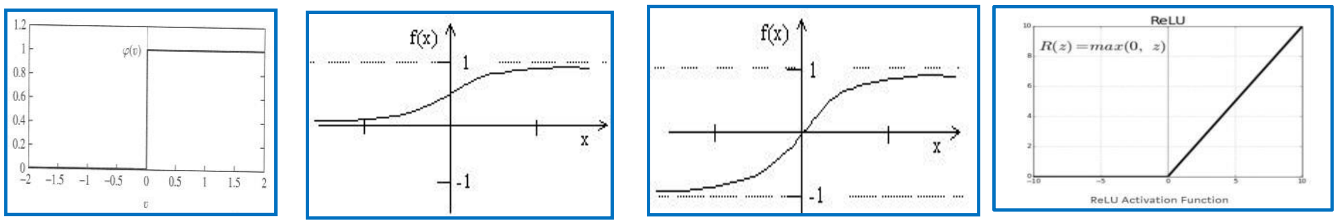
CAP 6619 Deep Learning

2024 Summer

Question 1

Figure 1 shows four activation functions in the neural network, please show the mathematical formulation of each activation function [0.5], and explain the characteristics of each activation function.



(a) (b) (c) (d)

Figure 1 Activation Functions.

Solution:

(a) Step Function (binary step function activation)

Mathematical formulation:

$$f(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

Characteristics:

Output: The output is 0 or a non-zero value that signifies the detection efficiency. Formalizing this more, we have Output = F (S) where S are the inputs to function) and f can take on values either in {1} when an event is detected but may not be correct such as false positive; OR at least zero processing power i.e., due to severe limitations only occasional results make sense -> Yes), otherwise!

Usage: Originally used in the first generation of artificial neurons but is now seldom employed in modern neural nets because it becomes non-different at x=0, making optimization way harder using gradient.

Problem: It does not help with learning via gradient descent, since its derivative is nearly zero everywhere.

(b) Sigmoid Activation Function.

Math formulation:

$$\mathrm{f(x)} = \frac{1}{1 + e^{-x}}$$

Characteristics:

Output: Squashes the input value into (0, 1).
Usage: It is commonly used in binary classification task where an output has to be within this interval (0 and 1).
Problems: The sigmoid activation suffers from a problem called the vanishing gradient problem, which occurs when large values of x make the gradient so small that learning takes place at a very slow pace.

(c) Tanh Activation Function

Math formulation:

$$\mathrm{f(x)} = \mathrm{tanh(x)} = \frac{1}{1 + e^{-2x}} - 1$$

Characteristics:

Output: The output values vary from -1 to 1.
Usage: Like sigmoid, it passes through zero; hence fast convergence can typically be achieved.
Problem: But unfortunately, the tanh activation function also suffers from the vanishing gradient problem just like sigmoid, particularly in large input values.

(d) ReLU (Rectified Linear Unit)

Math formulation:

$$\mathrm{f(x)} = \mathbf{max}(0, \mathrm{x})$$

Characteristics:

Output: Deep neural networks widely employ this activation function since it is simple and effective.
Usage: It doesn’t saturate on positive values making it a prevention for vanishing gradient issue.
Problem: For large segments of the dataset, neurons may remain inactive (doing nothing) giving rise to the ‘dying ReLU’ problem.

Question 2

Drive gradient of the Sigmoid activation function Figure 1(b) and the gradient of the ReLU activation function Figure1(d). Explain the advantages vs. disadvantages of each of the activation functions, respectively.

Solution:

Gradients of Sigmoid and ReLU Activation Functions

(1) Sigmoid Activation Function Gradient:

The sigmoid function is given by:

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

To find its gradient $\frac{d}{dx}f(x)$, we use the chain rule:

$$\frac{d}{dx}f(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

By differentiating:

$$\frac{d}{dx}f(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

Since $f(x) = \frac{1}{1+e^{-x}}$, we can simplify the gradient further as:

$$\frac{d}{dx}f(x) = f(x)(1-f(x))$$

So, the gradient of the sigmoid function is:

$$f'(x) = f(x)(1-f(x))$$

(2) ReLU Activation Function Gradient:

The ReLU function is defined as:

$$f(x) = \max(0, x)$$

The gradient $f'(x)$ of ReLU is:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Advantages vs. Disadvantages of Sigmoid and ReLU Activation Functions

Sigmoid Activation Function

Advantages:
Outputs are between 0 and 1, making it useful for models with probabilities (e.g., binary classification).
A smooth curve is better for gradient-based optimizations.

