

# Deep Learning

## Activation Functions

Tiago Vieira

Institute of Computing  
Universidade Federal de Alagoas

# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

Why Do We Need Activation Functions?

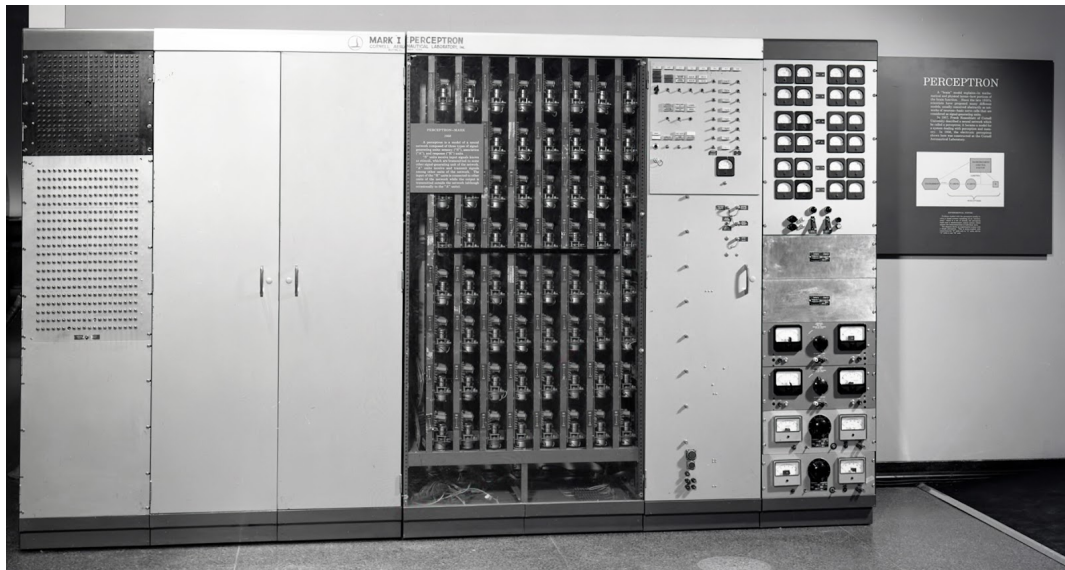
Factors

Examples of Activation Functions

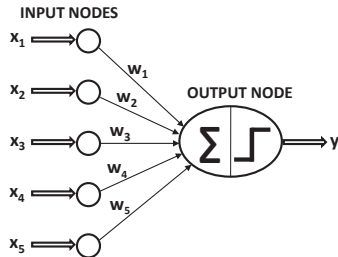
# Binary Classification and Linear Regression Problems

- ▶ In the binary classification problem, each training pair  $(\bar{X}, y)$  contains feature variables  $\bar{X} = (x_1, \dots, x_d)$ , and label  $y$  drawn from  $\{-1, +1\}$ .
  - Example: Feature variables might be frequencies of words in an email, and the class variable might be an indicator of spam.
  - Given labeled emails, recognize incoming spam.
- ▶ In linear regression, the *dependent* variable  $y$  is real-valued.
  - Feature variables are frequencies of words in a Web page, and the dependent variable is a prediction of the number of accesses in a fixed period.
- ▶ Perceptron is designed for the binary setting.

# The Perceptron (proposed by Frank Rosenblatt in 1958)



# The Perceptron: Earliest Historical Architecture



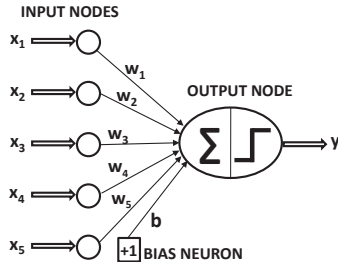
- ▶ The  $d$  nodes in the input layer only transmit the  $d$  features  $\overline{X} = [x_1 \dots x_d]$  without performing any computation.
- ▶ Output node multiplies input with weights  $\overline{W} = [w_1 \dots w_d]$  on incoming edges, aggregates them, and applies *sign activation*:

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X}\} = \text{sign}\left\{\sum_{j=1}^d w_j x_j\right\}$$

## What is the Perceptron Doing?

- ▶ Tries to find a *linear separator*  $\overline{W} \cdot \overline{X} = 0$  between the two classes.
- ▶ Ideally, all positive instances ( $y = 1$ ) should be on the side of the separator satisfying  $\overline{W} \cdot \overline{X} > 0$ .
- ▶ All negative instances ( $y = -1$ ) should be on the side of the separator satisfying  $\overline{W} \cdot \overline{X} < 0$ .

## Bias Neurons



- In many settings (e.g., skewed class distribution) we need an invariant part of the prediction with bias variable  $b$ :

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\left\{\sum_{j=1}^d w_j x_j + b\right\} = \text{sign}\left\{\sum_{j=1}^{d+1} w_j x_j\right\}$$

- On setting  $w_{d+1} = b$  and  $x_{d+1}$  as the input from the bias neuron, it makes little difference to learning procedures  $\Rightarrow$  Often implicit in architectural diagrams

# Training a Perceptron

- ▶ Go through the input-output pairs  $(\bar{X}, y)$  one by one and make updates, if predicted value  $\hat{y}$  is different from observed value  $y \Rightarrow$  Biological readjustment of synaptic weights.

$$\bar{W} \Leftarrow \bar{W} + \underbrace{\alpha (y - \hat{y})}_{\text{Error}} \bar{X}$$

$$\bar{W} \Leftarrow \bar{W} + (2\alpha)y\bar{X} \text{ [For misclassified instances } y - \hat{y} = 2y]$$

- ▶ Parameter  $\alpha$  is the learning rate.
- ▶ One cycle through the entire training data set is referred to as an *epoch*  $\Rightarrow$  Multiple epochs required
- ▶ How did we derive these updates?



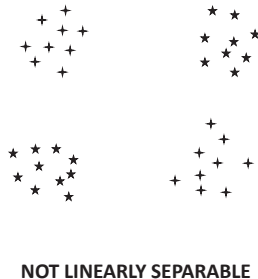
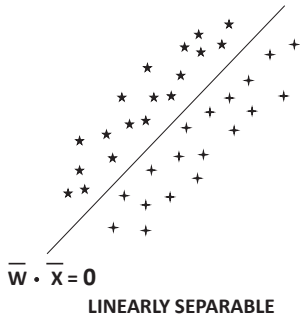
# What Objective Function is the Perceptron Optimizing?

- ▶ At the time, the perceptron was proposed, the notion of loss function was not popular  $\Rightarrow$  Updates were heuristic
- ▶ Perceptron optimizes the perceptron criterion for  $i$ th training instance:

$$L_i = \max\{-y_i(\overline{W} \cdot \overline{X}_i), 0\}$$

- Loss function tells us how far we are from a desired solution  $\Rightarrow$  Perceptron criterion is 0 when  $\overline{W} \cdot \overline{X}_i$  has same sign as  $y_i$ .
- ▶ Perceptron updates use *stochastic gradient descent* to optimize the loss function and reach the desired outcome.
  - Updates are equivalent to  $\overline{W} \leftarrow \overline{W} - \alpha \left( \frac{\partial L_i}{\partial w_1} \dots \frac{\partial L_i}{\partial w_d} \right)$

