CSE473: Computational Intelligence

Bipolar Perceptron Learning & Optimization

by:

Hossam Abd El Munim

Computer & Systems Engineering Dept.,

Ain Shams University,

1 El-Sarayat Street, Abbassia, Cairo 11517

Many slides and images are taken from the internet.

Dataset split

Training Images

- Train classifier

Validation Images

- Measure error
- Tune model hyperparameters

Testing Images

- Secret labels
- Measure error

Random train/validate splits = cross validation

Training

Training Images

Training Labels

Image Features

Training

Learned classifier

Testing

Test Image

Image Features

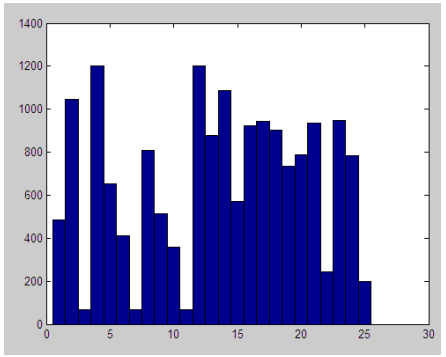
Apply classifier

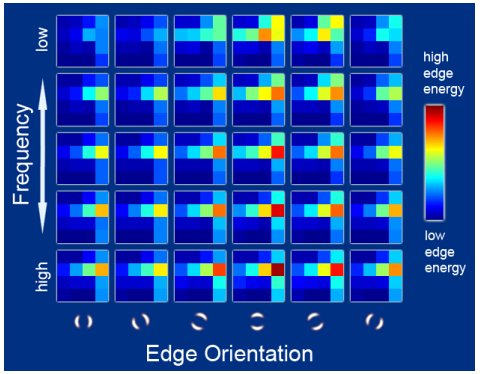
Prediction

Slide credit: D. Hoiem and L. Lazebnik

Features

- Raw pixels
- Histograms
- Templates
- SIFT descriptors
  - GIST
  - ORB
  - HOG….





Training

Training Images

Training Labels

Image Features

Training

Learned classifier

Testing

Test Image

Image Features

Apply classifier

Prediction

Slide credit: D. Hoiem and L. Lazebnik

Training

Training Images

Training Labels

Image Features

Training

Learned classifier

Testing

Test Image

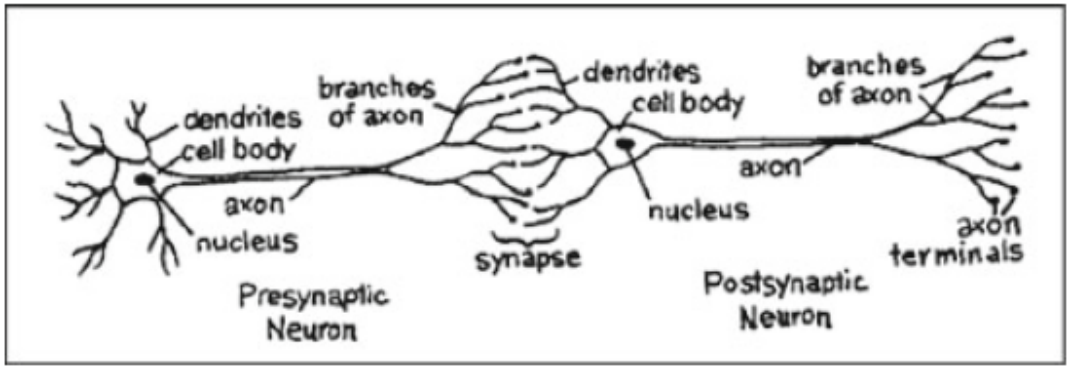
Image Features

Apply classifier

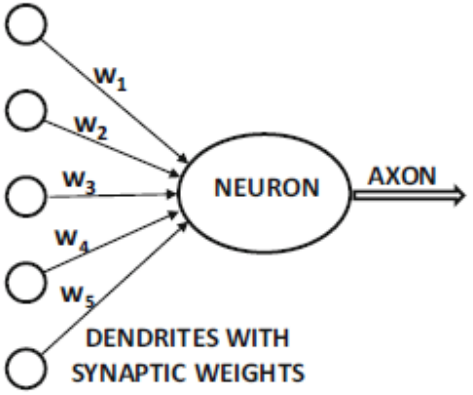
Prediction

Slide credit: D. Hoiem and L. Lazebnik

Simulation of biological mechanism in artificial neural networks



(a) Biological neural network



(b) Artificial neural network

Figure 1.1: The synaptic connections between neurons. The image in (a) is from "*The Brain: Understanding Neurobiology Through the Study of Addiction* [598]." Copyright ©2000 by BSCS & Videodiscovery. All rights reserved. Used with permission.

Deep Learning and Others

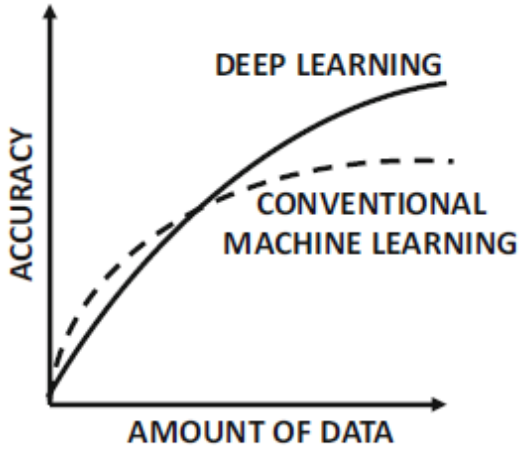
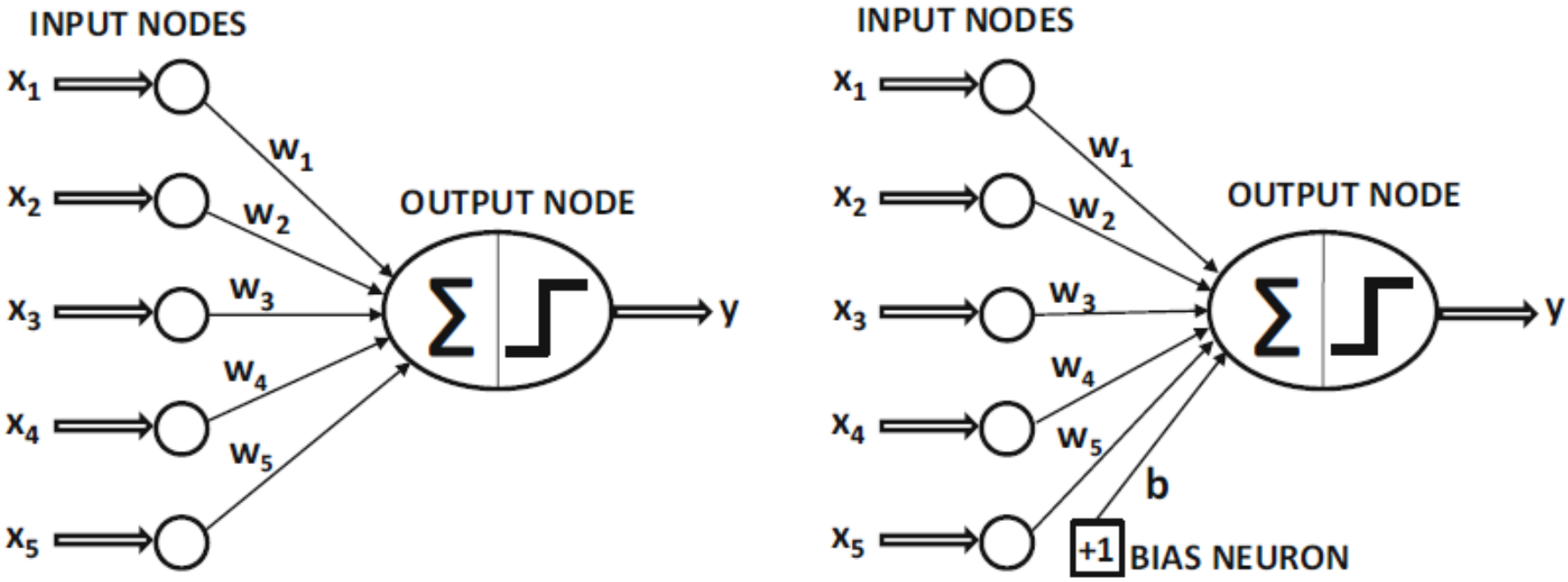


Figure 1.2: An illustrative comparison of the accuracy of a typical machine learning algorithm with that of a large neural network. Deep learners become more attractive than conventional methods primarily when sufficient data/computational power is available. Recent years have seen an increase in data availability and computational power, which has led to a "Cambrian explosion" in deep learning technology.

