

University of Thessaly



Neuro-Fuzzy Computing

ECE447

2nd Problem Set

Alexandra Gianni Nikos Stylianou

ID: 3382

ID: 2917

January 28, 2024

Problem 1

In this exercise we need to find the minimum of the given 2-dimensional function:

$$F(\mathbf{w}) = w_1^2 + w_2^2 + (0.5w_1 + w_2)^2 + (0.5w_1 + w_2)^4 \quad (1)$$

with the Conjugate Gradient (Fletcher-Reeves) method.

Initially, we can conclude that the function $F(w)$ is not in quadratic form because of the term $(0.5w_1 + w_2)^4$. A function is said to be in quadratic form if it can be expressed as a second-degree polynomial where all the terms are either squared terms or cross-products of the variables. The presence of the fourth-degree term $(0.5w_1 + w_2)^4$ makes this function a higher-degree polynomial, specifically a quartic function with respect to $(0.5w_1 + w_2)$, which means it cannot be classified as quadratic.

Also, the independent values in this function are w_1, w_2 , because only with them we can manipulate the $F(w)$.

As an initial guess we have $w(0) = [3, 3]^T$.

The steps we have to use are specific for each iteration

FIRST ITERATION $k = 0$

Step1: Calculate the Gradient at $w(k)$

$$\nabla f(w_1, w_2) = \begin{pmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{pmatrix} = \begin{pmatrix} 2w_1 + (0.5w_1 + w_2) + 2(0.5w_1 + w_2)^3 \\ 2w_2 + 2(0.5w_1 + w_2) + 4(0.5w_1 + w_2)^3 \end{pmatrix} = \begin{pmatrix} 2.5w_1 + w_2 + 2(0.5w_1 + w_2)^3 \\ w_1 + 4w_2 + 4(0.5w_1 + w_2)^3 \end{pmatrix}$$

where at the point $w(0) = [3, 3]^T$ we have $\nabla f(x) = \begin{pmatrix} -53 \\ -19 \end{pmatrix}$

Problem 2

Problem 3

For the given neural network, we have:

- learning rate $LR = 1$,
- $w^1(0) = -3$, $w^2(0) = -1$,
- $b^1(0) = 2$, $b^2(0) = -1$ and
- input/target pair $\{p = 1, t = 0\}$

FIRST ITERATION

Step 1: Calculate first layer's output

$$n^1 = w^1 p + b^1 = (-3)(1) + 2 = -1$$

$$a^1 = \text{Swish}(n^1) = \text{Swish}(-1) = \frac{n^1}{1 + e^{-n^1}} = \frac{-1}{1 + e} = -0.2689$$

Step 2: Calculate second layer's output

$$n^2 = w^2 a^1 + b^2 = (-1)(-0.2689) + (-1) = -0.7311$$

$$a^2 = \text{LReLU}(n^2) = \text{LReLU}(-0.7311) = -0.000731$$

Step 3: Calculate error

$$e = t - a^2 = (0 - (-0.000731)) = 0.000731$$

Step 4: Calculate sensitivity on second layer

$$s^2 = -2 \text{LReLU}'(n^2) (t - a^2) = -2 (0.001) (0.000731) = -1.462\text{e} - 6$$

LReLU's derivative is 1 for $x > 0$ and 0.001 for $x < 0$.

Step 5: Calculate sensitivity on first layer using back-propagation

$$s^1 = \text{Swish}'(n^1) (w^2)^T s^2 = \text{Swish}'(-1) (-1) (-1.462\text{e} - 6) = 0.0723(-1)(-1.462\text{e} - 6)$$

$$s^1 = 1.0570\text{e} - 7$$

Step 6: Update weights and biases

$$w^2(1) = w^2(0) - LR s^2 (a^1)^T = -1 - 1(-1.462\text{e} - 6)(-0.2689) \approx -1$$

$$b^2(1) = b^2(0) - LR s^2 = -1 - 1(-1.462\text{e} - 6) \approx -1$$

$$w^1(1) = w^1(0) - LR s^1 (a^0)^T = -3 - 1(1.0570\text{e} - 7)(-1) \approx -3$$

$$b^1(1) = b^1(0) - LR s^1 = 2 - 1(1.0570\text{e} - 7) \approx 2$$

Since there were no changes on the biases and weights, the next iteration will not change the parameters of the given neural network, but we will calculate them anyway.