# Where does the Perceptron Fail?

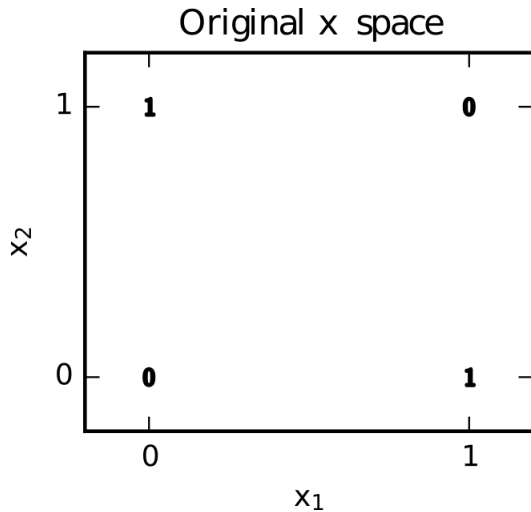


- ▶ The perceptron fails at similar problems as a linear SVM
  - **Classical solution:** Feature engineering with Radial Basis Function network  $\Rightarrow$  Similar to kernel SVM and good for noisy data
  - **Deep learning solution:** Multilayer networks with nonlinear activations  $\Rightarrow$  Good for data with a lot of structure

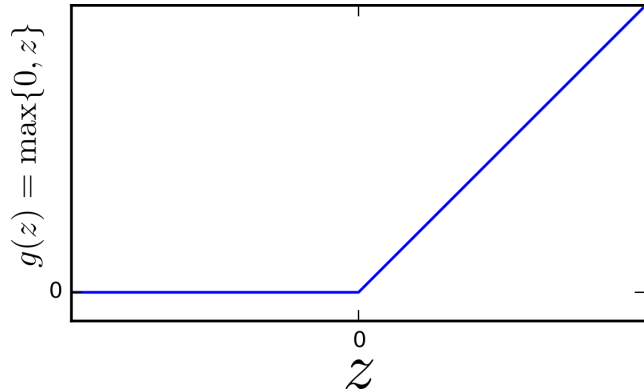
# The XOR Problem

- ▶ “*Perceptrons*” by Marvin Minsky and Seymour Papert (1969).
- ▶ Perceptrons cannot solve the XOR problem.
- ▶ Significant decline in interest and funding of neural network research.

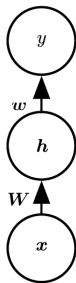
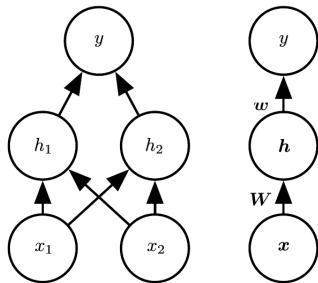
## The XOR Problem



## Rectified Linear Activation

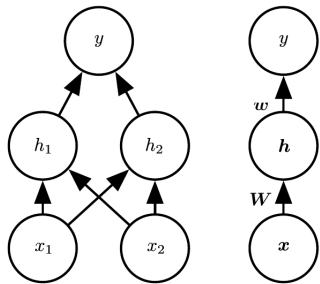


## Network Diagrams



$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$
$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

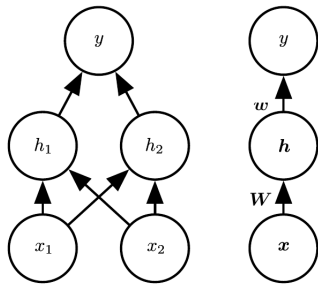
## Solving XOR



$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$
$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

$$X = [\mathbf{x}]_{i=1}^4 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$
$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$
$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$
$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$
$$b = 0$$

## Solving XOR



$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$
$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

$$H = \max(0, \mathbf{W}^T X + \mathbf{c})$$

$$H = \max\left(0, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

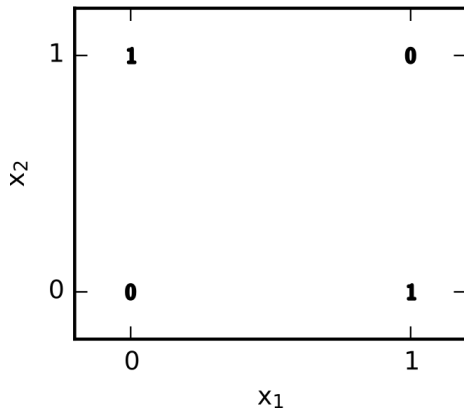
$$H = \max\left(0, \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

$$H = \max\left(0, \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}\right)$$

