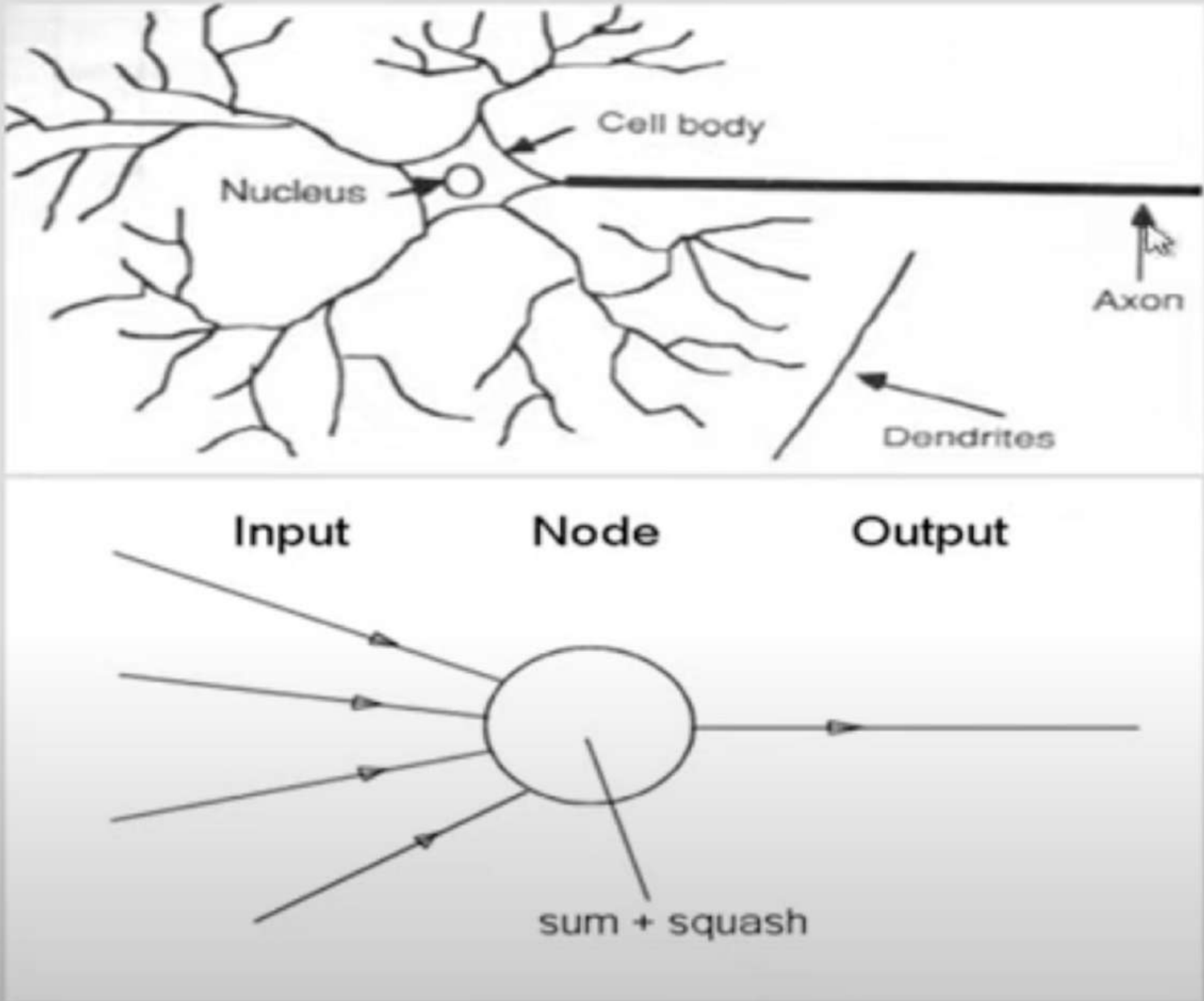
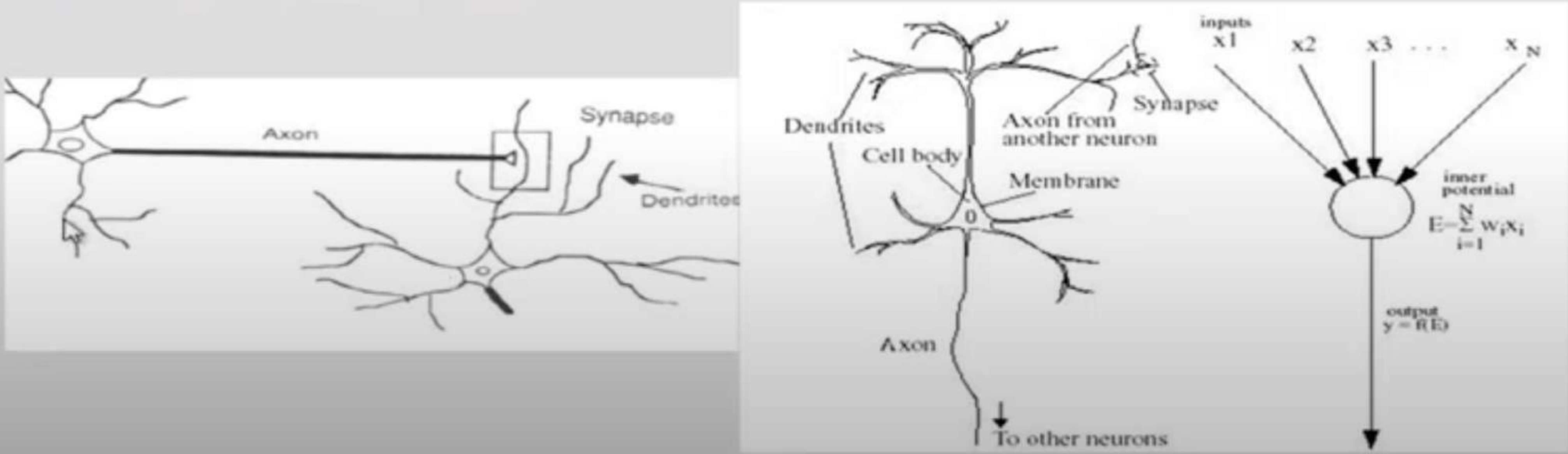
Neuron



