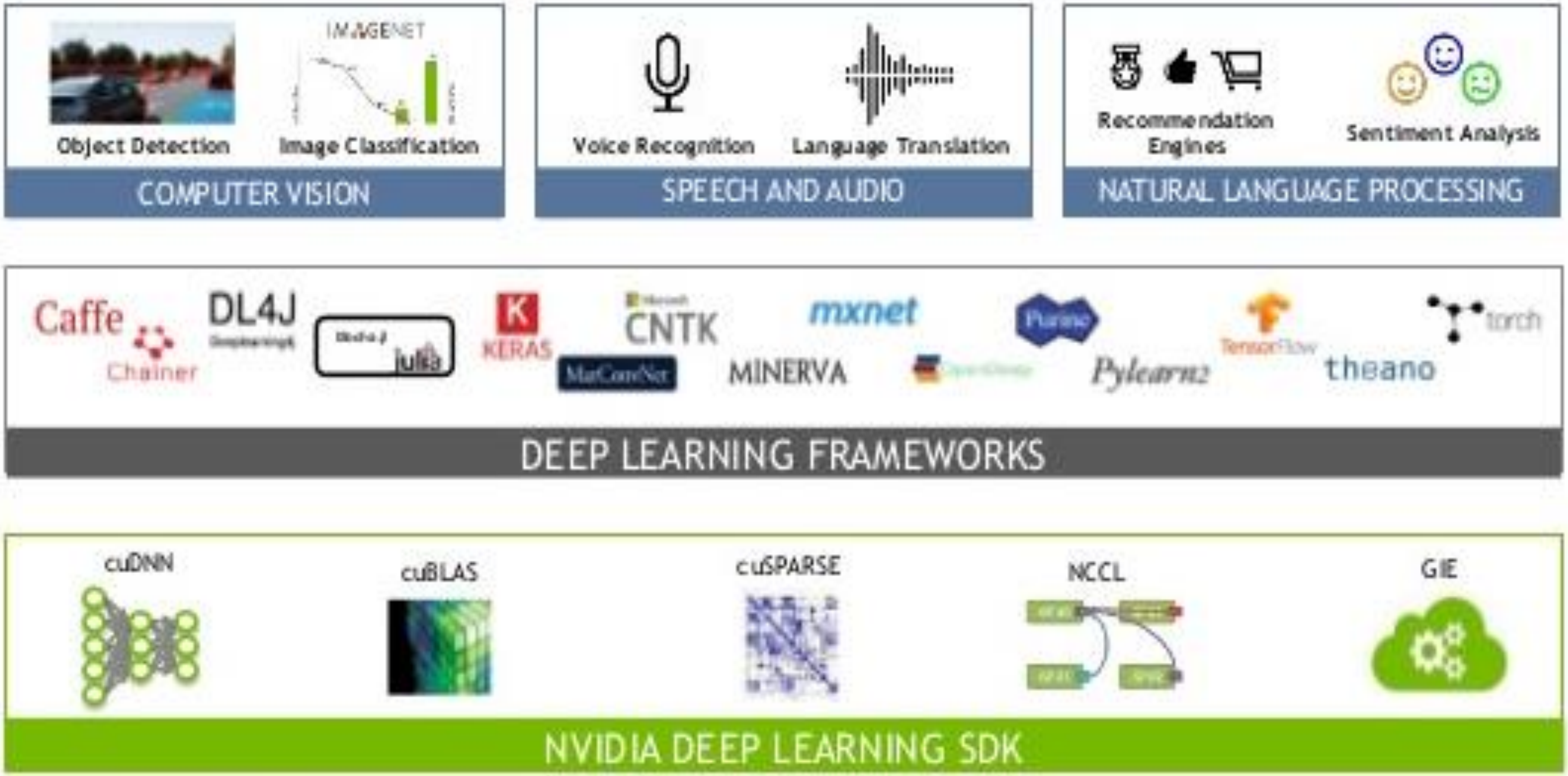
Without bias: (3*4)+(4*2)=20

With bias: (4*4)+(5*2)=26

If network is not fully connected you can count the connections.

