



activation-
functions...

Deep Learning
Activation Functions
Tiago Vieira
Institute of Informatics
Universidade Federal do Rio de Janeiro
March 21, 2023

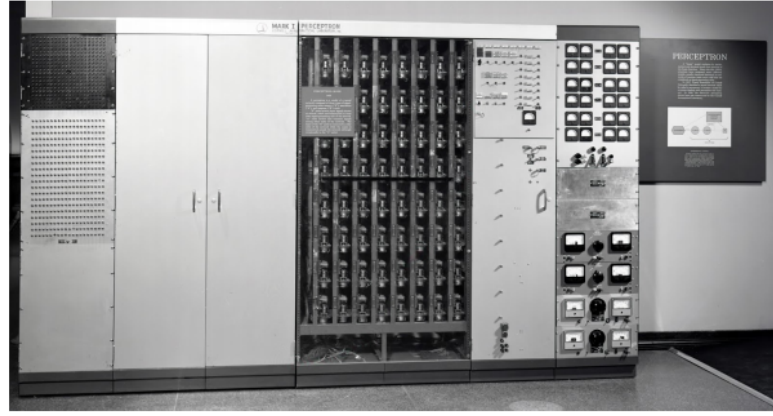
Summary

Binary Classification and Linear Regression Problems
The XOR Problem
Why Do We Need Activation Functions?
Factors
Examples of Activation Functions

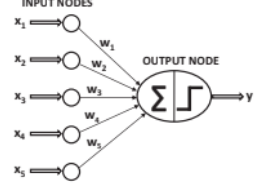
Binary Classification and Linear Regression Problems

- In the binary classification problem, each training pair (\vec{X}, y) contains feature variables $\vec{X} = (x_1, \dots, x_d)$, 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.
- In linear regression, the dependent variable y is real-valued.
Example: Feature variables might be a patient's age, weight, and the dependent variable is a prediction of the number of strokes in a fixed period.
- Perceptron is designed for the binary setting.

The Perceptron (proposed by Frank Rosenblatt in 1958)



The Perceptron: Earliest Historical Architecture

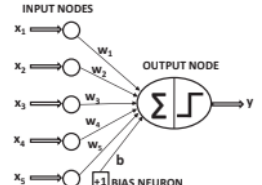


- The d nodes in the input layer only transmit the d features $\vec{X} = [x_1, \dots, x_d]$ without performing any computation.
- Output node multiplies input with weights $\vec{W} = [w_1, \dots, w_d]$ on incoming edges, aggregates them, and applies sign activation:
$$\hat{y} = \text{sign}(\vec{W} \cdot \vec{X}) = \text{sign}\left(\sum_{j=1}^d w_j x_j\right)$$

What is the Perceptron Doing?

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

Bias Neurons



- In many settings (e.g., skewed class distribution) we need an invariant part of the prediction with bias variable b :
$$\hat{y} = \text{sign}(\vec{W} \cdot \vec{X} + b) = \text{sign}\left(\sum_{j=1}^d w_j x_j + b\right) = \text{sign}\left(\sum_{j=1}^{d+1} w_j x_j\right)$$
- On setting $w_{d+1} = b$ and $x_{d+1} = 1$ 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 (\vec{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:
$$\vec{W} \leftarrow \vec{W} + \eta (y - \hat{y}) \vec{X}$$

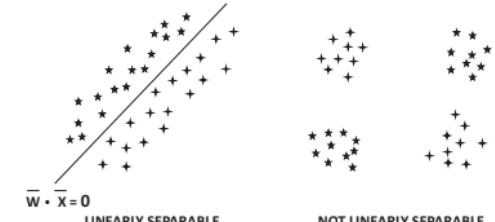
$$\vec{W} \leftarrow \vec{W} + \eta (y - \hat{y}) \vec{X} \text{ Error}$$
- Parameter η is the learning rate.
- One cycle through the entire training data set is referred to as an epoch \Rightarrow Multiple epochs required.
- How did we derive these updates?

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

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

Where does the Perceptron Fail?



- The perceptron fails on similar problems as a linear SVM.
Classical solution: Feature engineering with Radial Basis Function network \Rightarrow Similar to kernel.
Only required for noisy data.
Other learning solution: Multilayer networks with nonlinear activation \Rightarrow Good for data with a lot of outliers.