Neural Unit

ANNs

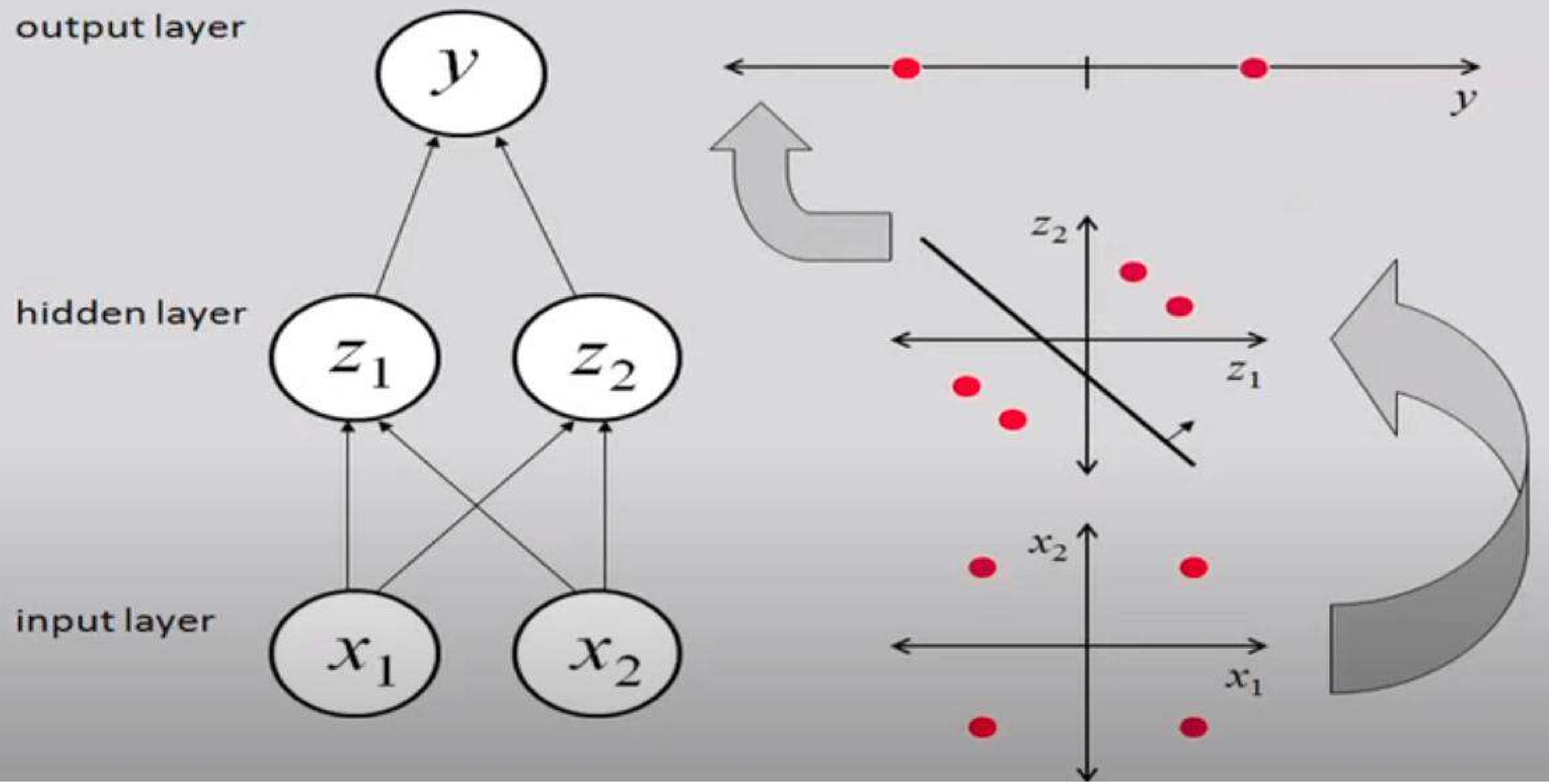
- ANNs incorporate the two fundamental components of biological neural nets:
  1. Nodes - Neurones
  2. Weights - Synapses



Limitations of Perceptrons

- Perceptrons have a *monotinicity* property:
  If a link has positive weight, activation can only increase as the corresponding input value increases (*irrespective* of other input values)
- Can't represent functions where input *interactions* can cancel one another's effect (e.g. XOR)
- Can represent only linearly separable functions

