Peeping Inside Black ~~hole~~ box

Ashwani Kushwaha[*]

February 17, 2024

Abstract

This review notes explores the inner workings of deep learning vs. traditional ML algorithms. While basic models struggle with complex patterns, deep networks, like a pair of neurons, can unlock hidden secrets via non-linear magic. Using math, code and mathematical intuition, we delve into their ability to capture curves and its relationships, leaving us eager to push the boundaries of understanding and unlock even deeper insights.

[*]ashwani_kush@outlook.com, adu3839@gmail.com

1 Introduction

The field of machine learning has come a long way in recent decades. Traditional algorithms were easier to understand, offering clear insights into their internal workings. However, the rise of deep neural networks, with their vast and complex architectures, has presented a new challenge: deciphering their hidden mechanisms. While some techniques exist, akin to local minima analysis in traditional ML, a deeper understanding of deep networks remains tantalisingly out of reach. This lack of clarity casts a shadow on our comprehension of various other algorithms as well.

Though I won't delve into the intricate details of deep learning algorithms, I aim to shed some light on their internal workings with shallow networks, drawing parallels to my past experience in theoretical physics. In my interactions with fellow physicists, grappling with the meaning and significance of complex mathematical expressions with physical relevance was a recurring challenge. A similar question arose regarding the nature of neurons in artificial neural networks: can we truly understand and explain these fundamental building blocks? This very curiosity ignites my desire to explore this realm further.

The magic of deep learning lies in its ability to bypass the tedious, manual process of feature engineering. Instead, by employing cleverly designed architectures and optimisation techniques, the network itself learns to identify and extract relevant features from data.

In the initial section, we'll delve into regression problems, followed by an expansion into classification problems in the subsequent section. Then, we'll delve into insights on the well-known Sigmoid activation function, examining both one-dimensional and two-dimensional scenarios. We'll also explore feature representation from these models. Subsequently, we'll present a simple real-life problem from NLP, marking the beginning of a more intricate journey. In the next work, we'll delve into more complex deeper networks and the renowned transformer architecture.

2 Machine Learning Approach

To illustrate the concept of Machine learning, let's dive into a practical example. We'll take a deep dive into a specific dataset (1).

Input	**Output**
2	4.04
2.5	6.28
2.9	8.45
1.2	1.4
5.3	28.0
4.30	18.60
6.75	45.50
7.50	56.00
7.9	62.3
6.4	41.00
....	.....

Table 1: Input-Output Table

We have a collection of data points scattered on a graph, with each point representing a relationship between two variables. Your goal is to find a line or curve that best captures the overall trend of these points. This line or curve is your model, and the process of finding it is called "fitting" the model to the data.
This dataset holds immense potential for uncovering hidden patterns and insights. To unlock its secrets, we'll embark on a journey of discovery using the powerful tools of machine learning. Let's take it step-by-step:
Step 1: Building a Simple Model

Imagine we have a machine that takes numbers as input (x) and outputs other numbers (X). To understand the relationship between x and X, we can start with a simple model:

Model: $Y = aX$

This equation says that Y is simply a scaled version of X, determined by the coefficient "a." It's like stretching or shrinking X according to the value of a.

Step 2: Testing the Model's Fit

Can this simple model handle all the data? We plug in different values of X and compare the predicted Y values (using our model) to the actual Y values in the dataset. This comparison reveals:

Limitations: If the data exhibits complex patterns beyond just scaling, the Y predictions from our simple model will likely deviate significantly from the actual values.

Step 3: Introducing Complexity

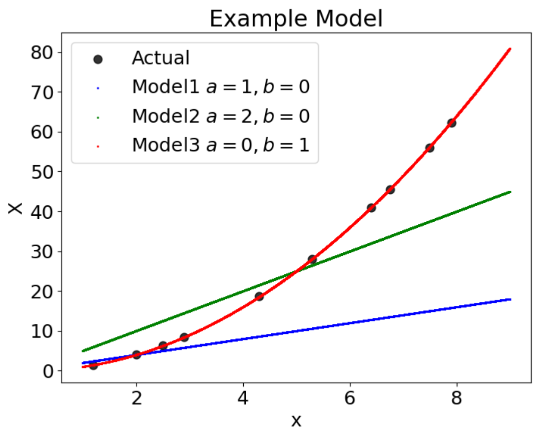


Figure 1: The Comparison between Actual and Model.

To capture these intricate patterns, we can add more sophistication to our model. One way is to introduce a quadratic term:

Model: $Y = aX + bX^2$

This equation adds a "bendiness" factor to the model, controlled by the coefficient "b." Now, by adjusting both a (slope) and b (bendiness), we can potentially achieve a much better fit to the data. It appears that $a \approx 0$ and $b \approx 1$ remarkably align with our dataset Fig.(1).

However, it's lamentable that we had to infer on our own which polynomial terms would best suit our model's fit. Choosing the right terms and complexity remains an exciting challenge in machine learning. Through continued exploration and analysis, we can unlock the secrets hidden within this dataset and gain valuable insights from its hidden patterns.

3 Deep Learning Approach

Let's explore this problem using a neural network. Our objective? Discovering the simplest network suitable for our dataset. Any specific normalization conditions or other considerations?

Let's start with neurons in a basic architecture:

$$z = ax + b$$

These neurons follow a linear path. No matter how many layers I add, the solution remains linear. But let's set aside the technical terms for a moment and put it to the test.

Imagine we have just two neurons, and we combine them:

$$\begin{aligned}
z_1 &= a_1x + b_1 \\
z_2 &= a_2x + b_2 \\
\ldots & \\
z_3 &= z_1 + z_2 \\
z_3 &= (a_1 + a_2)x + (b_1 + b_2)
\end{aligned}$$

Whoops! It seems the result remains linear. Even when we bring these neurons together, the path they follow stays just as straight. Nonlinearity? Not happening here!

3.1 Sigmoid:Non Linearity

What next, we need non-linearity in our model. Let's consider the Sigmoid function and study it a bit.

$$\begin{aligned}
g(x) &= \frac{1}{1+e^{-x}} && (1) \\
\text{For our case:} \quad z &= y + b \quad ; \text{where } y = ax && (2) \\
g(z) &= \frac{1}{1+e^{-(y+b)}} && (3)
\end{aligned}$$

This looks complex... Can we really make sense out of it...!!!

To assess the suitability of the sigmoid function in our context, we perform a Taylor series expansion:

The Taylor series expansion of a function $f(x)$ around a point a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

In this expansion:

- $f(a)$ represents the value of the function at $x = a$.
- $f'(a)$ represents the first derivative of $f(x)$ evaluated at $x = a$.
- $f''(a)$ represents the second derivative of $f(x)$ evaluated at $x = a$.
- Similarly, $f'''(a)$ represents the third derivative, and so on.

The nth term in the expansion involves the nth derivative of $f(x)$ evaluated at $x = a$ divided by $n!$ times $(x-a)^n$.

The Taylor series expansion is a way to represent a function as an infinite sum of terms involving its derivatives evaluated at a specific point a. This expansion is particularly useful in calculus and mathematical analysis for approximating functions around a given point.