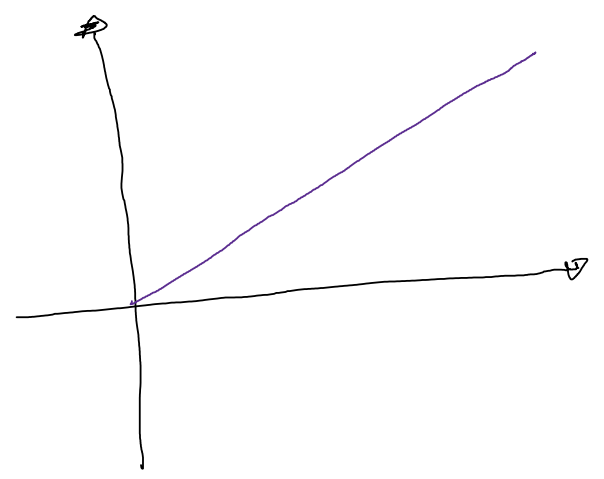
The XOR Problem

- "Perceptrons" by Marvin Minsky and Seymour Papert (1969).
- Perceptrons cannot solve the XOR problem.
- Significant decline in interest and funding of neural network research.
 \rightarrow Parte da falha da perceptron não pode nos levar a conclusões desprezíveis como a falha da XOR \rightarrow as funções de ativação

* Composição entre as funções de ativação

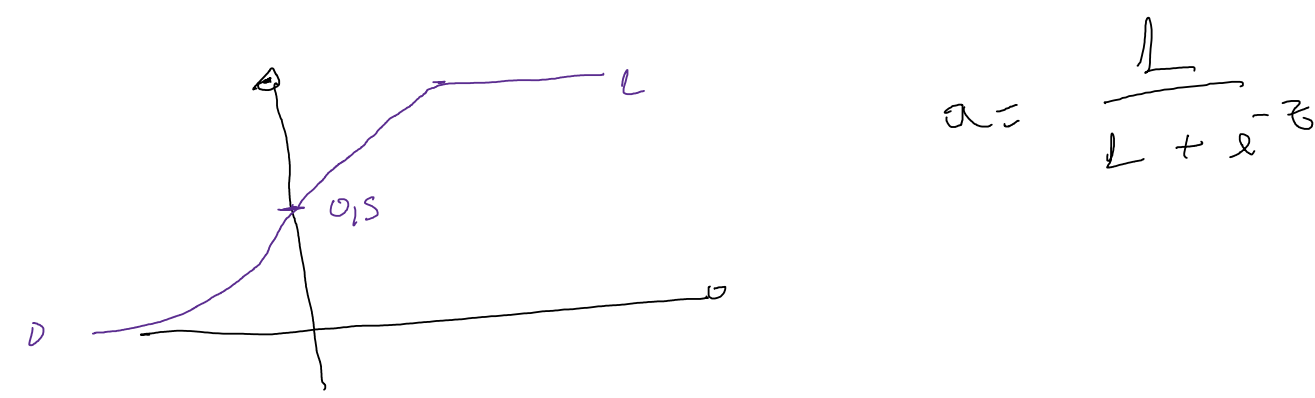
\rightarrow Identity / Linear Function

- \hookrightarrow Usado quando o output são números reais
 $\hookrightarrow f(x) = x$
- \hookrightarrow Usada na regressão linear se usamos o square loss



\rightarrow Sigmoid / Logistic

- \hookrightarrow Resulta em algo entre 0 e 1
- \hookrightarrow Boa p/ probabilidades
- \hookrightarrow É derivável
- \hookrightarrow Modelo da regressão logística quando usamos log loss