SECOND ITERATIONStep 1:

$$n^1 = w^1 p + b^1 = (-3)(1) + 2 = -1$$

$$a^1 = \text{Swish}(n^1) = \text{Swish}(-1) = \frac{n^1}{1 + e^{-n^1}} = \frac{-1}{1 + e} = -0.2689$$

Step 2:

$$n^2 = w^2 a^1 + b^2 = (-1)(-0.2689) + (-1) = -0.7311$$

$$a^2 = \text{LReLU}(n^2) = \text{LReLU}(-0.7311) = -0.000731$$

Step 3:

$$e = t - a^2 = (0 - (-0.000731)) = 0.000731$$

Step 4:

$$s^2 = -2 \text{LReLU}'(n^2) (t - a^2) = -2 (0.001) (0.000731) = -1.462\text{e} - 6$$

Step 5:

$$s^1 = \text{Swish}'(n^1) (w^2)^T s^2 = \text{Swish}'(-1) (-1) (-1.462e - 6) = 0.0723(-1)(-1.462e - 6)$$

$$s^1 = 1.0570e - 7$$

Step 6:

$$w^2(1) = w^2(0) - LR s^2 (a^1)^T = -1 - 1(-1.462e - 6)(-0.2689) \approx -1$$

$$b^2(1) = b^2(0) - LR s^2 = -1 - 1(-1.462e - 6) \approx -1$$

$$w^1(1) = w^1(0) - LR s^1 (a^0)^T = -3 - 1(1.0570e - 7)(-1) \approx -3$$

$$b^1(1) = b^1(0) - LR s^1 = 2 - 1(1.0570e - 7) \approx 2$$

Problem 4

Problem 5

Problem 7

A continuous piecewise linear function is a function that is linear on every segment of its domain.

To show that a Multi-Layer Perceptron (MLP) using only the ReLU (Rectified Linear Unit) or pReLU (Parametric Rectified Linear Unit) activation functions constructs a continuous linear function, we must first review the properties of these activation functions.

Let's consider the ReLU activation function for this explanation.

The ReLU activation function is defined as:

$$\text{ReLU}(x) = \max(x, 0) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise,} \end{cases} \quad (2)$$

We need to check if they meet the prerequisites of continuity and linearity.

- Is it Continuous?

Yes it is, because it has no break points for the various values of x

- Is it Linear?

Yes it is, because it consists of only two linear parts. ReLU is linear within its segments.

In an MLP, the output of each neuron is computed by applying an affine transformation (multiplying the weights and adding the bias), followed by ReLU activation. The key property of ReLU activation is that it is a piecewise linear function. When you consider a single neuron with ReLU activation, it essentially performs two operations:

1. For inputs x where $x > 0$, the output is x .
2. For inputs x where $x \leq 0$, the output is 0

Having a closer look, the first operation ($x > 0$) is a linear transformation with a slope of 1 (output is $y = x$), and the second operation ($x \leq 0$) is a constant zero (output is $y = 0$).

By composing several such neurons in an MLP architecture, we effectively create a composition of linear transformations and constant zeros. Since the operations of the individual ReLU neurons are piecewise linear, the combination of these operations is naturally also a piecewise linear function.

The breakpoints in the piecewise linear function occur where the activations of the neurons go from 0 to the actual linear operation -when the input x exceeds 0-. As you move from one layer to the next in the network, we are effectively combining multiple piecewise linear functions, resulting in a more complex piecewise linear function overall.

The activation function pReLU behaves similarly, but it introduces a learnable parameter a for the negative slope that allows a continuous range of slopes for the linear part when x is negative.

To summarize, an MLP that uses only ReLU (or pReLU) activation functions constructs a continuous piecewise linear function because the operations performed by these activation functions are individually piecewise linear and the composition of these operations across the layers results in a piecewise linear function that approximates complex mappings between inputs and outputs.

We can see also the graphical explanation here:

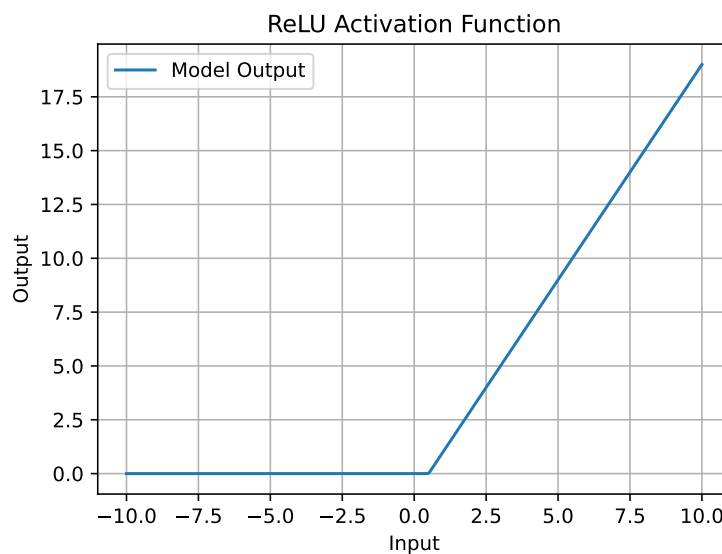


Figure 1: Plot of the MLP using the ReLU activation function

Problem 11

Convolutional Neural Networks (CNNs) have revolutionized in the field of image processing and computer vision and are widely utilized.

