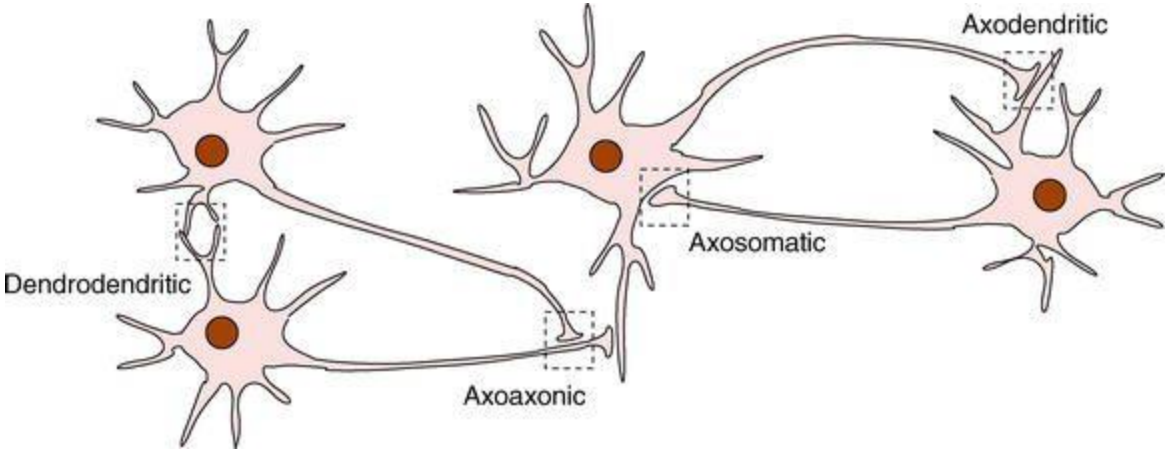
Neural Networks

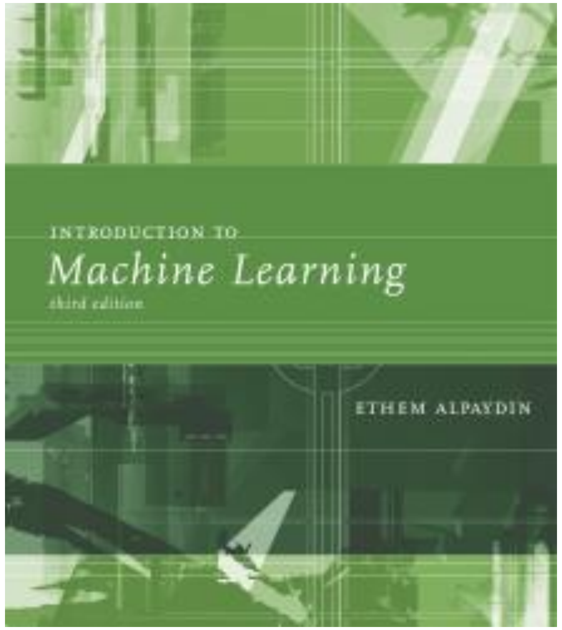
- Simulate how the brain does reasoning
- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: 10^{10}
- Large connectitivity: 10^5
- Parallel processing
- Distributed computation/memory
- Robust to noise, failures



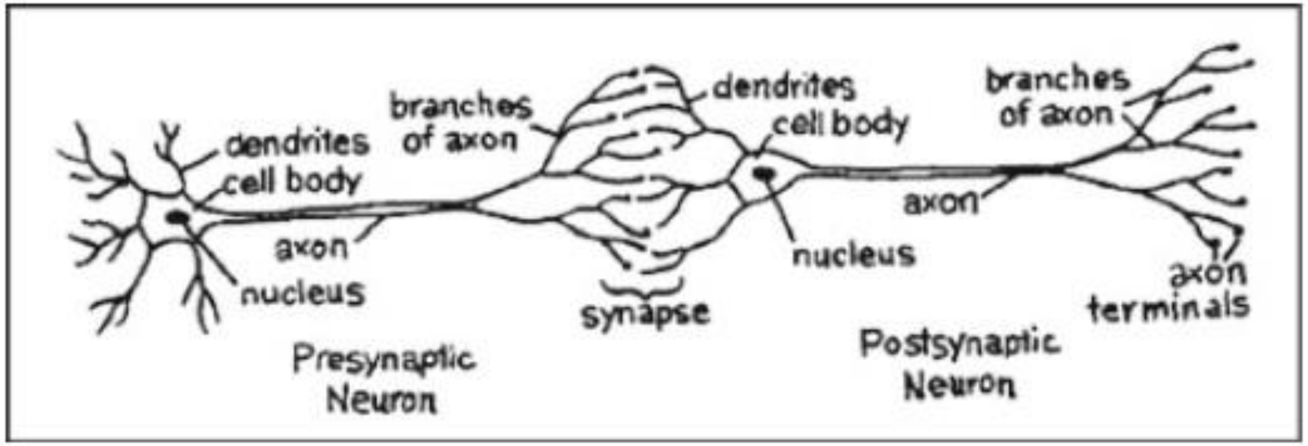
- Credit: Some slides from E. Alpaydin. Introduction to Machine Learning. MIT Press, 2014. Ref. [2].

alpaydin@boun.edu.tr

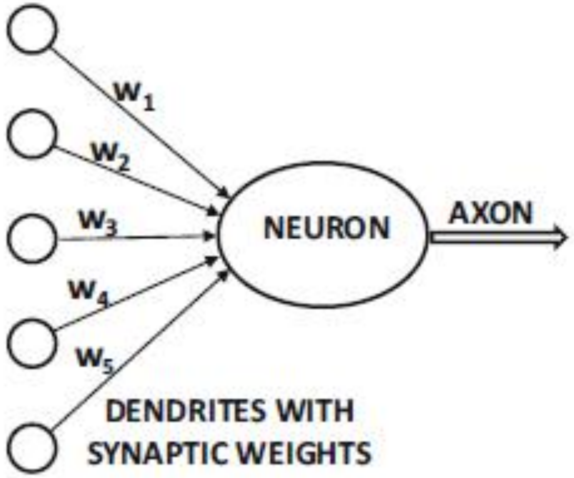
http://www.cmpe.boun.edu.tr/~ethem/i2ml3e



From Biological to Artificial NN



(a) Biological neural network



(b) Artificial neural network

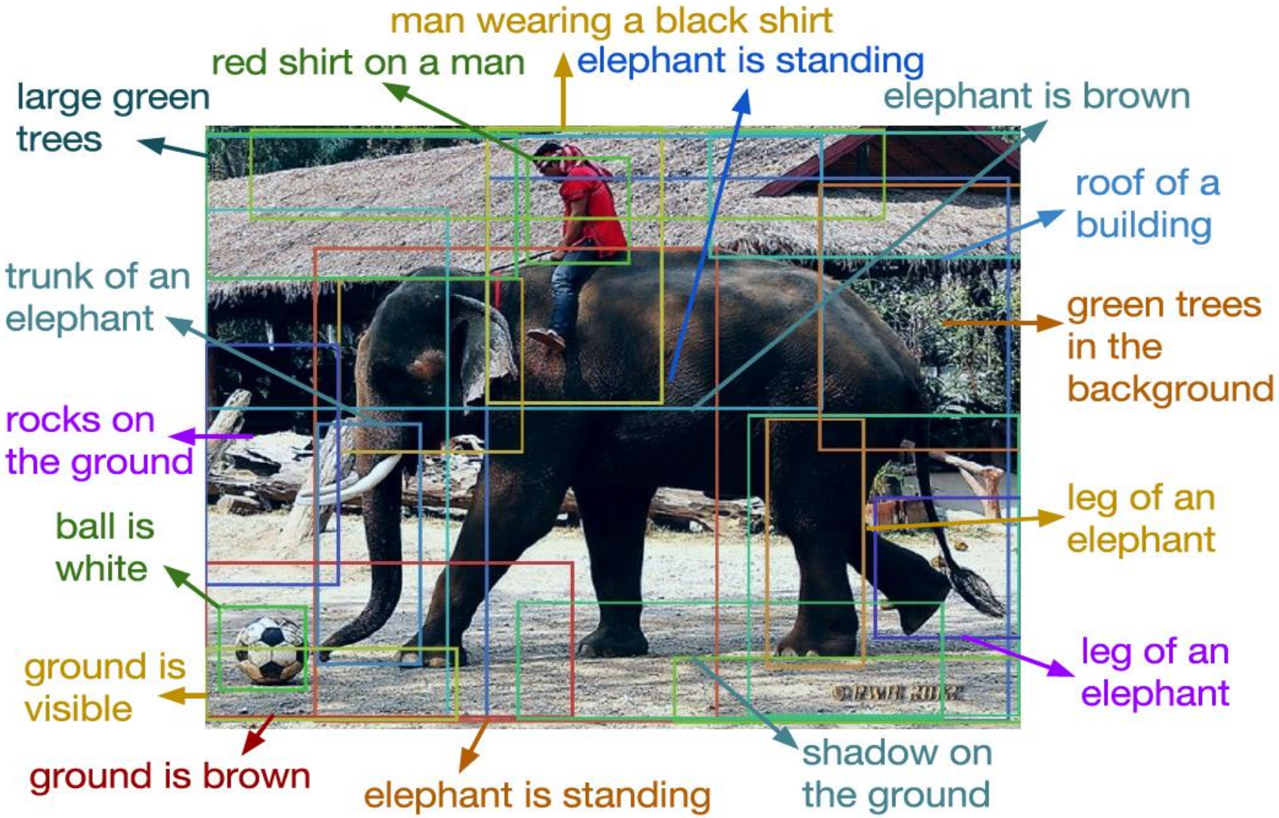
- Image from ref. [1]

Deep Learning - Recap

- Neural networks
  - Not a new theory
  - Date back from 1940s
  - McCulloch/Rosenblatt
- Emerged in the 1980s:
  - multi-layer perceptron
  - Convolutional NN
- Declined due to emergence of SVM
- Recently emerged due to:
  - Big data
  - Efficient hardware (GPUs, multi-processing, multicore)
  - Applications:
    - many

Example:

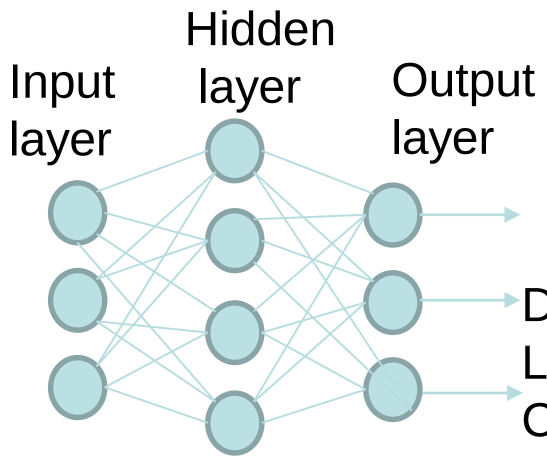
- DenseCap: From images to natural language
- Classification, object detection, to full sentences in natural language
- Uses convolutional and recurrent NN



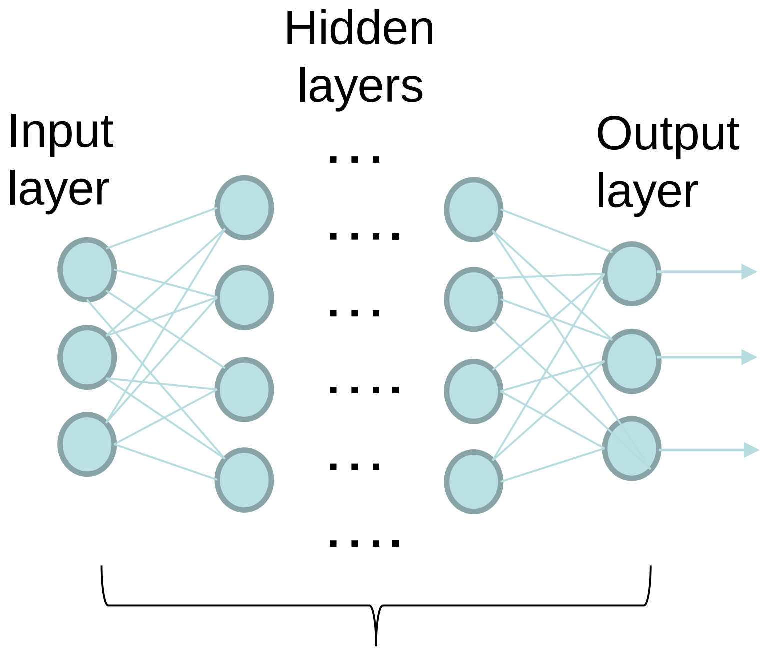
J. Johnson et al., DenseCap, IEEE CVPR 2016

Deep Learning - Artificial Neural Networks

- Deep Learning is based on deeper artificial neural networks



DenseCap: Fully Convolutional Localization Networks for Dense Captioning



Deeper means increased number of hidden layers

- Due to the advancement in computing resources (CPUs, GPUs, memory, etc..) ➔ Deep learning becomes feasible.
- More hidden layers ➔ More optimization & learning ➔ revealing more of the intrinsic features.

Neural Networks

Tasks NN can do:

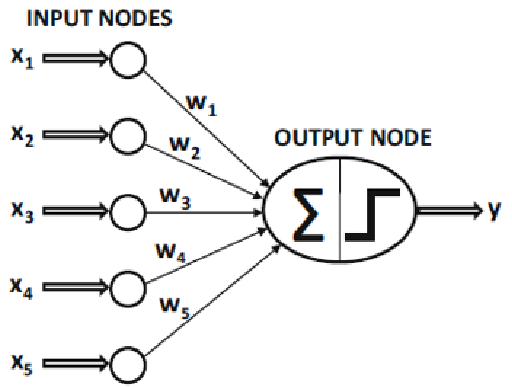
- Classification - Prediction
  - One-class
  - Two-class
  - Multi-class
- Regression
  - Linear
  - Nonlinear
  - Multiple linear
- Dimensionality Reduction (DR)
- Clustering:
  - Not really – but done through DR
- Reinforcement Learning

Types of NN (main):

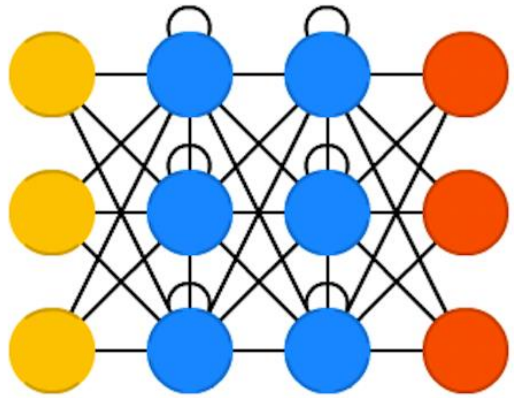
- Single layer vs Multilayer
- Recurrent NN (RNN)
- Auto Encoder (AE)
- Convolutional NN (CNN)
- Markov Chain (MC)
- Hopfield Network (HN)
- Boltzmann Machine (BM)
- Generative Adversarial Network (GAN)
- Graph Convolutional Network (GCN)