For the Sigmoid function around $y \approx 0$ or $|y| < 1$:

Approximate Sigmoid Function

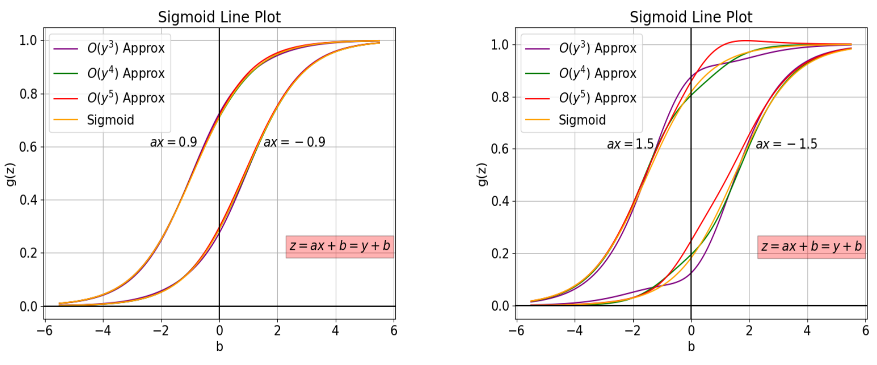
$$g(z) \approx g(b) + yg'(b) + \frac{y^2}{2!}g''(b) + \frac{y^3}{3!}g'''(b) + \frac{y^4}{4!}g''''(b) + \ldots \tag{4}$$

where, $g'(b), g''(b), g'''(b)$ and $g''''(b)$ are the first, second, third, and fourth derivatives of the sigmoid function $g(x)$ Eq.(40) evaluated at b.

Evaluation of Approximation:

The accuracy of the sigmoid function approximation, as defined in Eq. (4), is dependent on the magnitude of y:

- **Suitable Approximation ($|y| < 1$):** For $|y| < 1$, the approximation closely matches the sigmoid function for any value of b (e.g., Fig. 2(a)).
- **Inadequate Approximation ($|y| > 1$):** When $|y| > 1$, the approximation doesn't hold for all values of b (e.g., Fig. 2(b)).



(a) (b)

Figure 2: (a) The approximated sigmoid function works pretty good for $|y| < 1$, for all the values of b. (b) The Sigmoid approximation for $|y| > 1$, is only satisfied for range of b values where $|y + b| < 1$.

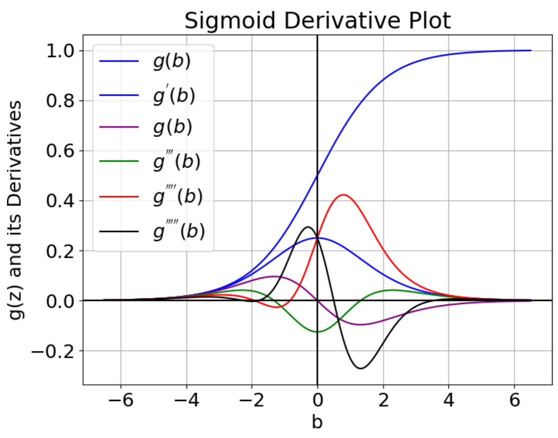


Figure 3: The derivatives of Sigmoid Function.

Key Implications:

Model Conditioning:** To guarantee model accuracy, we prioritize situations where $|y| < 1$, effectively imposing a constraint on the model bias.

Data Normalization:** This finding emphasizes the crucial role of data normalization or rescaling within a range satisfying $|ax| < 1$.

Deep Learning Connection:** This principle aligns with established deep learning practices, such as batch normalisation and data scaling.

Conclusion:

The analysis underscores the vital importance of data normalisation for the validity of the sigmoid function approximation. This ensures model accuracy and aligns with standard practices in deep learning.

While a formal proof falls outside the scope of our current discussion, visual examination of the derivative of sigmoid functions, as depicted in Figure 3, reveals a salient characteristic: their values consistently remain bounded below unity.
The same goes for sigmoid most closest ally *tanh* activation function. We have shown that in Appendix (A.A)

3.2 The Quest Begins: Embracing Complexity to Capture $X = x^2$

Our journey delves into the intriguing realm of neural networks, where we embark on a mission to conquer the challenge of approximating a non-linear relationship: $X \approx x^2$. Through this exploration, we unveil the limitations and hidden potential of different network architectures, paving the way for a deeper understanding of their expressive power.

3.2.1 Single Neuron's Stumble: The Linearity Barrier

Initial Exploration: We commence our quest with a single neuron shown in Fig. (4), armed with linear activation:

$$z = ax + b$$
$$X = g(z)$$

- **Taylor Series Unveiling:** To illuminate the neuron's behavior, we wield the tool of a Taylor series expansion

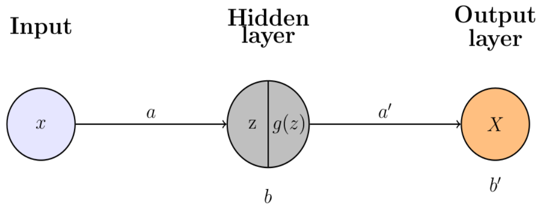


Figure 4: The One neuron architecture of DL.

around the bias b:

$$X \approx g(b) + (ax)g'(b) + \frac{(ax)^2}{2!}g''(b) + \dots$$

- Focusing on the Leading Terms: Scrutinizing the dominant terms, we encounter a crucial insight:

$$X \approx g(b) + (ax)g'(b) + \frac{(ax)^2}{2!}g''(b)$$

Hitting the Wall: Despite the non-linear activation function g, the single neuron's inherent linearity shackles its ability to fully capture the quadratic curvature of $X \approx x^2$. . This limitation impedes our network from accurately modeling the desired relationship.

3.2.2 Enhancing Expressive Capacity with Two Neurons

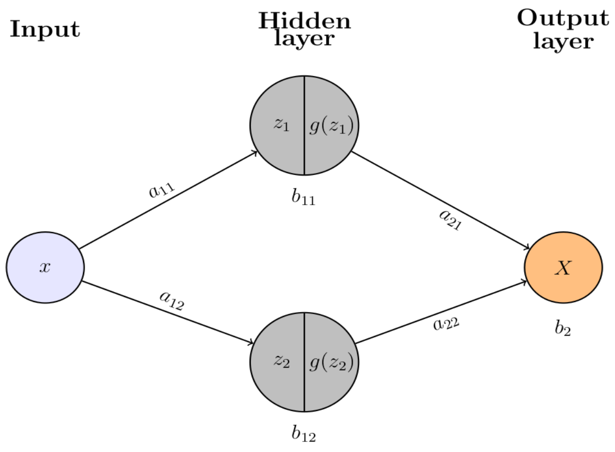


Figure 5: The Two neuron architecture of DL.

To transcend the limitations posed by a solitary neuron, we advance towards a more potent architecture as shown in Fig.(5); dual-neuron network:

$$\begin{aligned}
z_1 &= a_{11}x + b_{11} \\
z_2 &= a_{12}x + b_{12} \\
X &= a_{21}g(z_1) + a_{22}g(z_2) + b_2 \\
X &\approx b_2 + a_{21}g(b_{11}) + a_{22}g(b_{12}) + x(a_{11}a_{21}g^{'}(b_{11}) + a_{12}a_{22}g^{'}(b_{12})) \\
&\quad + \frac{x^2}{2}(a_{11}^2a_{21}g^{''}(b_{11}) + a_{12}^2a_{22}g^{''}(b_{12})) + \frac{x^3}{6}(a_{11}^3a_{21}g^{'''}(b_{11}) + a_{12}^3a_{22}g^{'''}(b_{12})) + ...
\end{aligned}$$

Focusing in leading terms:

$$X \approx b_2 + a_{21}g(b_{11}) + a_{22}g(b_{12}) + x(a_{11}a_{21}g^{'}(b_{11}) + a_{12}a_{22}g^{'}(b_{12})) \tag{5}$$

$$+ \frac{x^2}{2}(a_{11}^2a_{21}g^{''}(b_{11}) + a_{12}^2a_{22}g^{''}(b_{12})) + ... \tag{6}$$

$$\tag{7}$$

Exploring the Taylor series for both neurons and focusing on the leading terms, we derive the weight and bias conditions enabling $X \approx x^2$:

$$\begin{aligned}
a_{21} &= \frac{2g'(b_{12})}{a_{11}^2g'(b_{12})g''(b_{11}) - a_{11}a_{12}g'(b_{11})g''(b_{12})} \\
a_{22} &= -\frac{2g'(b_{11})}{a_{11}a_{12}g'(b_{12})g''(b_{11}) - a_{12}^2g'(b_{11})g''(b_{12})} \\
b_2 &= \frac{2(a_{11}g(b_{12})g'(b_{11}) - a_{12}g(b_{11})g'(b_{12}))}{a_{11}^2g'(b_{12})g''(b_{11}) - a_{11}a_{12}g'(b_{11})g''(b_{12})}
\end{aligned}$$

Simplifying further, we refine these conditions by imposing constraints like $b_{11} = b_{12}$ and $a_{11} = -a_{12}$:

$$a_{21} = a_{22} = \frac{1}{a_{12}^2 g''(b_{12})}$$
$$b_2 = -\frac{2g(b_{12})}{a_{12}^2 g''(b_{12})}$$

Ensuring $|a_{11}x| < 1$ guarantees the convergence of the Taylor series expansion, adding a layer of stability to the model.