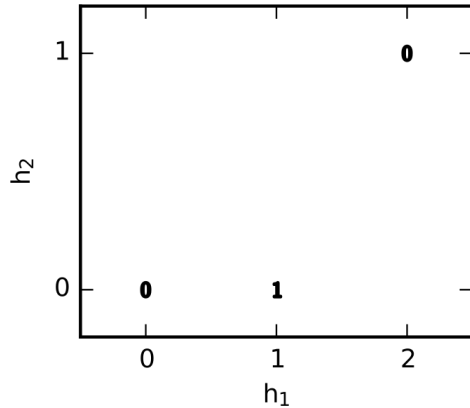
$$H = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

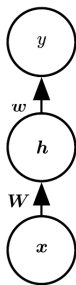
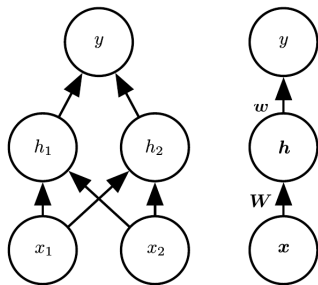


Original x space



Learned h space





$$\mathbf{h} = \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$

$$f(\mathbf{x}; (\mathbf{W}; \mathbf{c}); (\mathbf{w}, b)) = \mathbf{w}^T \mathbf{h} + b$$

$$Y = \max(0, \mathbf{w}^T H + \mathbf{b})$$

$$Y = \max \left( \begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right)$$

$$Y = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

## Rectified Linear Activation (ReLU)

- ▶ Applying this function to the output of a linear transformation yields a nonlinear transformation.
- ▶ Very close to linear.
- ▶ Very simple nonlinearity (2 pieces – piecewise-linear).
- ▶ Sufficient to represent any function if enough hidden units are connected.
- ▶ Default activation function recommended for use with most feedforward NNs.

### Why ReLU is so effective?

- ▶ Strong gradient. Gradient descent can compute large gradients.
- ▶ Consistent behavior across its whole domain.
- ▶ Historical reasons.

## Why Do We Need Activation Functions?

- ▶ A neural network with any number of layers but only linear activations can be shown to be equivalent to a single-layer network.
- ▶ An activation function  $\Phi(v)$  in the output layer can control the nature of the output (e.g., probability value in  $[0, 1]$ )
- ▶ In *multilayer* neural networks, activation functions bring nonlinearity into hidden layers, which increases the complexity of the model.
- ▶ Activation functions required for inference may be different from those used in loss functions in training.
  - Perceptron uses sign function  $\Phi(v) = \text{sign}(v)$  for prediction but does not use any activation for computing the perceptron criterion (during training).

## Why Do We Need Loss Functions?

- ▶ The loss function is typically paired with the activation function to quantify how far we are from a desired result.
- ▶ An example is the perceptron criterion.

$$L_i = \max\{-y(\overline{W} \cdot \overline{X}), 0\}$$

- ▶ Note that loss is 0, if the instance  $(\overline{X}, y)$  is classified correctly.
- ▶ Even though many machine learning problems have discrete outputs, a smooth and continuous loss function is required to enable *gradient-descent* procedures.
- ▶ Gradient descent is at the heart of neural network parameter learning.

# Factors

- ▶ Nonlinearity.
- ▶ Continuously differentiable.
- ▶ Range.
- ▶ Monotonicity.
- ▶ Smooth.
- ▶ Approximating identity near the origin.

# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

Why Do We Need Activation Functions?

Factors

Examples of Activation Functions

## Identity Activation

- ▶ Identity activation  $\Phi(v) = v$  is often used in the output layer, when the outputs are real values.
- ▶ For a single-layer network, if the training pair is  $(\overline{X}, y)$ , the output is as follows:

$$\hat{y} = \Phi(\overline{W} \cdot \overline{X}) = \overline{W} \cdot \overline{X}$$

- ▶ Use of the squared loss function  $(y - \hat{y})^2$  leads to the *linear regression* model with numeric outputs and *Widrow-Hoff learning* with binary outputs.
- ▶ Identity activation can be combined with various types of loss functions (e.g., perceptron criterion) even for discrete outputs.



# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

Why Do We Need Activation Functions?

Factors

Examples of Activation Functions

## Sign Activation

- ▶  $\Phi(v) = \begin{cases} +1 & \text{if } v > 0; \\ -1 & \text{if } v < 0. \end{cases}$
- ▶ Can be used to map to binary outputs at prediction time.
- ▶ Its non-differentiability prevents its use for creating the loss function at training time.
- ▶ Eg. while the perceptron uses the sign function for prediction, in training it requires only the linear activation.

# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

Why Do We Need Activation Functions?

Factors

Examples of Activation Functions

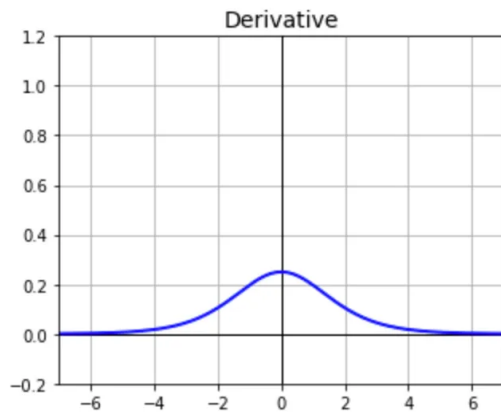
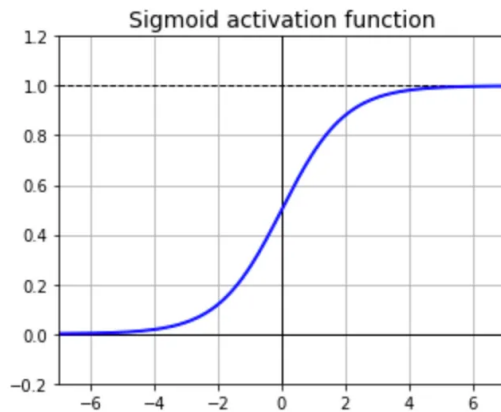
## Sigmoid Activation

- ▶ Sigmoid activation is defined as  $\Phi(v) = 1/(1 + \exp(-v))$ .
- ▶ For a training pair  $(\overline{X}, y)$ , one obtains the following prediction in a single-layer network:

$$\hat{y} = 1/(1 + \exp(-\overline{W} \cdot \overline{X}))$$

- ▶ Prediction is the *probability* that class label is +1.
- ▶ Paired with *logarithmic loss*, which  $-\log(\hat{y})$  for positive instances and  $-\log(1 - \hat{y})$  for negative instances.
  - $\mathcal{L}(\hat{y}, y) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$
- ▶ Resulting model is *logistic regression*.

# Sigmoid Activation



# Sigmoid Activation

Characteristics:

- ▶ The function is a common S-shaped curve.
- ▶ The output of the function is centered at 0.5 with a range from 0 to 1.
- ▶ The function is differentiable. That means we can find the slope of the sigmoid curve at any two points.
- ▶ The function is monotonic but the function's derivative is not.

# Sigmoid Activation

Drawbacks:

- ▶ Vanishing gradient.
- ▶ Computationally expensive.
- ▶ The output is not zero-centered.

# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

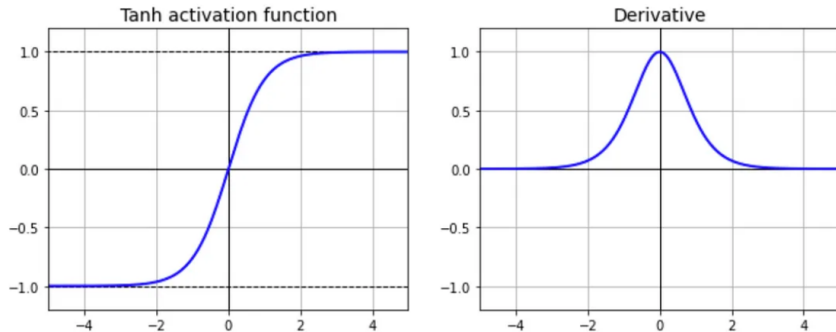
Why Do We Need Activation Functions?