$$\sigma = \frac{1}{1 + e^{-x}}$$

\rightarrow Sign Function

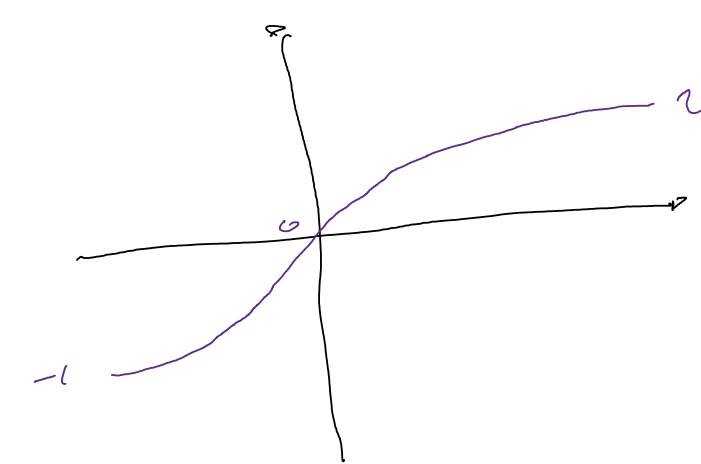
- \hookrightarrow Usado p/ converter o output em binário

$$\phi(v) = \begin{cases} 1 & \text{se } v > 0 \\ -1 & \text{se } v < 0 \end{cases}$$

- \hookrightarrow Usado pelo perceptron simples na predição

\rightarrow Tanh / Hyperbolic Function

- \hookrightarrow Outra versão, "melhorada" da função sigmoidal
- \hookrightarrow Range é de -1 a 1, centrado em zero
- \hookrightarrow Derivável



$$\sigma = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- \hookrightarrow Mais usada em hidden layers

- \hookrightarrow Fatores dela como a sigmoid são computacionalmente caros

- \hookrightarrow Vanishing Gradient

\rightarrow ReLU

- \hookrightarrow Rectified Linear Function

- \hookrightarrow Dois pedaços lineares para contornar a não linearidade

- \hookrightarrow Contínuo mas não derivável

- \hookrightarrow Problema: dying ReLU

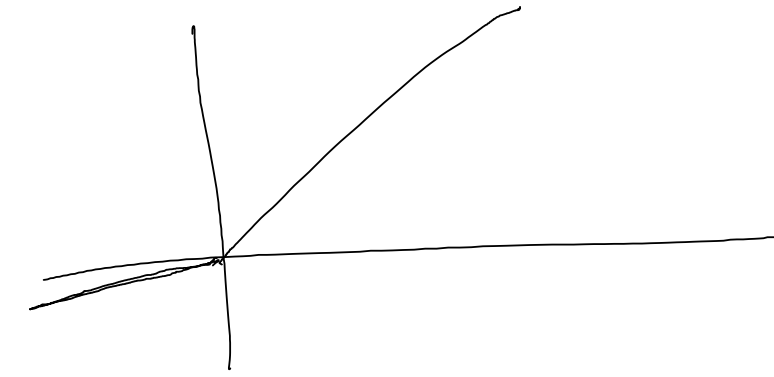
- \hookrightarrow Valores negativos viram zero, diminuindo a capacidade de aprendizagem

... não é totalmente linear

* Activation Function \rightarrow onheer

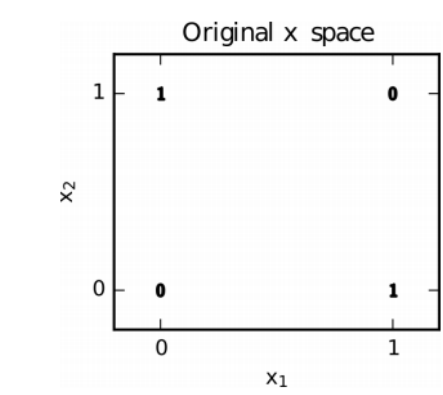
Rule of Thumb

- Output for binário \rightarrow Sigmoidal
- Outros unidades, o ReLU é geralmente o padrão
- Geralmente tenta ficar em hidden layers e sigmoidal em output layer binário
- Leaky ReLU \rightarrow não é igual a zero, tem um decaimento pequeno