Understanding the intuition behind these equations is pivotal. The combination of weighted non-linear activations from both neurons enriches the input representation. This capability enables the network to capture the quadratic curvature of $X \approx x^2$ that a single neuron cannot.

Consider plotting each neuron's activation function alongside the desired quadratic function. This visualization illustrates how the dual-neuron network can flex its combined output to more closely approximate the quadratic curve.

This two-neuron framework signifies a notable advancement in expressive potential compared to a solitary neuron. By thoughtfully configuring weights and biases, we harness the synergy of multiple neurons to tackle intricate non-linear relationships.

Let's evaluate the effectiveness of the following condition in a given model: Let's assess the performance and

```
class PolynomialRegression(nn.Module):
    def __init__(self):
        super(PolynomialRegression, self).__init__()
        self.linear1 = nn.Linear(1, 2)  # Define the weight matrix. Size: Number of features x Number of
        self.m = nn.Sigmoid()
        self.linear2 = nn.Linear(2, 1)  # Define the weight matrix. Size: Number of features x Number of


    def forward(self, x):
        x = self.linear1(x)
        x = self.m(x)
        x = self.linear2(x)

        return x
linear1.weight = [[-0.01],[0.01]]
linear2.weight = [[-3.73344*pow(10,5),-3.73344*pow(10,5)]]
linear1.bias = [3.5,3.5]
linear2.bias = [7.248*pow(10,5)]
```

Figure 6: Pytorch Module For Square Model

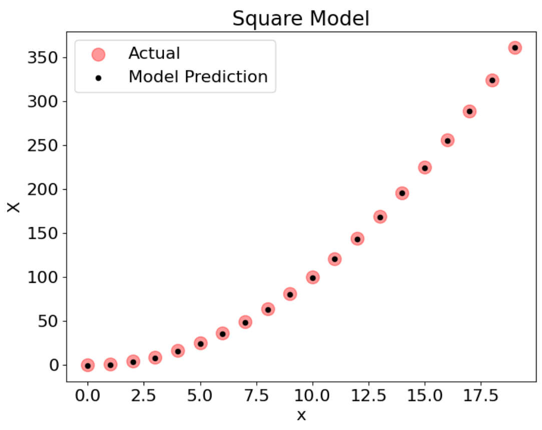


Figure 7: The Comparison between Actual and Model.

validity of this model. This code snippet defines a `Polynomial Regression` class using PyTorch's neural network module. It involves two linear layers along with a sigmoid activation function. The weights and biases are preset for each layer, with explicit configurations provided for better understanding and evaluation.

For values $a_{12} = -a_{11} = -0.01$ and, $b_{12} = 3.5$, we would get $a_{21} = a_{22} = -3.73344 \times 10^5$ and $b_2 = 7.248 \times 10^5$. This would provide a suitable fit as Shown in Fig.(7), where actual and model prediction coincide with minimum error. This setup aims to approximate a polynomial relationship within the data. We'll proceed to test this model.

While the current approach holds promise, could we potentially push the envelope further? Perhaps incorporating higher-order terms, such as cubic terms, into the model might yield even more accurate results.

3.2.3 Delving Deeper: Approximating $X \approx x^3$

Unveiling the Power of Two Can a mere pair of neurons conquer the cubic realm? Let's embark on this exploration, armed with mathematical insights and computational prowess.

Taylor Series Ascendancy We commence by invoking the revered Taylor series to unveil the hidden facets of our two-neuron architecture:

Equations Unveiled

$$\begin{aligned} z_1 &= a_{11}x + b_{11} \\ z_2 &= a_{12}x + b_{12} \\ X &= a_{21}g(z_1) + a_{22}g(z_2) + b_2 \end{aligned}$$

Meticulously expanding these equations, we isolate the leading terms, revealing a pathway towards cubic approximation:

$$\begin{aligned} X \approx\ & b_2 + a_{21}g(b_{11}) + a_{22}g(b_{12}) + x(a_{11}a_{21}g'(b_{11}) + a_{12}a_{22}g'(b_{12})) \\ & + \frac{x^2}{2}(a_{11}^2 a_{21}g''(b_{11}) + a_{12}^2 a_{22}g''(b_{12})) + \frac{x^3}{6}(a_{11}^3 a_{21}g'''(b_{11}) + a_{12}^3 a_{22}g'''(b_{12})) + \ldots \end{aligned}$$

Condition for Cubic Conquest To achieve the coveted $X \approx x^3$, we meticulously derive a set of constraints upon the weights and biases:

Equations Imposed

$$\begin{aligned} a_{12} &= \frac{a_{11}g''(b_{11})g'(b_{12})}{g'(b_{11})g''(b_{12})} \\ a_{21} &= \frac{6g'(b_{11})g''(b_{12})^2}{a_{11}^3(g'(b_{11})g''(b_{12})^2 g'''(b_{11}) - g'(b_{12})g''(b_{11})^2 g'''(b_{12}))} \\ a_{22} &= -\frac{6g'(b_{11})^3 g''(b_{12})^3}{a_{11}^3 g'(b_{12})^2(g'(b_{11})g''(b_{11})g''(b_{12})^2 g'''(b_{11}) - 1g'(b_{12})g''(b_{11})^3 g'''(b_{12}))} \\ b_{21} &= -\frac{6g'(b_{11})g''(b_{12})^2(g(b_{11})g'(b_{12})^2 g''(b_{11}) - g(b_{12})g'(b_{11})^2 g''(b_{12}))}{a_{11}^3 g'(b_{12})^2(g'(b_{11})g''(b_{11})g''(b_{12})^2 g'''(b_{11}) - g'(b_{12})g''(b_{11})^3 g'''(b_{12})} \end{aligned}$$

Numerical Manifestation We translate these theoretical insights into the realm of computation, employing Python to craft a tangible model:

```
class PolynomialRegression(nn.Module):
    def __init__(self):
        super(PolynomialRegression, self).__init__()
        self.linear1 = nn.Linear(1, 2) # Weight matrix definition: Size - Number of features x Number of
        self.m = nn.Sigmoid()
        self.linear2 = nn.Linear(2, 1) # Weight matrix definition: Size - Number of features x Number of

    def forward(self, x):
        x = self.linear1(x)
        x = self.m(x)
        x = self.linear2(x)
```

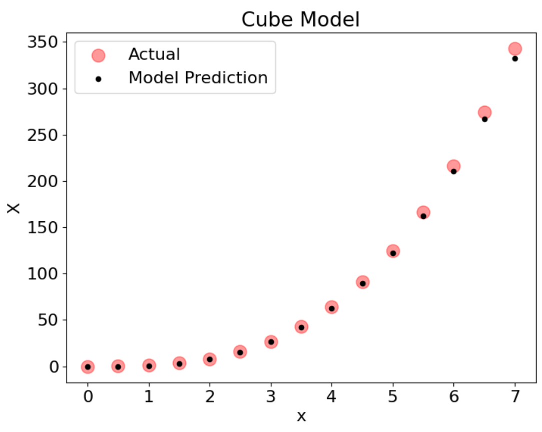


Figure 8: The Comparison between Actual and Model.

```
        return x
data1 = [[0.01],[-0.0490245]]
data2 = [[2.21685*pow(10,6),4.2613*pow(10,5)]]
bias1 = [0.5,-0.1]
bias2 = [-1.58232*pow(10,6)]
```

Let's evaluate the efficacy and reliability of this model. The following code snippet establishes a `PolynomialRegression` class utilizing PyTorch's neural network module. It comprises two linear layers accompanied by a sigmoid activation function. Explicitly predefined weights and biases for each layer are included to enhance comprehension and facilitate assessment.