Humans Versus Computers: Stretching the Limits of Artificial Intelligence

- Humans and computers are inherently suited to different types of tasks.
- For example, computing the cube root of a large number is very easy for a computer, but it is extremely difficult for humans.
- On the other hand, a task such as recognizing the objects in an image is a simple matter for a human, but has traditionally been very difficult for an automated learning algorithm.
- It is only in recent years that deep learning has shown an accuracy on some of these tasks that exceeds that of a human.
- In fact, the recent results by deep learning algorithms that surpass human performance [184] in (some narrow tasks on) image recognition would not have been considered likely by most computer vision experts as recently as 10 years ago.

The Basic Architecture of Neural Networks



(a) Perceptron without bias

(b) Perceptron with bias

Figure 1.3: The basic architecture of the perceptron

Perceptron as a Binary Classifier
Only Two Categories

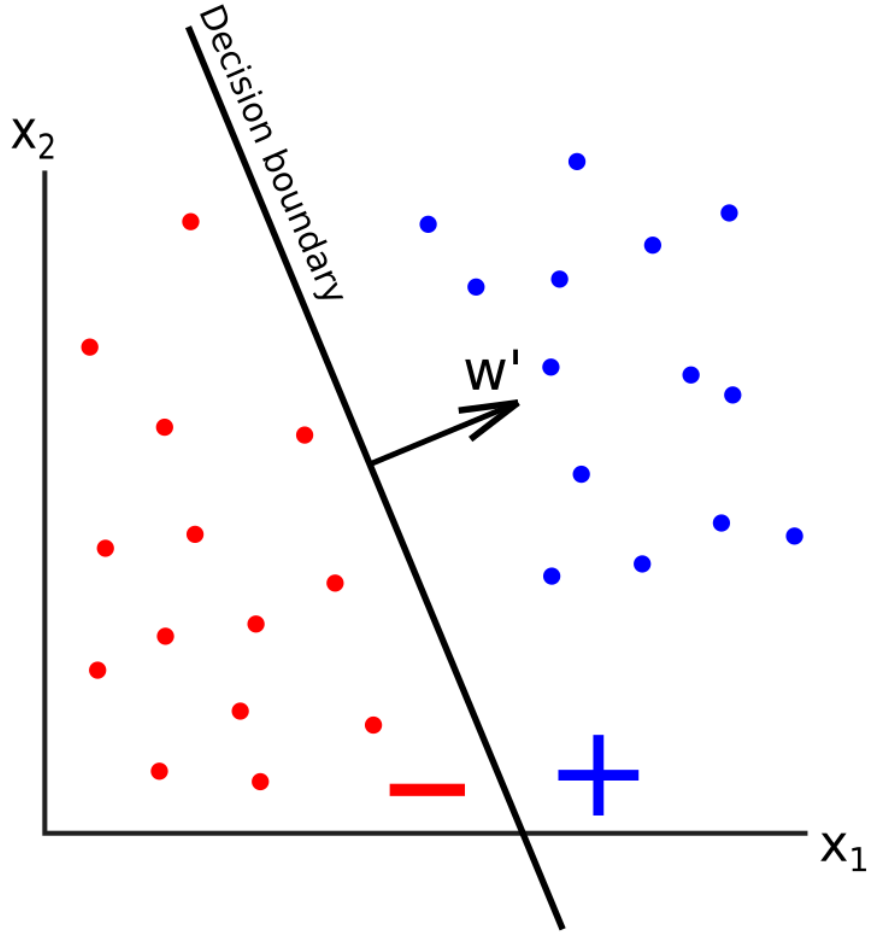
Perceptron as a Binary Classifier (2D Case)

$\mathbf{X} = [x_1\ x_2\ 1]^T$

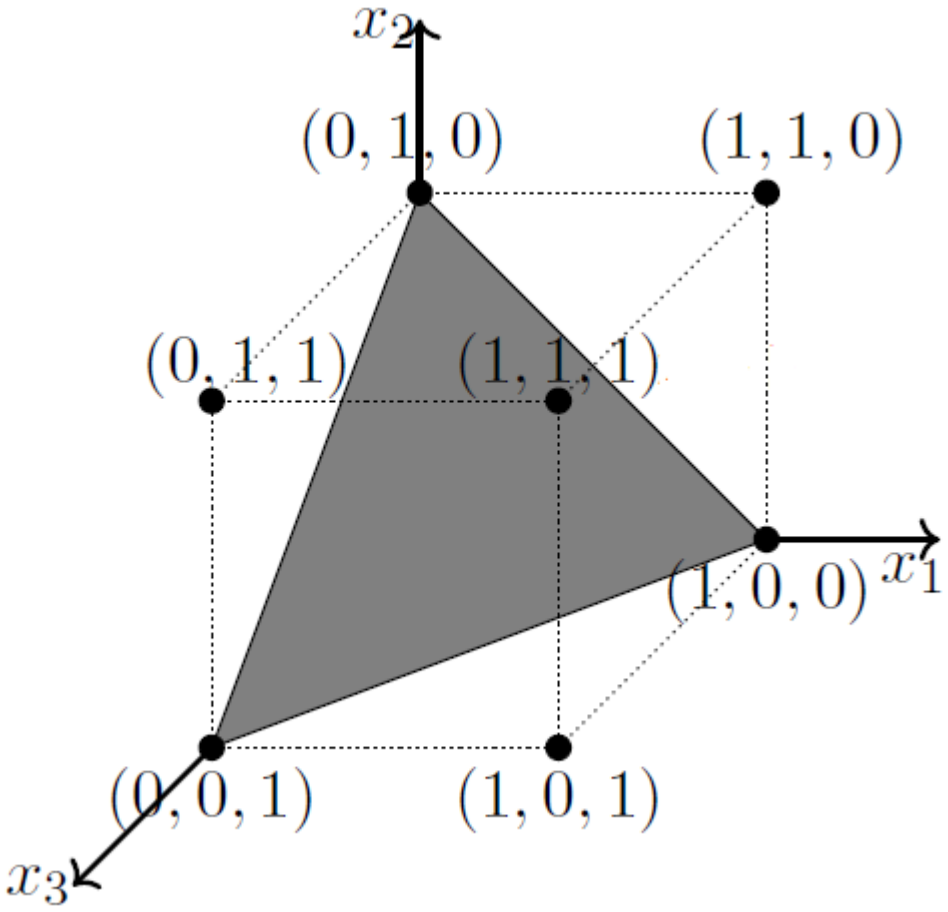
$\mathbf{W} = [w_1\ w_2\ w_0]^T$

$\mathbf{W.X = W^TX = 0}$

$w_1x_1 + w_2x_2 + w_0 = 0$



Perceptron as a Binary Classifier (3D Case)



$\mathbf{X} = [x_1\ x_2\ x_3\ 1]^T$

$\mathbf{W} = [w_1\ w_2\ w_3\ w_0]^T$

$\mathbf{W.X = W^TX = 0}$

$w_1x_1 + w_2x_2 + w_3x_3 + w_0 = 0$

What do you think about higher dimensions?

AND Function & Parameter Estimation

$(-1)w_1+(-1)w_2+w_0 < 0$

$(-1)w_1+(+1)w_2+w_0 < 0$

$(+1)w_1+(-1)w2+w_0 < 0$

$(+1)w_1+(+1)w2+w_0 \geq 0$

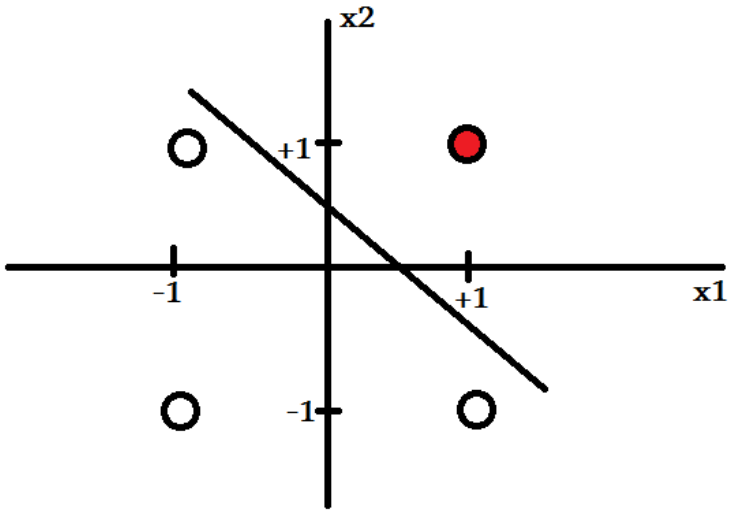
x_1	x_2	y
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1