Disadvantages:
Vanishing Gradient Problem: If the input of the sigmoid function is too large or too small, then the gradient becomes close to zero causing gradients to vanish, therefore slowing down learning especially in deep networks.
Non-zero centered output: The output is always positive between [0, 1] which may cause slow updates in gradients resulting in inefficient optimization.

ReLU Activation Function:

Advantages:
Its computation is simple since there are only two possible gradients i.e 1 or 0
No vanishing gradient for positive inputs: This leads to fast learning in deeper networks since Reinforced Learning doesn’t slow down the learning rate like Sigmoid;

Sparsity: Because some neurons are inactive (output being equal to zero), sparsity is induced along paths that enhance efficient learning.

Disadvantages:

Dying ReLU Problem: When x =< 0, the gradient reduces to zero making it impossible for perturbed updates which leads to non-updating neurons the other hand called ‘dead neurons’ during training especially when large portions of input give negative outputs

Question 3

Figure 2 shows a set of samples (dots) that are labeled as red and green (red dots belong to class C2, and green dots belong to class C1).

- Explain how a neuron can be trained to correctly classify dots into different classes (C1 vs. C2)
- What are the roles of the weight values of the neuron
- Assume a neuron with weight values [w_0, w_1, w_2] is used to learn the decision surface to separate the two group instances (w_0 is the weight value for bias), Draw decision surfaces corresponding to [0, 1, 1], [0, 1, -1], and [-1, 1, 1], respectively (mark each line on the plot)
- Explain how each weight values [w_0, w_1, w_2] control the decision surface, respectively [0.25 pt]. Among the three decision surfaces, which line is the best decision surface to separate instances, why? [0.25 pt]

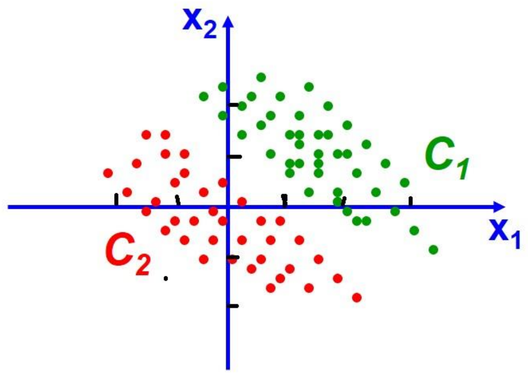


Figure 2. Examples of a linearly separable classification task with two feature dimensions

Solution:

1. How a Neuron Can be Taught to Categorize Dots into Classes (C1 vs. C2)

By taking advantage of a supervised learning approach, a neuron is capable of differentiating between the data points (green indicates class C1 whereas red represents class C2). The process involves:

Input Features: Inputs for the neuron are feature x1 and feature x2 (seen in Figure 2 axes).

Weights and Bias: Each input feature is associated with corresponding weights w1, and w2 while there exists a bias term denoted by w0.

The calculated value is sent through an activation function,(which is usually a sigmoid or step function) to determine the type of class involving this value z.

Training: The weights w0, w1, and w2 are adjusted during training using a technique like gradient descent, minimizing the error between the predicted and actual class labels. Consequently, the neuron takes multiple samples which allows it to learn what the optimal decision boundary would be.

2. Roles of the Weight Values of the Neuron

The weight values control the orientation and position of the decision boundary:

w1: This weight affects the slope of the decision boundary to the x1 feature.

W2: This weight affects the slope of the decision boundary for the feature.

W0: This is the bias term and controls the intercept of the decision boundary with the axes, shifting the decision boundary up or down.