NN – Main Types

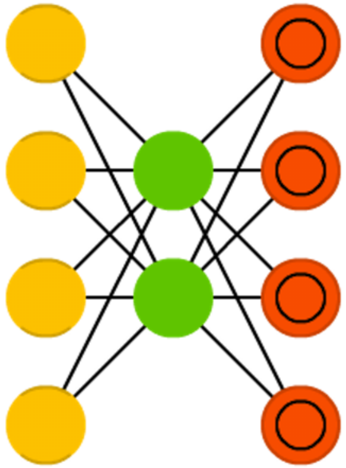
Perceptron



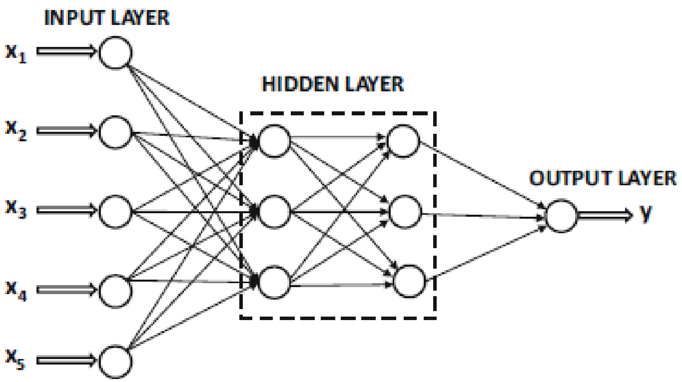
RNN



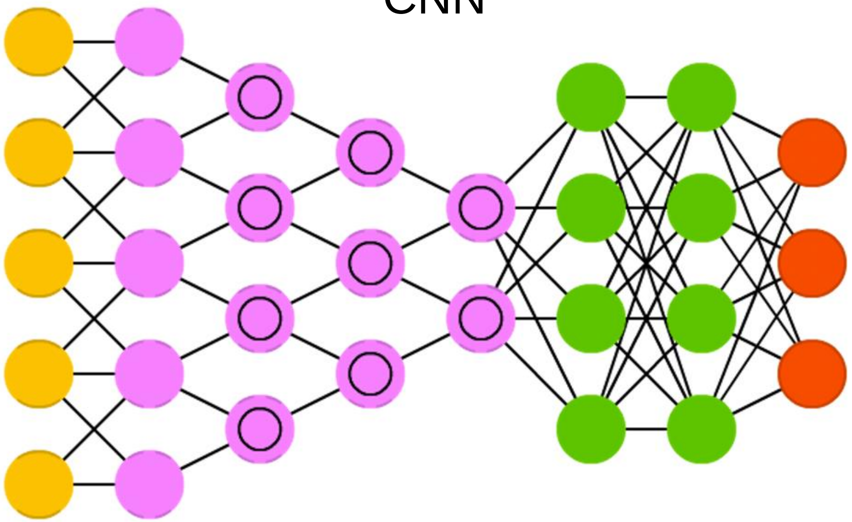
Autoencoder



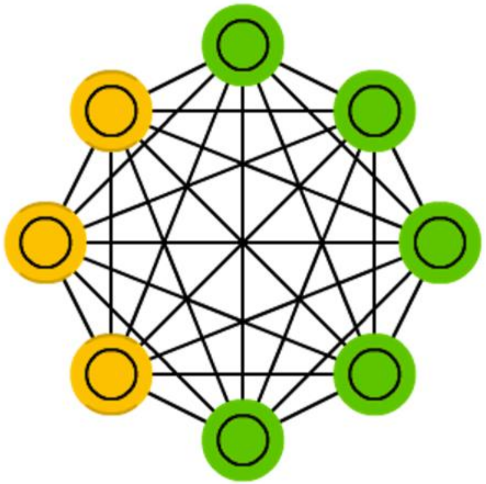
Multilayer



CNN



Boltzmann machine



Source: ref. [1] and http://www.asimovinstitute.org/neural-network-zoo/ ref. [4]

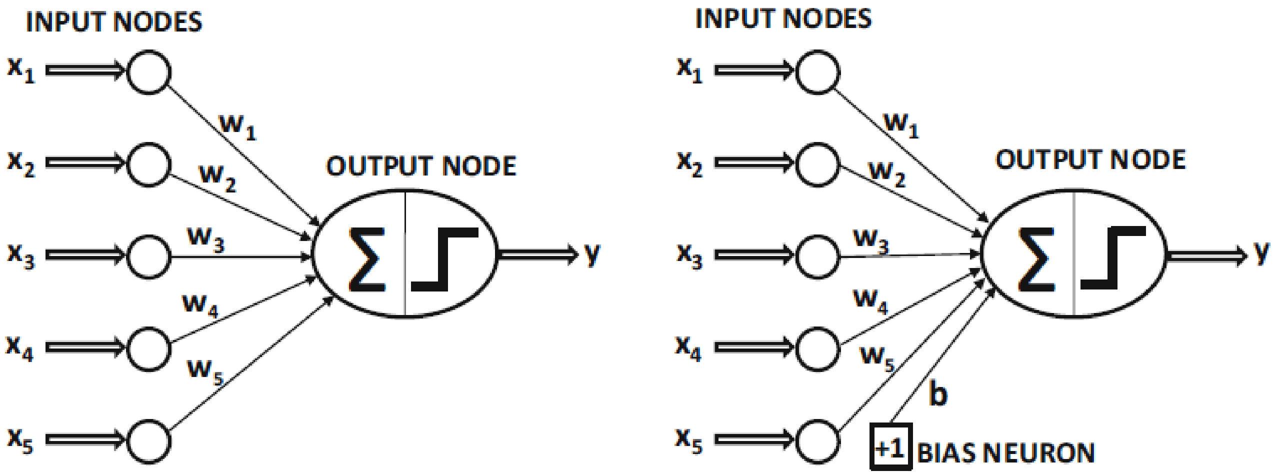
Single Layer NN

- Input:
  - $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^d$
  - Each $\mathbf{x}_i$ belongs to a class $y = \{-1, +1\}$
- Given $\mathbf{x}$ output a value $\hat{y}$ based on a linear function:
  - without bias: $y = \mathbf{w}^t\mathbf{x}$
  - with bias: $y = \mathbf{w}^t\mathbf{x} + b$
- Aim: minimize the prediction error

$$E(\mathbf{X}) = y - \hat{y}$$

Classification:

$$\hat{y} = sign\{\mathbf{w}^t\mathbf{x} + b\}$$



(a) Perceptron without bias (b) Perceptron with bias

Generalized Linear Classifiers

Augmented Feature Vector

- Lets change the notation a bit
- We've seen that the form of a linear classifier is:

$$g(\mathbf{x}) = y = w^t x + b$$

- we can write this function as:

$$g(x) = y = b + \sum_{i=1}^{d} w_i x_i$$

where $x_0 = 1$

- In NN, b is called "bias"

Bias: The *augmented feature vector* is:

$$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$

- Similarly, the *augmented weight vector*:

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}$$

- This implies a mapping from the *d*-dim. space onto the (*d*+1)-dim. Space
- For simplicity, the bias will be omitted in this chapter

The Perceptron Algorithm

- Find a criterion function to solve the set of inequalities $\mathbf{w}^t \mathbf{y}_i > 0$
- Solution: $J(.)$ is the number of samples "misclassified" by **w**.
- It gives a "piecewise" criterion function, for which the gradient descent algorithm can be used
- The *perceptron criterion function*:

$$J_p(\mathbf{w}) = \sum_{\mathbf{x} \in \mathbf{X}(\mathbf{w})} -\mathbf{w}^t \mathbf{x}$$

where $X(\mathbf{w})$ is the set of samples *misclassified* by **w**.

Algorithm **Batch Perceptron**

Input: A threshold θ

begin Initialize **w**, α, $k \leftarrow 0$

  repeat

    $k \leftarrow k + 1$

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{\mathbf{x} \in \mathbf{X}(\mathbf{w})} \mathbf{x}$

  until $\left|\alpha \sum_{\mathbf{x} \in \mathbf{X}(\mathbf{w})} \mathbf{x}\right| < \theta$

end

(Rosenblatt, 1958)

Activation Function

- Activation function converts output into outcome (class label, regression value, prediction, etc.)
- Sign function used in linear perceptron
- Other functions are useful in other applications

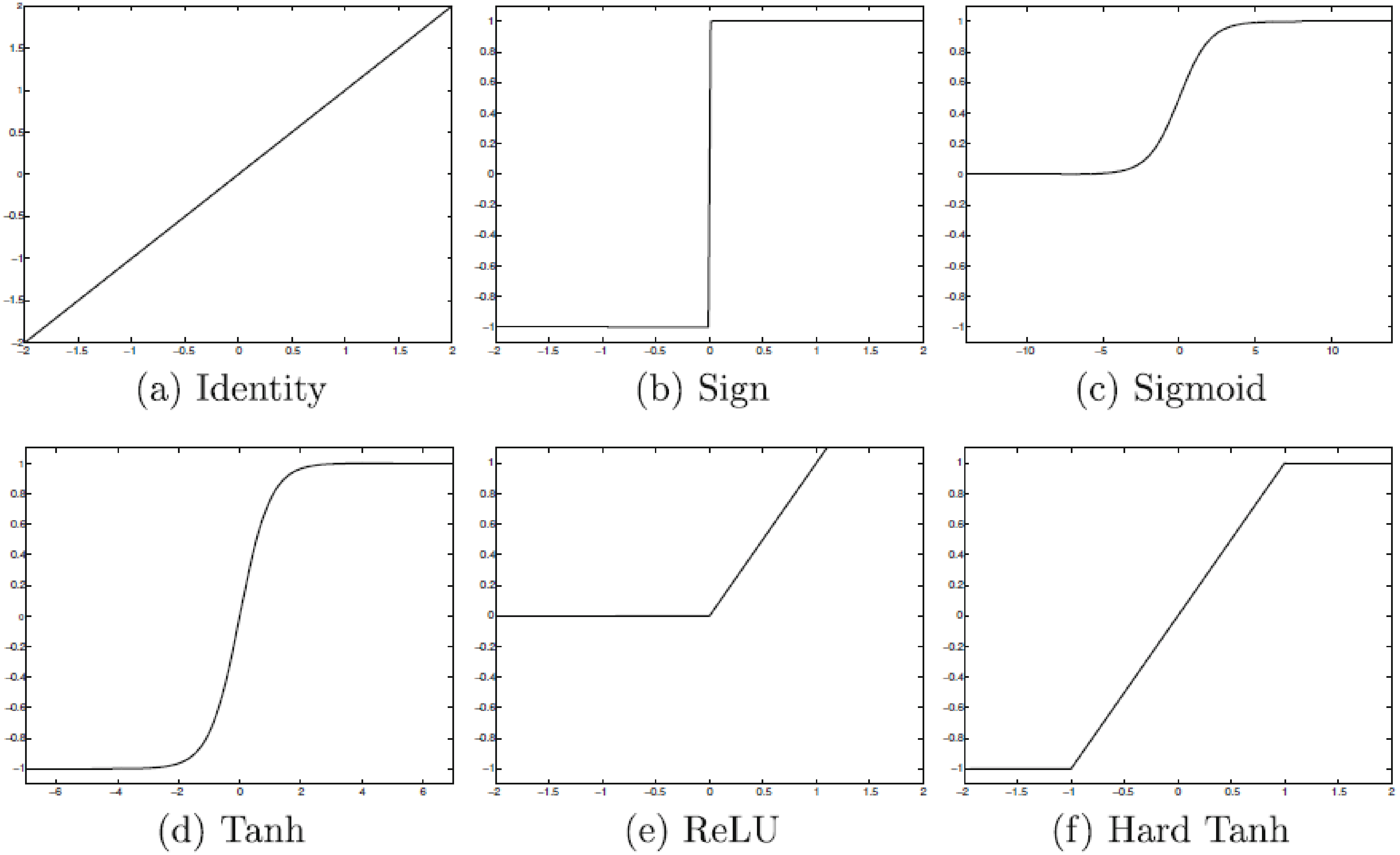
- Types:
  - Sign
  - Sigmoid
  - Tang
  - ReLU

$$\Phi(v) = \text{sign}(v) \text{ (sign function)}$$

$$\Phi(v) = \frac{1}{1+e^{-v}} \text{ (sigmoid function)}$$

$$\Phi(v) = \frac{e^{2v}-1}{e^{2v}+1} \text{ (tanh function)}$$

Various Activation Functions



(a) Identity (b) Sign (c) Sigmoid

(d) Tanh (e) ReLU (f) Hard Tanh

Ref [1]

Single Layer NN

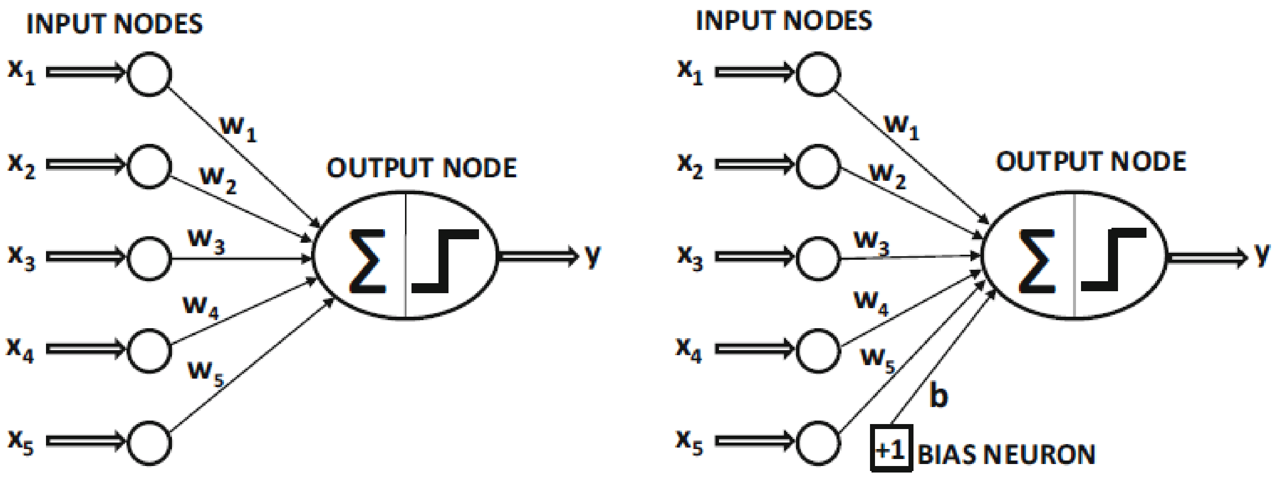
- NN include a loss function
- In case of single layer NN:

$$L = E(\mathbf{X}) = \sum_{\mathbf{x}\in\mathbf{X}} (y - \hat{y})\mathbf{x}$$

- where

$$\hat{y} = sign\{\mathbf{w}^t\mathbf{x}\}$$

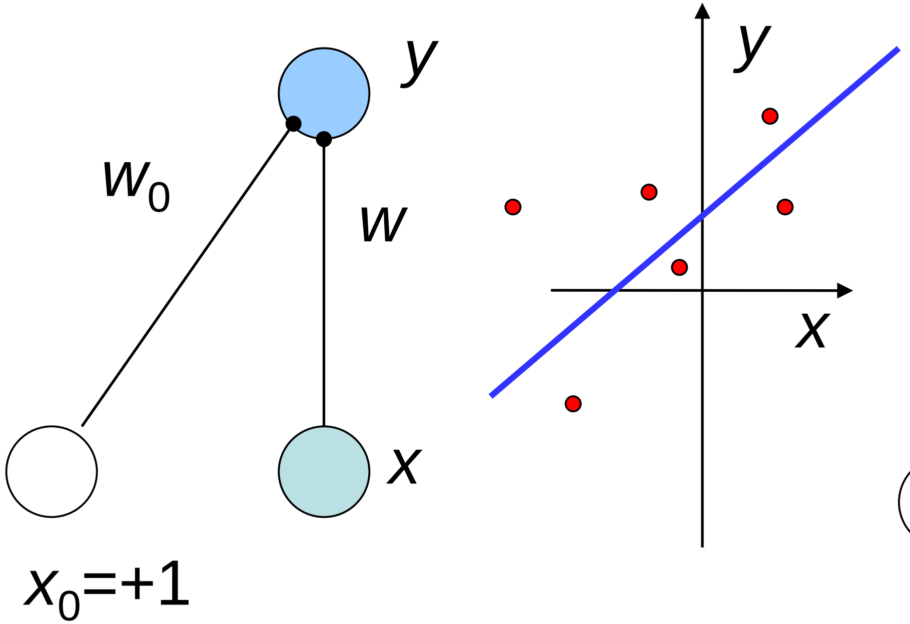
- The network can be trained using the gradient descent approach
- As we've seen, Algorithm Batch Perceptron does it this way
- Works well for linearly separable problems
- But struggles for nonlinearly separable problems