Second + Third = $2w_0 < 0$

First + Third = $-2w_2 + 2w_0 < 0$

Take $w_0 = -1$ & then $w_2 > -1$…Set $w_2 = 2$

Substitute in the first $w_1 > -3$…Set $w_1 = 2$



Perceptron Training: Learning W

+1

-1

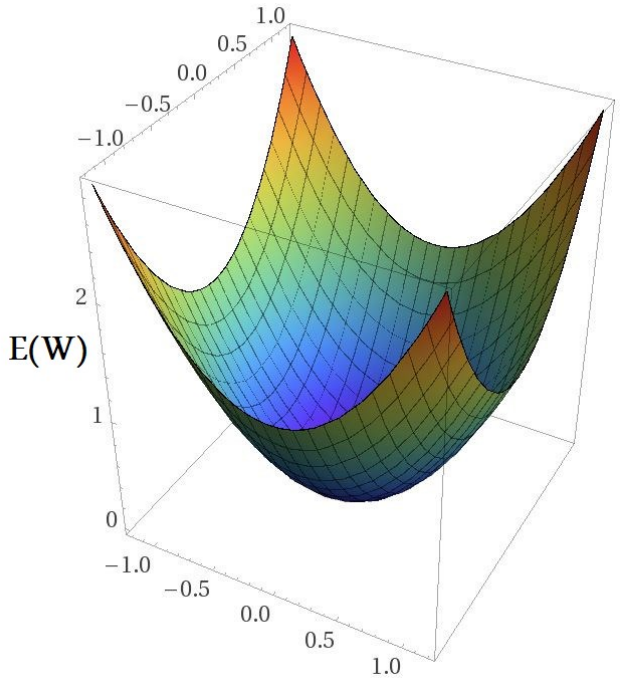
Given a data set, D = {($\mathbf{X}_1$, y_1), ($\mathbf{X}_2$, y_2)... ($\mathbf{X}_N$, y_N)} of labelled feature vectors where y ϵ {-1, +1}, we to estimate the vector **W** that minimizes the following objective function which measure the difference between the actual and desired outputs:-

$$E(\mathbf{W}) = (y_1\text{-sign}(W.\ \mathbf{X}_1))^2 + (y_2\text{-sign}(W.\ \mathbf{X}_2))^2 + \ldots + (y_N\text{-sign}(W.\ \mathbf{X}_N))^2$$

Training & Optimization

Given a data set, D = {($\mathbf{X}_1$, y_1), ($\mathbf{X}_2$, y_2)... ($\mathbf{X}_N$, y_N)} of labelled feature vectors where y ϵ {-1, +1}, we to estimate the vector **W** that minimizes the following objective function which measure the difference between the actual and desired outputs:-

$$J(\mathbf{W}) = (y_1\text{-sign}(\mathbf{W}.\mathbf{X}_1))^2 + (y_2\text{-sign}(\mathbf{W}.\mathbf{X}_2))^2 + \ldots + (y_N\text{-sign}(\mathbf{W}.\mathbf{X}_N))^2$$

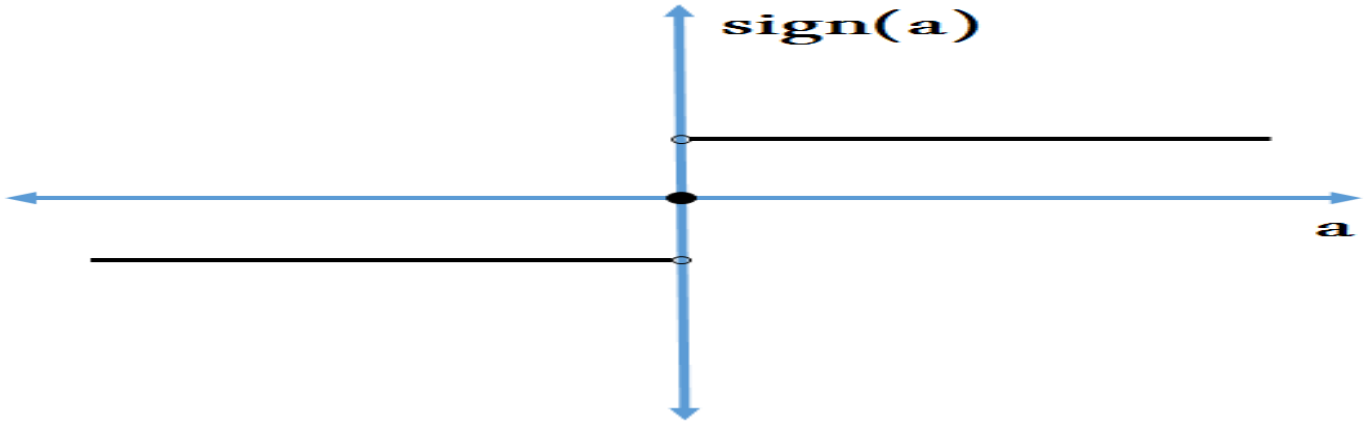


Derivative-based Optimization

- We need to find W that satisfies the following condition:-

$$\frac{\partial J}{\partial W} = 0$$

- First, how can we solve such an equation in multi dimensions?
- Second, the sign function has a derivative of zero every where except at the origin. Can we use a better objective function?



Again Learning W: An Optimization Problem

- Formulate learning problem as an optimization problems
  - Given: A set of *N* training examples

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, yn)\}$$

  - Design a loss function *L*
  - Find the weight vector $\mathbf{w}$ that minimizes the expected/average loss on training data

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} L(\mathbf{w}^T\mathbf{x}_m, y_m)$$

Better Loss Function

- 0/1 Loss function: $J_{0/1}(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} L(sign(\mathbf{w}^T\mathbf{x}_m), y_m)$

  where $L(y', y) = 0$ when $y' = y$, otherwise $L(y', y) = 1$

- Issue: does not produce useful gradient since the surface of J is piece-wise flat
- Instead we will consider the "**perceptron criterion**"

$$J_p(w) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

- The term $\max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$ is 0 when y_m is predicted correctly, otherwise it is equal to $|\mathbf{w}^T \mathbf{x}_m|$, which can be viewed as the "confidence" in the mis-prediction
- Has a nice gradient leading to the solution region

Gradient Descent

- The objective function consists of a sum over data points---we can update the parameter after observing each example
- This is the Stochastic gradient descent approach

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

$$J_m(\mathbf{w}) = \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

$$\nabla J_m = \begin{cases} 0 & \text{if } y_m \mathbf{w} \cdot \mathbf{x}_m > 0 \\ -y_m \mathbf{x}_m & \text{otherwise} \end{cases}$$

After observing $(\mathbf{x}_m, y_m)$, if it is a mistake $\mathbf{w} \leftarrow \mathbf{w} + y_m \mathbf{x}_m$

Batch Training

Input: $(\mathbf{X}_m, y_m)$, m = 1, 2, …,N

Set $\mathbf{W} = [0, 0,.., 0]^T$

Repeat

    delta $= [0, 0,.., 0]^T$

    for m = 1 to N do

        if $y_m \mathbf{W}.\mathbf{X}_m <= 0$

            delta = **delta** – $y_m \mathbf{X}_m$

    delta = **delta** / N

    W = **W** – **delta**

Until ||**delta**|| < ϵ

Online Training

Input: $(\mathbf{X}_m, y_m)$, m = 1, 2, …,N

Set $\mathbf{W} = [0, 0,.., 0]^T$

Repeat

    delta $= [0, 0,.., 0]^T$

    for m = 1 to N do

        if $y_m \mathbf{W}.\mathbf{X}_m <= 0$

            delta = **delta** – $y_m \mathbf{X}_m$

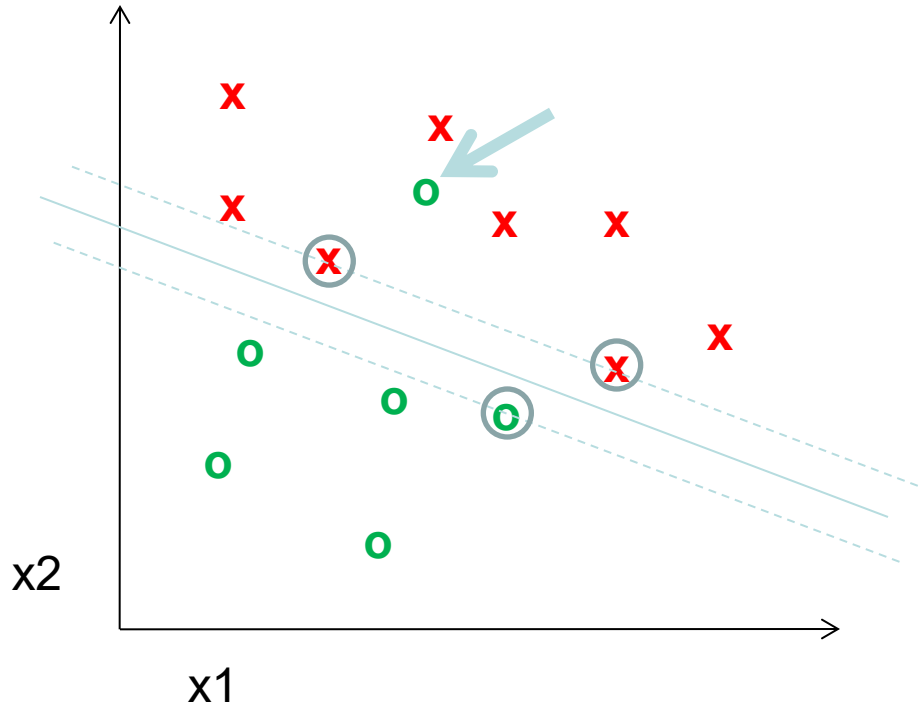
            W = **W** – **delta** / N

Until ||**delta**|| < ϵ

Bipolar Perceptron and SVM

Find a *linear function* to separate the classes:

$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$

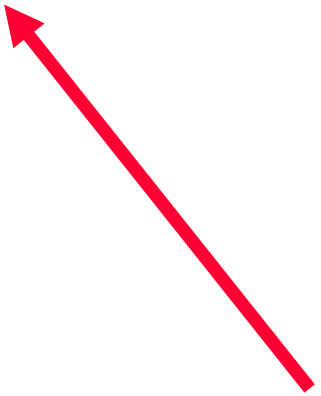


What if my data are not linearly separable?

