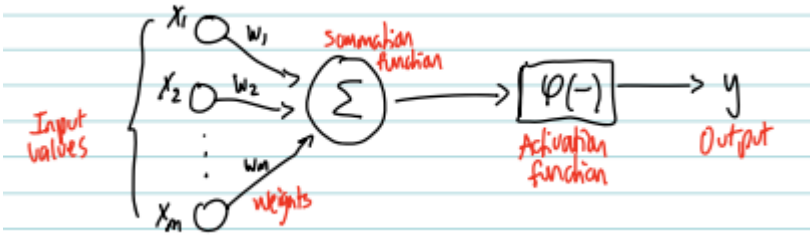
Assignment 1

CAP6619 - Deep Learning Summer 2024

Benjamin Luo

May 27, 2024

Problem 1. Perceptron Model



Inputs: Values that the perceptron receives from either other perceptrons or an existing model (i.e. a dataset, an image, etc.)
Weights: Parameters of the perceptron model that can be adjusted to determine how much influence each incoming input value will have on the final output
Summing function: Function that serves to calculate a weighted sum for the model by adding the products of the corresponding input values and weights together
Activation Function: Function that is employed to determine how the weighted sum calculated by the summing function will be interpreted in the the final output value
Output: The outgoing value of the perceptron, after the weights and input values have been summated and put through the activation function

Problem 2. Activation Functions

(a) Hard-limiter

$$\phi(v) = \begin{cases} 1 \text{ if } v \geq 0 \\ 0 \text{ if } v \leq 0 \end{cases}$$

In the hard-limiter activation function, the output is not continuous; it can only be 1 of 2 possible outputs, which is denoted by the horizontal sections of the graph.

(b) Sigmoid

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

The sigmoid function is a function bounded between values 0 and 1. It is continuous, as the output can be any input within the bounds of the function.

(c) Hyperbolic tangent

$$\phi = \tanh(v)$$

The hyperbolic tangent function is similar to the sigmoid function with some key differences; it is zero-centered, meaning that positive values will be positive, and negative values will similarly be negative. The base hyperbolic tangent function lies within the range of -1 and 1.

(d) Rectified Linear Unit (ReLU)

$$R(x) = max(0, x)$$

The rectified linear unit (ReLU) function is simply a linear function that normalizes the negative values into 0. Essentially, it is a linear function that creates different output based on the positive values of the activation inputs. For any results that are less than or equal to 0, the activation function output is 0.

Problem 3. Activation Functions

(a) Figure 1(b) - Sigmoid

For a given input x, the output can be calculated through this equation:

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

Using the quotient rule, we get:

$$\frac{d}{dx}\sigma(x) = \frac{(1 + e^{-x})(0) + (1)(-e^{-x})}{(1 + e^{-x})^2} = \frac{e^{-x}}{(1 + e^{-x})^2}$$

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

$$\frac{1}{(1 + e^{-x})^2} - \frac{1}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}}\left(1 - \frac{1}{1 + e^{-x}}\right)$$

Substituting σ in for the function, we get:

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

- It is continuous and differentiable as well as bounded, meaning that it will always have a meaningful output (between 0 and 1).
- However, the gradient of the function vanishes as output approaches the extremities of the function (0,1); in addition to that, sigmoid can be computationally expensive as well.

(b) Figure 1(d) - Rectified Linear Unit (ReLU)

The derivative of the gradient of the ReLU activation function is split into two parts. We can first treat the function $ReLU = max(0, x)$ like a piece-wise function, with the sections where the max is determined based on 0 and x, respectively. With this in mind, the derivative of 0 is similarly 0, while the derivative of x is 1. We can represent this as:

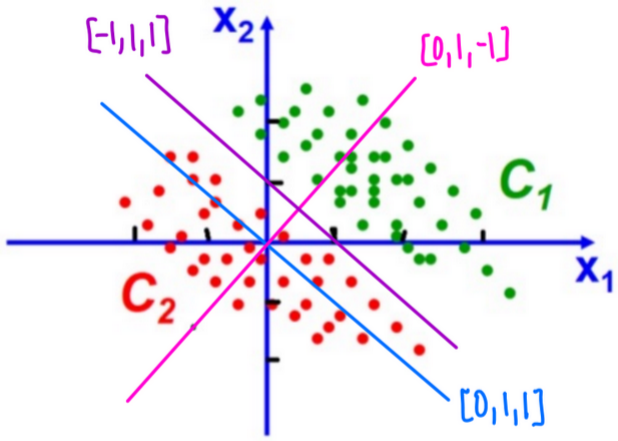
$$R(x) = max(0, x)$$

$$R'(x) = \begin{cases} 0 \text{ if } x \leq 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