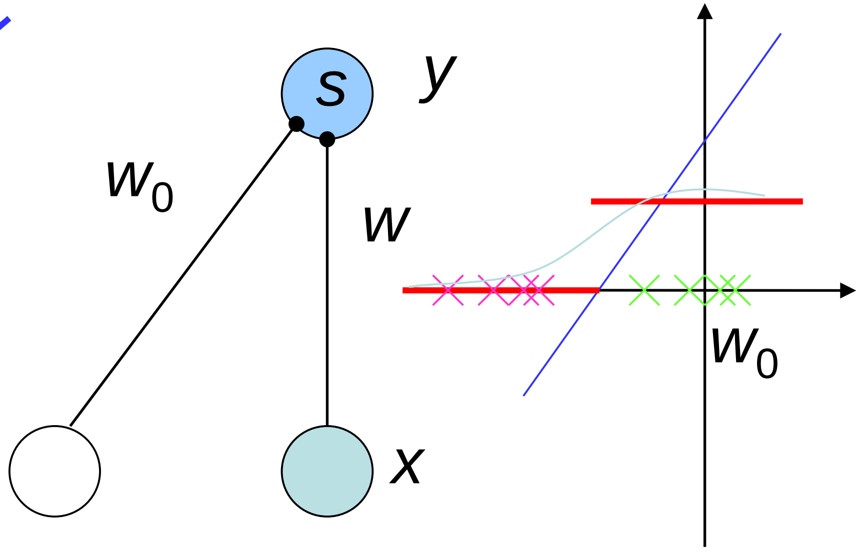


(a) Perceptron without bias (b) Perceptron with bias

What a Perceptron Does

- Regression: $y = w^t x + w_0$
- Classification: $y = 1(w^t x + w_0 > 0)$





$$y = \text{sigmoid}\,(o) = \frac{1}{1 + \exp\left[-\mathbf{w}^T \mathbf{x}\right]}$$

K Outputs

Regression:

$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = \mathbf{w}_i^t \mathbf{x}$$

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

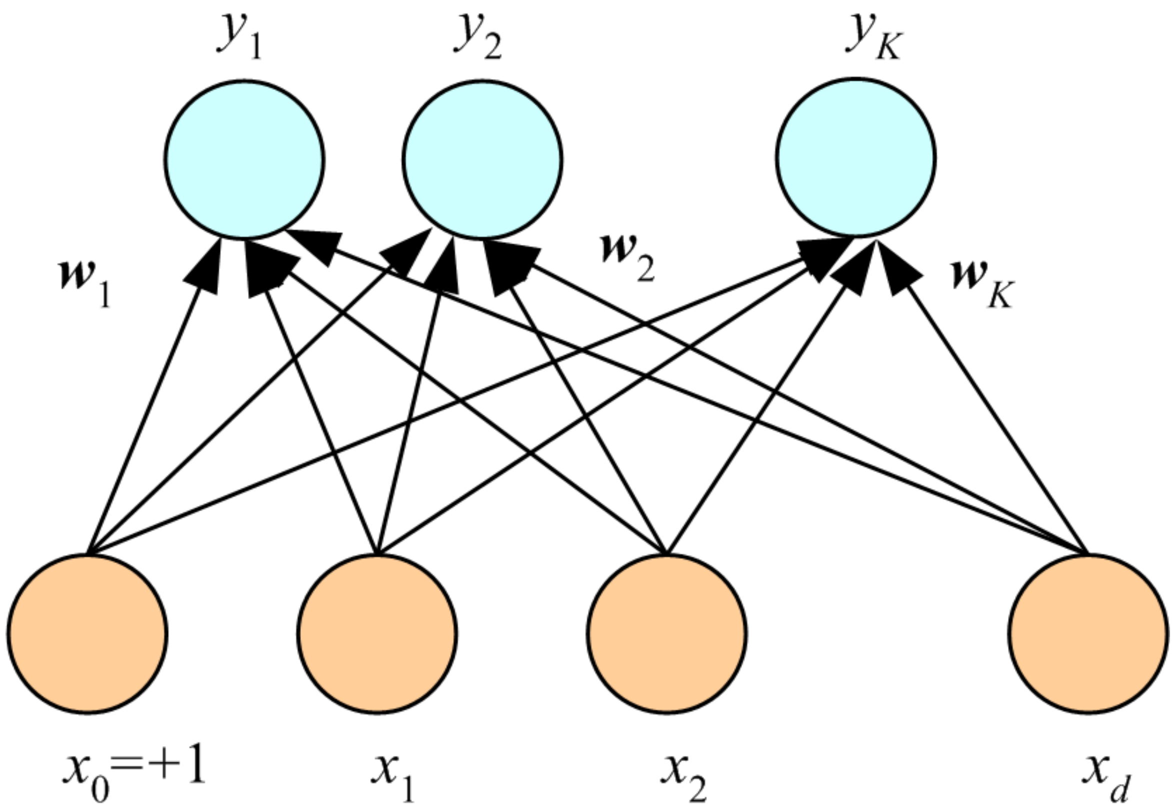
Classification:

$$o_i = \mathbf{w}_i^T \mathbf{x}$$

$$y_i = \frac{\exp o_i}{\sum_k \exp o_k}$$

choose C_i

if $y_i = \max_k y_k$



Training

- Online (instances seen one by one) vs batch (whole sample) learning:
  - No need to store the whole sample
  - Problem may change in time
  - Wear and degradation in system components
- Stochastic gradient-descent: Update after a single pattern
- Generic update rule (LMS rule):

$$\Delta w_{ij}^t = \eta \left(r_i^t - y_i^t \right) x_j^t$$

$$\text{Update} = \text{LearningFactor} \cdot \left(\text{DesiredOutput} - \text{ActualOutput} \right) \cdot \text{Input}$$

Training a Perceptron: Regression

- Regression (Linear output):

$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, r^t\right) = \frac{1}{2}\left(r^t - y^t\right)^2 = \frac{1}{2}\left[r^t - \left(\mathbf{w}^T \mathbf{x}^t\right)\right]^2$$

$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

Classification

- Single sigmoid output

$$y^t = \text{sigmoid}\left(\mathbf{w}^T\mathbf{x}^t\right)$$

$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, \mathbf{r}^t\right) = -r^t \log y^t - \left(1 - r^t\right)\log\left(1 - y^t\right)$$

$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

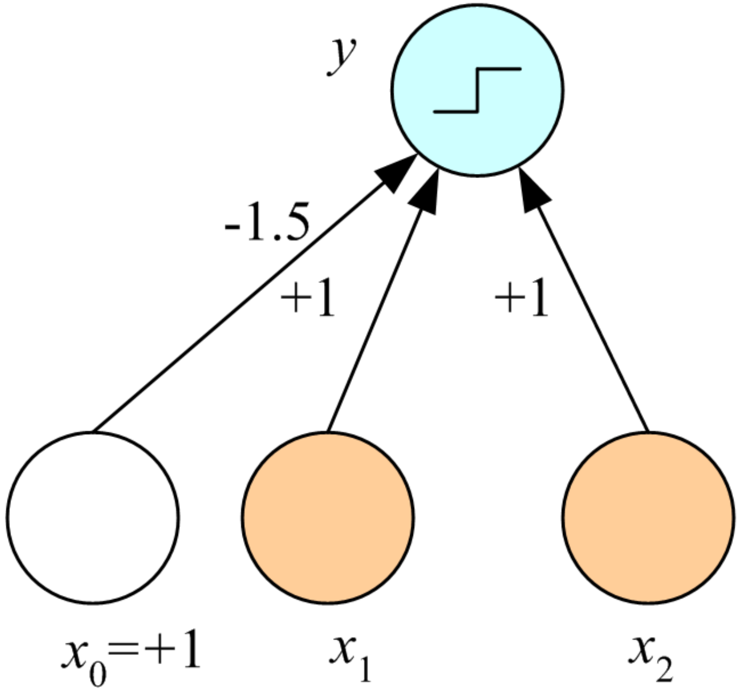
- K>2 softmax outputs

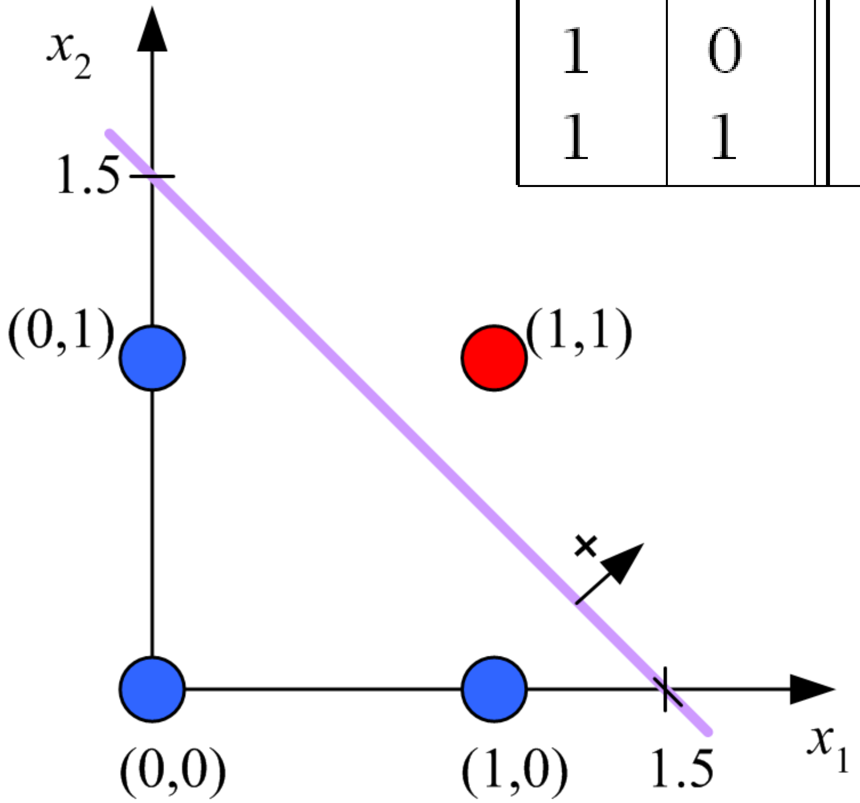
$$y^t = \frac{\exp \mathbf{w}_i^T\mathbf{x}^t}{\sum_k \exp \mathbf{w}_k^T\mathbf{x}^t} \qquad E^t\left(\{\mathbf{w}_i\}_i \mid \mathbf{x}^t, \mathbf{r}^t\right) = -\sum_i r_i^t \log y_i^t$$

$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right)x_j^t$$

Learning Boolean AND

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1



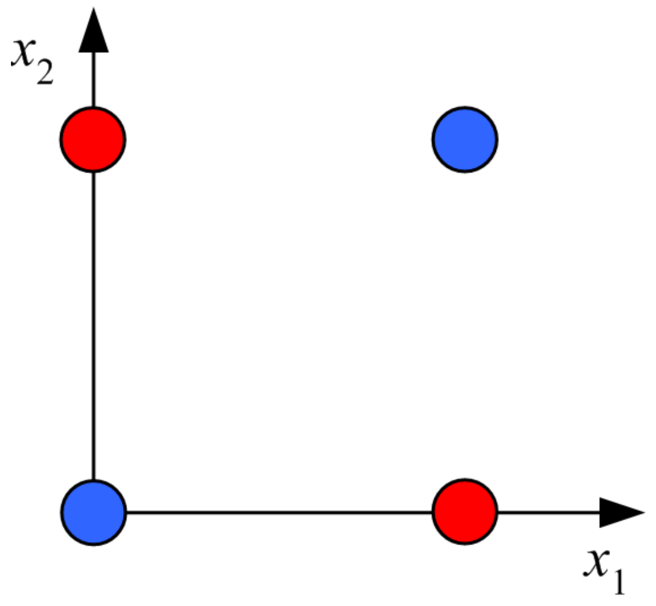


XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

- No w_0, w_1, w_2 satisfy:

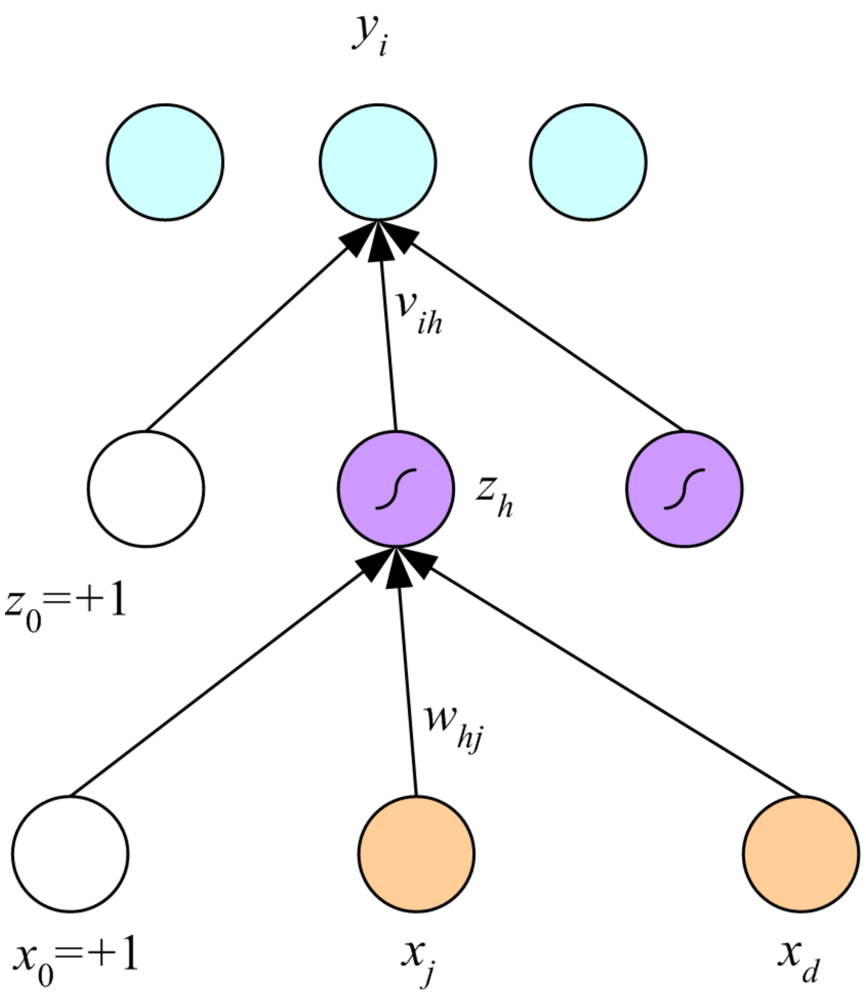
$$\begin{aligned} & & w_0 &\le 0 \\ & w_2 + & w_0 &> 0 \\ w_1 + & & w_0 &> 0 \\ w_1 + & w_2 + & w_0 &\le 0 \end{aligned}$$



(Minsky and Papert, 1969)

Recall: we can solve it using an SVM + Poly2 kernel!