3. Decision Surfaces for Different Weights

The decision boundary is defined by the equation:

[w0 + w1 x1 + w2 x_2 = 0]

This represents the line that separates the two classes.

For [0, 1, 1]: The equation is (x1 + x2 = 0) or (x2 = -x1).

This is a line with slope -1 and intercept 0.

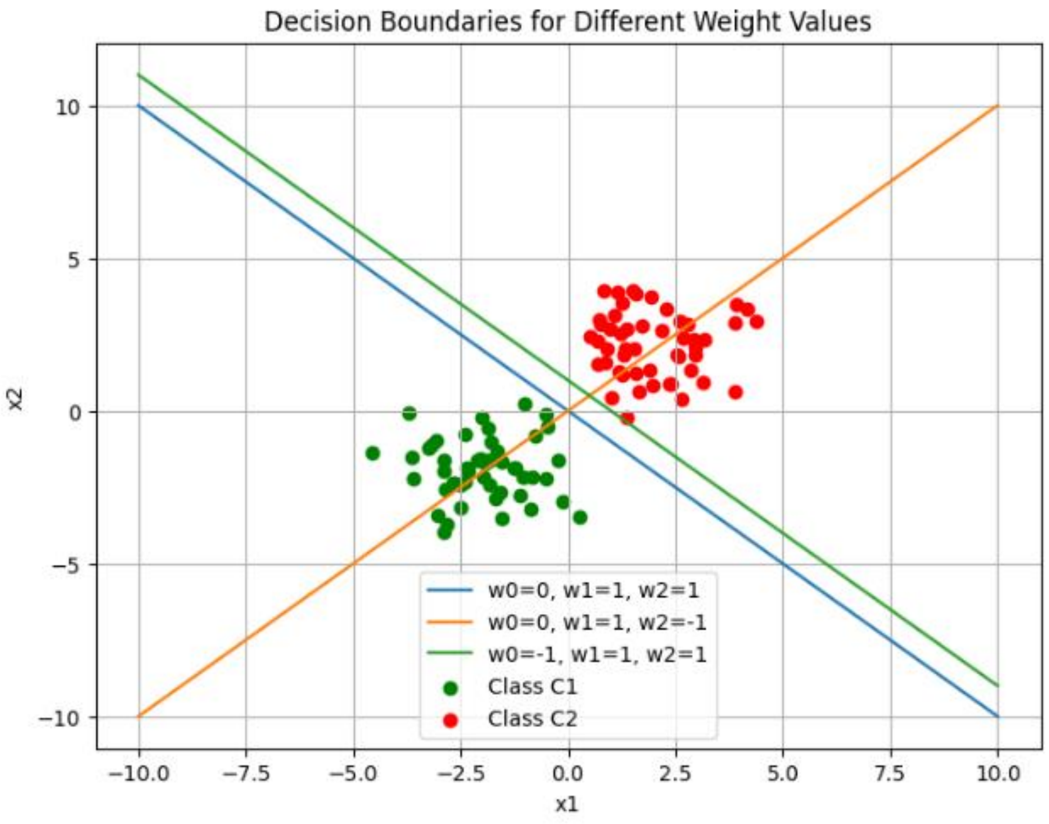
For [0, 1, -1]: The equation is (x1 - x2 = 0) or (x2 = x1).

This is a line with slope 1 and intercept 0.

For [-1, 1, 1]: The equation is (1 + x1 + x2 = 0) or (x2 = -x1 - 1).

This is a line with slope -1 and intercept -1.

plot the decision surfaces and mark each line:



4. How Each Weight Determines the Decision Surface

w0: This parameter can be used for moving the line up or down. Negative value makes a slide down, a positive one makes it slide up.
w1: The slope of the line along x1-axis depends on this parameter as it controls weight of feature x1.
w2: This parameter determines how steeply it slopes in relation to x2-axis since this is responsible for weighting feature x2

Optimal Decision Surface

Out of all three lines above, the most ideal one is that which most efficiently separates members of classes C1 and C2 from each other. In this instance, [0, 1, -1] appears likely to do best by passing between data points and thus minimizing errors in distinguishing between class C1 (green dots) and class C2 (red dots).