Introduce flexible ‘hinge’ loss (or ‘soft-margin’)

Relation to SVM: Modified Loss Function

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} \max(0,\ 1 - y_m \mathbf{w}^T \mathbf{x}_m)$$



Introduce flexible ‘hinge’ loss (or ‘soft-margin’)

Relation to SVM: Modified Loss Function

Does this affect the training algorithm?

Choice of Activation Function (1)

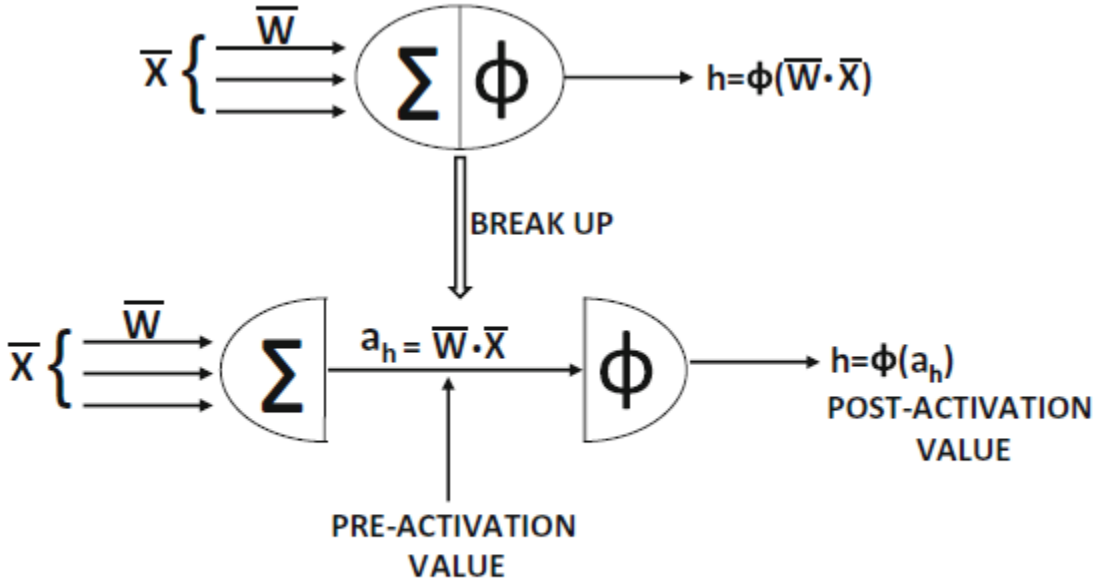


Figure 1.7: Pre-activation and post-activation values within a neuron

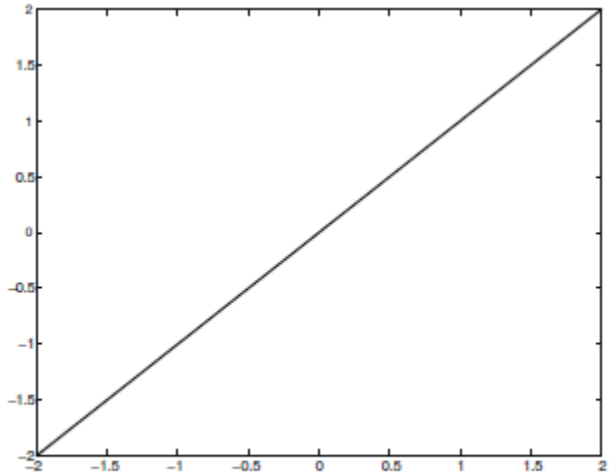
Choice of Activation Function (2)

The classical activation functions that were used early in the development of neural networks were the sign, sigmoid, and the hyperbolic tangent functions:

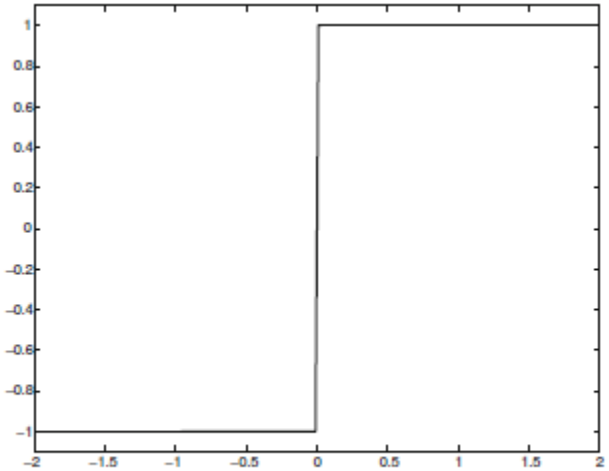
$$\Phi(v) = \text{sign}(v) \text{ (sign function)}$$

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

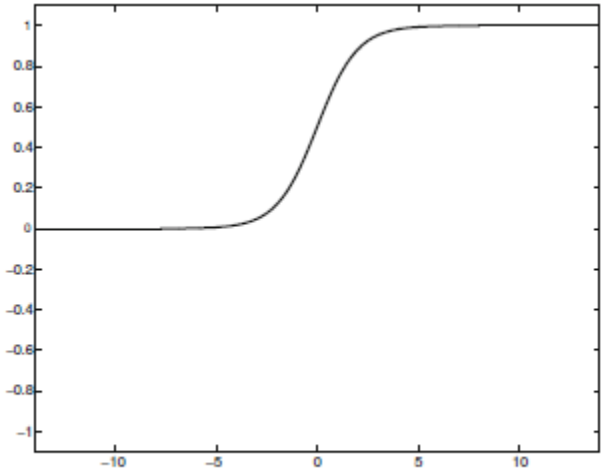
$$\Phi(v) = \frac{e^{2v}-1}{e^{2v}+1} \text{ (tanh function)}$$



(a) Identity



(b) Sign



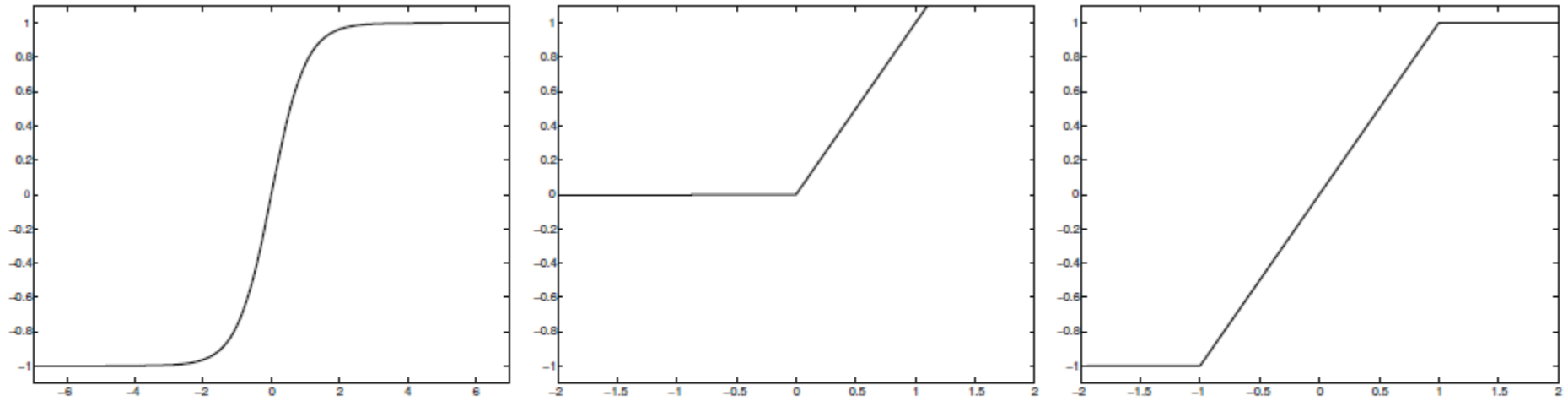
(c) Sigmoid

Choice of Activation Function (3)

The sigmoid and the tanh functions have been the historical tools of choice for incorporating nonlinearity in the neural network. In recent years, however, a number of piecewise linear activation functions have become more popular:

$$\Phi(v) = \max\{v, 0\} \text{ (Rectified Linear Unit [ReLU])}$$
$$\Phi(v) = \max\{\min[v, 1], -1\} \text{ (hard tanh)}$$

The ReLU and hard tanh activation functions have largely replaced the sigmoid and soft tanh activation functions in modern neural networks because of the ease in training multi-layered neural networks with these activation functions.