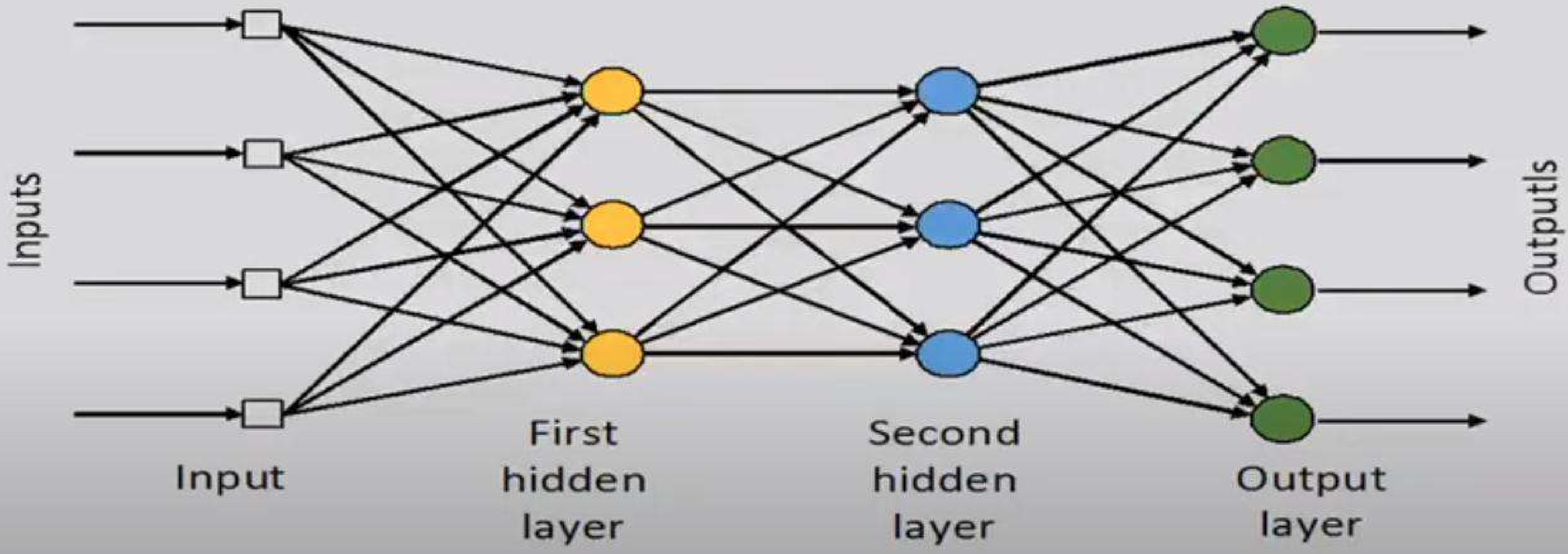
A solution: multiple layers

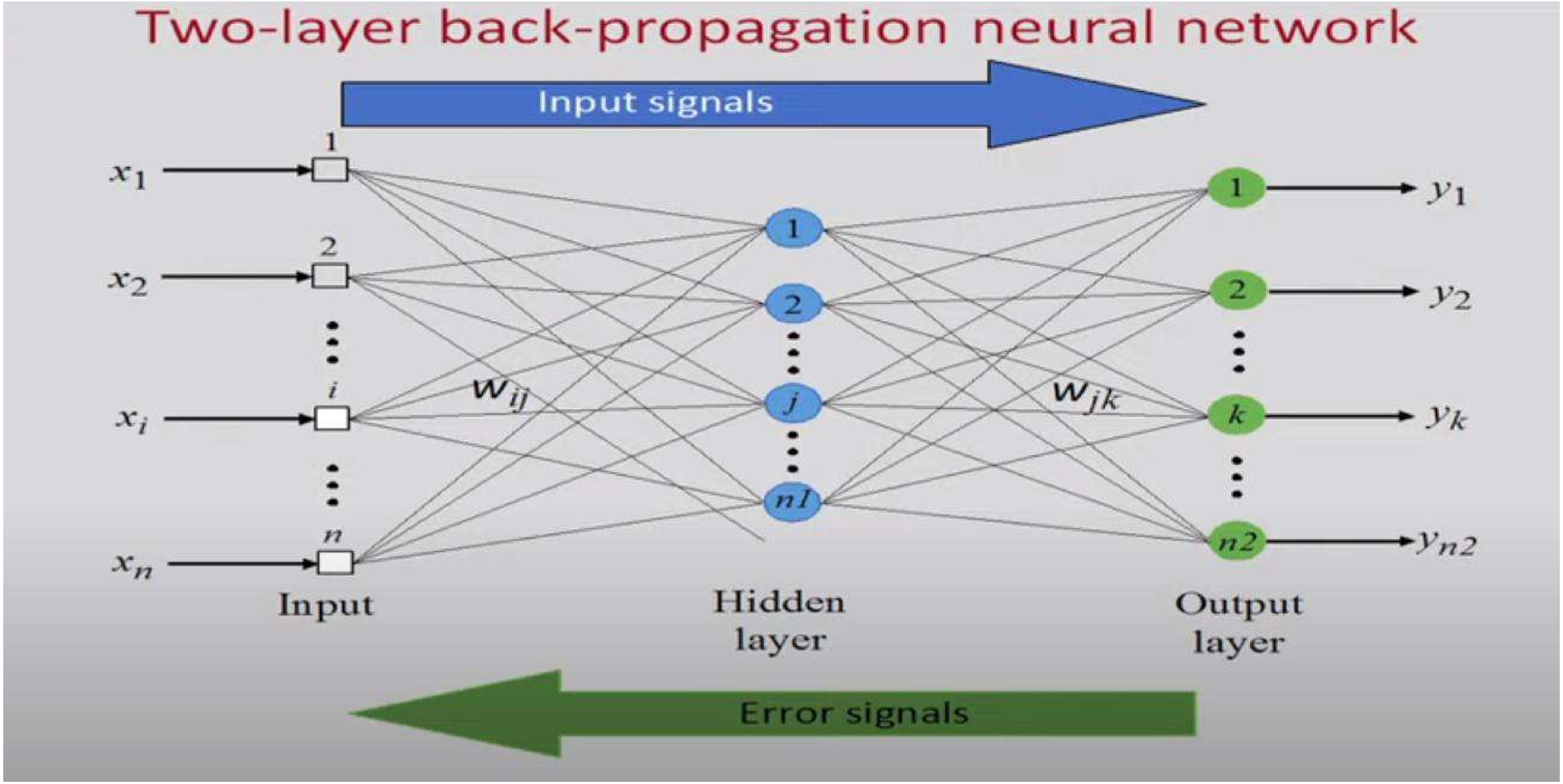


Power/Expressiveness of Multilayer Networks

- Can represent interactions among inputs
- Two layer networks can represent any Boolean function, and continuous functions (within a tolerance) as long as the number of hidden units is sufficient and appropriate activation functions used
- Learning algorithms exist, but weaker guarantees than perceptron learning algorithms

Multilayer Network




Two-layer back-propagation neural network
Input signals
x_1
x_2
x_i
x_n
1
2
i
n
w_{ij}
w_{jk}
1
2
j
n1
1
2
k
n2
y_1
y_2
y_k
y_{n2}
Input
Hidden layer
Output layer
Error signals

Representation Capability of NNs

- Single layer nets have limited representation power (linear separability problem). Multi-layer nets (or nets with non-linear hidden units) may overcome linear inseparability problem.
- Every Boolean function can be represented by a network with a single hidden layer.
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers.

Derivation

- For one output neuron, the error function is

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

- For each unit j, the output o_j is defined as

$$o_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

The input net_j to a neuron is the weighted sum of outputs o_k of previous n neurons.

- Finding the derivative of the error:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

For one output neuron, the error function is $E = \frac{1}{2}(y - o)^2$

For each unit j, the output o_j is defined as

$$o_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$= \sum_{l} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial net_{z_l}} w_{jz_l}\right) \varphi(net_j)\left(1 - \varphi(net_j)\right) o_i$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

with

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - y_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron} \\ \left(\sum_{Z} \delta_{z_l} w_{jl}\right) o_j (1 - o_j) & \text{if } j \text{ is an inner neuron} \end{cases}$$

To update the weight w_{ij} using gradient descent, one must choose a learning rate η.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, do

- For each training example, do
  - Input the training example to the network and compute the network outputs
  - For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(y_k - o_k)$$

  - For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}, \delta_k,$$

  - Update each network weight w_i, j

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

x_d = input

y_d = target output

o_d = observed unit output

w_{ij} = wt from i to j

Backpropagation

- Gradient descent over entire network weight vector
- Can be generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
- May include weight momentum α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Training may be slow.
- Using network after training is very fast

Training practices: batch vs. stochastic
vs. mini-batch gradient descent

- **Batch gradient descent**:
  1. Calculate outputs for the entire dataset
  2. Accumulate the errors, back-propagate and update

Too slow to converge
Gets stuck in local minima

- **Stochastic/online gradient descent**:
  1. Feed forward a training example
  2. Back-propagate the error and update the parameters

Converges to the solution faster
Often helps get the system out of local minima

- **Mini-batch gradient descent**:

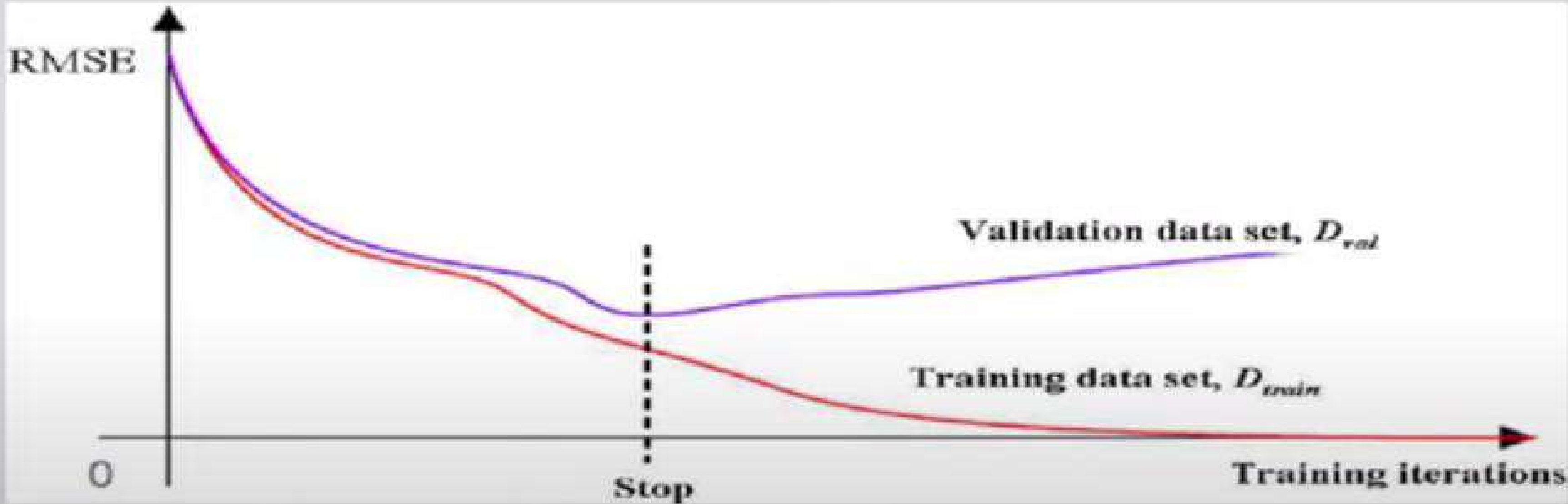
Learning in *epochs*
Stopping

- Train the NN on the entire training set over and over again
- Each such episode of training is called an "epoch"