Question 4

Figure 3 shows a single layer neural network with three weight values (including bias). Given a training instance x(n), assume desired label of the instance is d(n),

1. Define squared error of the instance with respect to the network [0.5 pt].
2. Use gradient descent learning to derive weight updating rules for w_0, w_1, and w_2, respectively [0.5 pt]

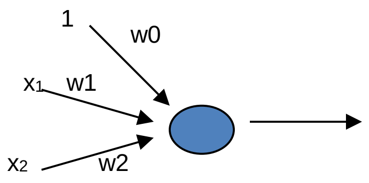


Figure 3: Single layer neural network

Solution:

1. Squared Error of the Instance with Respect to the Network

For a given training instance $x(n)$ with the desired label $d(n)$, the output of the neural network can be represented as:

$$y(n) = w_0 + w_1x_1(n) + w_2x_2(n)$$

The squared error $E(n)$ is the difference between the desired output $d(n)$ and the actual output $y(n)$, squared:

$$E(n) = \frac{1}{2}\left(d(n) - y(n)\right)^2$$

The factor $\frac{1}{2}$ is included to simplify the derivative calculation when applying gradient descent.

2. Weight Updating Rules Using Gradient Descent

The goal is to minimize the squared error $E(n)$ with respect to the weights w_0, w_1, and w_2 by updating the weights iteratively using gradient descent. The general update rule for gradient descent is:

$$w_i \leftarrow w_i - \eta \frac{\partial E(n)}{\partial w_i}$$

where η is the learning rate, and w_i is the weight being updated (w_0, w_1, or w_2).

We will now derive the partial derivatives of the squared error with respect to each weight.

1. Updating w_0 (bias weight):

We need to calculate $\frac{\partial E(n)}{\partial w_0}$:

$$\frac{\partial E(n)}{\partial w_0} = \frac{\partial}{\partial w_0}\left(\frac{1}{2}\left(d(n) - y(n)\right)^2\right) = -(d(n) - y(n))$$

Thus, the update rule for w_0 is:

$$w_0 \leftarrow w_0 + \eta(d(n) - y(n))$$

2. Updating w_1:

We need to calculate $\frac{\partial E(n)}{\partial w_1}$:

$$\frac{\partial E(n)}{\partial w_1} = \frac{\partial}{\partial w_1}\left(\frac{1}{2}\left(d(n) - y(n)\right)^2\right) = -(d(n) - y(n))x_1(n)$$

Thus, the update rule for w_1 is:

$$w_1 \leftarrow w_1 + \eta(d(n) - y(n))x_1(n)$$

3. Updating w_2:

We need to calculate $\frac{\partial E(n)}{\partial w_2}$:

$$\frac{\partial E(n)}{\partial w_2} = \frac{\partial}{\partial w_2}\left(\frac{1}{2}\left(d(n) - y(n)\right)^2\right) = -(d(n) - y(n))x_2(n)$$

Thus, the update rule for w_2 is:

$$w_2 \leftarrow w_2 + \eta(d(n) - y(n))x_2(n)$$

Question 5

The following figure shows a quadratic function $y=2x^2-4x+1$. Assume we are at the point $x=-1$, and is searching for the next movement to find the minimum value of the quadratic function using gradient descent (the learning rate is 0.1).

- What is the gradient at point $x=-1$? (Show your solutions) [0.5 pt]
- Following gradient decent principle, find the next movement towards the global minimum [0.5 pt]

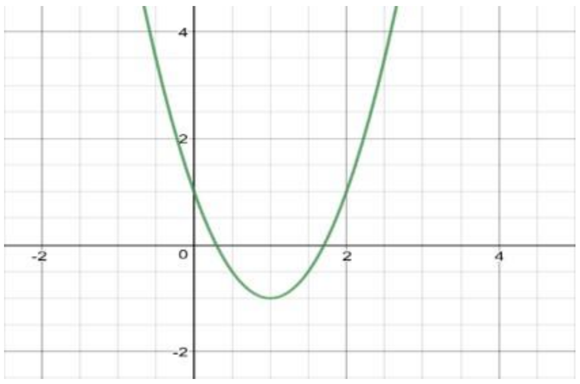


Figure 4. A quadratic function

Solution:

1. Given Quadratic Function:

The function is $y = 2x^2 - 4x + 1$.