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

Figure 1.8: Various activation functions

Sigmoid and tanh Derivatives

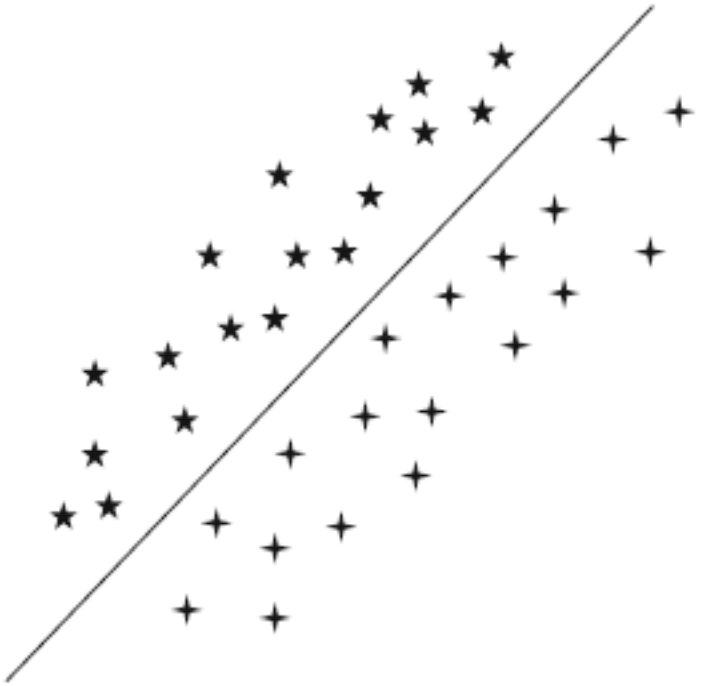
Sigmoid (u) = f(u)

df/du = f(1-f)

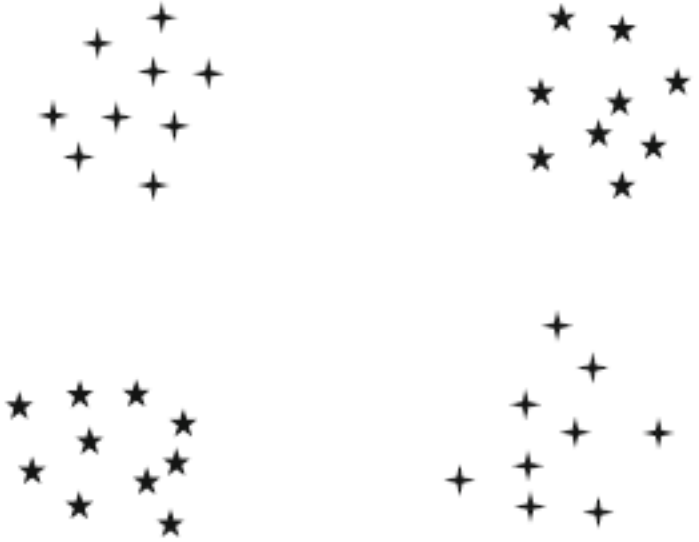
tanh (v) = g(v)

dg/dv = $(1-g^2)$

Will it work? Do need a multilayer network?

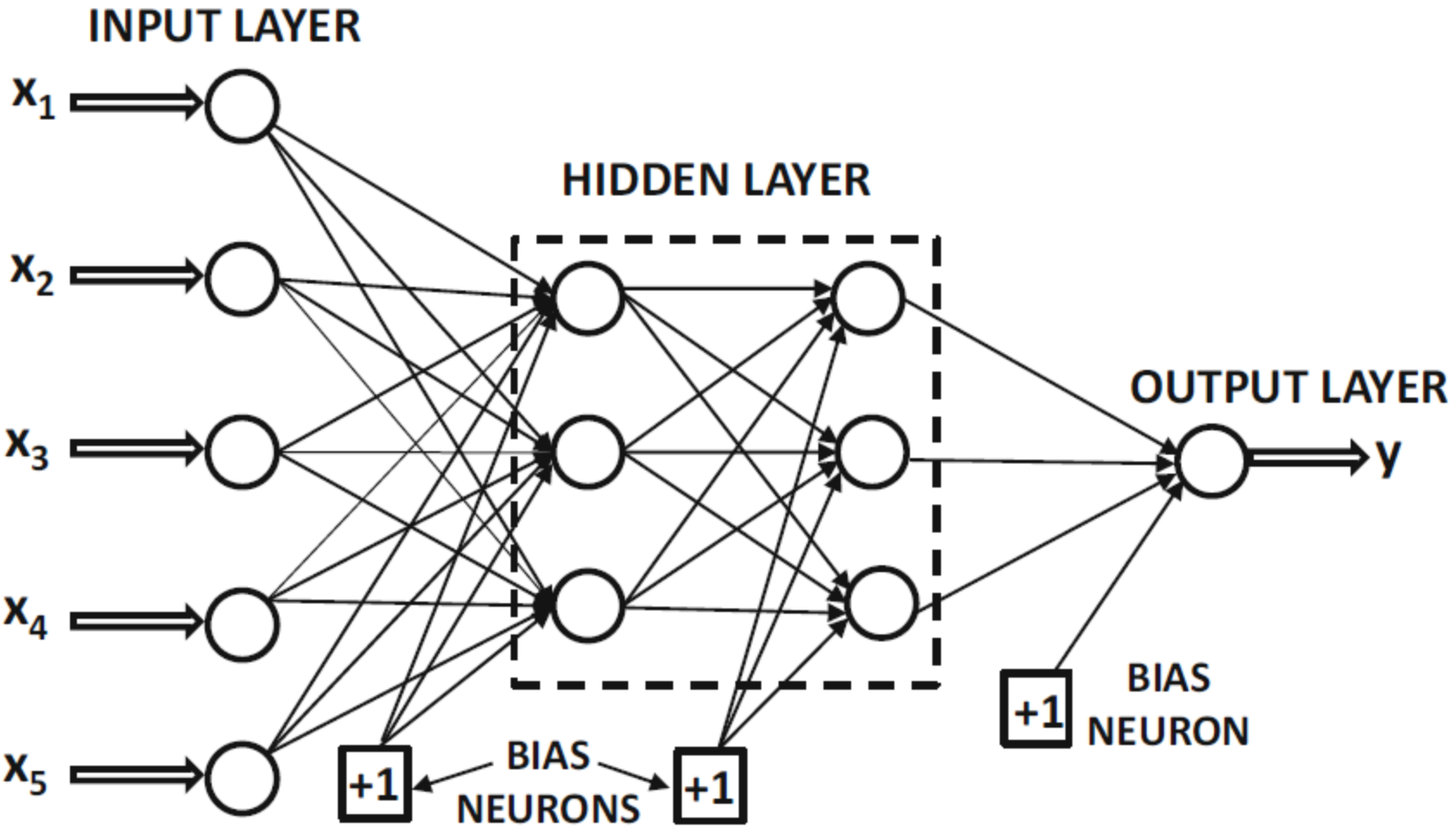


LINEARLY SEPARABLE



NOT LINEARLY SEPARABLE

A Multilayer Network



Multiple Outputs

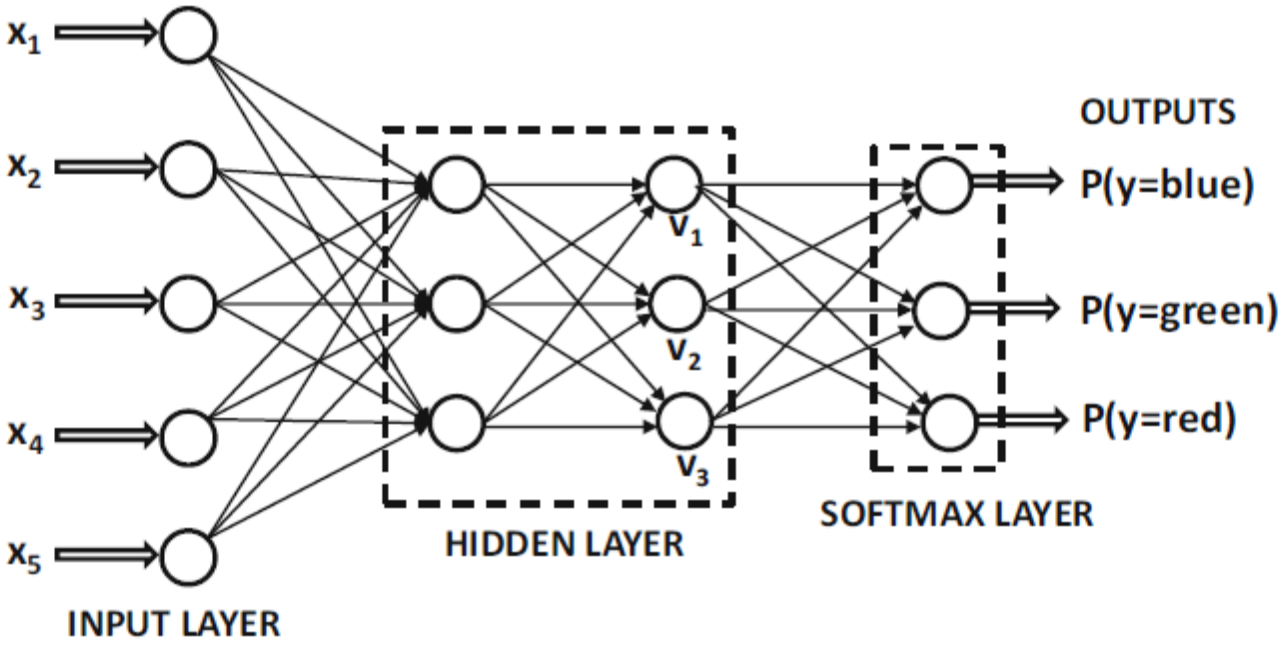


Figure 1.9: An example of multiple outputs for categorical classification with the use of a softmax layer