Factors

Examples of Activation Functions



# Tanh Activation



- The tanh activation is a scaled and translated version of sigmoid activation.

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

- Often used in hidden layers of multilayer networks

# Tanh Activation

## Characteristics.

- ▶ The function is a common S-shaped curve as well.
- ▶ The difference is that the output of Tanh is zero centered with a range from  $-1$  to  $+1$  (instead of 0 to 1 in the case of the Sigmoid function).
- ▶ The same as the Sigmoid, this function is differentiable.
- ▶ The same as the Sigmoid, the function is monotonic, but the function's derivative is not.

# Tanh Activation

Problems:

- ▶ Vanishing gradient.
- ▶ Computationally expensive.

# Summary

Binary Classification and Linear Regression Problems

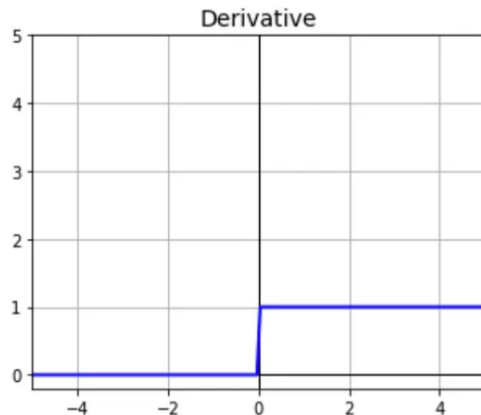
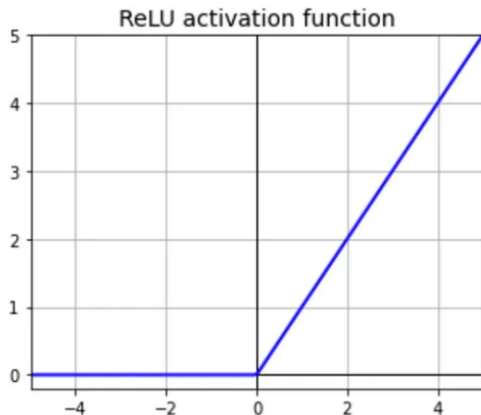
The XOR Problem

Why Do We Need Activation Functions?

Factors

Examples of Activation Functions

## Piecewise Linear Activation Functions – ReLU



$$\Phi(v) = \max\{v, 0\}$$

- Piecewise linear activation functions are easier to train than their continuous counterparts.

# Piecewise Linear Activation Functions – ReLU

Characteristics:

- ▶ Graphically, the ReLU function is composed of two linear pieces to account for non-linearity.
- ▶ The ReLU function is continuous, but it is not differentiable because its derivative is 0 for any negative input.
- ▶ The output of ReLU does not have a maximum value (It is not saturated) and this helps Gradient Descent.
- ▶ The function is very fast to compute.

# Piecewise Linear Activation Functions – ReLU

Problem:

- ▶ Dying ReLU.

# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

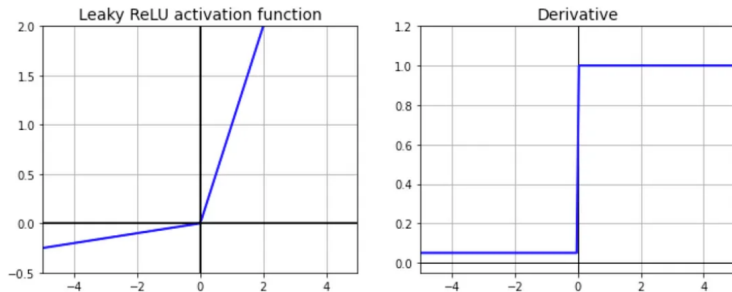
Why Do We Need Activation Functions?