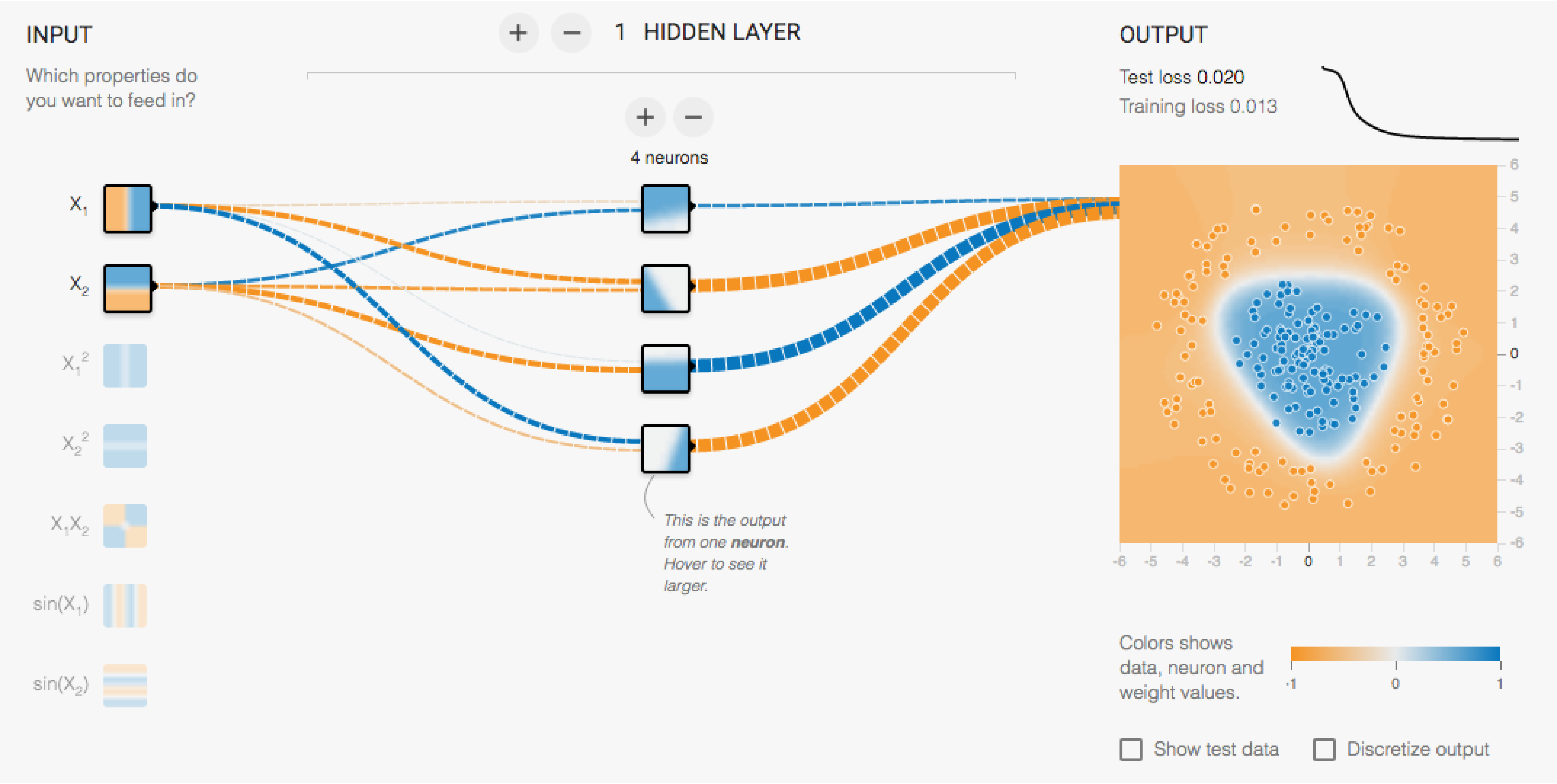
Powering the Deep Learning Ecosystem

NVIDIA SDK Accelerates Every Major Framework



per.nvidia.com/deep-learning-software

Multi-Layer Networks Demo



http://playground.tensorflow.org/

Summary

- We have learned Neural Networks(NN) and back-propagation (BP) algorithms
- Visualized the impact of inputs (features) and hidden layers for NN model
- Mathematical model and BP approach for a NN
- Momentum, batch learning method, and network parameters for training NN
- Deep Learning (DL) ecosystem

- What's next:
  - Computational graph
  - Efficient gradient calculation methods
  - Examples