For values $a_{11} = 0.01$, $b_{11} = 0.5$ and $b_{12} = -0.1$, we would get $a_{12} = -0.0490245$, $a_{21} = 2.21685 \times 10^6$, $a_{22} = 4.2613 \times 10^5$ and $b_{21} = -1.58232 \times 10^6$. This would provide a suitable fit as Shown in Fig.(8), where actual and model prediction coincide with minimum error.
This configuration endeavors to mimic a polynomial association within the dataset. Our next step involves testing this model's performance and meticulously analyzing its accuracy in capturing the inherent data patterns.

3.3 Unveiling the Art of Multiplication with Neurons: A Symphony of Hidden Layers

The Elusive Nature of Multiplication in Single Neurons Individual neurons are great at understanding simple connections between things, but they struggle when it comes to directly understanding something like multiplying two things together, like xy. How can we solve this with two inputs, x and y? If we use just one

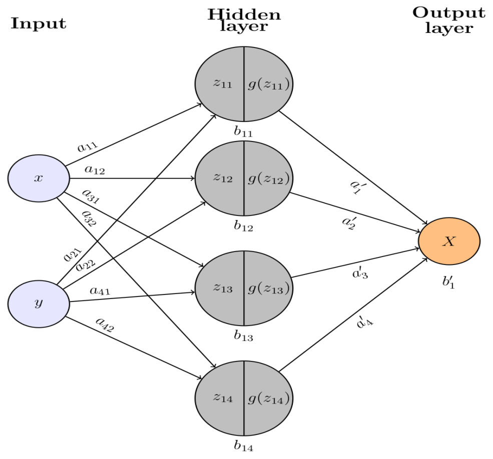


Figure 9: The Four neuron architecture of DL.

neuron:

$$
\begin{aligned}
z &= a_{11}x + a_{12}y + b_{11} \\
g(z) &\approx f(b_{11}) + a_{11}xg^{'}(b_{11}) + a_{12}yg^{'}(b_{11}) + \frac{1}{2}a_{11}^2x^2g^{''}(b_{11}) + \frac{1}{2}a_{12}^2y^2g^{''}(b_{11}) + a_{12}a_{11}xyg^{''}(b_{11}) + ...
\end{aligned}
$$

We're looking for a condition where the term that involves xy is the most important. This is quite a challenging problem, but there might be a clever way to simplify it. This kind of math is trickier for a single neuron because it's not just a straightforward relationship. To tackle this, we need a smarter setup: a team of neurons that can work together to handle these complex tasks.

Constructing a Concerto of Neurons The architectural masterpiece we unveil unfolds in two acts as Shown in Fig.(9):

Act I: The First Hidden Layer - A Quartet of Emphasis Visualize four specialized neurons, each finely tuned to magnify the representation of xy in their activities. We consider 4 neurons with x and y as input. The first hidden layer would be:

$$
\begin{aligned}
z_{11} &= a(x+y) + b \\
g(z_{11}) &= \frac{1}{2}a^2x^2g^{''}(b) + a^2xyg^{''}(b) + \frac{1}{2}a^2y^2g^{''}(b) + axg^{'}(b) + ayg^{'}(b) + g(b) \quad (8) \\
z_{12} &= a(-x-y) + b \\
g(z_{12}) &= \frac{1}{2}a^2x^2g^{''}(b) + a^2xyg^{''}(b) + \frac{1}{2}a^2y^2g^{''}(b) - axg^{'}(b) - ayg^{'}(b) + g(b) \quad (9) \\
z_{13} &= a(-x+y) + b \\
g(z_{13}) &= \frac{1}{2}a^2x^2g^{''}(b) - a^2xyg^{''}(b) + \frac{1}{2}a^2y^2g^{''}(b) - axg^{'}(b) + ayg^{'}(b) + g(b) \quad (10) \\
z_{14} &= a(x-y) + b \\
g(z_{14}) &= \frac{1}{2}a^2x^2g^{''}(b) - a^2xyg^{''}(b) + \frac{1}{2}a^2y^2g^{''}(b) + axg^{'}(b) - ayg^{'}(b) + g(b) \quad (11) \\
& \quad (12)
\end{aligned}
$$

These neurons combine different ways of adding and subtracting x and y, boosted by a carefully chosen number 'a'. This special number helps make the desired product stronger in each neuron's activity.

Act II: The Unifying Maestro - Harmonising the Ensemble A second neuron emerges, playing the role of the maestro, orchestrating the outputs of the first layer:

$$
\begin{aligned}
X &= cg(z_{11}) + cg(z_{12}) - cg(z_{13}) - cg(z_{14}) \\
&= 4a^2cxyg^{''}(b)
\end{aligned}
$$

Here, $g(z)$ represents the activation function, guiding the flow of information within the network. c acts as a critical conductor, carefully tuning the contributions of each neuron. By setting $c = \frac{1}{4a^2g''(b)}$, where $g''(z)$ denotes the second derivative of the activation function, we achieve a remarkable transformation: $X \approx xy$, harmoniously blending the outputs of the first layer through the chosen activation function and meticulously selected parameters.

3.4 From Theory to Code: Translating the Concerto into Python

```python
class MultiplicationRegression(nn.Module):
    def __init__(self):
        super(MultiplicationRegression, self).__init__()
        self.linear1 = nn.Linear(2, 4)  # Define the weight matrix. Size: Number of features x Number of
        self.m = nn.Sigmoid()
        self.linear2 = nn.Linear(4, 1)  # Define the weight matrix. Size: Number of features x Number of


    def forward(self, x):
        x = self.linear1(x)
```

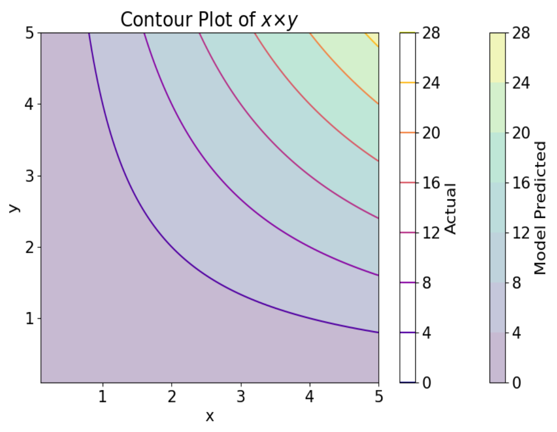


Figure 10: The Comparison between Actual and Model.

```
        x = self.m(x)
        x = self.linear2(x)

        return x
data1 = [[0.01,0.01],[-0.01,-0.01],[-0.01,0.01],[0.01,-0.01]]
data2 = [[2.63907*pow(10,4),2.63907*pow(10,4),-2.63907*pow(10,4),-2.63907*pow(10,4)]]
bias1 = [-1.5,-1.5,-1.5,-1.5]
bias2 = [0.0]
```

This code snippet defines a `Multiply` class in PyTorch, incorporating two linear layers to construct a neural network. The first layer has four neurons to emphasise the interaction between x and y, while the second layer produces the output based on this interaction. This configuration aims to capture the relationship xy within the dataset. For $a = 0.01$, $b - 1.5$, we get $c = -2.6390 \times 10^4$, would provide a suitable fit as Shown in Fig.(10), where actual and model prediction coincide with minimum error. Neural networks can approximate non-linear functions like multiplication. Strategically designed neuron ensembles can isolate specific terms and relationships. Understanding these techniques empowers us to tackle increasingly intricate problems in neural computation.