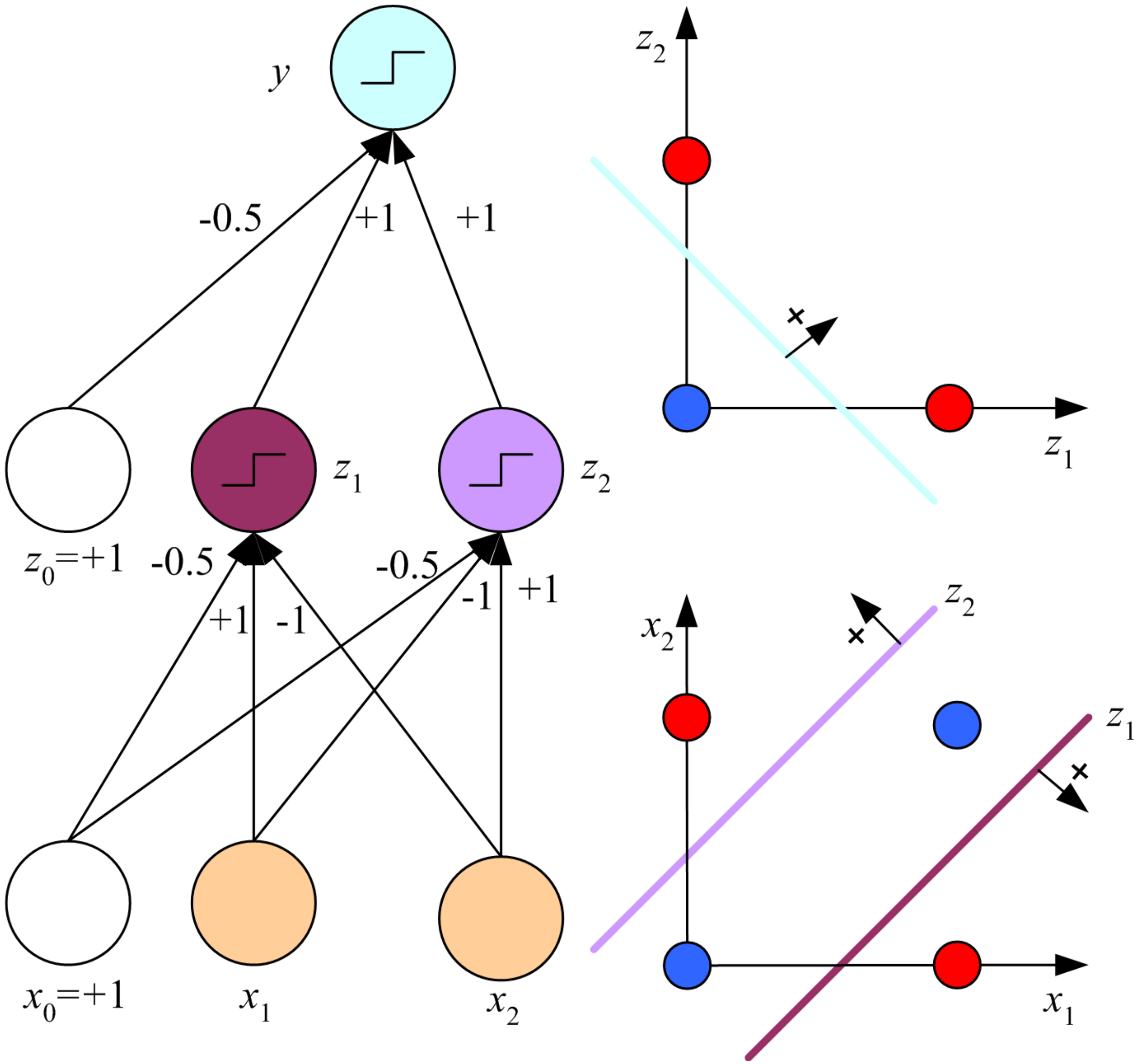
Multilayer Perceptrons



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

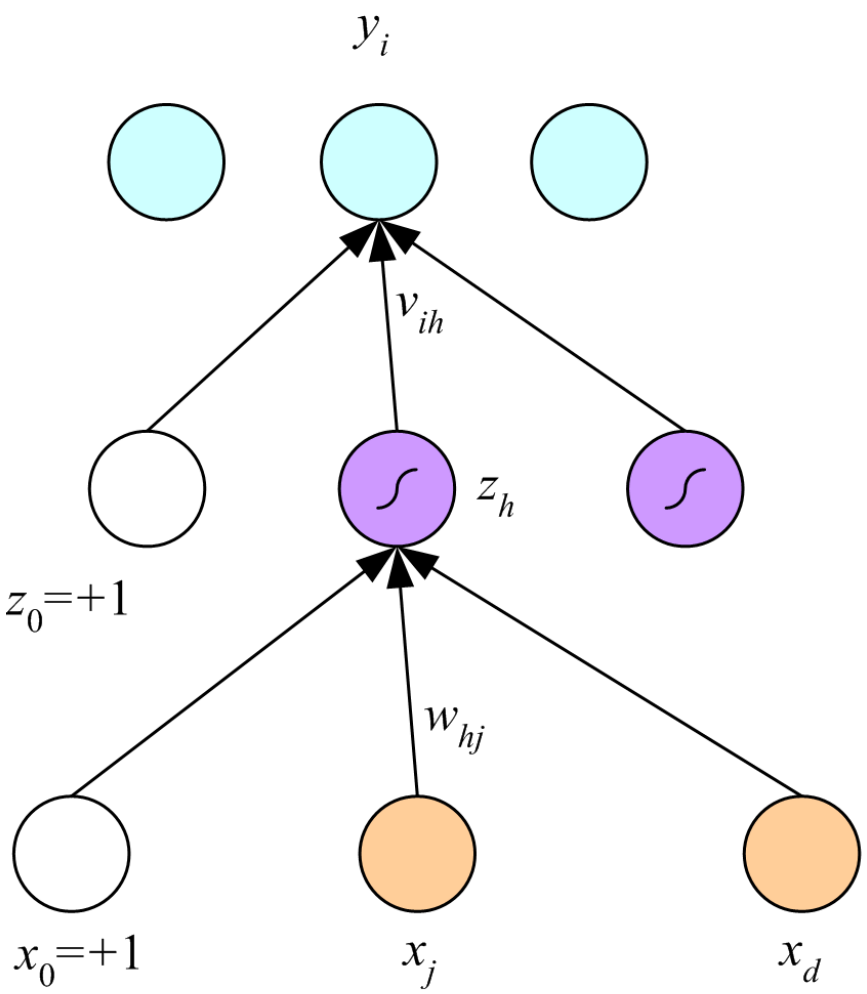
$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

(Rumelhart et al., 1986)



x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

Backpropagation



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right)$$

$$= \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

$$\boxed{\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}}$$

Regression

$$E(\mathbf{W},\mathbf{v}|\mathcal{X})=\frac{1}{2}\sum_t\left(r^t-y^t\right)^2$$

$$y^t=\sum_{h=1}^{H}v_h z_h^t+v_0$$

$$\boxed{\Delta v_h=\sum_t\left(r^t-y^t\right)z_h^t}$$

Backward

Forward

$$z_h=\boxed{\text{sigmoid}}\left(\mathbf{w}_h^T\mathbf{x}\right)$$

$$\boldsymbol{x}$$

$$\begin{aligned}\Delta w_{hj}&=-\eta\frac{\partial E}{\partial w_{hj}}\\&=-\eta\sum_t\frac{\partial E}{\partial y^t}\frac{\partial y^t}{\partial z_h^t}\frac{\partial z_h^t}{\partial w_{hj}}\\&=-\eta\sum_t-\left(r^t-y^t\right)v_h\boxed{z_h^t\left(1-z_h^t\right)}x_j^t\\&=\eta\sum_t\left(r^t-y^t\right)v_h z_h^t\left(1-z_h^t\right)x_j^t\end{aligned}$$

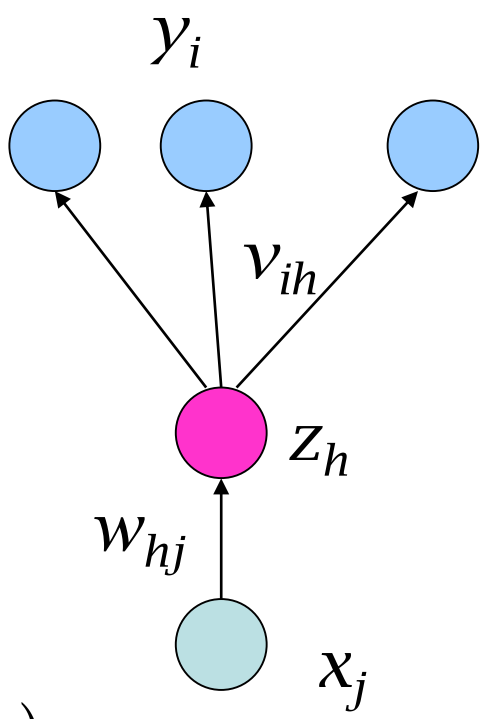
Regression with Multiple Outputs

$$E(\mathbf{W},\mathbf{V}\,|\,\mathcal{X})=\frac{1}{2}\sum_t\sum_i\left(r_i^t-y_i^t\right)^2$$

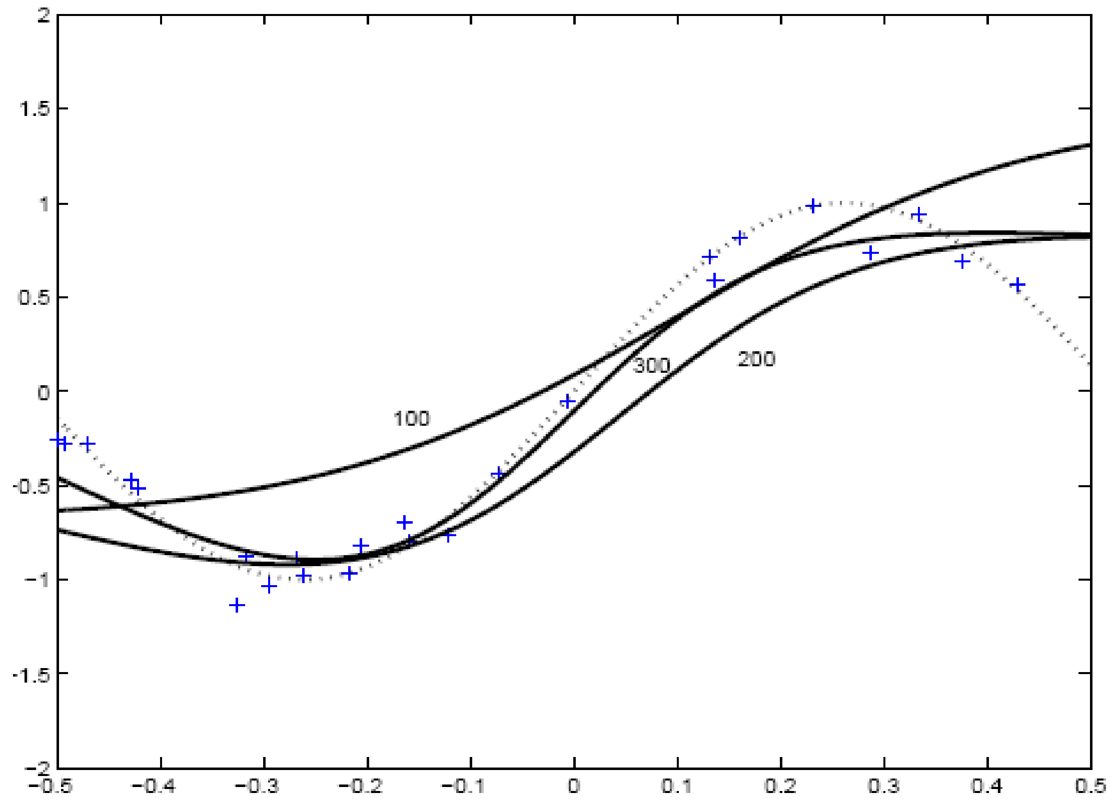
$$y_i^t=\sum_{h=1}^{H}v_{ih}z_h^t+v_{i0}$$

$$\Delta v_{ih}=\eta\sum_t\left(r_i^t-y_i^t\right)z_h^t$$

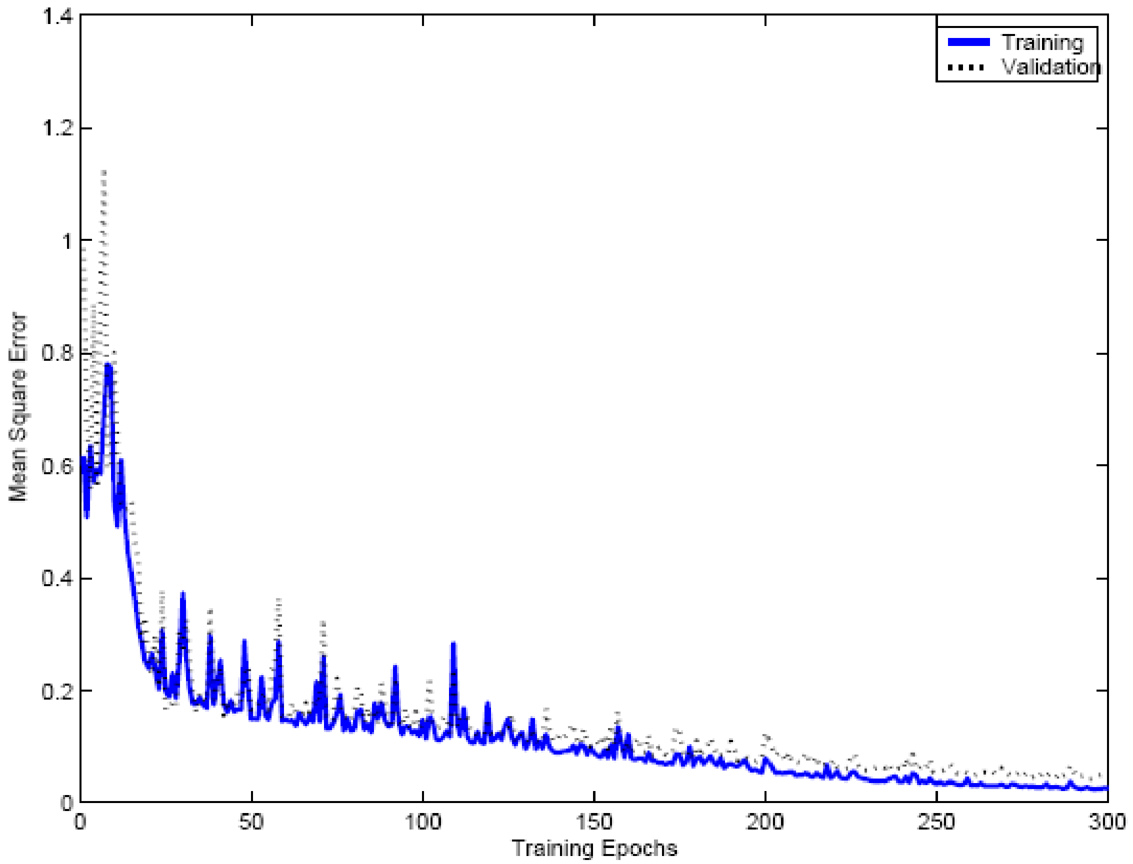
$$\Delta w_{hj}=\eta\sum_t\left[\sum_i\left(r_i^t-y_i^t\right)v_{ih}\right]z_h^t\left(1-z_h^t\right)x_j^t$$