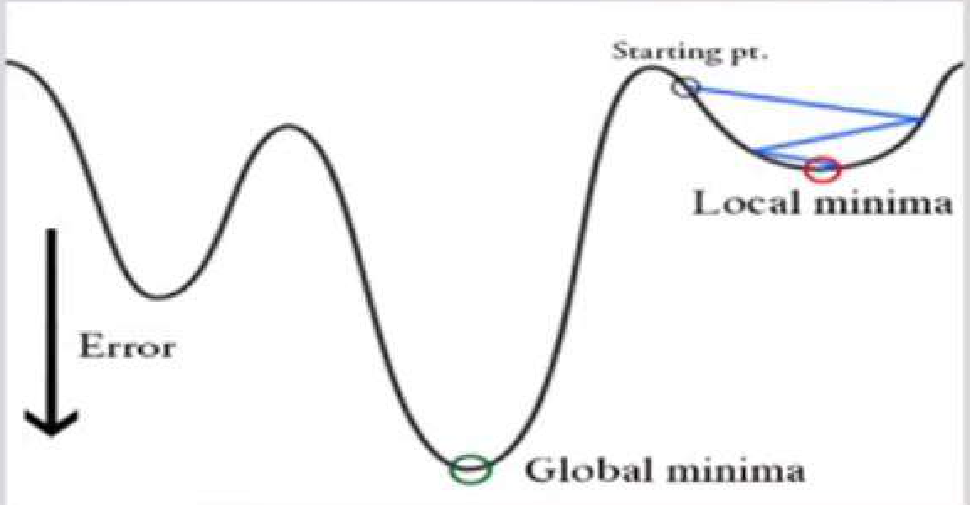
Stopping

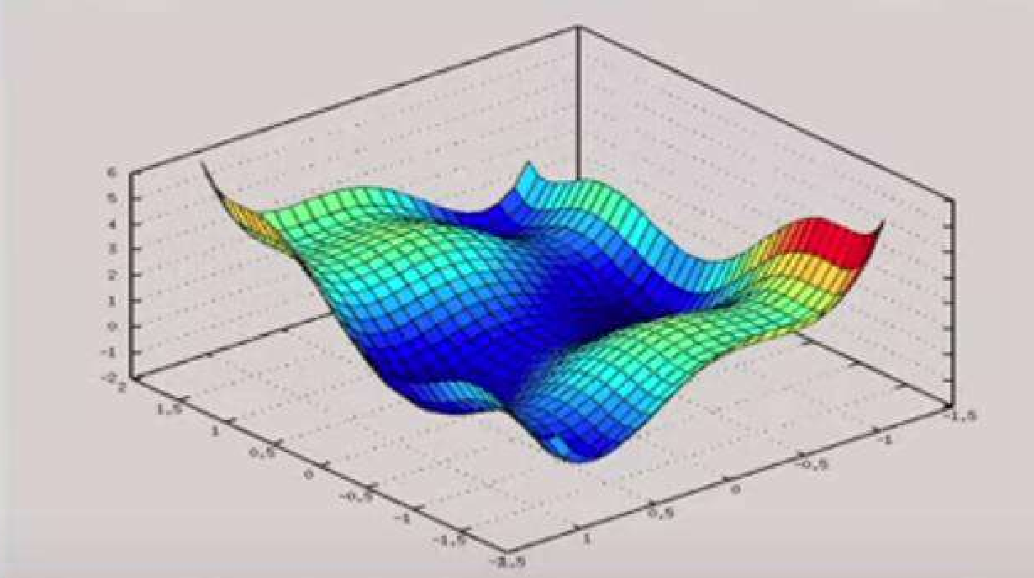
1. Fixed maximum number of epochs: most naïve
2. Keep track of the training and validation error curves.

Overfitting in ANNs



Local Minima





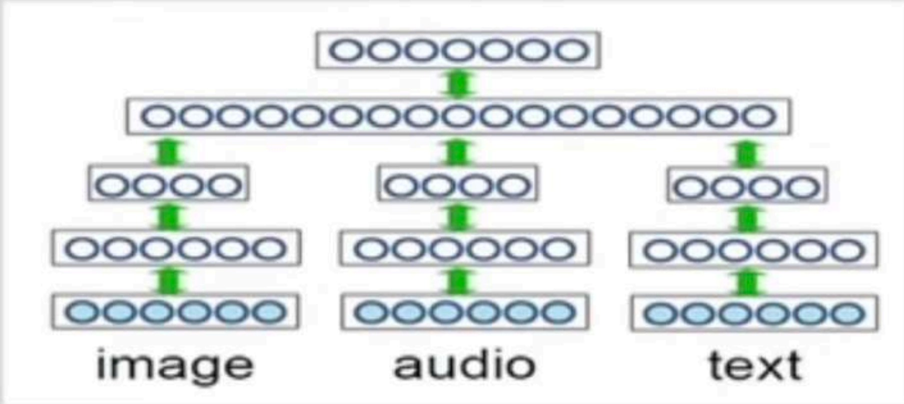
- NN can get stuck in local minima for small networks.
- For most large networks (many weights) local minima rarely occurs.
- It is unlikely that you are in a minima in every dimension simultaneously.

ANN

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizes sum of squared training errors
- Can add a regularization term (weight squared)
- Local minima
- Overfitting

Deep Learning

- Breakthrough results in
  - Image classification
  - Speech Recognition
  - Machine Translation
  - Multi-modal learning

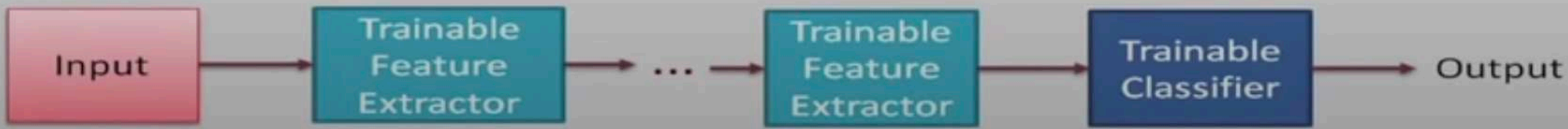


Deep Neural Network

- Problem: training networks with many hidden layers doesn't work very well
- Local minima, very slow training if initialize with zero weights.
- Diffusion of gradient.

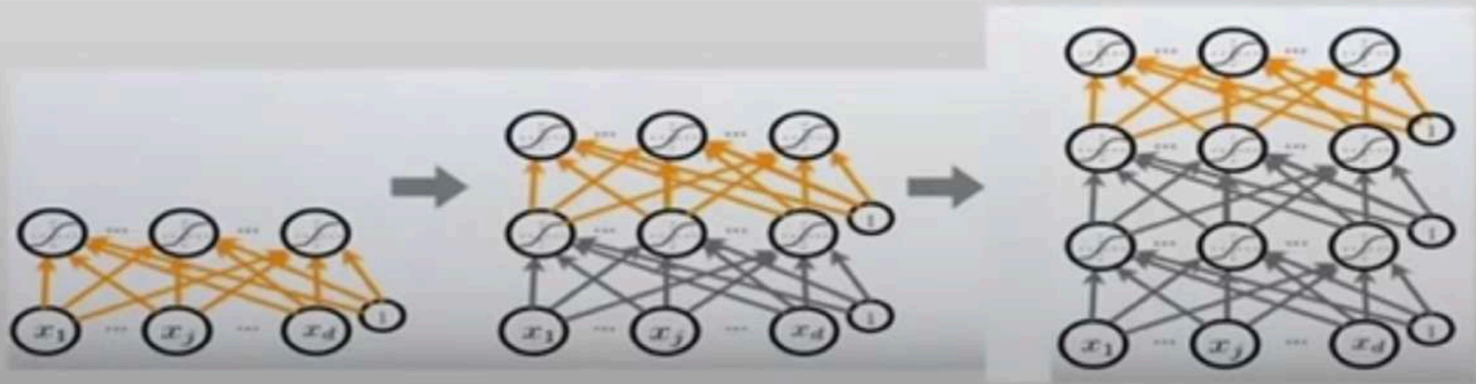
Hierarchical Representation

- Hierarchical Representation help represent complex functions.
- NLP: character ->word -> Chunk -> Clause -> Sentence
- Image: pixel > edge -> texton -> motif -> part -> object
- Deep Learning: learning a hierarchy of internal representations
- Learned internal representation at the hidden layers (trainable feature extractor)
- Feature learning