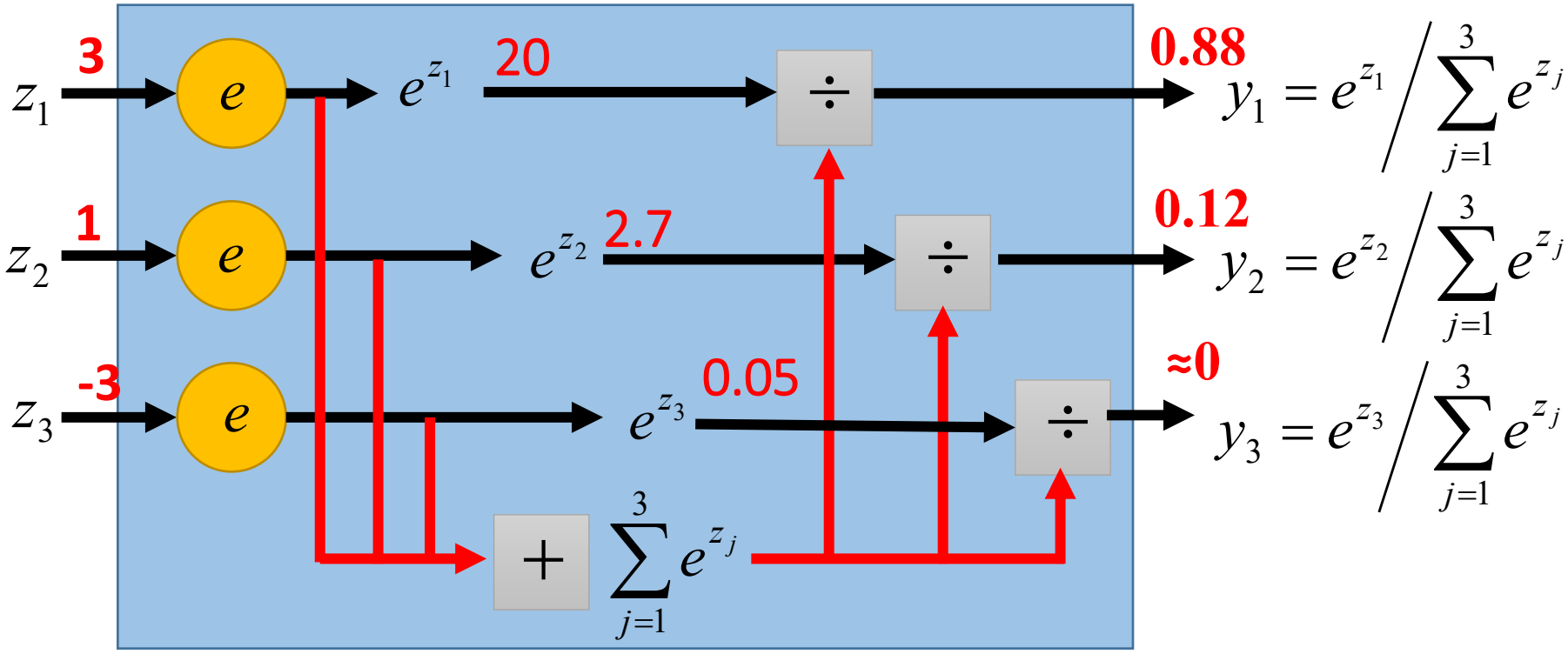
Probabilistic Prediction
Index of Maximum is the Prediction

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



Choice of Loss Function for Probabilistic Prediction

Two Categories: Binary Classification Identity Activation Function

Binary targets (logistic regression): In this case, it is assumed that the observed value y is drawn from $\{-1, +1\}$, and the prediction $\hat{y}$ is a an arbitrary numerical value on using the identity activation function. In such a case, the loss function for a single instance with observed value y and real-valued prediction $\hat{y}$ (with identity activation) is defined as follows:

$$L = \log(1 + \exp(-y \cdot \hat{y})) \tag{1.14}$$

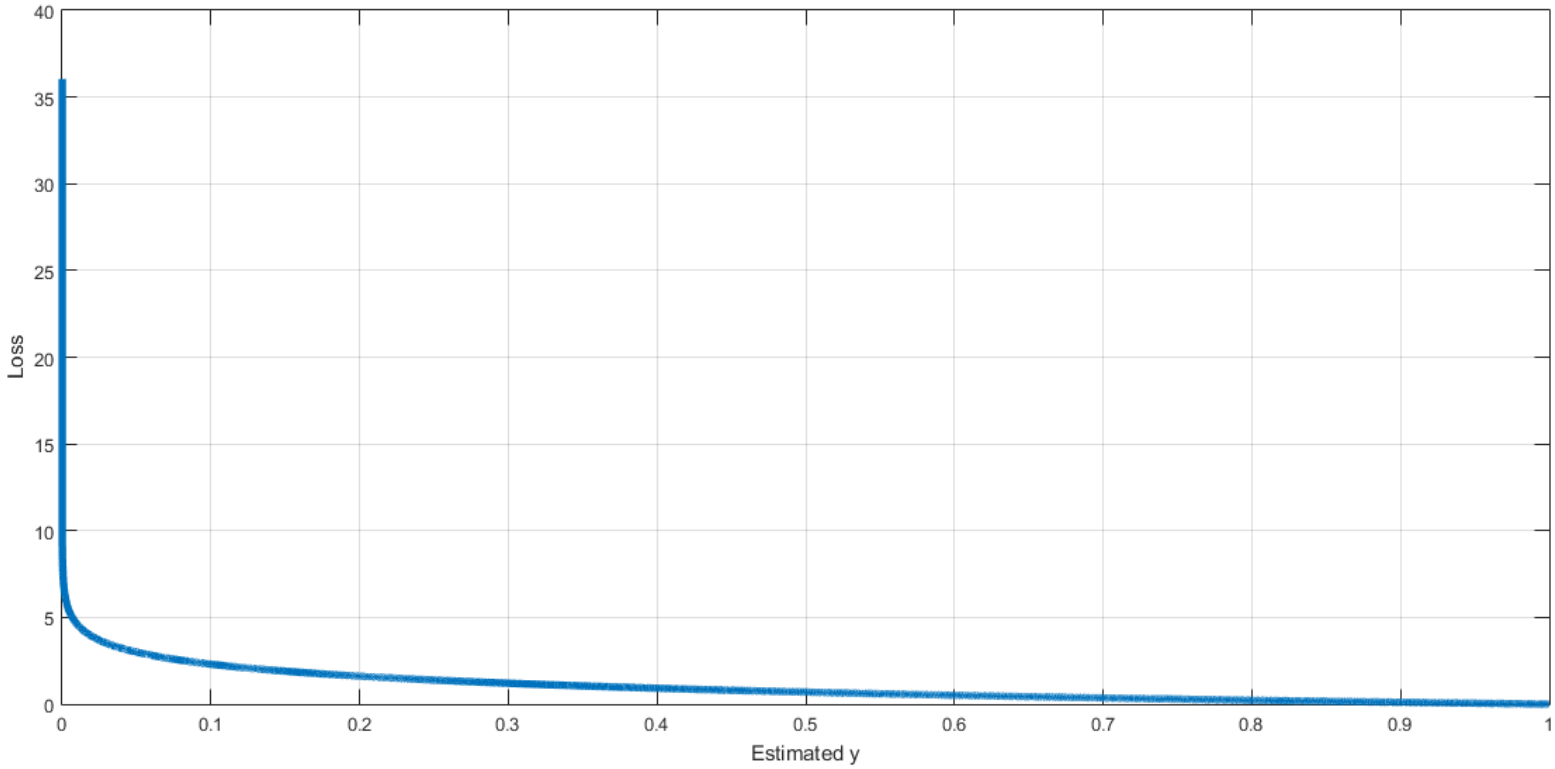
Two Categories: Binary Classification Sigmoid Activation Function

This type of loss function implements a fundamental machine learning method, referred to as *logistic regression*. Alternatively, one can use a sigmoid activation function to output $\hat{y} \in (0, 1)$, which indicates the probability that the observed value y is 1. Then, the negative logarithm of $|y/2 - 0.5 + \hat{y}|$ provides the loss, assuming that y is coded from $\{-1, 1\}$. This is because $|y/2 - 0.5 + \hat{y}|$ indicates the probability that the prediction is correct. This observation illustrates that one can use various combinations of activation and loss functions to achieve the same result.