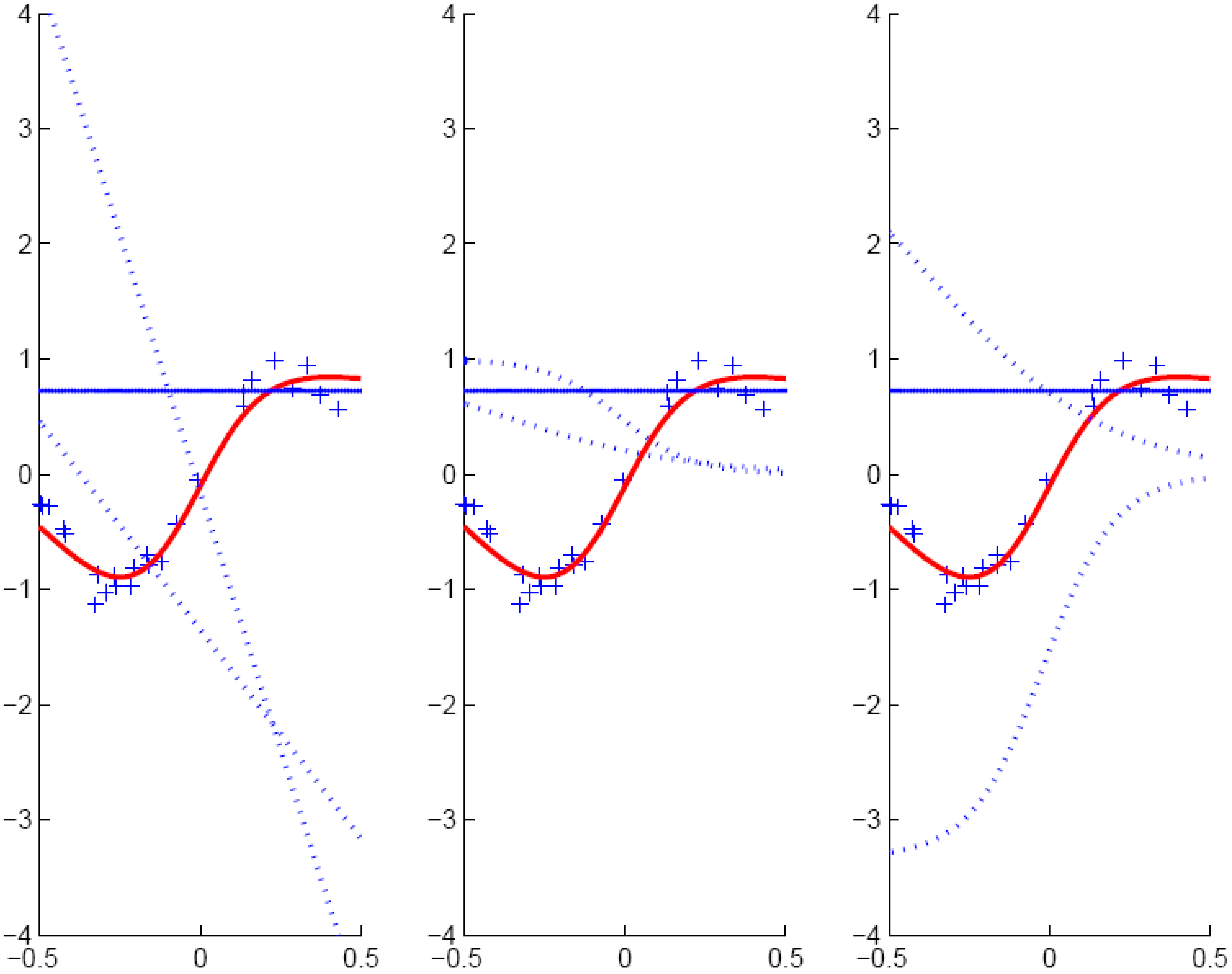
Initialize all v_{ih} and w_{hj} to rand$(-0.01, 0.01)$
Repeat
    For all $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ in random order
        For $h = 1, \ldots, H$
            $z_h \leftarrow \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$
        For $i = 1, \ldots, K$
            $y_i = \boldsymbol{v}_i^T \boldsymbol{z}$
        For $i = 1, \ldots, K$
            $\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$
        For $h = 1, \ldots, H$
            $\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$
        For $i = 1, \ldots, K$
            $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$
        For $h = 1, \ldots, H$
            $\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$
Until convergence



100
200
300



Training
Validation
Mean Square Error
Training Epochs


4
3
2
1
0
−1
−2
−3
−4
−0.5
0
0.5

Two-Class Discrimination

- One sigmoid output y^t for $P(C_1|\boldsymbol{x}^t)$ and $P(C_2|\boldsymbol{x}^t) \equiv 1-y^t$

$$y^t = \text{sigmoid}\left(\sum_{h=1}^{H} v_h z_h^t + v_0\right)$$

$$E(\mathbf{W},\mathbf{v}|\mathcal{X}) = -\sum_t r^t \log y^t + \left(1-r^t\right)\log\left(1-y^t\right)$$

$$\Delta v_h = \eta \sum_t \left(r^t - y^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left(r^t - y^t\right) v_h z_h^t \left(1 - z_h^t\right) x_j^t$$

k>2 Classes

$$o_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \qquad y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t} \equiv P\left(C_i \mid \mathbf{x}^t\right)$$

$$E\left(\mathbf{W}, \mathbf{v} \mid \mathcal{X}\right) = -\sum_t \sum_i r_i^t \log y_i^t$$

$$\Delta v_{ih} = \eta \sum_t \left(r_i^t - y_i^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) v_{ih}\right] z_h^t \left(1 - z_h^t\right) x_j^t$$

Multiple Hidden Layers

- MLP with one hidden layer is a universal approximator (Hornik et al., 1989), but using multiple layers may lead to simpler networks

$$z_{1h} = \text{sigmoid}\left(\mathbf{w}_{1h}^T \mathbf{x}\right) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\right), h = 1, \ldots, H_1$$

$$z_{2l} = \text{sigmoid}\left(\mathbf{w}_{2l}^T \mathbf{z}_1\right) = \text{sigmoid}\left(\sum_{h=1}^{H_1} w_{2lh} z_{1h} + w_{2l0}\right), l = 1, \ldots, H_2$$

$$y = \mathbf{v}^T \mathbf{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0$$

Improving Convergence

- Momentum

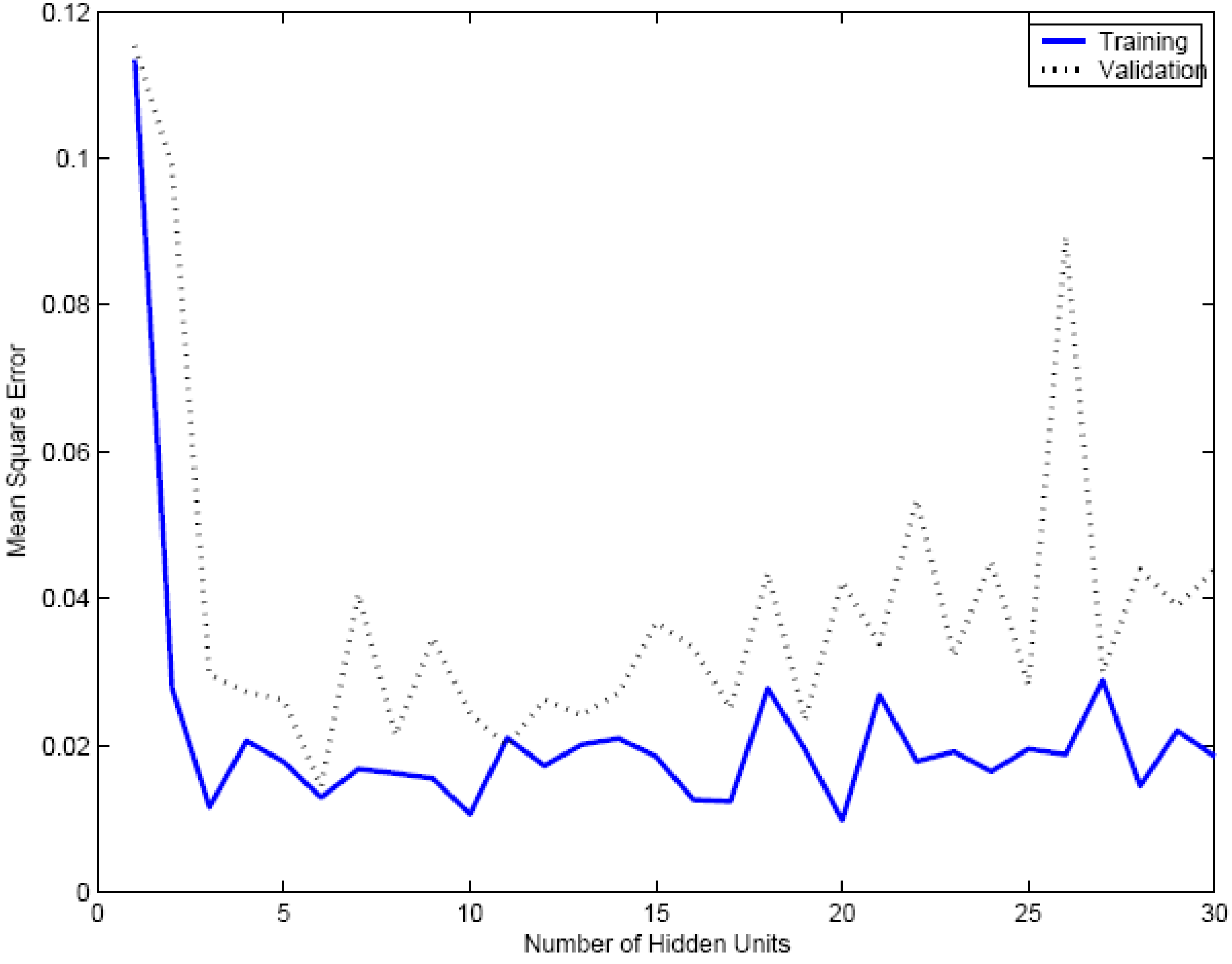
$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

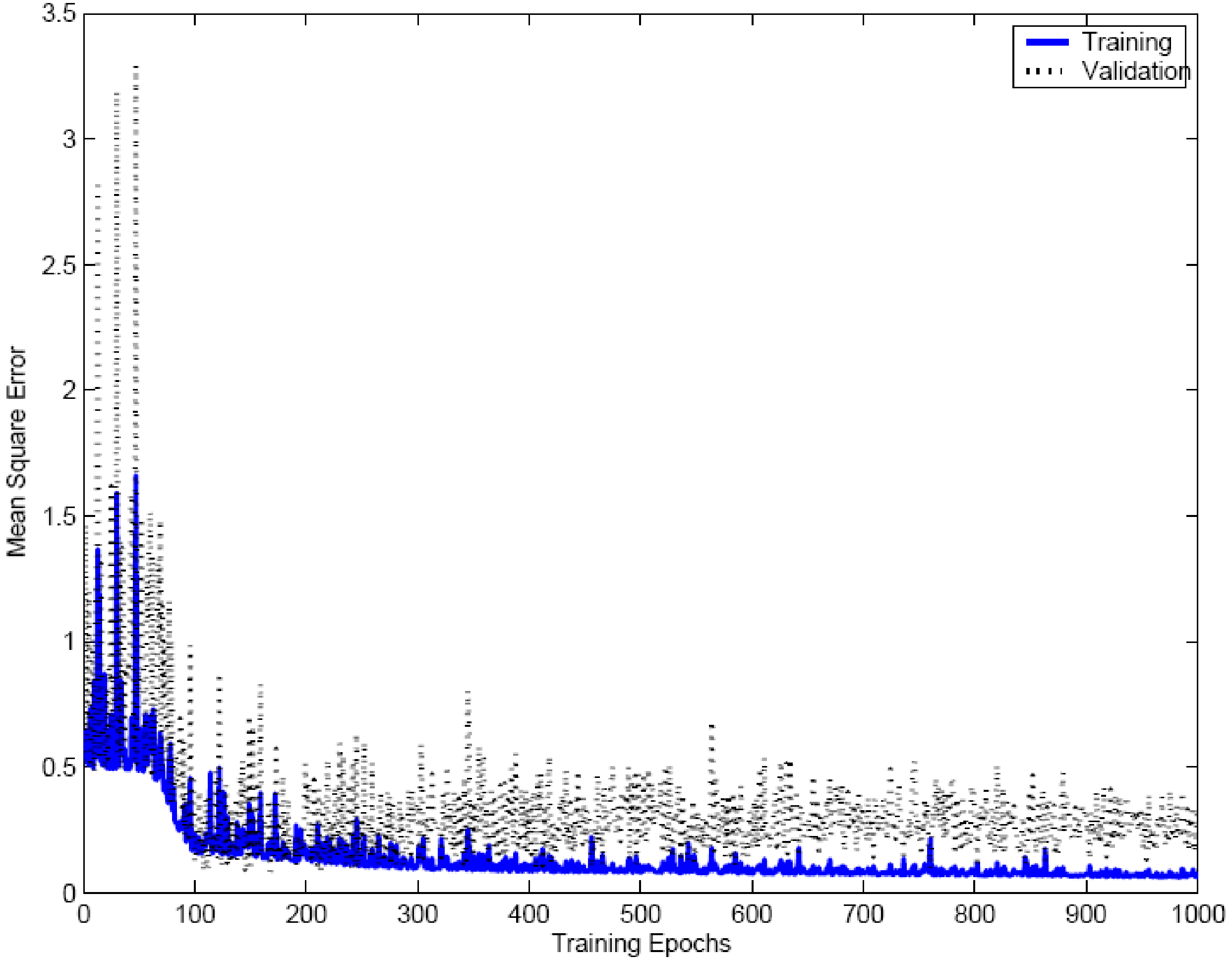
- Adaptive learning rate

$$\Delta \eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases}$$

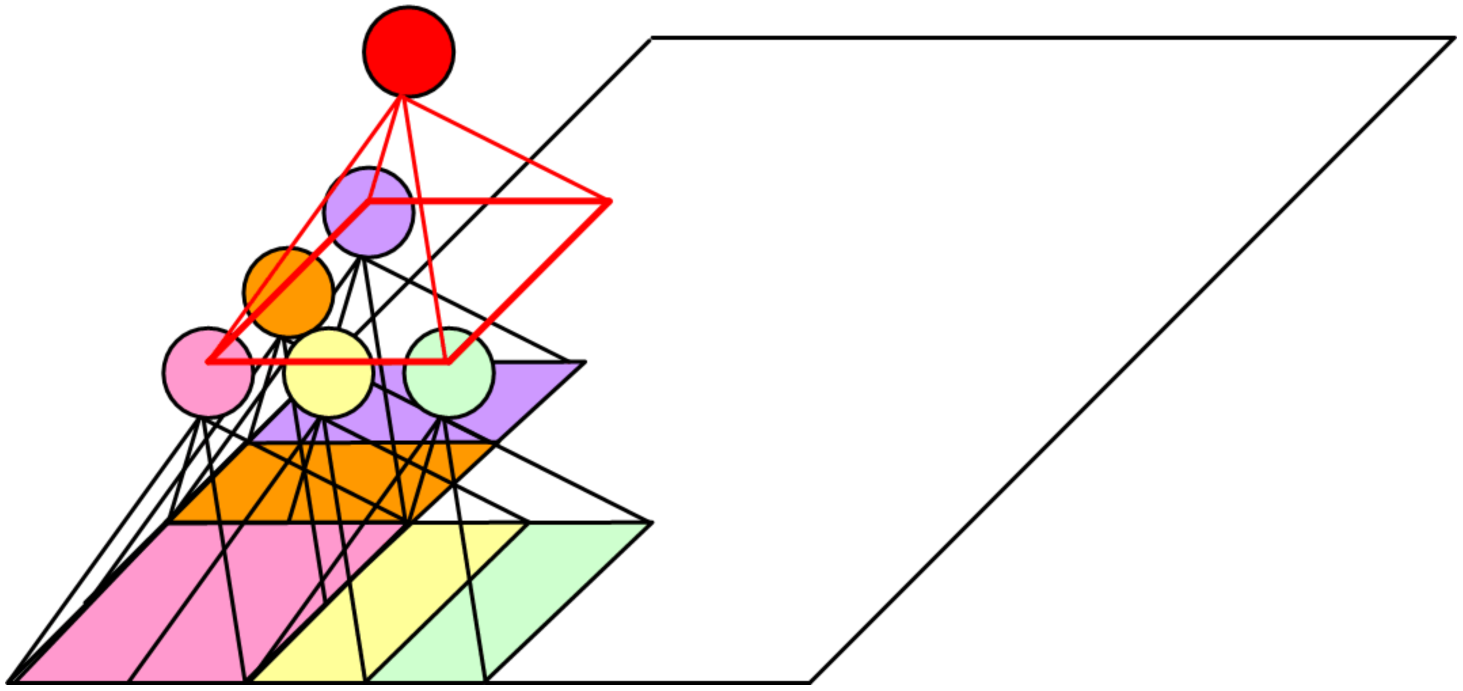
Overfitting/Overtraining

Number of weights: $H\,(d+1)+(H+1)K$





Convolutional Neural Networks (CNN)