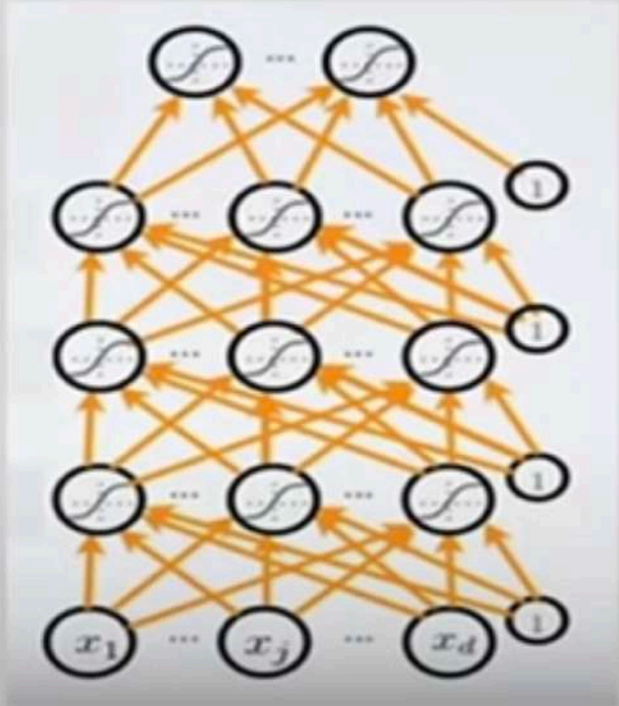
Unsupervised Pre-training

- We will use greedy, layer wise pre-training
  - Train one layer at a time
  - Fix the parameters of previous hidden layers
  - Previous layers viewed as feature extraction
- find hidden unit features that are more common in training input than in random inputs



Tuning the Classifier

- After pre-training of the layers
  - Add output layer
  - Train the whole network using supervised learning (Back propagation)



Deep neural network

- Feed forward NN
- Stacked Autoencoders (multilayer neural net with target output = input)
- Stacked restricted Boltzmann machine
- Convolutional Neural Network

A Deep Architecture: Multi-Layer Perceptron

Output Layer

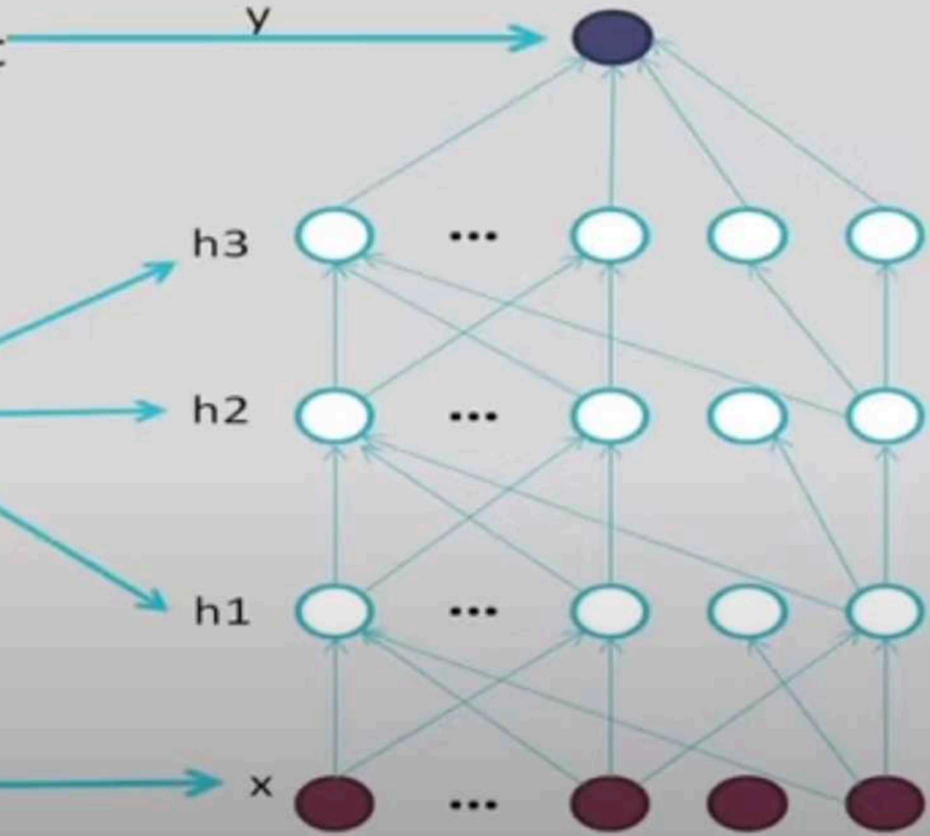
Here predicting a supervised target

Hidden layers

These learn more abstract representations as you head up

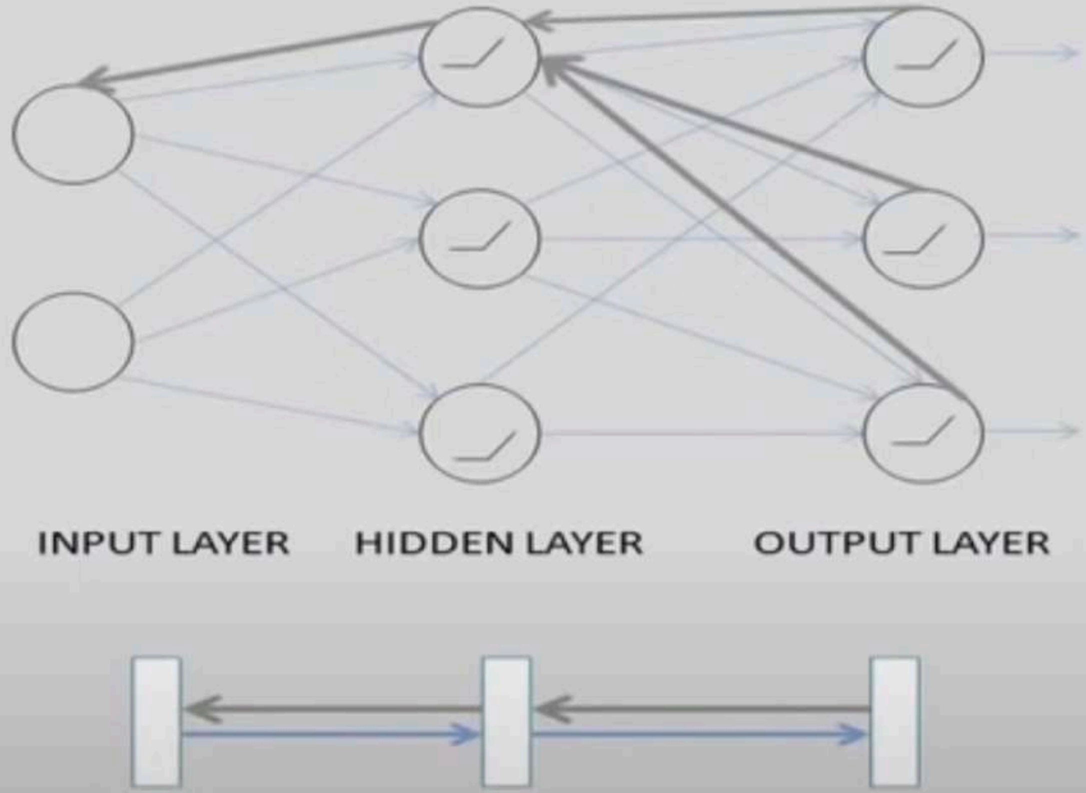
Input layer

Raw sensory inputs



A Neural Network

- Training : Back Propagation of Error
  - Calculate total error at the top
  - Calculate contributions to error at each step going backwards
  - The weights are modified as the error is propagated