2. Calculate the Gradient:

To calculate the gradient at $x = -1$, we first need the derivative of the function with respect to x. The derivative of $y = 2x^2 - 4x + 1$ is:

$$\frac{dy}{dx} = 4x - 4$$

Now, plug $x = -1$ into the derivative:

$$\frac{dy}{dx} = 4(-1) - 4 = -4 - 4 = -8$$

So, the gradient at $x = -1$ is -8.

3. Gradient Descent Step:

Using gradient descent, the next position is determined by:

$$x_{\text{new}} = x_{\text{current}} - \text{learning rate} \times \text{gradient}$$

The learning rate is given as 0.1, and the current gradient at $x = -1$ is -8. So the next movement is:

$$x_{\text{new}} = -1 - 0.1 \times (-8) = -1 + 0.8 = -0.2$$

Question 6

Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. Assuming the class label for C1 and C2 are 1 and -1, respectively, the learning □=0.1, and the initial weights are w_0=1, w_1=1, and w_2=1. Please use gradient learning rule to learn a linear decision surface to separate the two classes. List the results in the first two rounds by using tables in the following form (Report the mean squared errors of all instances with respect to the initial weight values, and the mean squared errors E(W) AFTER the weight updating for each round).

Solution:

Classes:

C1={(1,0),(1,1),(0,−1)} (class label = 1)

C2={(0,1),(−1,0),(−1,−1)} (class label = -1)

Initial weights: w0=1, w1=1, w2=1

Learning rate: η=0.1

Linear Model:

he decision function or output vvv for an input x=x0,x1,x2x is:

$$v = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2$$

Update Rule:

The Perceptron learning rule updates the weights according to:

$$\Delta w_i = \eta \cdot (d - o) \cdot x_i$$

Initial Mean Squared Error (MSE)

The initial MSE is calculated as:

$$E(W) = \frac{1}{N}\sum_{i=1}^{N}(d_i - o_i)^2$$

First Round:

Input	Weight	v	Desired (d)	Output (o)	Δw
(1, 1, 0)	(1, 1, 1)	v=1·1+1·1+1·0=2	1	1	Δw = 0 (no change)
(1, 1, 1)	(1, 1, 1)	v=1·1+1·1+1·1=3	1	1	Δw = 0 (no change)
(1, 0, -1)	(1, 1, 1)	v=1·1+1·0+1·(−1)=0	1	0	Δw0=0.1·(1−0)·1=0.1, Δw2=0.1·(1−0)·(−1)=−0.1
(1, 0, 1)	(1, 1, 1)	v=1·1+1·0+1·1=2	-1	1	Δw0=0.1·(−1−1)·1=−0.2, Δw2=0.1·(−1−1)·1=−0.2
(1, -1, 0)	(1, 1, 1)	v=1·1+1·(−1)+1·0=0	-1	0	Δw0=0.1·(−1−0)·1=−0.1, Δw1=0.1·(−1−0)·(−1)=0.1
(1, -1, -1)	(1, 1, 1)	v=1·1+1·(−1)+1·(−1)=−1	-1	-1	Δw = 0 (no change)

New Weights After First Round:

w0=1+0.1−0.2−0.1=0.8

w1=1+0.1=1.1

w2=1−0.1−0.2=0.7

Mean Squared Error After First Round:

We compute the MSE after updating the weights:

$$E(W) = \frac{1}{6}\sum_{i=1}^{6}(d_i - o_i)^2$$

$$E(W) = \frac{1}{6}\left(0+0+0+4+0+0\right) = \frac{4}{6} = 0.6667$$

- **After the first round:**
  w0=0.8w_0 = 0.8w0=0.8, w1=1.1w_1 = 1.1w1=1.1, w2=0.7w_2 = 0.7w2=0.