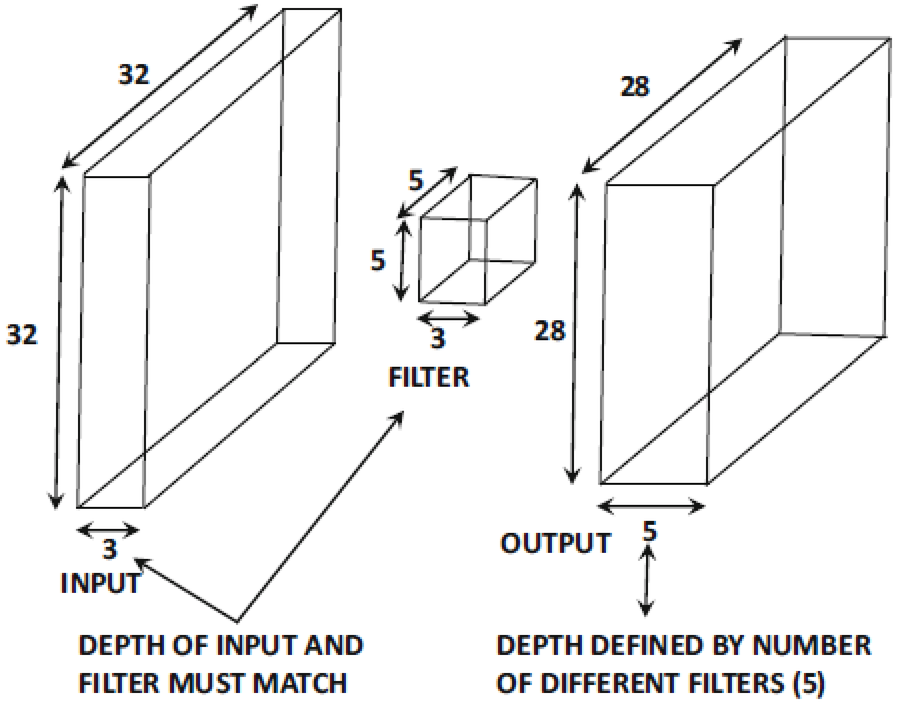


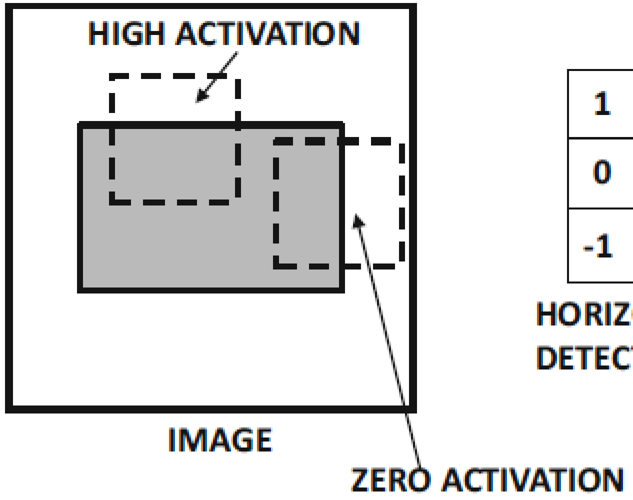
Proposed by Le Cun et al. in 1989
Main principle:
- Convolution
- Activation
- Pooling

(Le Cun et al, 1989)

CNN – Basic Structure

- Convolution between layers
- No. of layers affects output
- Form of filter helps detect particular features of image
- Ref. [1]



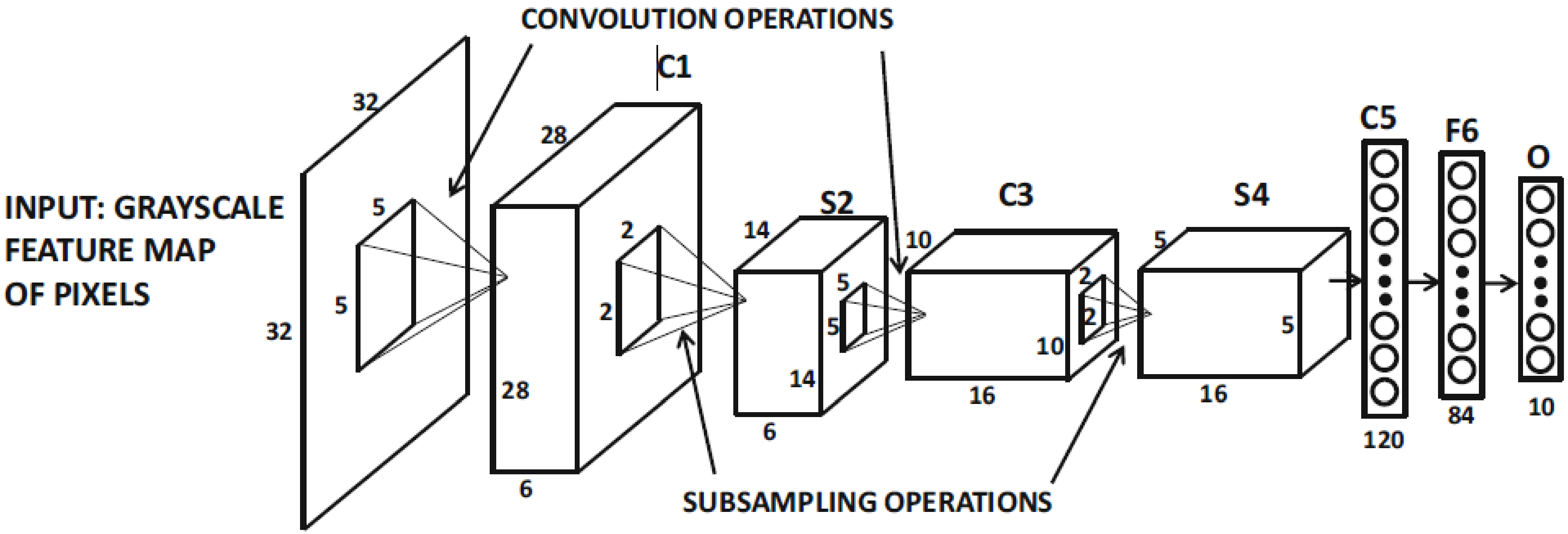


1	1	1
0	0	0
-1	-1	-1

HORIZONTAL EDGE DETECTING FILTER

CNN - Architecture

A simple Convolutional Neural Network on a grayscale image [1]
Input Layer → Convolutional Layer → Activation Function → Pooling Layer → Fully Connected Layer → … → Output



Simple architecture of a Convolutional Neural Network [1]

Convolutional Neural Network- Convolution Layer

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

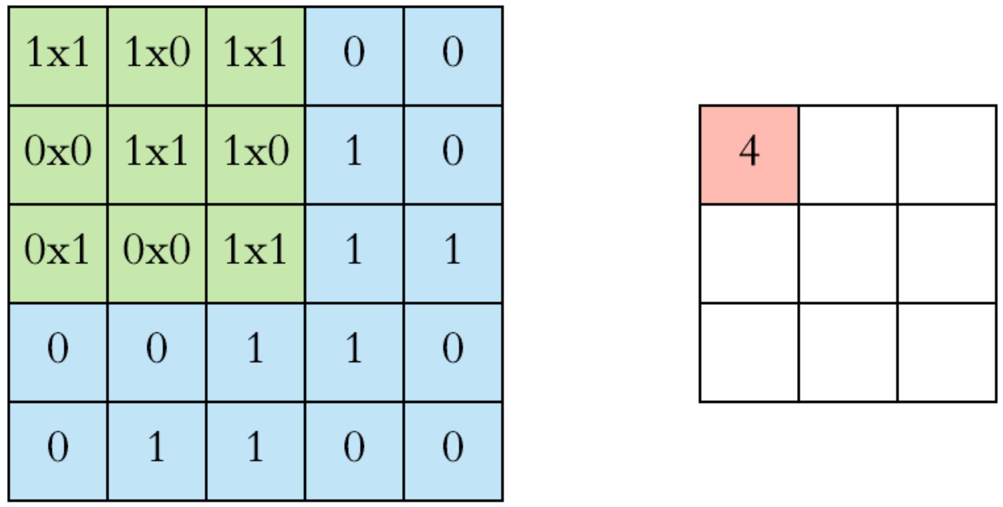
1	0	1
0	1	0
1	0	1

Filter / Kernel

The Filter/Kernel convolutes through the entire depth of the Input grid.

Left: Input grid, Right: Filter/Kernel. [4]

Convolutional Neural Network- Convolution Layer



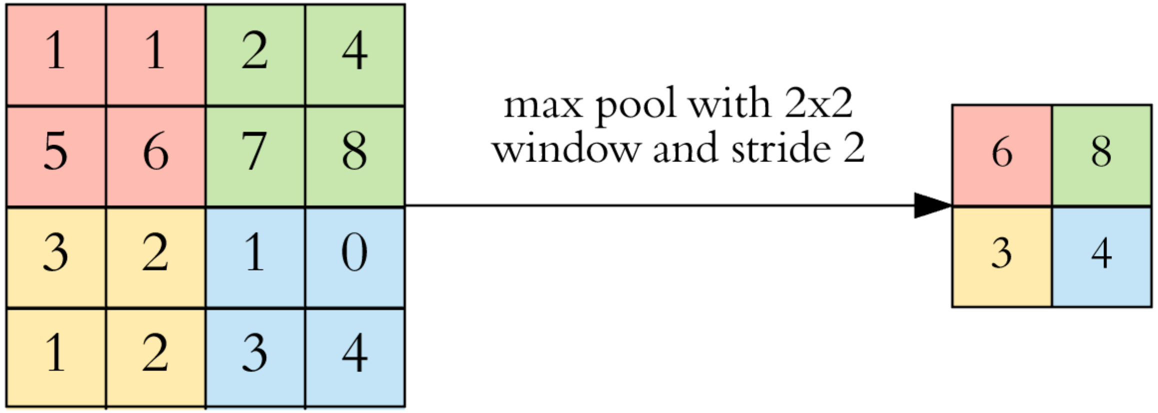
Left: The Filter/Kernel convolutes through the entire depth of the Input grid, Right: An Activation map created by element wise matrix multiplication and summation of results.[4]

To create an Activation map convolute the Filter/Kernel over the input grid.

At every location do element-wise matrix multiplication and sum the results.

Convolutional Neural Network- Pooling Layer

Types of pooling:
MaxPool
AvgPool
Etc.

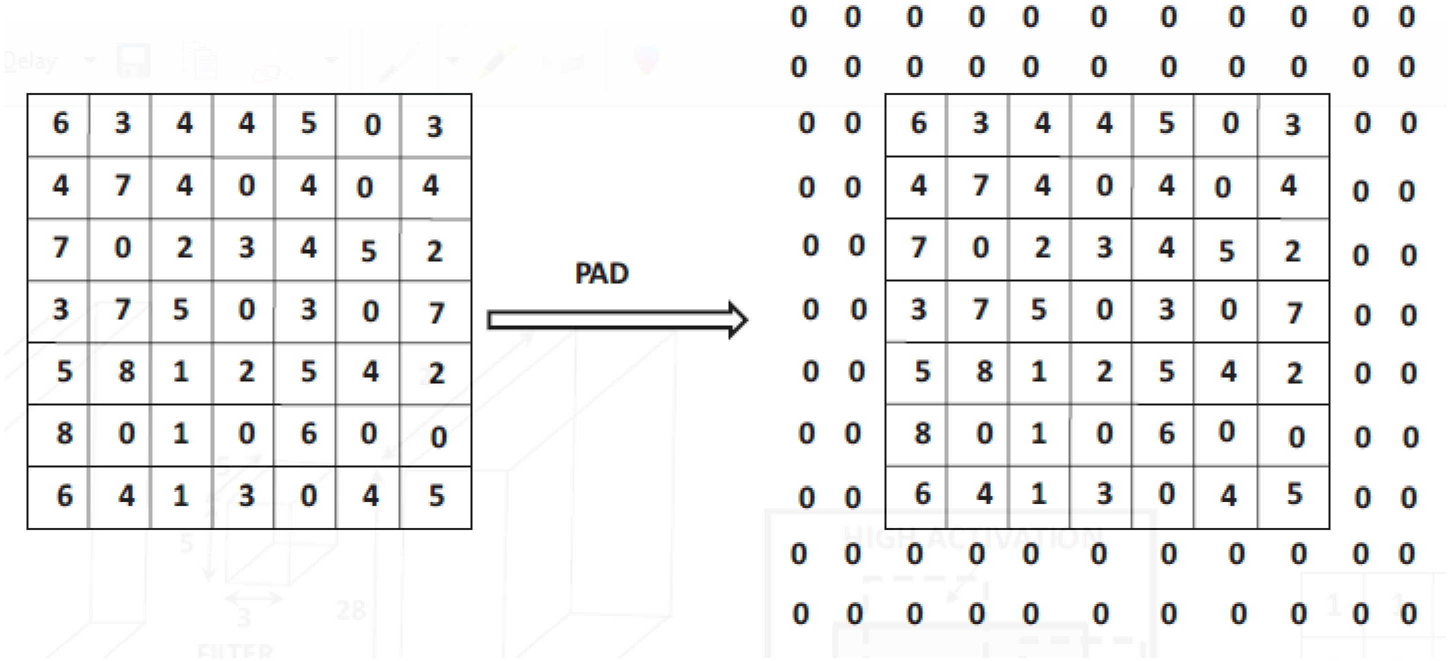


The Max Pooling layer takes the Activation map, the stride, and the window and down samples the Activation map by taking the maximum value in each pooling window. [4]

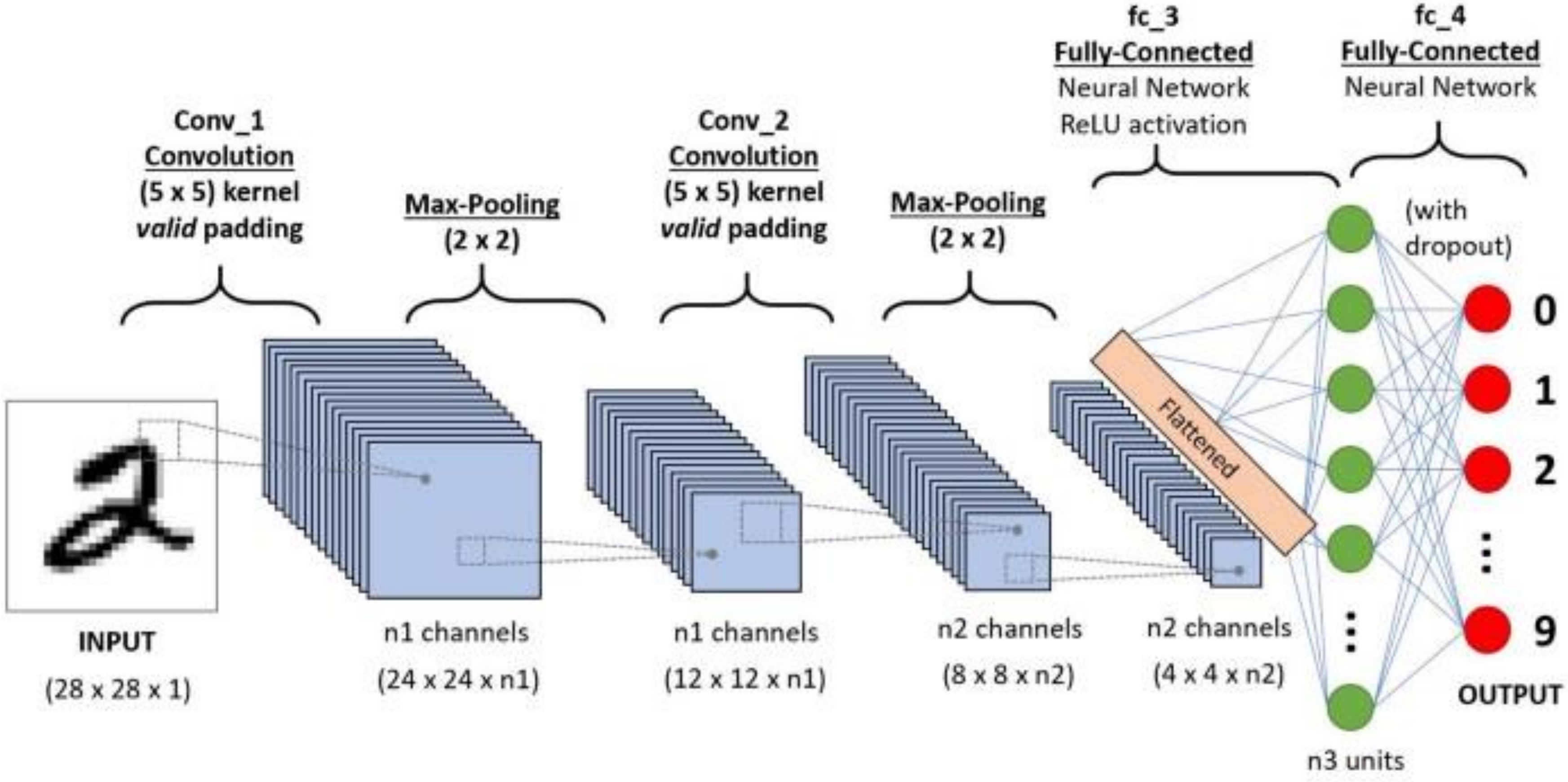
- Pooling is done to **reduce the dimensionality**.
- Pooling layers **down sample each Activation map** independently by reducing the height and width while keeping the depth intact.
- The most common type of pooling is **max pooling** which takes the maximum value in the pooling window.