4 Classification with Neural network

Machine Learning Classification approach is based on creating an plane or probability distribution approach. This include SVM and Logistic Regression, which are based in separation plain approach and naive base is based on probability distribution approach.

We will explore what could be the approach taken by Deep learning network. Can it be something other than these approach ? Lets start with simpler problem with two blob in 2-D plain (11). This would pave us the path to approach for complex problem and problem in higher dimensions.

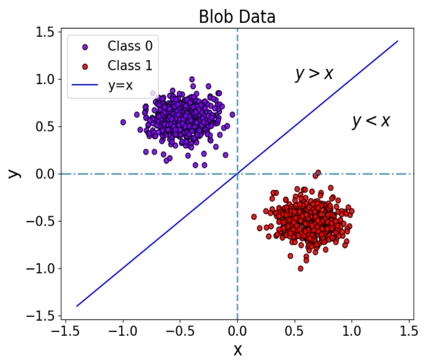


Figure 11: Two blobs of Class 0 and Class 1 in 2-D plane.

The figure presents with a dataset which are described by coordinate x and y, the first blob is under the class 0 and another class 1. We can easily make out of it, that $y = x$ Fig. (11), separate this data in equal proportion.

Ho do we understand this quantitatively?
We create a neural network which accept two input x and y, Let's start with neurons in a basic architecture:

$$z = ax + by + c$$

1. **Separating Line (y=x):**

   - The line $y = x$ serves as a visual boundary between the two clusters. - Points above the line generally belong to one cluster, while points below belong to the other.

2. **Plane z = x - y: Fig.** (12)**(a)**

   - This plane extends the concept of separation into three-dimensional space. - It's defined by the equation $z = x - y$, meaning it calculates the difference between the x and y coordinates of each point.

3. **Class Separation:**

   - Points with $z < 0$ (where $x - y < 0$) are classified as belonging to Class 0. - Points with $z > 0$ (where $x - y > 0$) are classified as belonging to Class 1.

for a=1, b=-1 and c=0, we get:

$$z = x - y$$

The corresponding equation gives plot with z contour. After applying Sigmoid function:

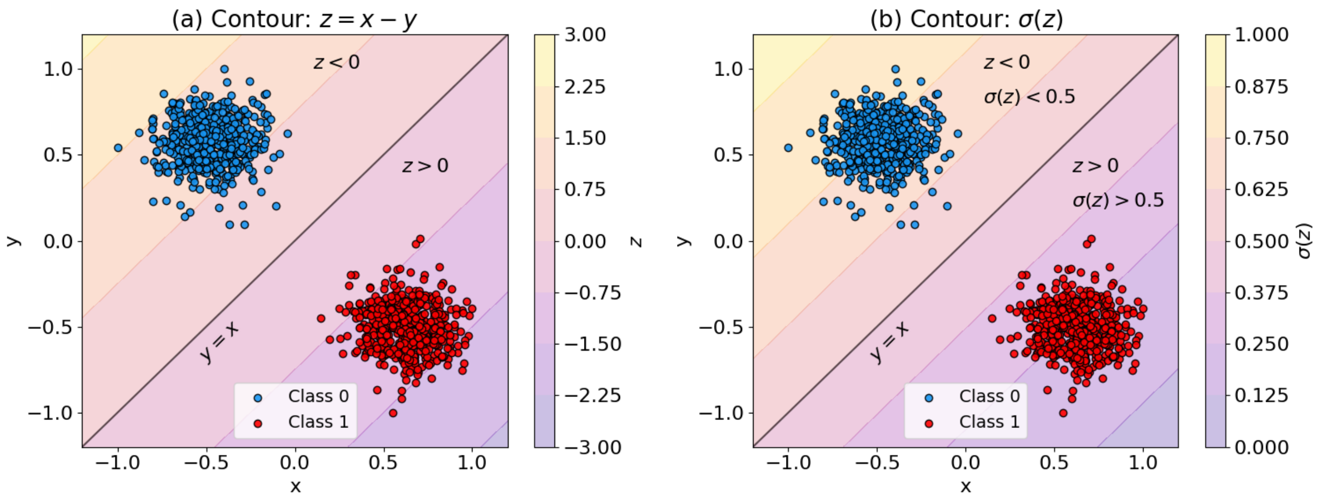


Figure 12: (a) Contour plot of function z. (b) Contour plot of function sigmoid(z).

$$\sigma(z) = sigmoid(z)$$

Binomial Distribution: The sigmoid function transforms the linear decision boundary z = x - y into a form resembling a probability distribution, specifically a binomial distribution with two possible outcomes (0 or 1).
Classification using Sigmoid: Fig. (12)**(b)**

- For $Sigmoid(z) < 0.5$: Points with $z < 0$ are classified as Class 0.
- For $Sigmoid(z) > 0.5$: Points with $z > 0$ are classified as Class 1.

Key Concepts

- **Linear Separation:** Straight line (or plane) effectively divides data.
- **Decision Boundary:** Surface or threshold determining class membership.
- **Normalization:** Transforming data to a common scale for analysis.
- **Binomial Distribution:** Probability distribution with two possible outcomes.
- **Sigmoid Function:** Used for classification and logistic regression.

Summary

- The image demonstrates linear separation of data clusters using a contour plane.
- The sigmoid function introduces a probabilistic approach, mapping data to class probabilities.
- This combination of techniques is fundamental in various machine learning classification algorithms.

4.1 Image Analysis: Decision Boundaries and Softmax Function

This problem we dealt with the sigmoid function. However, we can also use softmax function. How do we approach for it. Two outputs are required, which is compared and provide, the probability for the class. A simple architecture is presented in Fig. (13) Let's start with neurons in a basic architecture:

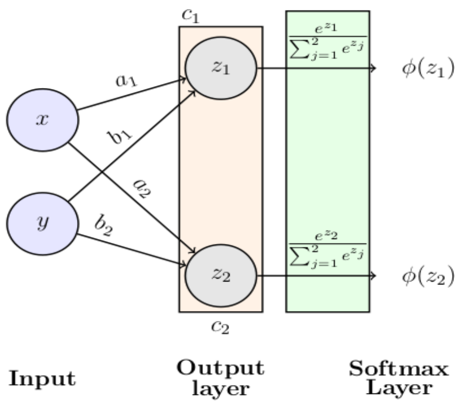


Figure 13: Simple Architecture for Classification Problem.

$$z_1 = a_1x + b_1y + c_1$$

$$z_2 = a_2x + b_2y + c_2$$

for $a_1 = 1, a_2 = -1, b_1 = -1, b_2 = 1$ and $c_1 = 0, c_2 = 0$

- **Two Blobs:** Distinct clusters of data points in 2D space. (Include image of two blobs)
- **Separation Line (y=x):** Visually divides the blobs, suggesting a boundary.
- **Contour Planes (z1=x-y, z2=x-y):** Define two surfaces that effectively partition the blobs in different ways.
- **Softmax Function:** Applied to both z1 and z2 to create normalized probability distributions, aiding in class assignment. (Include image of softmax function)

The z_1 and z_2 are the outputs of the model, We further apply softmax function over this output, this would give the probability distribution of both the classes. The Softmax function is defined as:

The softmax function for a vector $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ is given by

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

For our problem the equation would transform as:

$$\phi(z_1) = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$

$$\phi(z_2) = \frac{e^{z_2}}{\sum_{j=1}^{2} e^{z_j}}$$

$$\phi(z_2) + \phi(z_1) = 1$$

The Equations defines the Fig.(14) as

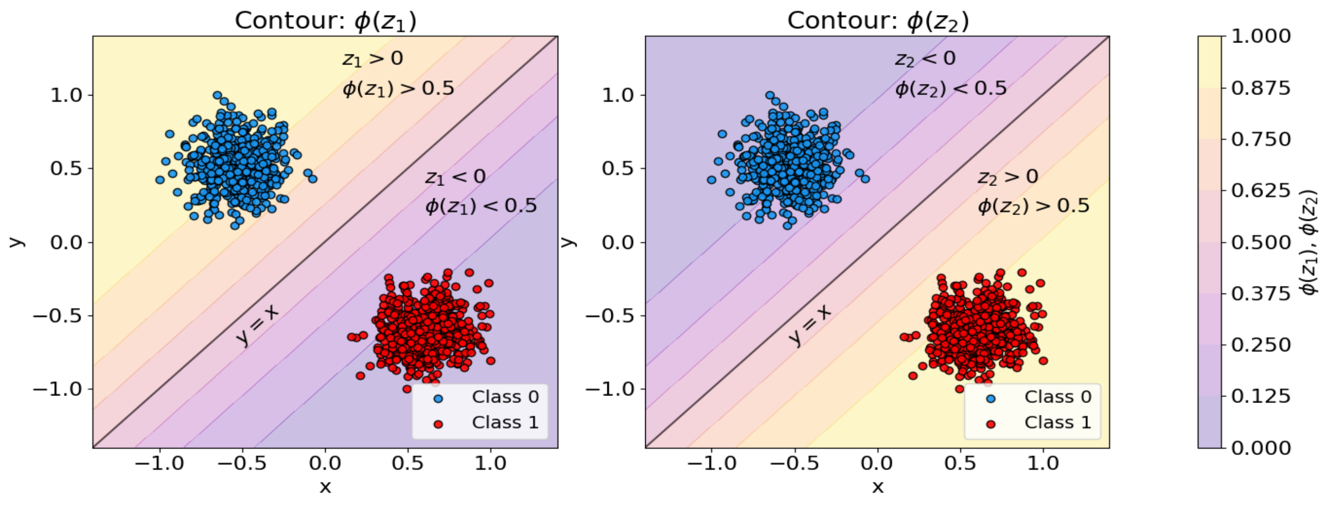


Figure 14: 2-D Classification with softmax. (a) The first plot defines the contour z_1 with softmax function $\phi(z_1)$. (b) The second plot defines the contour z_2 with softmax function $\phi(z_2)$.

Plot (a): Contour $\phi(z_1)$

- **Class 0:** Data points with $z_1 > 0$, falling above the plane.
- **Class 1:** Data points with $z_1 < 0$, positioned below the plane.
- **Probabilities:**
  - $z_1 > 0$: $\phi(z_1) > 0.5$, indicating a higher probability of Class 0.
  - $z_1 < 0$: $\phi(z_1) < 0.5$, suggesting a greater likelihood of Class 1.

Plot (b): Contour $\phi(z_2)$

- **Class 0:** Data points with $z_2 < 0$, falling below the plane.
- **Class 1:** Data points with $z_2 > 0$, positioned above the plane.
- **Probabilities:**
  - $z_2 > 0$: $\phi(z_2) > 0.5$, indicating a higher probability of Class 1.
  - $z_2 < 0$: $\phi(z_2) < 0.5$, suggesting a greater likelihood of Class 0.

Key Concepts

- **Multiple Decision Boundaries:** The use of two contour planes with different conditions demonstrates the ability to create complex decision boundaries for classification.
- **Softmax Function for Multinomial Distributions:** Specifically designed for multi-class classification, ensuring probabilities sum to 1.
- **Probabilistic Interpretation:** Provides a measure of confidence in class predictions.

Summary

- The image showcases the combination of multiple decision boundaries and the softmax function for multi-class classification.
- This approach is essential for handling scenarios with more than two distinct classes.

The plot gives the output, Now we need to compare both output, if the data is $y > x$, we observe, $\phi(z_1) > 0.5$ and $\phi(z_2) < 0.5$, hence the prediction would be Class 0 and vice versa. This has been quite fun, however, more complex await.

4.2 Nonlinear classification

Dive into the fascinating world of nonlinear classification! We'll explore its nuances using a straightforward one-dimensional dataset. Observe how intricate decision boundaries, coupled with activation functions, effectively segregate distinct classes within this seemingly simple space.

Key Elements of the Image and Setup **One-Dimensional Data**: Data points are plotted along a single horizontal axis; showcased by the data distribution depicted in Fig.(15)
Two Distinct Classes: Data is divided into Class 0 (blue points) and Class 1 (red points).
Initial Linear Separation: Dashed lines at $x = -0.5$ and $x = 0.5$ suggest a potential linear boundary but prove

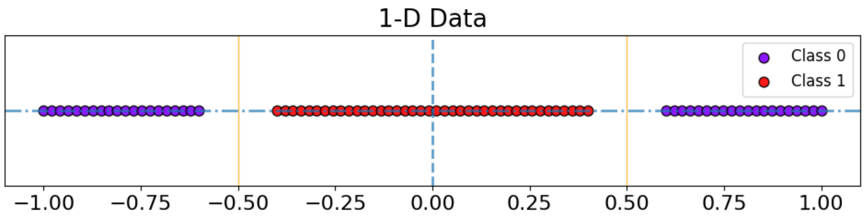


Figure 15: Simple One Dimensional Problem.

inadequate for this distribution.
Linear Separation: It is evident from the dataset we cannot create an linear boundary which could separate the dataset.

To Solve this problem, other than taking some complex activation for non linearity, we take x^2 as an activation. Our choice is to make the understanding and intuition much easier.

$$g(x) = x^2 \tag{13}$$

Network Architecture

There is only one input parameter x

First Layer : **Two Neurons**,

$$z_{11} = a_1x + b_1 \tag{14}$$
$$z_{12} = a_2x + b_2 \tag{15}$$

Second Layer :**Two Neurons**: After using the activation function:

$$\begin{aligned}
z_1 &= d_1g(z_{11}) + e_1g(z_{12}) + f_1 \\
z_1 &= d_1(a_1x + b_1)^2 + e_1(a_2x + b_2)^2 + f_1 \\
z_2 &= d_2g(z_{11}) + e_2g(z_{12}) + f_2 \\
z_2 &= d_2(a_1x + b_1)^2 + e_2(a_2x + b_2)^2 + f_2
\end{aligned} \tag{16}$$

For a condition that at x = 0.5, and x = -0.5, the boundaries are created, we can impose a condition that $z_1 = 0$ and $z_2 = 0$ at x = 0.5 and x = -0.5. We also put a condition that $z_1 > z_2$ in between $x = 0.5 : -0.5$ and vice versa in the other region. This gives us two solutions:

First Solution (17)

$$\begin{aligned}
e_1 &= \frac{-a_1^2d_1 - 4.f_1}{a_2^2} \\
e_2 &= \frac{-a_1^2d_2 - 4.f_2}{a_2^2} \\
b_1 &= 0 \\
b_2 &= 0
\end{aligned}$$

Second Solution (18)

$$\begin{aligned}
a_1 &= 0, b_1 = 0 \\
b_2 &= 0 \\
e_1 &= -\frac{4.f_1}{a_2^2} \\
e_2 &= -\frac{4.f_2}{a_2^2}
\end{aligned}$$

First Solution

With $a_1 = 4, a_2 = -4, d_1 = 4, d_2 = -4, f_1 = -8$ and $f_2 = 8$, we can calculate $e_1 = -2$ and $e_2 = 2$ for first solution. This provides the **Nonlinear Contour Planes**, with two parabolic curves as:

$$\begin{aligned} z_1 &= -8 + 32x^2 \\ z_2 &= 8 - 32x^2 \end{aligned} \tag{19}$$

which, act as decision boundaries.