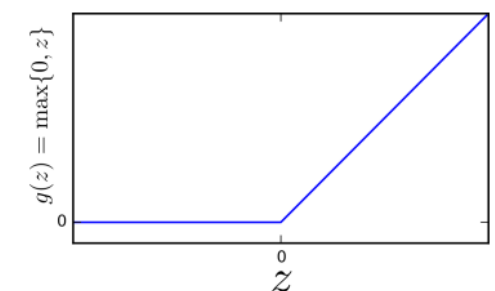


~

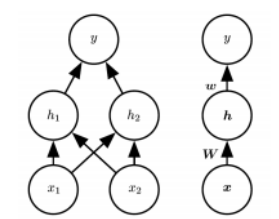
The XOR Problem



Rectified Linear Activation



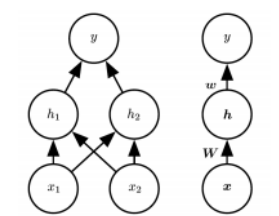
Network Diagrams



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

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

Solving XOR

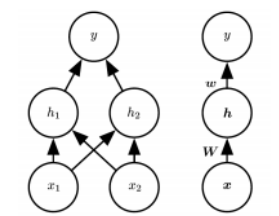


$$\mathbf{h} = \text{ID3X}(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$

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

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

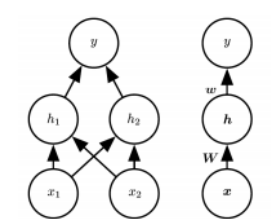
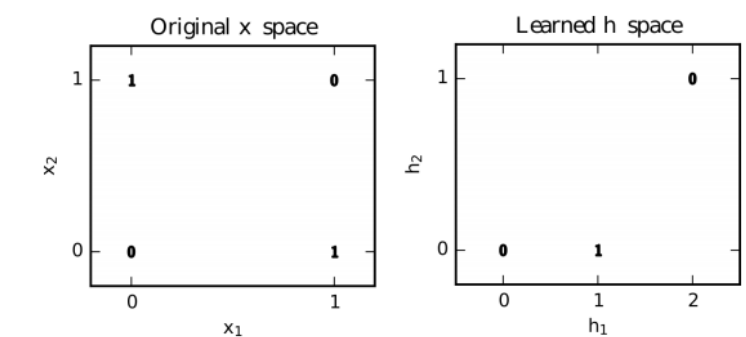
Solving XOR



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

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

$$\begin{aligned} H &= \max(0, \mathbf{W}^T \mathbf{X} + \mathbf{c}) \\ H &= \max\left(0, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right) \\ H &= \max\left(0, \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right) \\ H &= \max\left(0, \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}\right) \\ H &= \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$



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

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

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

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

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?

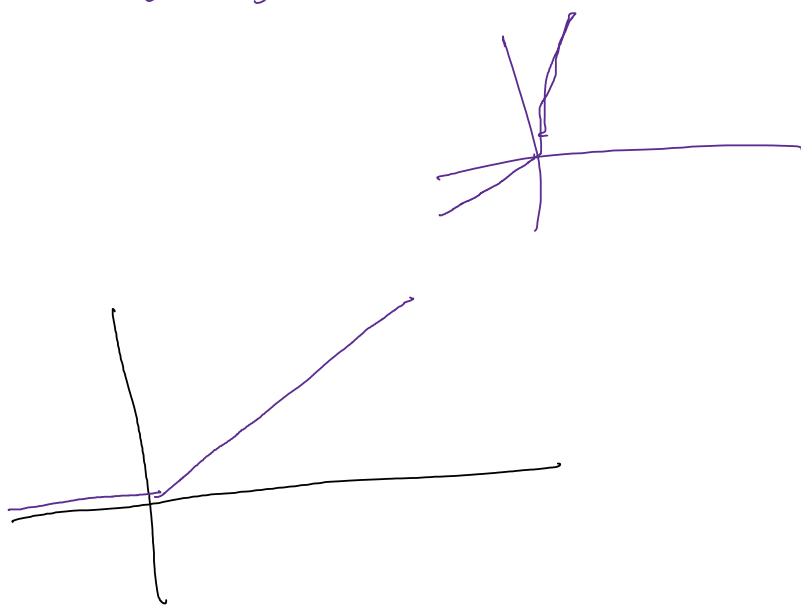
- ▶ The loss function is typically paired with the activation function to quantify how far we are from a desired result.
 - ▶ An example is the perceptron criterion.
- $$L_i = \max\{-y(\vec{W} \cdot \vec{X}), 0\}$$
- ▶ Note that loss is 0, if the instance (\vec{X}, y) is classified correctly.
 - ▶ Even 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