Factors

Examples of Activation Functions



## Piecewise Linear Activation Functions – Leaky ReLU



$$\Phi(v) = \max\{\gamma v, v\}$$

Where typically:  $\gamma = 0.01$ .

# Summary

Binary Classification and Linear Regression Problems

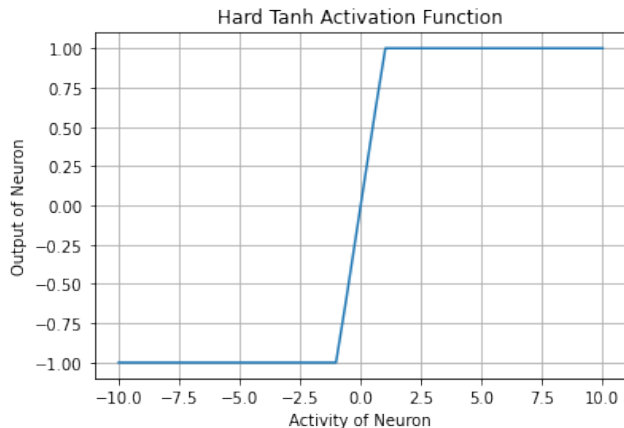
The XOR Problem

Why Do We Need Activation Functions?

Factors

Examples of Activation Functions

## Piecewise Linear Activation Functions – Hard TANH



$$\Phi(v) = \max \{ \min [v, 1], -1 \}$$

# Summary

Binary Classification and Linear Regression Problems

The XOR Problem

Why Do We Need Activation Functions?

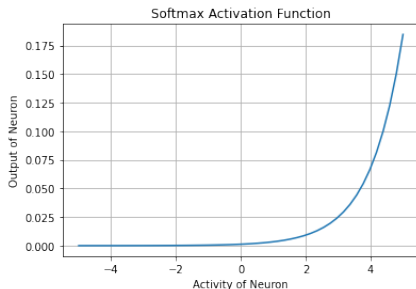
Factors

Examples of Activation Functions

# Softmax Activation Function

- ▶ All activation functions discussed so far map scalars to scalars.
- ▶ The softmax activation function maps vectors to vectors.
- ▶ Useful in mapping a set of real values to probabilities.
  - Generalization of sigmoid activation, which is used in *multiway* logistic regression.

$$\Phi(\bar{v})_i = \frac{e^{(v_i)}}{\sum_{j=1}^k e^{(v_j)}}$$



# Derivatives of Activation Functions

- ▶ Neural network learning requires gradient descent of the loss.
- ▶ Loss is often a function of the output  $o$ , which is itself obtained by using the activation function:

$$o = \Phi(v) \tag{1}$$

- ▶ Therefore, we often need to compute the partial derivative of  $o$  with respect to  $v$  during neural network parameter learning.
- ▶ Many derivatives are more easily expressed in terms of the output  $o$  rather than input  $v$ .

## Useful Derivatives

- ▶ Sigmoid:  $\frac{\partial o}{\partial v} = o(1 - o)$
- ▶ Tanh:  $\frac{\partial o}{\partial v} = 1 - o^2$
- ▶ ReLU: Derivative is 1 for positive values of  $v$  and 0 otherwise.
- ▶ Hard Tanh: Derivative is 1 for  $v \in (-1, 1)$  and 0 otherwise.

# Output Types

<b>Output Type</b>	<b>Output distribution</b>	<b>Out. Layer Act. Func.</b>	<b>Cost Function</b>
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	MSE
Continuous	Arbitrary	GAN, VAE, FVBN	Various



# Using Activation Functions

- ▶ The nature of the activation in output layers is often controlled by the nature of output
  - Identity activation for real-valued outputs, and sigmoid/softmax for binary/categorical outputs.
  - Softmax almost exclusively for output layer and is paired with a particular type of *cross-entropy* loss.
- ▶ Hidden layer activations are almost always nonlinear and often use the same activation function over the entire network.
  - Tanh often (but not always) preferable to sigmoid.
  - ReLU has largely replaced tanh and sigmoid in many applications.