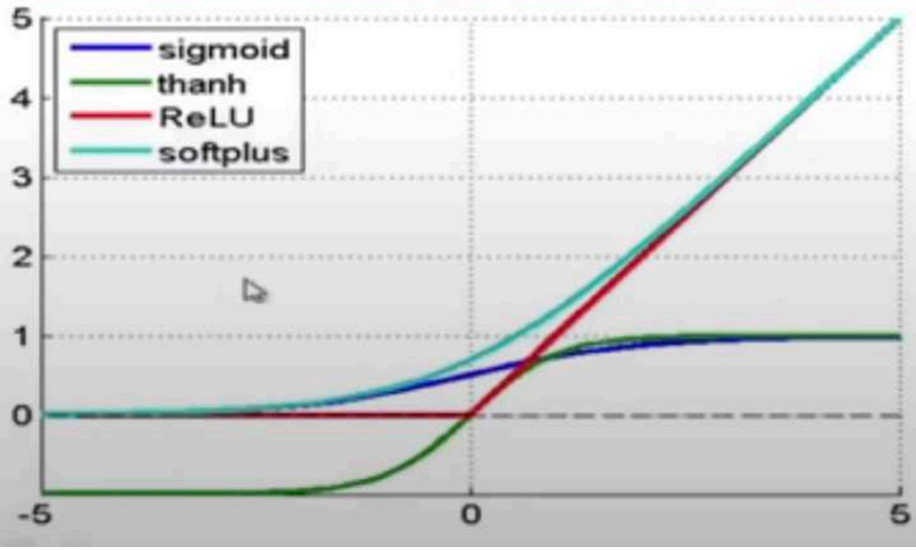
Training Deep Networks

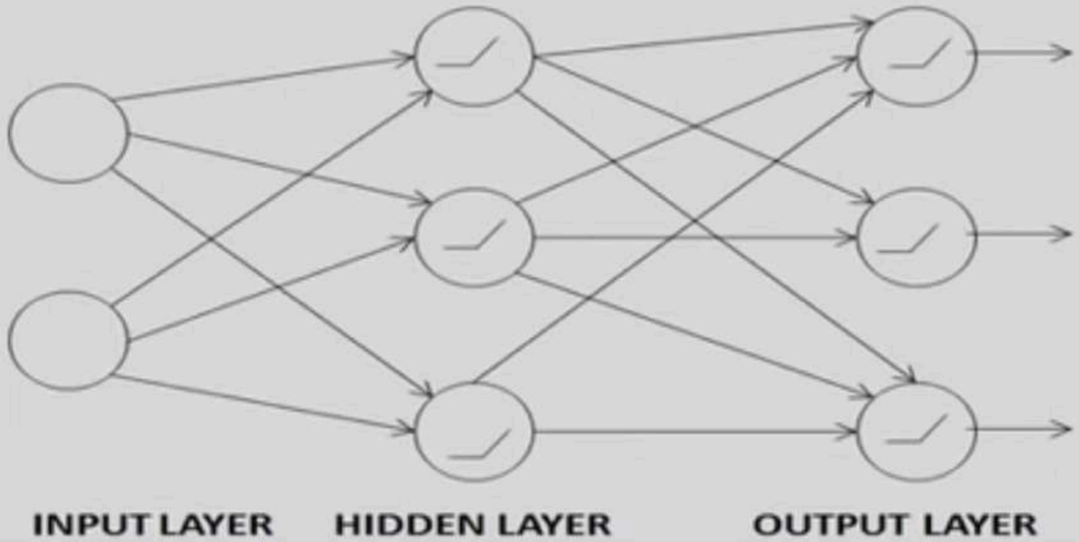
- Difficulties of supervised training of deep networks
  1. Early layers of MLP do not get trained well
     - Diffusion of Gradient – error attenuates as it propagates to earlier layers
     - Leads to very slow training
     - the error to earlier layers drops quickly as the top layers "mostly" solve the task
  2. Often not enough labeled data available while there may be lots of unlabeled data
  3. Deep networks tend to have more local minima problems than shallow networks during supervised training