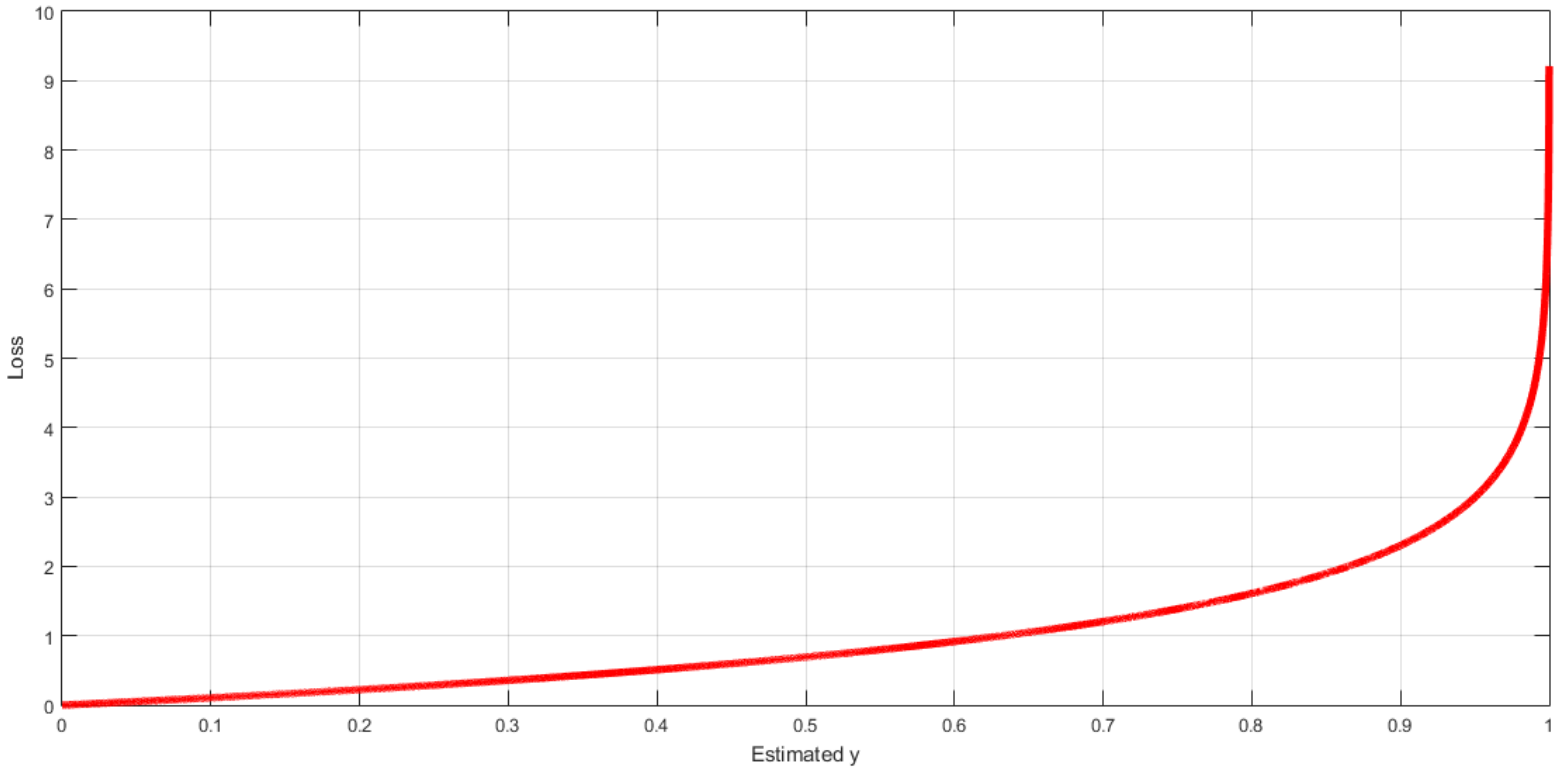
$$L = -\log(|\frac{y}{2} - \frac{1}{2} + \widehat{y}|)$$

$$\widehat{y} = \frac{1}{1 + e^{-(\mathbf{W}.\mathbf{X})}}$$

Two Categories: Binary Classification Sigmoid Activation Function (y=1)



Two Categories: Binary Classification Sigmoid Activation Function (y = -1)



Multi-Classifications

Categorical targets: In this case, if $\hat{y}_1 \ldots \hat{y}_k$ are the probabilities of the k classes (using the softmax activation of Equation 1.9), and the rth class is the ground-truth class, then the loss function for a single instance is defined as follows:

$$L = -\log(\hat{y}_r) \tag{1.15}$$

Machine Learning with Shallow Neural Networks

Neural Architectures for Binary Classification Models

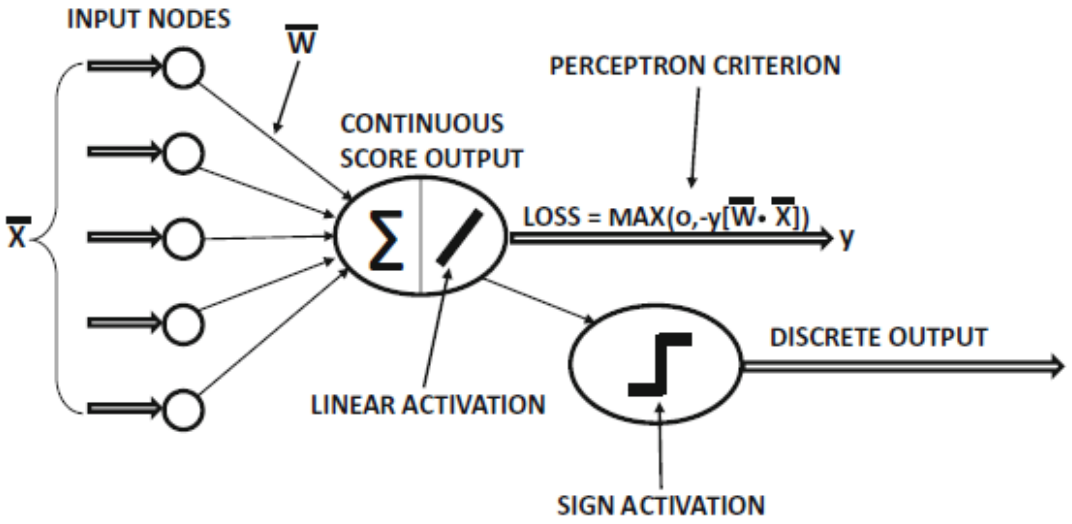
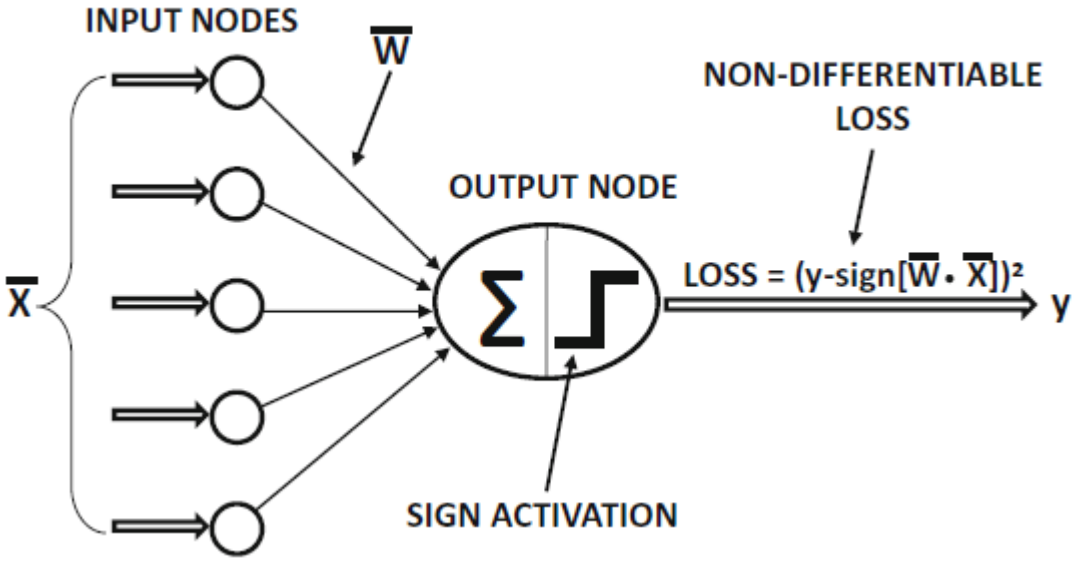
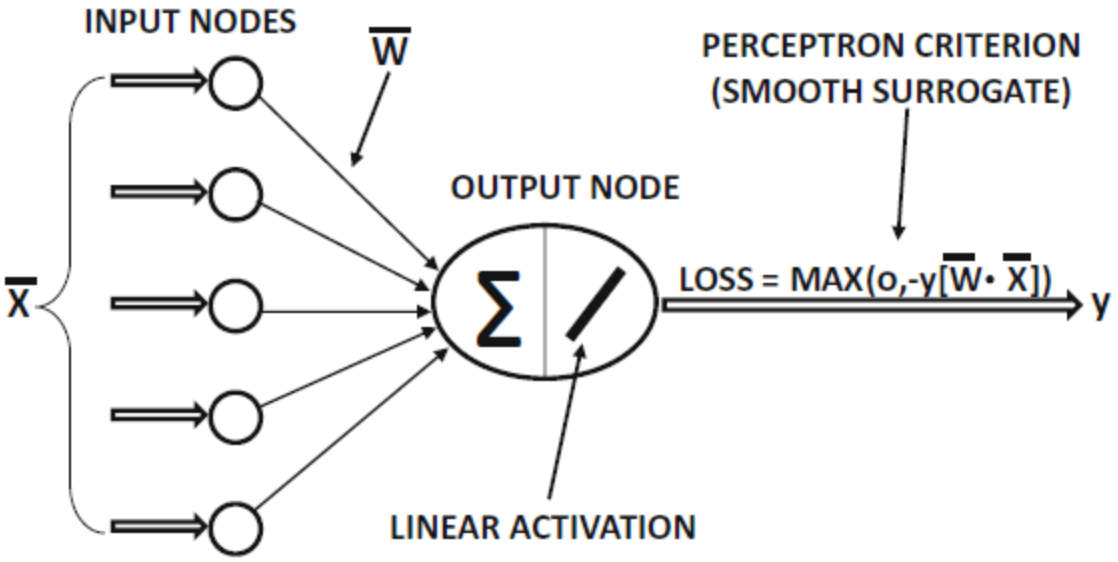