Second Round:

Input	**Weight**	**v**	**Desired (d)**	**Output (o)**	**Δw**
(1, 1, 0)	(0.8, 1.1, 0.7)	v=0.8·1+1.1·1+0.7·0=1.9	1	1	Δw = 0 (no change)
(1, 1, 1)	(0.8, 1.1, 0.7)	v=0.8·1+1.1·1+0.7·1=2.6	1	1	Δw = 0 (no change)
(1, 0, -1)	(0.8, 1.1, 0.7)	v=0.8·1+1.1·0+0.7·(−1)=0.1	1	0	Δw0=0.1·(1−0)·1=0.1, Δw2 =0.1·(1−0)·(−1)=−0.1
(1, 0, 1)	(0.8, 1.1, 0.7)	v=0.8·1+1.1·0+0.7·1=1.5	-1	1	Δw0=0.1·(−1−1)·1=−0.2, Δw2=0.1·(−1−1)·1=−0.2
(1, -1, 0)	(0.8, 1.1, 0.7)	v=0.8·1+1.1·(−1)+0.7·0=−0.3	-1	0	Δw0=0.1·(−1−0)·1=−0.1, Δw1 =0.1·(−1−0)·(−1)=0.1
(1, -1, -1)	(0.8, 1.1, 0.7)	v=0.8·1+1.1·(−1)+0.7·(−1)= −1	-1	-1	Δw = 0 (no change)

New Weights After Second Round:

Now, let's calculate the new weights after applying the updates in the second round:

- w0=0.8+0.1−0.2−0.1= 0.6
- w1=1.1+0.1=1.2
- w2=0.7−0.1−0.2=0. 4

Mean Squared Error After Second Round:

Once again, the MSE after the second round is computed using:

$$E(W) = \frac{1}{6}\sum_{i=1}^{6}(d_i - o_i)^2$$

$$E(W) = \frac{1}{6}(0+0+0+4+0+0) = \frac{4}{6} = 0.6667$$

Question 7

Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. Assuming the class label for C1 and C2 are 1 and -1, respectively, the learning rate □=0.1, and the initial weights are w_0=1, w_1=1, and w_2=1. Use Delta rule (AdaLine) to learn a linear decision surface to separate the two classes. List the results in the first round by using tables in the following form

(Please report the mean squared errors of all instances with respect to the initial weight values, and also report the mean squared errors E(W) AFTER the weight updating of the last instance).

Solution:

Classes:

C1={(1,0),(1,1),(0,−1)} (class label = 1)

C2={(0,1),(−1,0),(−1,−1)} (class label = -1)

Initial weights: w0=1, w1=1, w2=1

Learning rate: η=0.1

Delta Rule Formula:

- The formula for updating weights is

$$\Delta w = \eta \cdot (d - v) \cdot x$$

Input	**Weights (w0, w1, w2)**	**v**	**Desired**	**Output**	**Δw**	**New Weights (w0, w1, w2)**
(1, 1, 0)	(1, 1, 1)	2	1	1	(-0.1, -0.1, 0)	(0.9, 0.9, 1)
(1, 1, 1)	(0.9, 0.9, 1)	0	1	1	(0.1, -0.1, 0)	(1, 0.8, 1)
(1, 0, -1)	(1, 0.8, 1)	0	1	1	(0.1, 0, -0.1)	(1.1, 0.8, 0.9)
(1, 0, 1)	(1.1, 0.8, 0.9)	2	-1	-1	(-0.3, 0, -0.3)	(0.8, 0.8, 0.6)
(1, -1, 0)	(0.8, 0.8, 0.6)	2.2	1	1	(-0.12, -0.12, -0.12)	(0.68, 0.68, 0.48)
(1, -1, -1)	(0.68, 0.68, 0.48)	-0.48	-1	-1	(-0.052, 0.052, 0.052)	(0.628, 0.732, 0.532)

Final MSE After First Round:

E(W)=2.285E(W) = 2.285E(W)=2.285

E(W) AFTER the weight updating of the last instance

E(W)=61(0.1296+1.218+0.816+4.6656+0.795+0.132)=61×7.7562=1.2927