Padding in CNNs

- Used to convolve over borders of image
- Helps run the learning process smoothly
- Most commonly known scheme know as “zero padding”
- Ref [1]

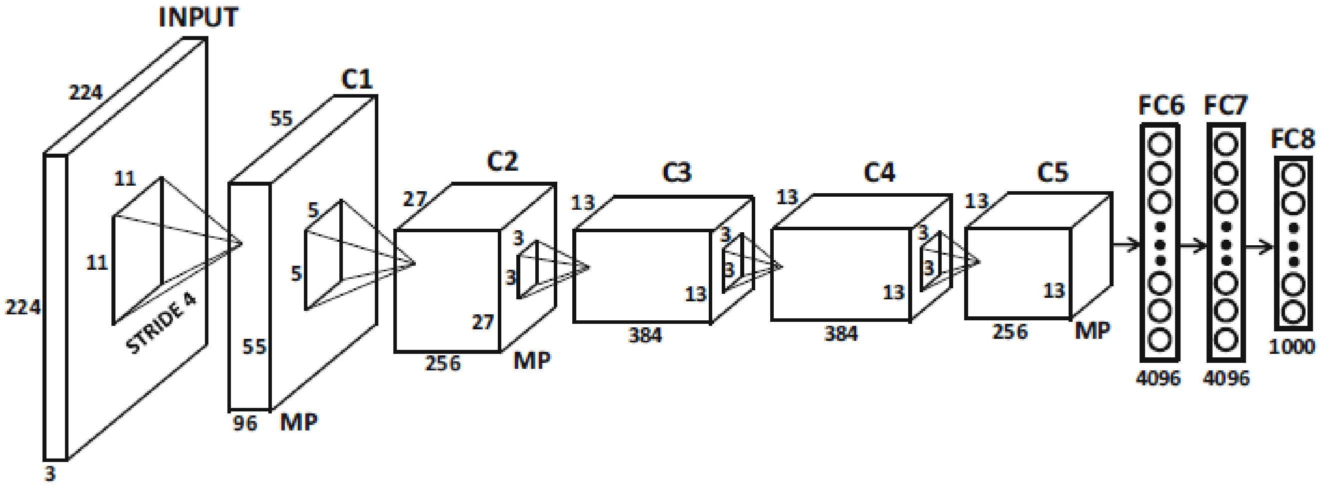


Example: Digit recognition

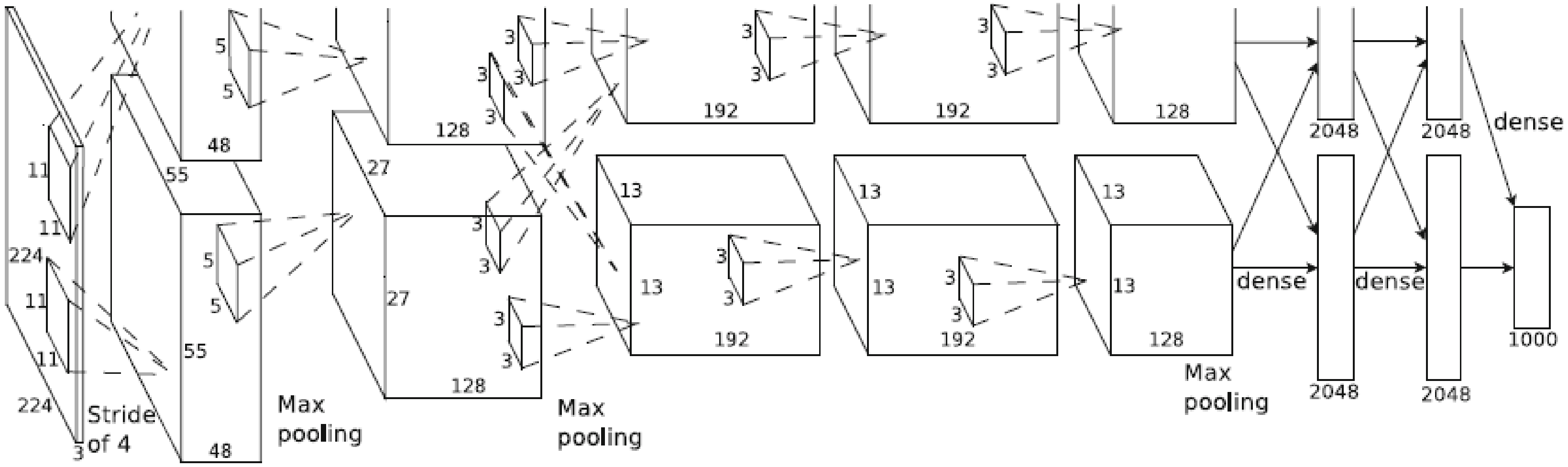


Implementation of CNNs

- The use of high-performance computing is crucial
- Partitioning can be done efficiently with the use of GPUs
- See Tools at the end of this chapter, and ref. [8] for example



(a) Without GPU partitioning

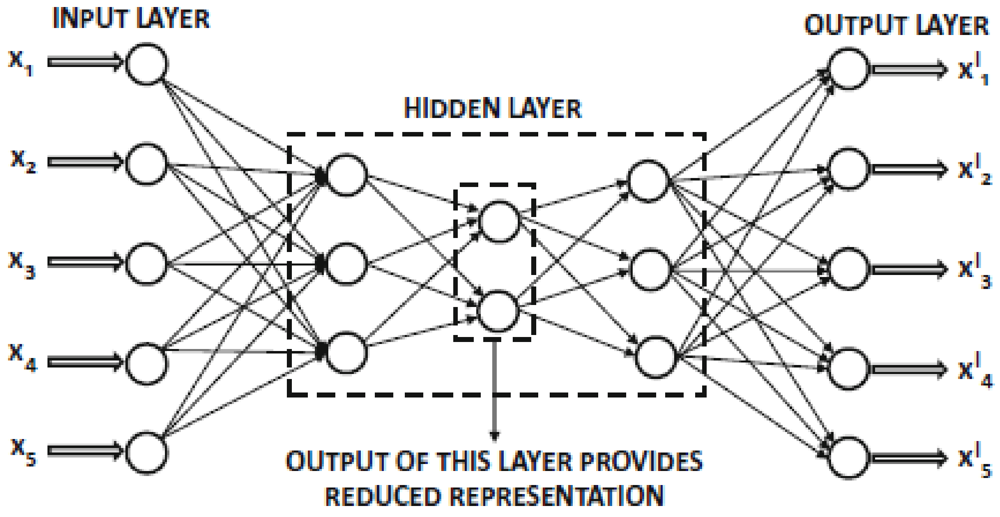


(b) With GPU partitioning (original architecture)

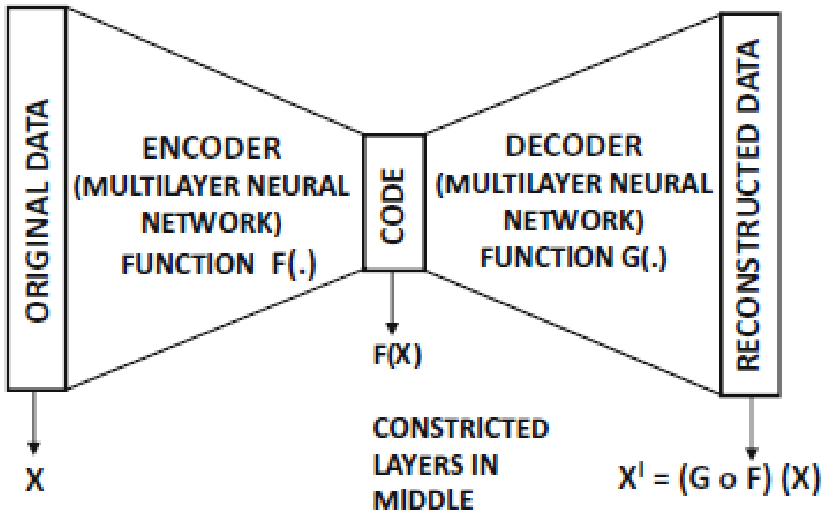
Autoencoder

- An autoencoder is a multilayer NN
  - Main condition: number of nodes in i^{th} hidden layer smaller than $(i-1)^{th}$
- Schematic of the autoencoder:

- Acts as a compression algorithms of the input:
  - Compression is typically lossy
  - Mostly used for dimensionality reduction
  - Pre-processing step for clustering



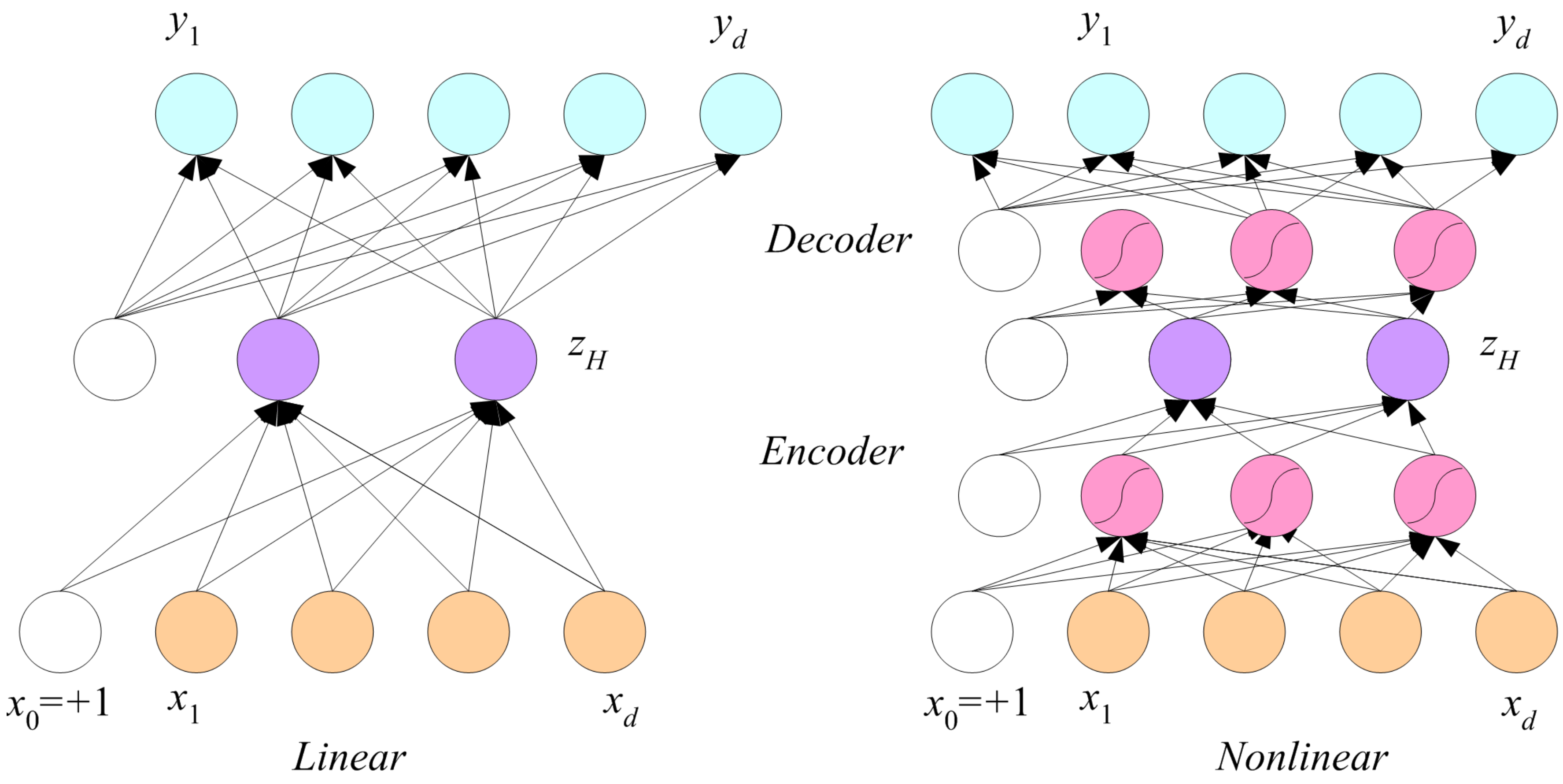
(a) Three hidden layers



(b) General schematic

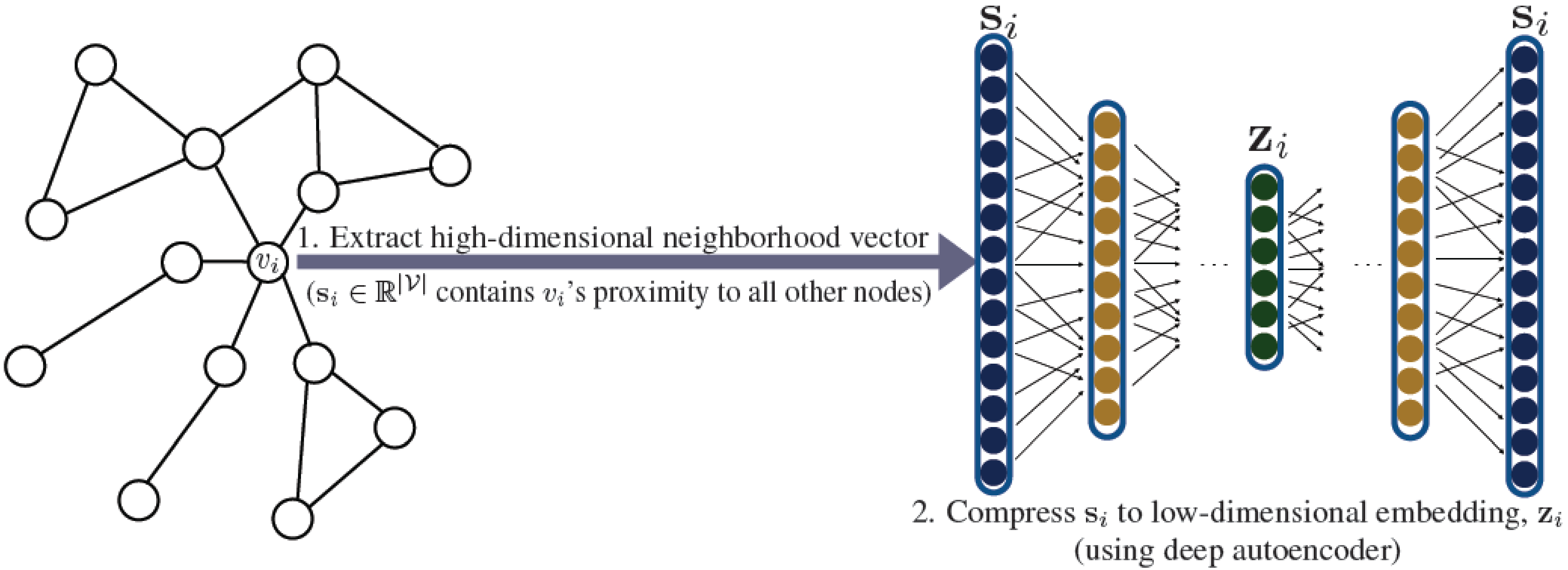
ref. [1]