## Why are Hidden Layers Nonlinear?

- ▶ A multi-layer network that uses only the identity activation function in all its layers reduces to a single-layer network that performs linear regression.

$$\bar{h}_1 = \Phi(W_1^T \bar{x}) = W_1^T \bar{x}$$

$$\bar{h}_{p+1} = \Phi(W_{p+1}^T \bar{h}_p) = W_{p+1}^T \bar{h}_p \quad \forall p \in \{1 \dots k-1\}$$

$$\bar{o} = \Phi(W_{k+1}^T \bar{h}_k) = W_{k+1}^T \bar{h}_k$$

- ▶ We can eliminate the hidden variable to get a simple linear relationship:

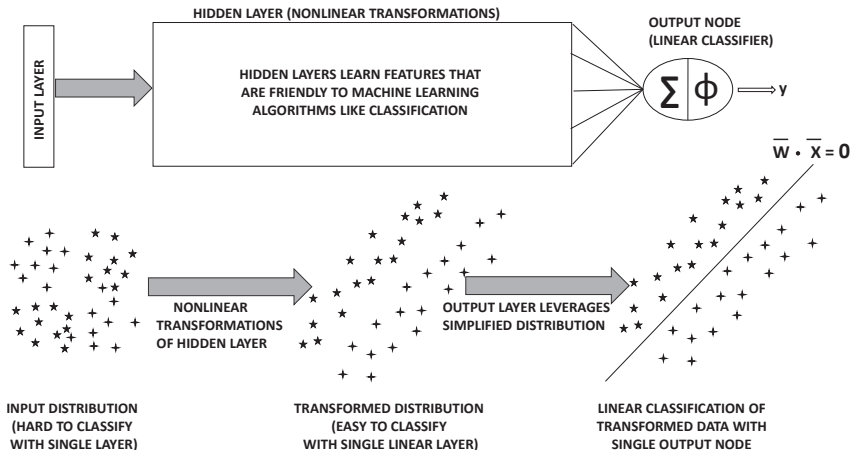
$$\begin{aligned}\bar{o} &= W_{k+1}^T W_k^T \dots W_1^T \bar{x} \\ &= \underbrace{(W_1 W_2 \dots W_{k+1})^T}_{W_{xo}^T} \bar{x}\end{aligned}$$

- ▶ We get a *single-layer* network with matrix  $W_{xo}$ .

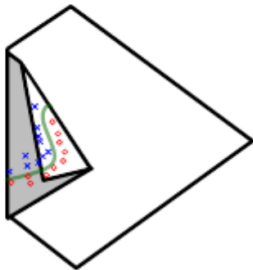
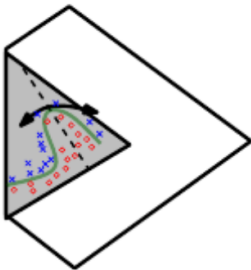
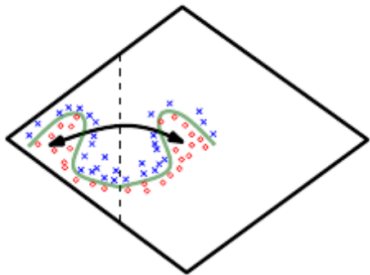
# Role of Hidden Layers

- ▶ Nonlinear hidden layers perform the role of hierarchical feature engineering.
  - Early layers learn primitive features and later layers learn more complex features
  - Image data: Early layers learn elementary edges, the middle layers contain complex features like honeycombs, and later layers contain complex features like a part of a face.
  - **Deep learners are masters of feature engineering.**
- ▶ The final output layer is often able to perform inference with transformed features in penultimate layer relatively easily.
- ▶ **Perceptron:** Cannot classify linearly inseparable data but can do so with *nonlinear* hidden layers.

# The Feature Engineering View of Hidden Layers



- Early layers play the role of feature engineering for later layers.



Thank you!  
[tvieira@ic.ufal.br](mailto:tvieira@ic.ufal.br)