In this exercise we are considering a 6×6 image I , where each entry represents the intensity of a pixel. The values are typically normalized, and the CNN would perform operations on this matrix to learn features and perform tasks like classification, detection, or segmentation. We will apply various layers and filters, so that we can extract higher-level features.

$$I = \begin{bmatrix} 20 & 35 & 35 & 35 & 35 & 20 \\ 29 & 46 & 44 & 42 & 42 & 27 \\ 16 & 25 & 21 & 19 & 19 & 12 \\ 66 & 120 & 116 & 154 & 114 & 62 \\ 74 & 216 & 174 & 252 & 172 & 112 \\ 70 & 210 & 170 & 250 & 170 & 110 \end{bmatrix} \quad (3)$$

Given the input matrix we can understand that it represents a grayscale image. In a grayscale image, each pixel is represented by a single intensity value, typically on a scale $[0, 255]$. The 2D input array contains such intensity values for each pixel in the image.

(a) Question A

The output of a convolution layer is a new matrix that's the result of the convolution operation. The convolution operation involves sliding the kernel over the input matrix, with a given stride $(1, 1)$, and for each position, computing the sum of elementwise multiplications.

The use of a stride in a convolutional layer is important, because it determines how much the filter or kernel moves across the input matrix. In our case, a stride of $(1, 1)$ means that the kernel moves one step at a time horizontally and vertically. This will result in an output matrix that is smaller than the input matrix by one less than the kernel size in each dimension. So, in our case the output will be a 4×4 . Also, the output's matrix size is smaller than the original because of the "valid" mode on our code. The "valid" mode means that the convolution product is only given for points where the kernels overlap completely with the input array. It doesn't add any padding to the input image.

In addition, the kernel we have defined is a 3×3 matrix with a zero in the center. This means that the convolution operation will sum up the values of the eight surrounding pixels and ignore the center pixel for each position in the input image.

So, in conclusion, with a

- $stride = (1, 1)$ and

- $kernel = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

The result of the convolution is a 4×4 matrix

$$result = \begin{bmatrix} 225 & 258 & 250 & 209 \\ 458 & 566 & 552 & 472 \\ 708 & 981 & 887 & 802 \\ 1000 & 1488 & 1320 & 1224 \end{bmatrix}$$

The resulting matrix, represents the features in the input image that the kernel was able to detect. In this case, the kernel seems to act like a filter that emphasizes the surrounding context of each pixel. The exact interpretation would depend on the specific values in the input image and the kernel.

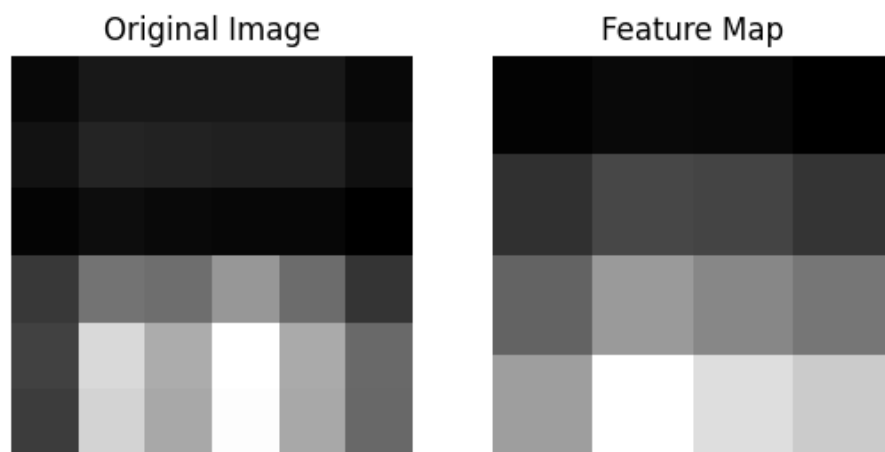


Figure 2: The Original Image and the Image after the convolution