↳ Lesby RELU comes in - 0.5, 1m



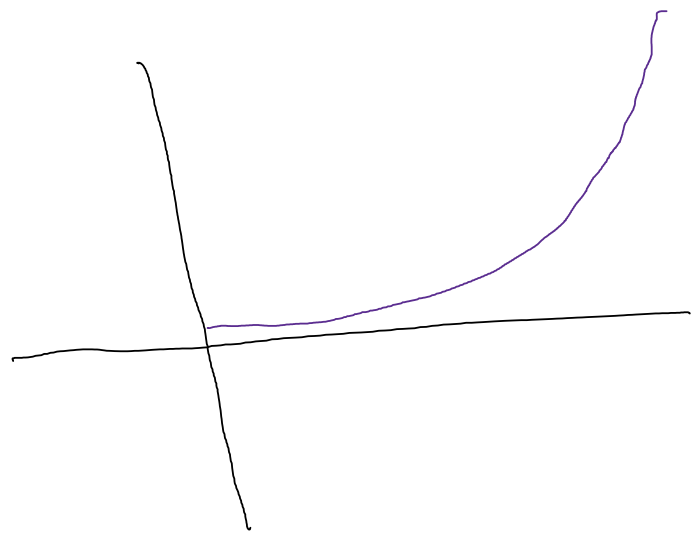
$$u = \max(0, z)$$

→ Softmax

↳ Usado para mapear vetores para vetores, no contexto de escala \rightarrow escala

↳ Bom X mapeo probabilístico

- ↳ Generalización de sigmoid, usado en multiclass logistic regression



→ Tabelas com output types

Identity Activation

- Identity activation $\Phi(x) = x$ is often used in the output layer, when the outputs are real values.
- For a single-layer network, if the training pair is (\vec{X}, y) , the output is as follows:
$$\hat{y} = \Phi(\vec{W}^T \cdot \vec{X}) = \vec{W}^T \cdot \vec{X}$$
- 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

Sign Activation

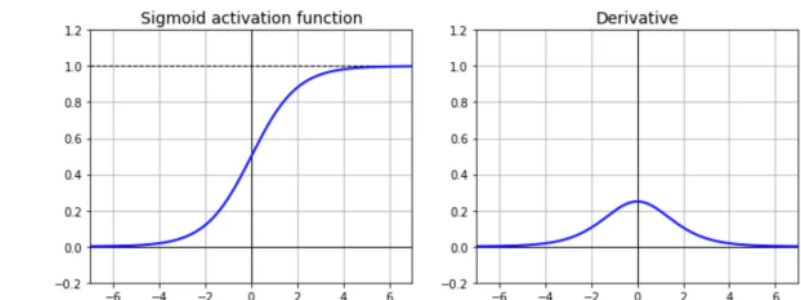
- $$\Phi(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}$$
- Can be used to map to binary outputs at prediction time.
- Its non-differentiability prevents its use for creating the loss function at training time.
- Eg. while the perceptron uses the sign function for prediction,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

Summary

Sigmoid Activation

- Sigmoid activation is defined as $\Phi(x) = 1/(1 + \exp(-x))$.
- For a training pair (\vec{X}, y) , one obtains the following prediction in a single-layer network:
$$\hat{y} = 1/(1 + \exp(-\vec{W}^T \cdot \vec{X}))$$
- Prediction is the probability that class label is +1.
- Pinned with logarithmic loss, which $-\log(y)$ for positive instances and $-\log(1 - y)$ for negative instances.
- $$f(x, y) = -y \ln \Phi(x) - (1 - y) \ln (1 - \Phi(x))$$
- Resulting model is logistic regression.