The advantages of ReLU is that its gradient is consistent, due to the derivatives being constant in nature, as well as being computationally efficient. However, it is disadvantaged by the fact that it is not differentiable at every point ($x = 0$). ReLU is also not bounded, as the gradient can blow up, since for positive values, it will keep going up.

Problem 4. Linear Line Decision

- The weight values of the neuron are designed to determine the optimal level of emphasis to place on various inputs to the neuron in order to have the most accuracy on the decision surface when it comes to determining the correct class for data inputs.



- w_0, or b, controls the origin that our decision surface is centered about.
- w_1 and w_2 control how large or small that specific dimension of the decision surface must be. In our case, with only two dimensions, it is easy to visualize with a line.
- Among the three decision surfaces mentioned in the question, the decision surface defined by $[-1, 1, 1]$ (depicted as the purple line in the above image) is the best decision to separate instances in this scenario. This is because, we notice that the data points tend to be differentiated into the different classes, either have both x_1 and x_2 values greater than 1 or not. Comparing the decision surface created, shown by the line $x_1 + x_2 - 1 \geq 0$ it seems to most accurately divide data points into their respective classes.

Problem 5. Gradient Descent Part 1

(a) Squared Error

The squared error of our instance x_n with respect to the network is as follows:

$$e(w) = d - (w_0 + w_1x_1 + w_2x_2)$$

Taking our error here and turning it into a second order function, we get the squared error:

$$E(w) = \frac{1}{2} \cdot e^2$$

$$E(w) = \frac{1}{2}(d - (w_0 + w_1x_1 + w_2x_2))^2$$

(b) Weight updating rules for w_0, w_1, w_2

$$f(w_0, w_1, w_2) = \frac{1}{2}(d - [w_0 + w_1x_1 + w_2x_2])^2$$

$$\text{for } w_0, \text{ gradient} = \frac{\partial f(w_0, w_1, w_2)}{\partial w_0}$$

$$= \frac{1}{2} \cdot 2 \cdot (d - [w_0 + w_1x_1 + w_2x_2]) \cdot \frac{\partial (d - [w_0 + w_1x_1 + w_2x_2])}{\partial w_1}$$

$$= (d - [w_0 + w_1x_1 + w_2x_2]) \cdot (-1)$$

$$w_0(k+1) = w_0(k) + (d - [w_0 + w_1x_1 + w_2x_2])$$

Based on Gradient = $(d - [w_0 + w_1x_1 + w_2x_2])$, the weight updating rules for w_1 and w_2 are:

$$w_1(k+1) = w_1(k) + (d - [w_0 + w_1x_1 + w_2x_2]) \cdot x_1$$

$$w_2(k+1) = w_2(k) + (d - [w_0 + w_1x_1 + w_2x_2]) \cdot x_2$$

Problem 6. Gradient Descent Part 2

To find the gradient at a point on the graph, we need to find the derivative first:

$$y = 2x^2 - 4x + 1$$

$$y' = 4x - 4$$

$$y'(-1) = 4(-1) - 4 = -8$$

The gradient at point $x = -1$ is equal to -8.

Following the gradient descent principle, we get:

$$w(k+1) = w(k) - \eta(\text{gradient})$$

$$w(k+1) = -1 - 0.1(-8) = -0.2$$

The next step towards the global minimum using the gradient descent principle is $x = -0.2$.