Autoencoder – two main types



Example: Autoencoder on Graph Embedding

Models: Deep Neural Graph Representation (DNGR) and Structural Deep Network Embeddings (SDNE) [6]
Nodes represented as vectors
Examples: Node2vec, DeepWalk, HARP, LINE

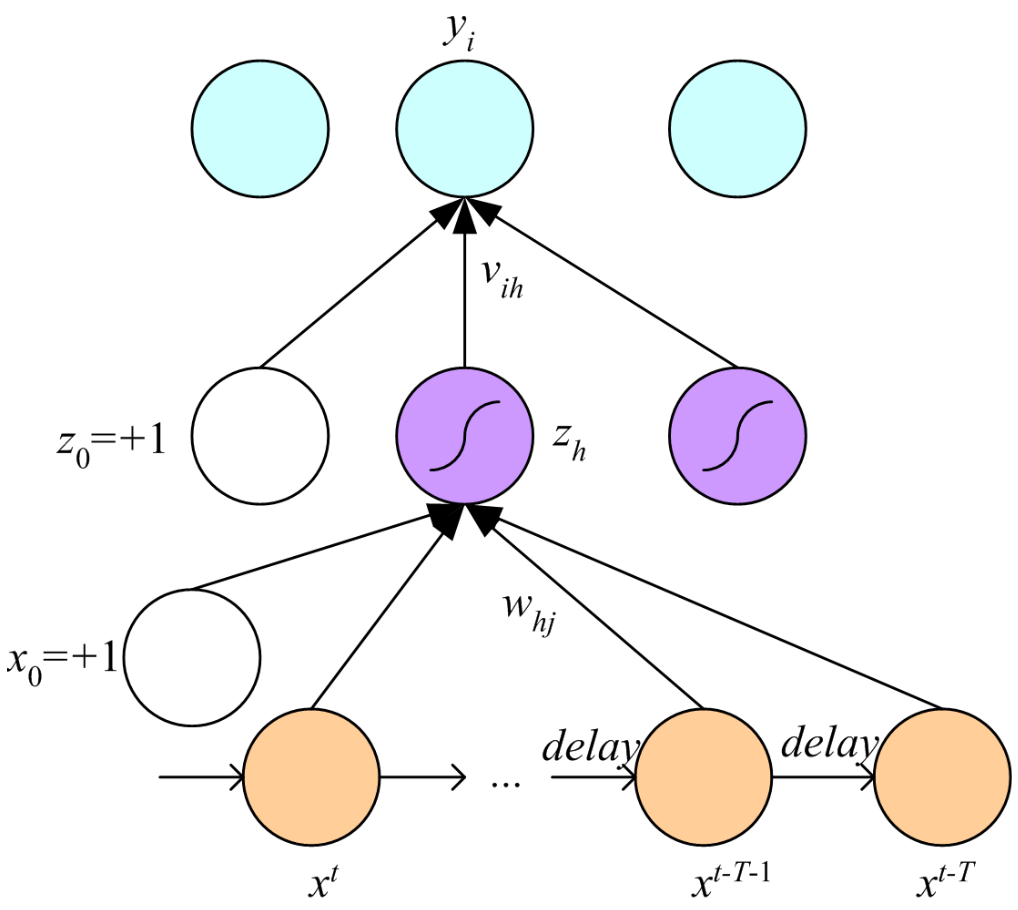


Aim of DNGR/SDNE: $DEC\big(ENC(s_i)\big) = DEC(z_i) \approx s_i$ or loss function: $\mathcal{L} = \sum_{v_i \in V} \|DEC(z_i) - s_i\|_2^2$

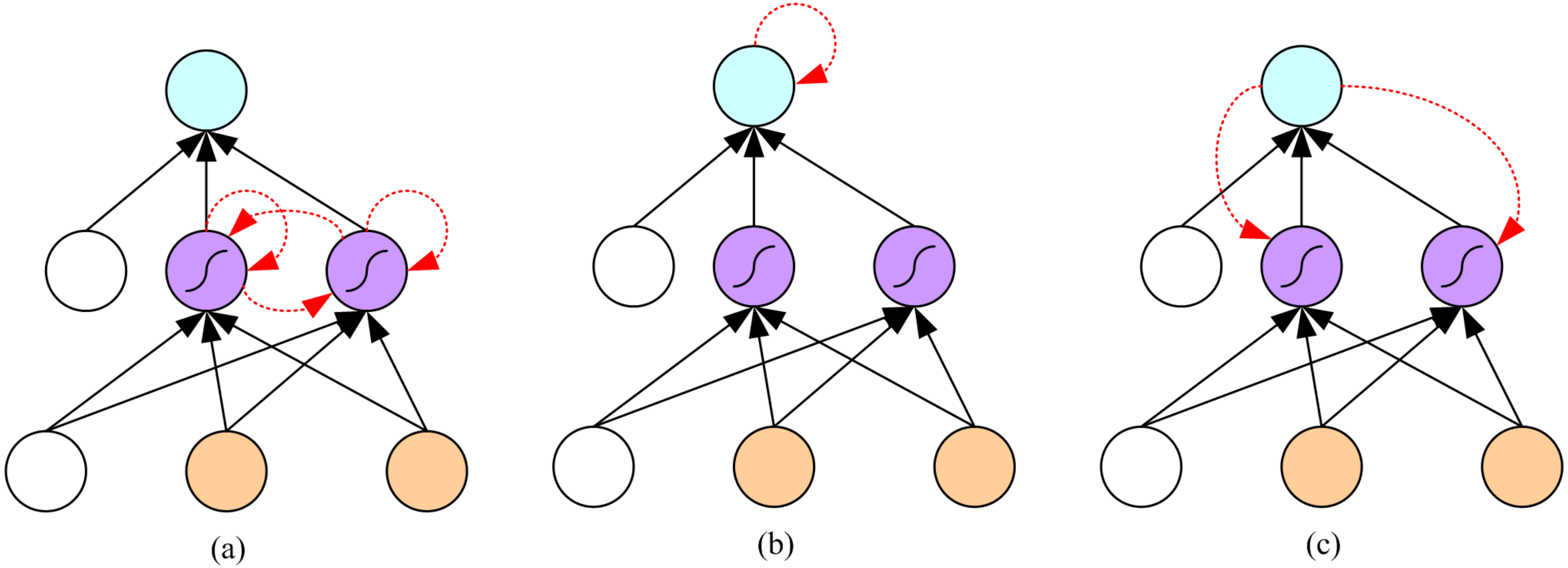
Learning Time - Sequences

- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association
  - Machine translation
- Network architectures
  - Time-delay networks
  - Recurrent neural networks (RNN)
    - Long-short term memory (LSTM)
    - Gated recurrent units (GRUs)
    - Others

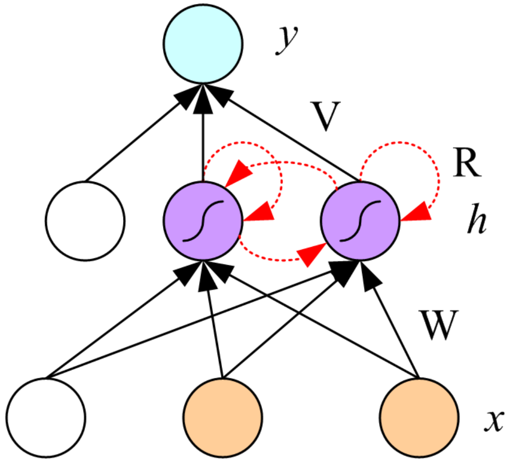
Time-Delay Neural Networks



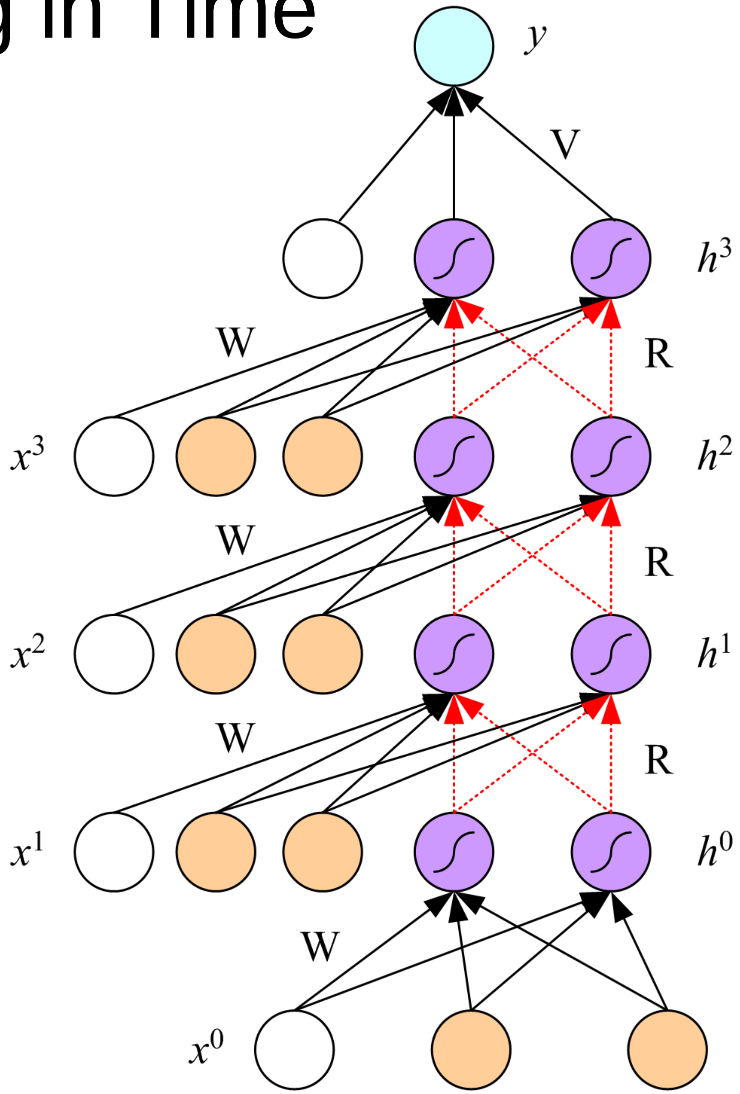
Recurrent Networks



Unfolding in Time



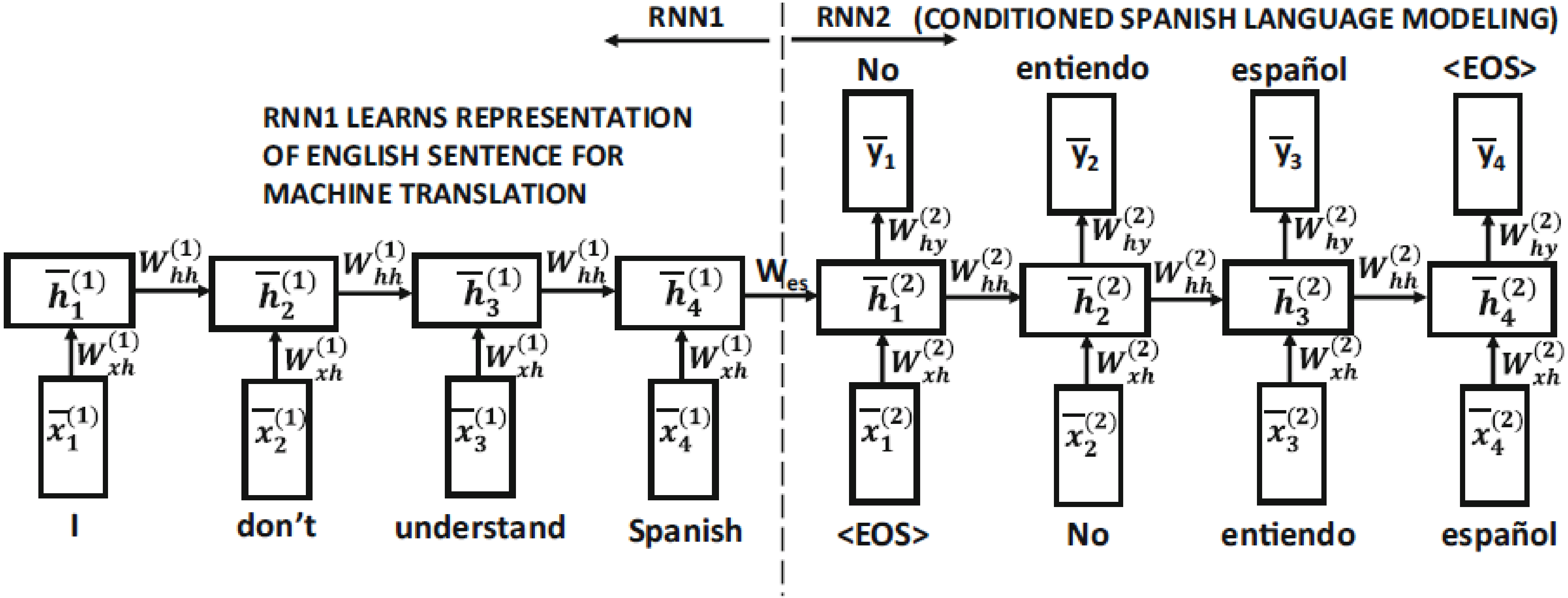
(a)



(b)

RNN Example - Translation

- 2 separate RNNs with their own sets of shared weights
  - One for each language



Tools for Deep Learning/NN

- Andrej Karpathy's blog – ref. [5]
  - https://karpathy.github.io/2019/04/25/recipe/
- Main Scikit reference:
  - Includes classification, regression and Boltzmann machine
  - Note: It's not intended for deep learning
  - https://scikit-learn.org/stable/modules/classes.html#module-sklearn.neural_network
- Sknn: an API compatible with Scikit-learn
  - https://scikit-neuralnetwork.readthedocs.io/en/latest/
- Tensoflow:
  - https://www.tensorflow.org/
- Tools by Google:
  - https://ai.google/tools/
- Deep learning on Sharcnet/Compute Canada:
  - https://www.sharcnet.ca/help/index.php/Machine_Learning_and_Data_Mining ref [8]
- IBM - Watson
  - https://www.ibm.com/cloud/deep-learning
- NVIDIA DIGITS
  - https://developer.nvidia.com/digits

Tools for Deep Learning

- **TensorFlow:** Open source library for machine learning and deep learning.
- **Keras:** NN API that runs on top of TensorFlow, **CNTK**, or **Theano**.
- **Caffe:** Open source written in C++ -- developed at Berkeley.
- **Torch 7:** Based on scripting language LuaJIT – good for CNN.
- Theano: Written in C++/Python – good for RNNs; slow compiling.

	Caffe	Theano	Torch7	TensorFlow
Core language	C++	Python, C++	LuaJIT	C++
Interfaces	Python, Matlab	Python	C	Python
Wrappers		Lasagne, Keras, sklearn-theano		Keras, Pretty Tensor, Scikit Flow
Programming paradigm	Imperative	Declarative	Imperative	Declarative
Well suited for	CNNs, Reusing existing models, Computer vision	Custom models, RNNs	Custom models, CNNs, Reusing existing models	Custom models, Parallelization, RNNs

References - Acknowledgments

The material presented in this chapter has been taken from the following sources (among others):

1. C. Aggarwal. Neural Networks and Deep Learning, Springer, 2018.
2. E. Alpaydin. Introduction to Machine Learning. Third Edition. 2014.
3. F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review. Vol. 65, No. 6. 1968.
4. F. Van Veen. The Neural Network Zoo. 2016. http://www.asimovinstitute.org/neural-network-zoo/
5. A. Karpathy. A recipe for training Neural Networks. Updated Apr 25, 2019. https://karpathy.github.io/2019/04/25/recipe/
6. W. Hamilton, R. Ying, J. Leskovec. Representation Learning on Graphs: Methods and Applications. Bulletin IEEE of the Computer Science Technical Committee on Data Engineering, 2018. https://arxiv.org/abs/1709.05584
7. CNN, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 /, Last accessed 21/05/2019.
8. Deep learning on Sharcnet: https://www.sharcnet.ca/help/index.php/Machine_Learning_and_Data_Mining