Training of neural networks

- Forward Propagation :
  - Sum inputs, produce activation
  - feed-forward

Activation Functions examples

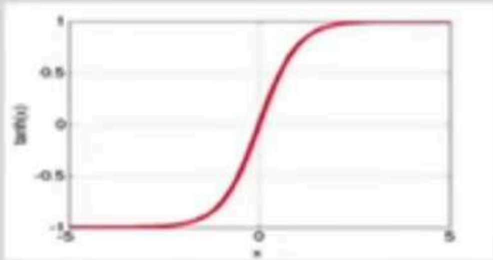


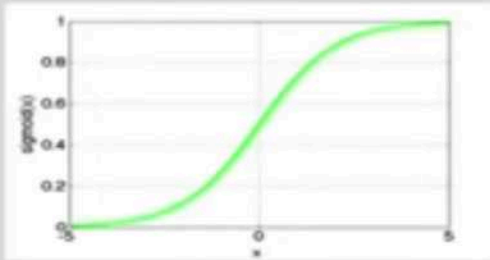


Activation Functions

Non-linearity

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



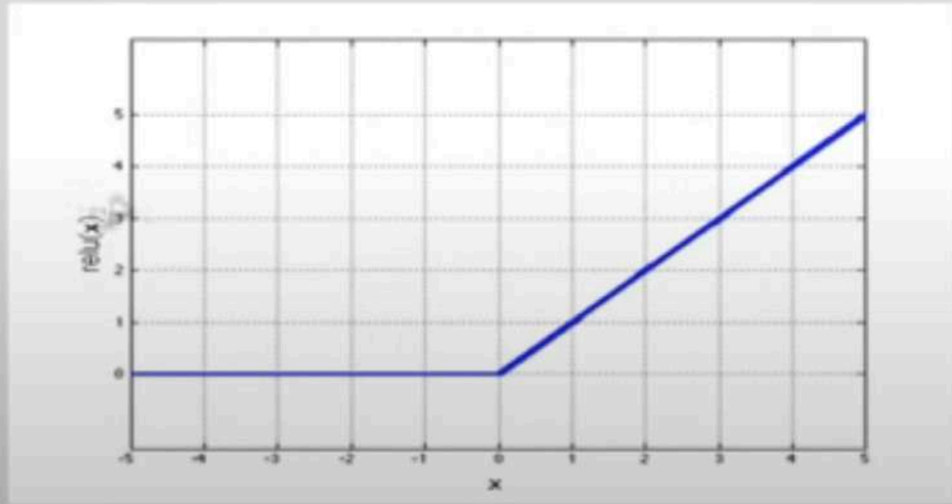


- $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$

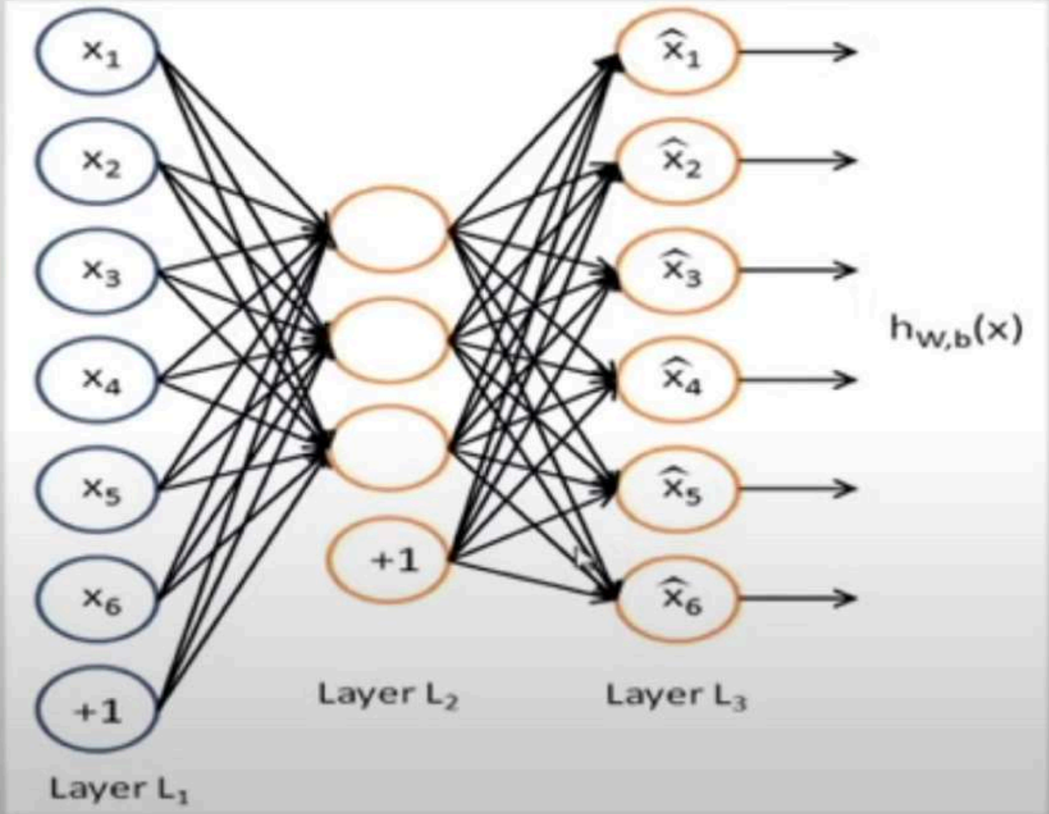
- Rectified linear
  relu(x) = max(0,x)

- Simplifies backprop
- Makes learning faster
- Make feature sparse

→ Preferred option



Autoencoder



Unlabeled training examples set $\{x^{(1)}, x^{(2)}, x^{(3)} \; ...\}$, $x^{(i)} \in \mathbb{R}^n$

Set the target values to be equal to the inputs. $y^{(i)} = x^{(i)}$

Network is trained to output the input (learn identify function).

$$h_{w,b}(x) \approx x$$

Solution may be trivial!

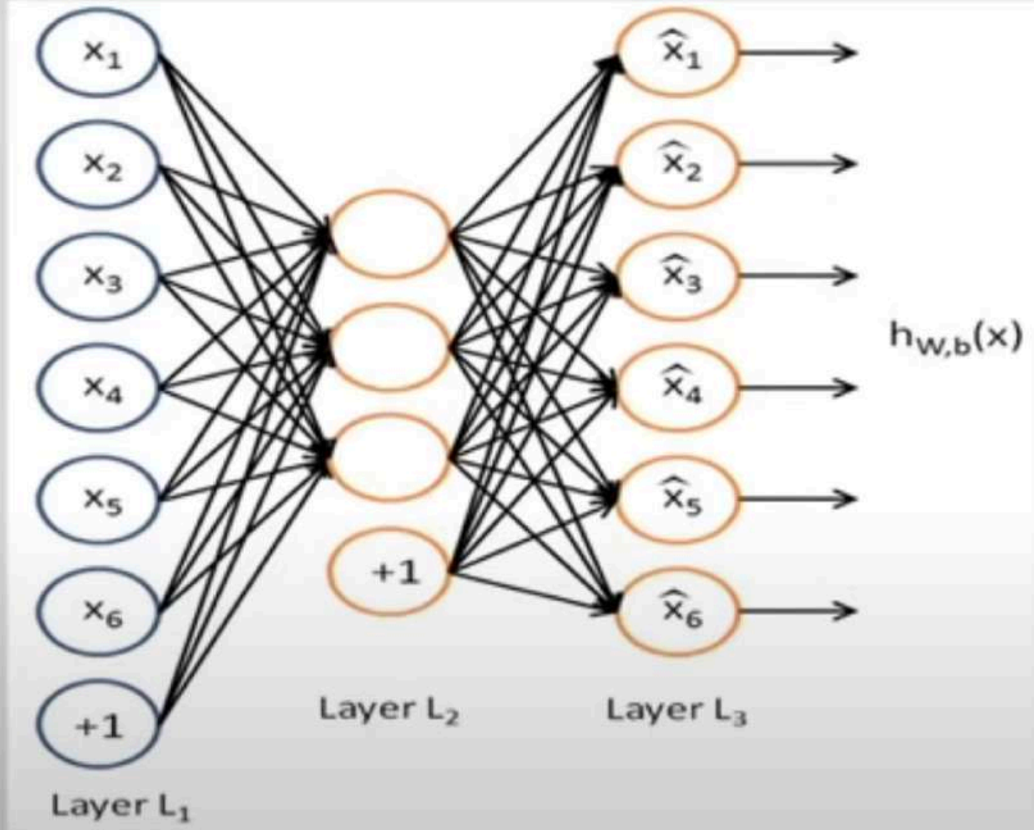
Autoencoders and sparsity

1. Place constraints on the network, like limiting the number of hidden units, to discover interesting structure about the data.
2. Impose sparsity constraint.

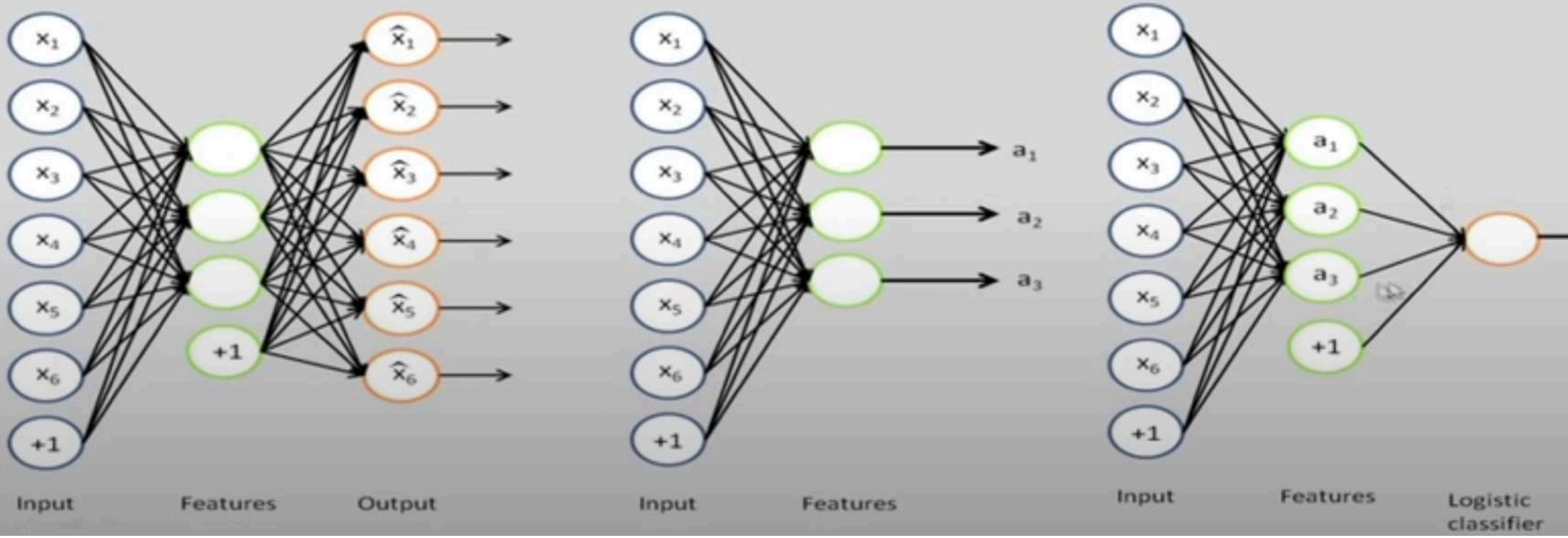
a neuron is “active” if its output value is close to 1

It is “inactive” if its output value is close to 0.

constrain the neurons to be inactive most of the time.