Figure 2.2: An extended architecture of the perceptron with both discrete and continuous predictions

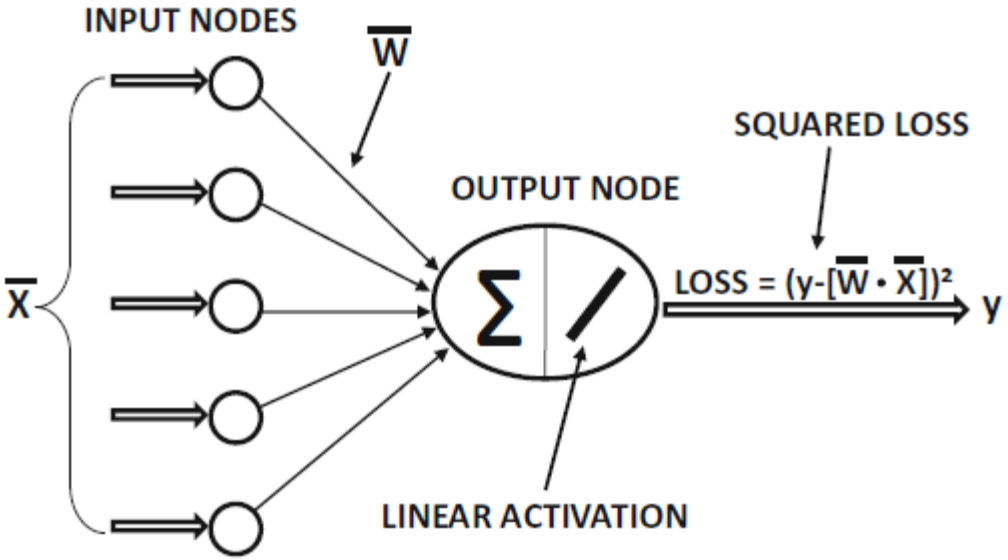
Different Variants of the Perceptron Binary Classifier – Bad Loss



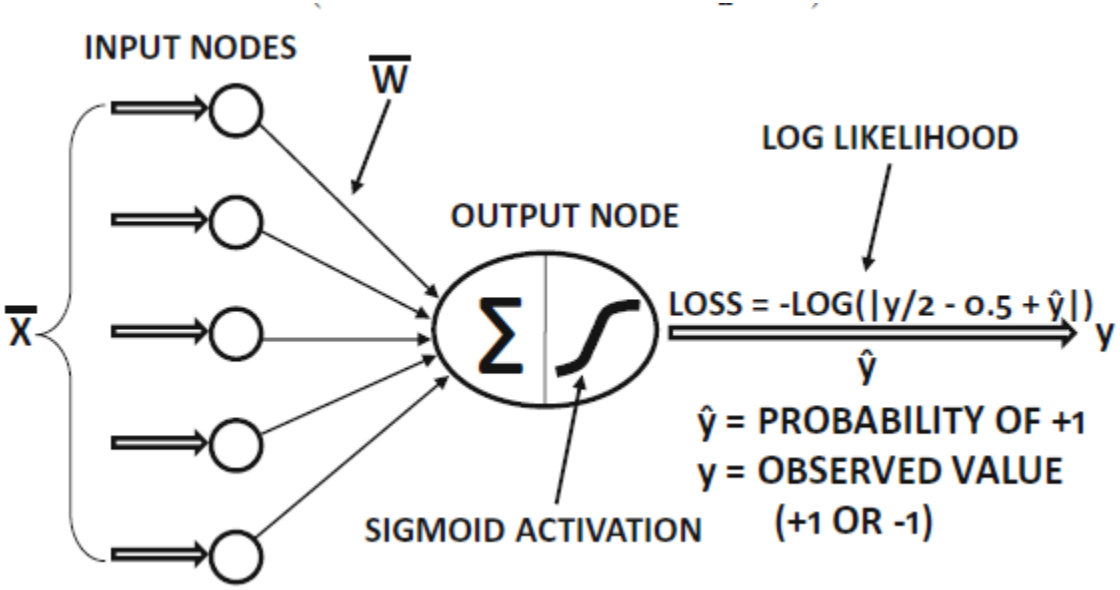
Different Variants of the Perceptron Binary Classifier – Smooth Loss Function



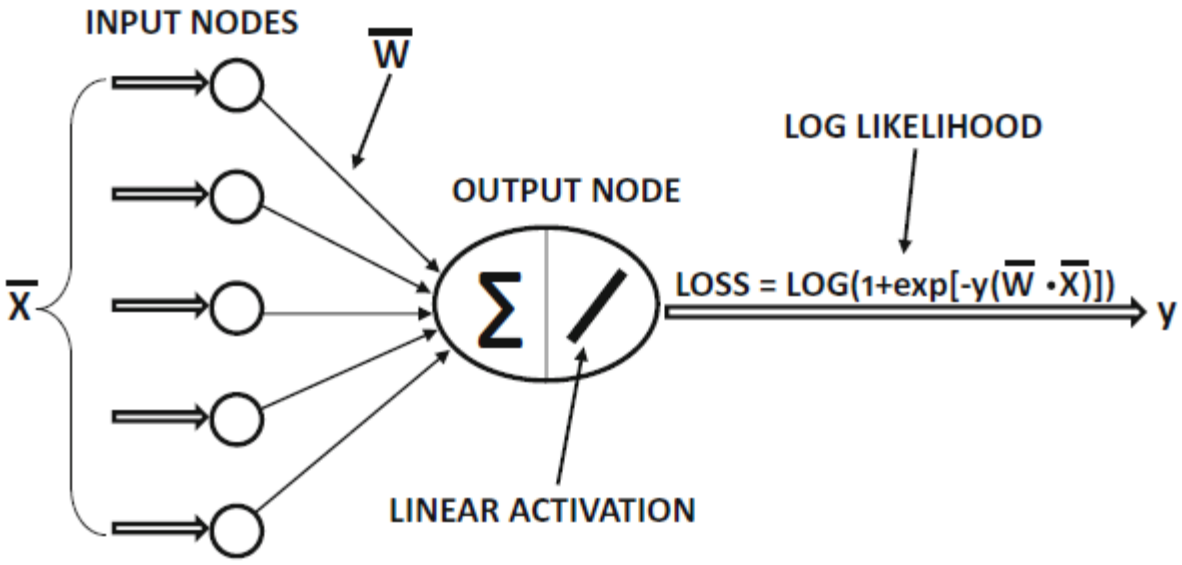
Different Variants of the Perceptron
Linear Regression – SSD Loss Function



Different Variants of the Perceptron Logistic Regression – LOG Likelihood Loss Function



Different Variants of the Perceptron Logistic Regression – LOG Likelihood Loss Function (Alternative)



Different Variants of the Perceptron Binary Classifier – Hinge Loss (SVM)