Sigmoid Activation



Sigmoid Activation

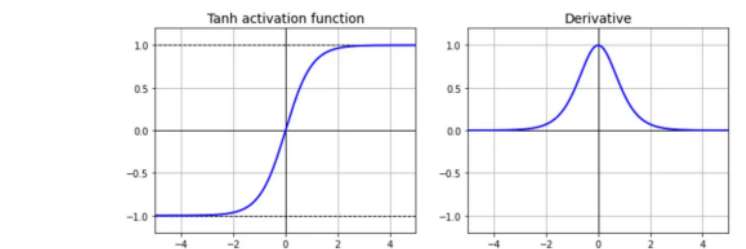
- Characteristics:
- The function is a common S-shaped curve.
 - The output of the function is centered at 0.5 with a range from 0 to 1.
 - The function is differentiable. That means we can find the slope of the sigmoid curve at any two points.
 - The function is monotonic but the function's derivative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

Sigmoid Activation

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

Summary

Tanh Activation



- The tanh activation is a scaled and translated version of sigmoid activation.
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = 2 \cdot \text{sigmoid}(2x) - 1$$
- 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

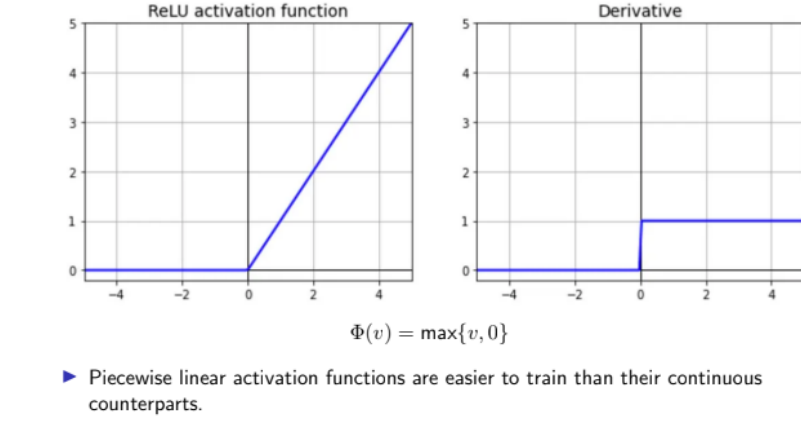
The XOR Problem

Why Do We Need Activation Functions?

Formulas

Examples of Activation Functions

Piecewise Linear Activation Functions – ReLU



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

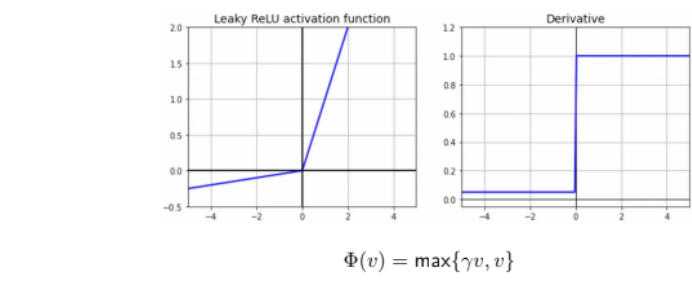
The XOR Problem

Why Do We Need Activation Functions?

Formulas

Examples of Activation Functions

Piecewise Linear Activation Functions – Leaky ReLU



Where typically: $\gamma = 0.01$.

Summary

Binary Classification and Linear Regression Problems

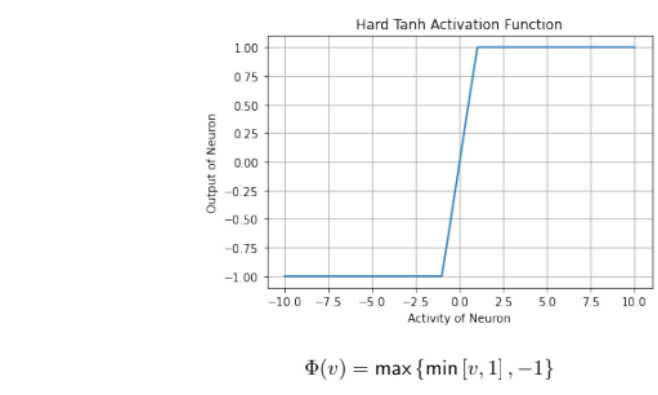
The XOR Problem

Why Do We Need Activation Functions?

Formulas

Examples of Activation Functions

Piecewise Linear Activation Functions – Hard TANH



Summary

Binary Classification and Linear Regression Problems

The XOR Problem

Why Do We Need Activation Functions?