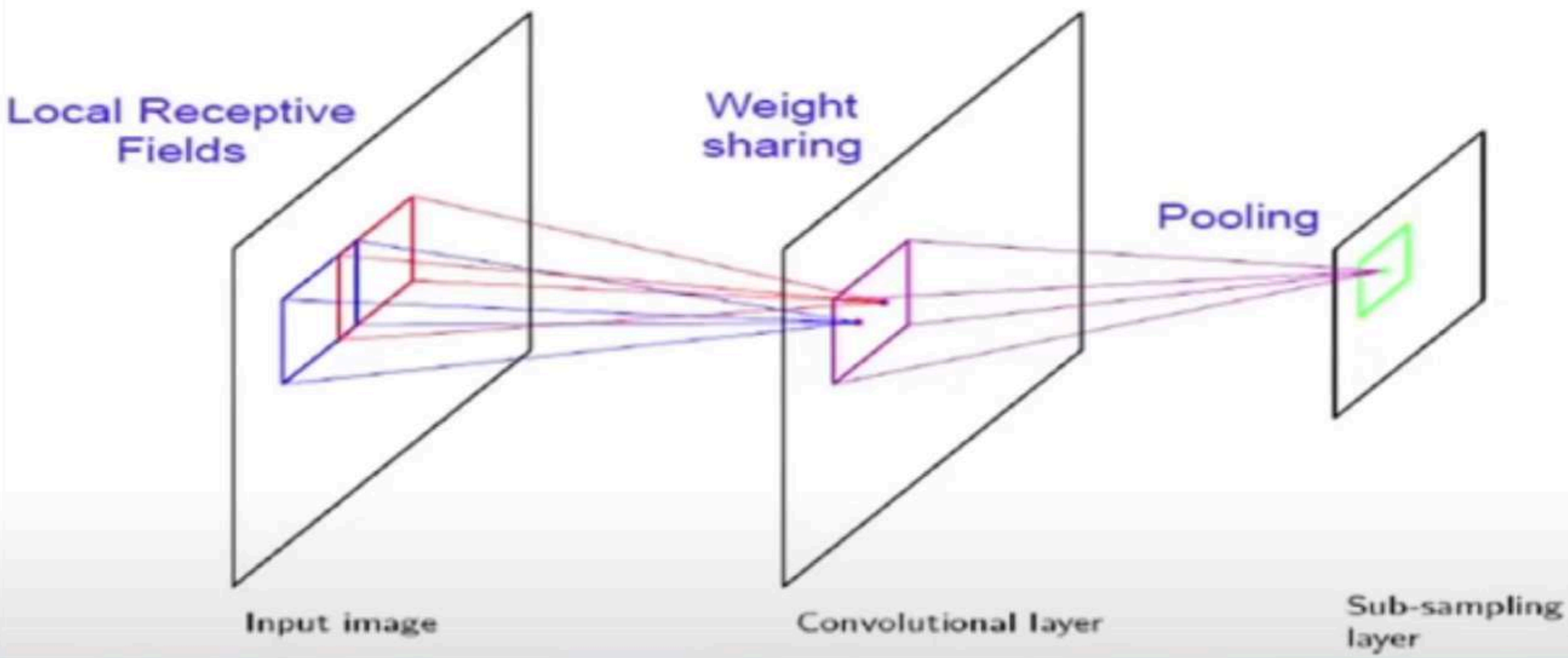
Auto-Encoders



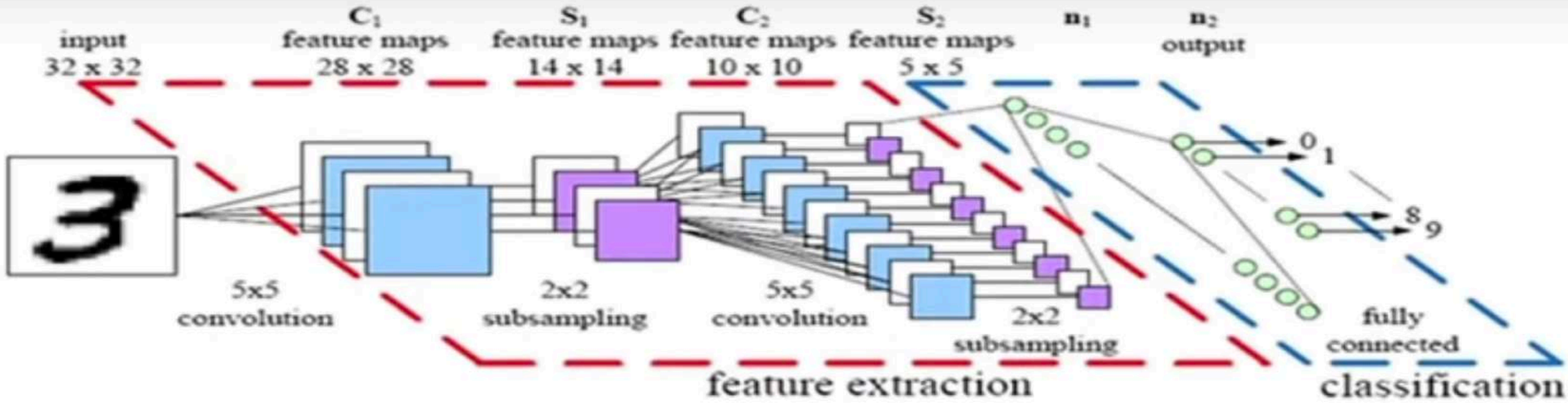
Convolutional Neural netwoks

- A CNN consists of a number of convolutional and subsampling layers.
- Input to a convolutional layer is a m x m x r image where m x m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3
- Convolutional layer will have k filters (or kernels)
- size n x n x q
- n is smaller than the dimension of the image and,
- q can either be the same as the number of channels r or smaller and may vary for each kernel

Convolutional Neural Networks

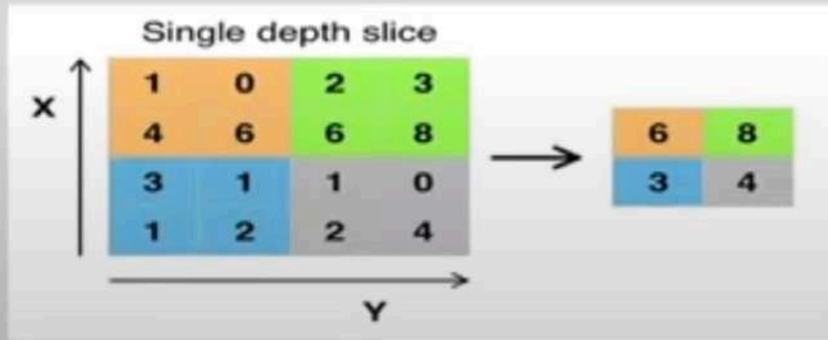


<u>Convolutional layers</u> consist of a rectangular grid of neurons
Each neuron takes inputs from a rectangular section of the previous layer
the weights for this rectangular section are the same for each neuron in the convolutional layer.



Pooling: Using features obtained after Convolution for Classification

The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block : max, average, etc.



CNN properties

- CNN takes advantage of the sub-structure of the input
- Achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features.

- CNN are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.