Problem 7. Gradient Descent Learning Rule Part 1

Mean squared errors $E(W)$ corresponding to the initial weights: $11/2$ $(= 5.5)$

First Round

Input	Weight	v	Desired, $d(n)$	Output, $o(n)$	Δw_i
$(1, 1, 0)$	$(1, 1, 1)$	$[1, 1, 0]$	1	2	$[-0.1, -0.1, 0]$
$(1, -1, 0)$	$(1, 1, 1)$	$[1, -1, 0]$	0	0	$[0, 0, 0]$
$(1, 0, -1)$	$(1, 1, 1)$	$[1, 0, -1]$	1	0	$[0.1, 0, -0.1]$
$(1, 0, 1)$	$(1, 1, 1)$	$[1, 0, 1]$	0	2	$[-0.2, 0, -0.2]$
$(1, 1, 1)$	$(1, 1, 1)$	$[1, 1, 1]$	1	3	$[-0.2, -0.2, -0.2]$
$(1, -1, -1)$	$(1, 1, 1)$	$[1, -1, -1]$	0	-1	$[0.1, -0.1, -0.1]$

New weight after first round: $[0.7, 0.6, 0.4]$
Mean squared errors $E(W)$ after the first-round weight updating: $2.38/2$ $(= 1.19)$

Second Round

Input	Weight	v	$d(n)$	$o(n)$	Δw_i
$(1, 1, 0)$	$(0.7, 0.6, 0.4)$	$[0.7, 0.6, 0]$	1	1.3	$[-0.03, -0.03, 0]$
$(1, -1, 0)$	$(0.7, 0.6, 0.4)$	$[0.7, -0.6, 0]$	0	0.1	$[0.09, -0.09, 0]$
$(1, 0, -1)$	$(0.7, 0.6, 0.4)$	$[0.7, 0, -0.4]$	1	0.3	$[0.07, 0, -0.07]$
$(1, 0, 1)$	$(0.7, 0.6, 0.4)$	$[0.7, 0, 0.4]$	0	1.1	$[-0.01, 0, -0.01]$
$(1, 1, 1)$	$(0.7, 0.6, 0.4)$	$[0.7, 0.6, 0.4]$	1	1.7	$[-0.07, -0.07, -0.07]$
$(1, -1, -1)$	$(0.7, 0.6, 0.4)$	$[0.7, -0.6, -0.4]$	0	-0.3	$[0.13, -0.13, -0.13]$

New weight after first round: $[0.88, 0.28, 0.12]$
Mean squared errors $E(W)$ after the second-round weight updating: $1.752/2$ $(= 0.876)$

Problem 8. Gradient Descent Learning Rule Part 2

Mean squared errors $E(W)$ corresponding to the initial weights: $11/2$ $(= 5.5)$

Input	Weight	v	d	o	Δw	New Weight
$(1, 1, 0)$	$(1, 1, 1)$	$[1, 1, 0]$	1	2	$[-0.1, -0.1, 0]$	$[0.9, 0.9, 1]$
$(1, -1, 0)$	$(0.9, 0.9, 1)$	$[0.9, -0.9, 0]$	0	0	$[0, 0, 0]$	$[0.9, 0.9, 1]$
$(1, 0, -1)$	$(0.9, 0.9, 1)$	$[0.9, 0, -1]$	1	-0.1	$[0.09, 0, -0.09]$	$[0.99, 0.9, 0.91]$
$(1, 0, 1)$	$(0.99, 0.9, 0.91)$	$[0.99, 0, 0.91]$	0	1.9	$[-0.19, 0, -0.19]$	$[0.8, 0.9, 0.72]$
$(1, 1, 1)$	$(0.8, 0.9, 0.72)$	$[0.8, 0.9, 0.72]$	1	2.42	[-.242,-.242,-.242]	[.558, .658, .478]
(1,-1,-1)	(.558, .658, .478)	[.558,-.658,-.478]	0	-.578	[.0578,-.0578,-.0578]	[.6158,.6002,.4302]

Mean squared errors $E(W)$ after the weight updating of the last instance: $\frac{1}{2}[1.98033] \approx 0.99016$

Problem 9. Programming Exercise 1

See the also additional attached file a1q9.pdf

Problem 10. Programming Exercise 2

See the also additional attached file a1q10.pdf for the scripts written.

Gradient descent learning can be very useful in searching for solutions to $f_1(x, y)$ and $f_2(x, y)$ as it helps us identify local and global maxima/minima. Learning rate is incredibly important in gradient descent learning, as it determines how well our model "learns". If the learning rate is too small, learning will be minimal or slow, and if the learning rate is too large, there is a high likelihood of oscillation and lack of real optimization due to being too imprecise in gradient descent.