Formulas

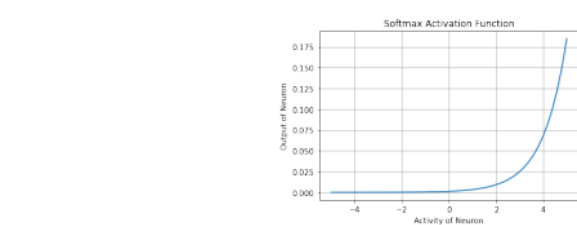
Examples of Activation Functions

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 multiclass logistic regression.

$$\Phi(\mathbf{V}) = \frac{e^{v_i}}{\sum_{j=1}^K e^{v_j}}$$



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

1)

Useful Derivatives

- Sigmoid: $\frac{\partial \sigma}{\partial x} = \sigma(1 - \sigma)$
- Tanh: $\frac{\partial \tanh}{\partial x} = 1 - x^2$
- ReLU: Derivative is 1 for positive values of x and 0 otherwise.
- Hard Tanh: Derivative is 1 for $x \in [-1, 1]$ and 0 otherwise.

Output Types

Output Type	Output distribution	Out. Layer Act. Func.	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross entropy
Discrete	Multinomial	Softmax	Cross entropy
Continuous	Gaussian	Linear	MSE
Continuous	Adversary	GAN, VAE, FNN	Variation

Using Activation Functions

- The nature of the activation in output layers is often controlled by the nature of output.
 - Identify activation for real-valued outputs, and sigmoid/softmax for binary/multinomial 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 does not always perform as expected.
 - ReLU has largely replaced tanh and sigmoid in many applications.

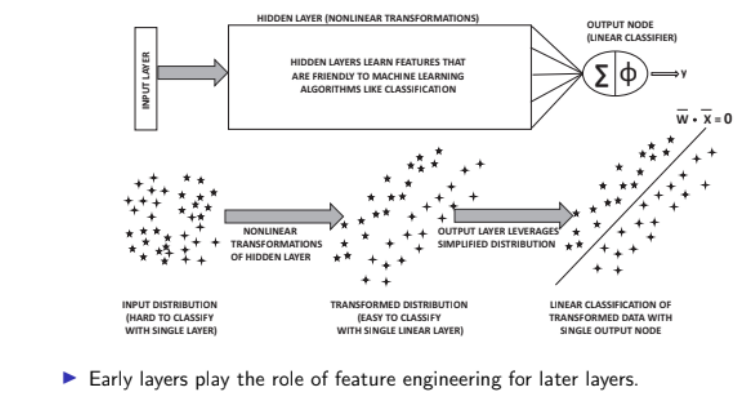
Why are Hidden Layers Nonlinear?

- 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.
$$\hat{y}_k = \Phi(W_k^T x) = W_k^T x$$
$$\hat{y}_{k+1} = \Phi(W_{k+1}^T \hat{y}_k) = W_{k+1}^T \hat{y}_k \quad \forall k \in \{1 \dots k-1\}$$
$$\hat{y} = \Phi(W_{k+1}^T \hat{y}_k) = W_{k+1}^T \hat{y}_k$$
- We can eliminate the hidden variable to get a simple linear relationship:
$$\hat{y} = W_{k+1}^T W_k^T \dots W_1^T x$$
$$= (W_{k+1}^T \dots W_1^T)^T x$$
- We get a single-layer network with matrix W_{k+1} .

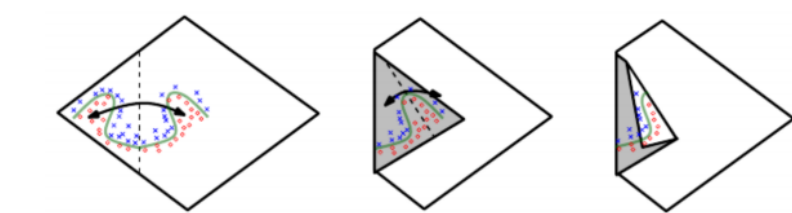
Role of Hidden Layers

- Nonlinear hidden layers perform the role of **hierarchical feature engineering**.
 - Early layers learn primitive features and later learn more **complex features**.
 - Young mice. Early layers learn information about the middle layer could learn complex features like bipeds, and later layers learn complex features like a part of a fox.
 - One hidden layer is enough for feature engineering.**
- The final output layer is often able to perform inference with transformed features in penultimate layer relatively easily.
- **Perceptrons**: 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!
twinkl@icloud.be