Second Solution We observe, one important point here, With the second solution, only **one neuron** is required in first hidden layer $a_1 = 0$ and $b_1 = 0$. This would imply the Eq.(16) would transform as:

$$\begin{aligned} z_1 &= e_1(a_2x)^2 + f_1 \\ z_2 &= e_2(a_2x)^2 + f_2 \end{aligned} \tag{20}$$

with appropriate choice $a_2 = 0$, $f_1 = -f_2 = 8$; we get the expression:

$$\begin{aligned} z_1 &= -8 + 32x^2 \\ z_2 &= 8 - 32x^2 \end{aligned} \tag{21}$$

Incorporating the softmax function over z_1 and z_2, we get the distribution of the $\phi(z_1)$ and $\phi(z_2)$ for two classes shown in Fig.(16)

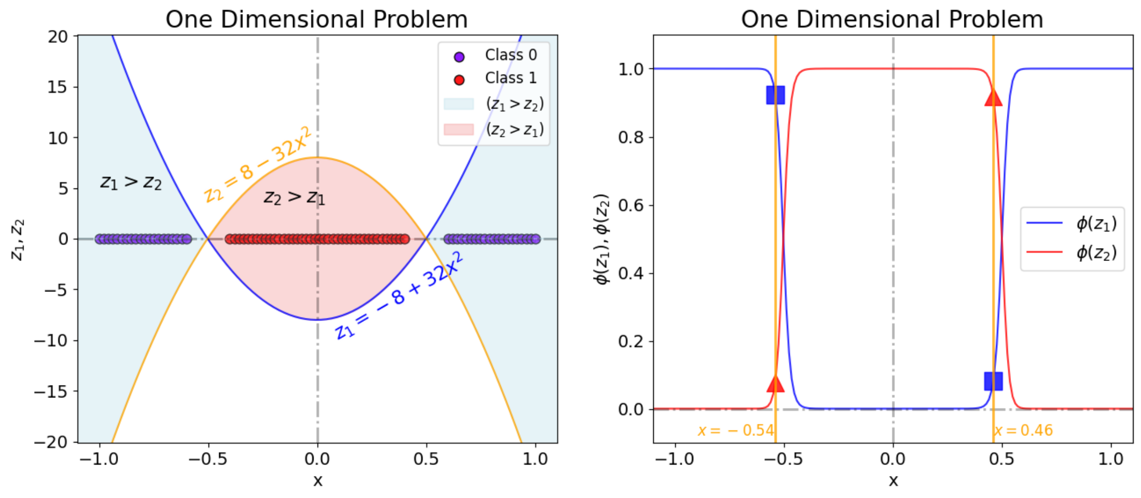


Figure 16: (a) The plot depicts the Eqn. $z_1 and z_2$ with respect to input x. (b) The Plot depicts the distribution of probabilities with respect to z_1 and z_2.

Class Assignment Rules **Class 1**: Points belong to Class 1 if $z_1 < 0$ AND $z_2 > 0$.

Class 0: Points belong to Class 0 if $z_1 > 0$ OR $z_2 < 0$ (or both).

Visual Interpretation of Fig.(16)**(a)**:

- Points above z_1 belong to Class 0.
- Points below z_2 belong to Class 0.
- Points between the curves belong to Class 1.

Visual Interpretation of Fig.(16)**(b)**:

- For point $x = -0.54$, probability of class 0 $\phi(z_1) = 0.9$ and probability of class 1 $\phi(z_2) = 0.1$.
- For point $x = 0.46$, probability of class 0 $\phi(z_1) = 0.1$ and probability of class 1 $\phi(z_2) = 0.9$

Key Concepts

- **Nonlinear Decision Boundaries**: Highlighting the power of nonlinear functions in capturing complex class relationships.
- **Multiple Decision Boundaries**: Utilizing multiple nonlinear boundaries with different conditions for nuanced classification.
- **Activation Functions**: Crucial role in introducing nonlinearity for learning complex patterns.
- **Visualizing Data Relationships**: Plotting data and decision boundaries aids in understanding classification logic.

Summary This image illustrates nonlinear classification in a 1D space, showcasing the combination of linear layers and nonlinear activation functions. It emphasizes the efficacy of nonlinear decision boundaries in handling complex data distributions and their significance in classification tasks.

4.3 Nonlinear classification overdose

Let's delve into a more intricate problem—because why not add a dash of complexity to our lives? Enter the challenging landscape of nonlinear classification, showcased by the data distribution depicted in Fig.(17).

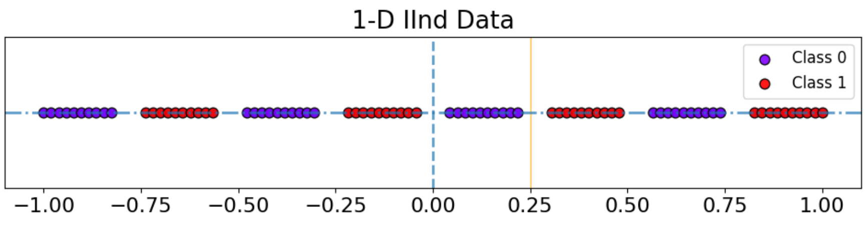


Figure 17: Complex One Dimensional Problem.

Two Distinct Classes: The data is segregated into Class 0 (blue points) and Class 1 (red points), evenly spaced. The dashed lines at $x = 0.25$ hint at a conceivable linear boundary.

This problem appears to demand complex functions with numerous monotonically increasing and decreasing segments. Could there be a solution of that nature?

$$g(x) = sin(x). \tag{22}$$

Network Architecture

There is only one input parameter x

First Layer : One Neurons

$$z_{11} = a_1x + b_1 \tag{23}$$
$$\tag{24}$$

Second Layer :Two Neurons: After using the activation function:

$$z_1 = d_1 g(z_{11}) + f_1 \tag{25}$$
$$z_1 = d_1 \sin(a_1x + b_1) + f_1 \tag{26}$$
$$z_2 = d_2 g(z_{11}) + f_2 \tag{27}$$
$$z_2 = d_2 \sin(a_1x + b_1) + f_2 \tag{28}$$

for a condition that at x = 0.25, and x = -0.25, the boundaries are created, we can impose a condition that $z_1 = 0$ and $z_2 = 0$ at x = 0.25 and x = -0.25. As the soution is periodic, it will take care of the rest of the boundaries. We also put a condition that $z_1 > z_2$ in between $x = 0.0 : 0.25$ and vice versa. As this solution is periodic, it can have infinite solutions. We get one solution as $a_1 = -12$, $b_1 = 0$,$f_1 = f_2 = 0$, $d_1 = -4$ and $d_2 = 4$. This would give us the solution as:

$$z_1 = 4\sin(12x)$$
$$z_2 = -4\sin(12x) \tag{29}$$

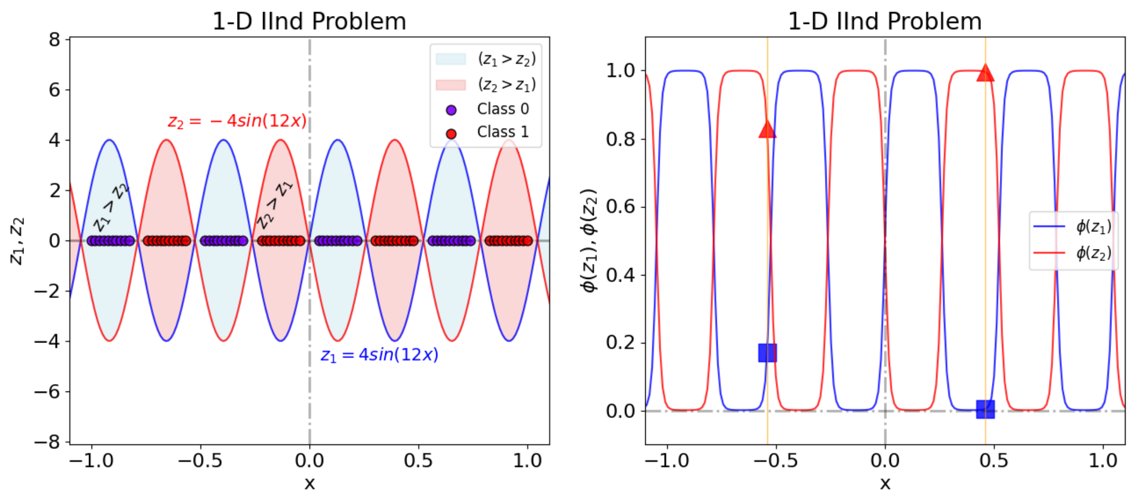


Figure 18: (a) The plot depicts the Eqn. z_1 and z_2 with respect to input x. (b) The Plot depicts the distribution of probabilities with respect to z_1 and z_2.

Incorporating the softmax function over z_1 and z_2, we get the distribution of the $\phi(z_1)$ and $\phi(z_2)$ for two classes Fig.(18). From the above example we can easily understand this problem, how boundaries separate class 0 and class 1.

We observe only one neuron is sufficient to solve this problem. It shows that, carefully articulating a activation function can solve the same problem with smaller networks. To solve the above problem we would require multiple neurons in first layer if consider sigmoid as an activation function(Can you think of how many, try this with your intuition).

Does this means, we should start using sin(x) for all the problems. Lets study the loss function of the above

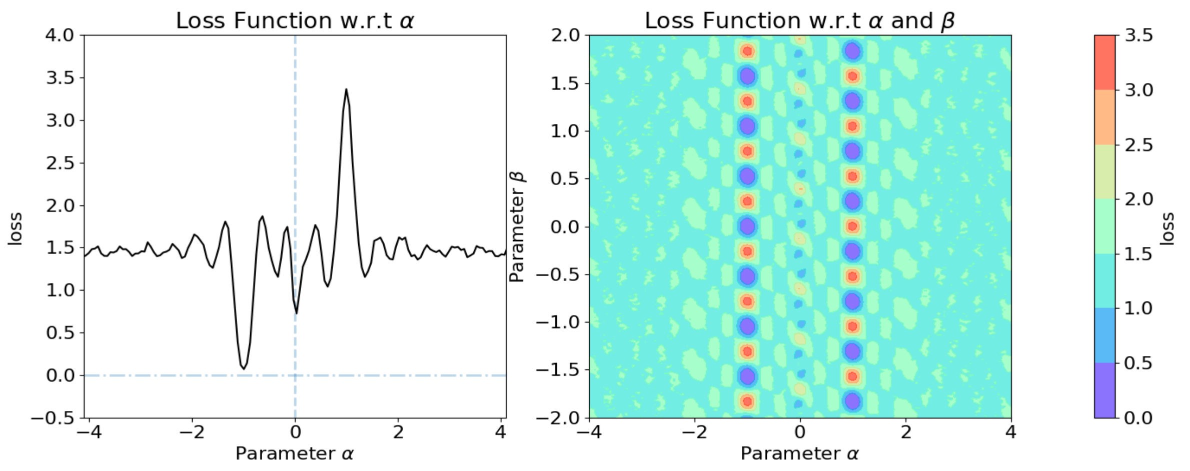


Figure 19: Loss function with respect to α and β, where $\alpha = -a_1/12$ and $\beta = b_1$

example Fig.(19), we have plot the loss landscape with respect to $\alpha = -a_1/12$ and $\beta = b_1$ for our understanding. There are multiple local minim's, The correct optimisation is crucial not to get stuck in local minima. Gradient-based optimization algorithms, commonly used in training neural networks, can struggle to escape these local traps and find the true global minimum, which is crucial for optimal performance.

While sin proved effective in capturing the repeating pattern of that particular dataset, it may not generalize well to other data distributions or classification tasks. Other activation functions like ReLU, Leaky ReLU, and tanh might be more suitable in different contexts.

4.4 Two D problem with Nonlinearity

Lets consider more complex problem with data distribution provided in fig(20).

Two Distinct Classes: Data is divided into Class 0 (blue points) and Class 1 (red points). They are equally spaced. The dashed lines as $x^2 + y^2 = 0.25$ and subsequently shows a potential boundary.

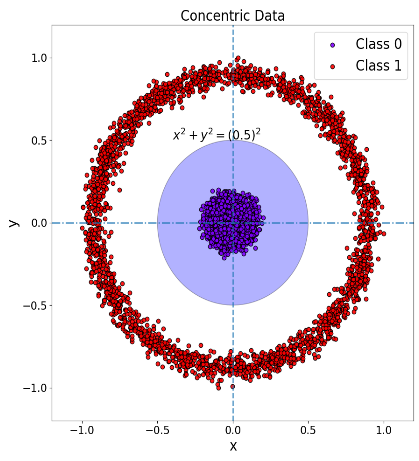


Figure 20: Two concentric circle data.

To Solve this problem, other than taking some complex activation for non linearity, we take:

$$g(z) = z^2 \tag{30}$$

as an activation. Our choice will make the understanding and intuition much easier. subsection*Network Architecture There is two input parameter x and y.

First Layer : Two Neurons

$$\begin{aligned} z_{11} &= a_1 x + b_1 y + c_1 \\ z_{12} &= a_2 x + b_2 y + c_2 \end{aligned} \tag{31}$$

Second Layer : Two Neurons

$$\begin{aligned} z_1 &= d_1 g(z_{11}) + e_1 g(z_{12}) + f_1 \\ z_2 &= d_2 g(z_{11}) + e_2 g(z_{12}) + f_2 \end{aligned} \tag{32}$$

$$\begin{aligned} z_1 &= d_1(a_1 x + b_1 y + c_1)^2 + e_1(a_2 x + b_2 y + c_2)^2 + f_1 \\ z_2 &= d_2(a_1 x + b_1 y + c_1)^2 + e_2(a_2 x + b_2 y + c_2)^2 + f_2 \end{aligned} \tag{33}$$

Given the condition that at $x^2 + y^2 = (0.5)^2$, the boundaries are delineated, we establish a criterion where $z_1 = 0$ and $z_2 = 0$. Additionally, we enforce a condition wherein $z_1 > z_2$ within the region where $x^2 + y^2 < (0.5)^2$. This condition creates contours where the probability of belonging to Class 0 is greater than that of Class 1, and vice versa.
We get one solution as $a_1 = a_2 = -5$, $b_1 = -b_2 = -5$,$f_1 = -f_2 = 2.5$,$c_1 = c_2 = 0$ and $d_1 = -d_2 = -0.2$. This would give us the solution as:

$$\begin{aligned} z_1 &= 2.5 - 10x^2 - 10y^2 \\ z_2 &= -2.5 + 10x^2 + 10y^2 \end{aligned} \tag{34}$$

Incorporating the softmax function over z_1 and z_2, we get the distribution of the $\phi(z_1)$ and $\phi(z_2)$ for two classes (21).

Interpretation of Neural Network Plots with x^2 Activation

Key Observations

- Concentric data distribution: The plots visualize data points arranged in two concentric circles, suggesting a radial classification task.

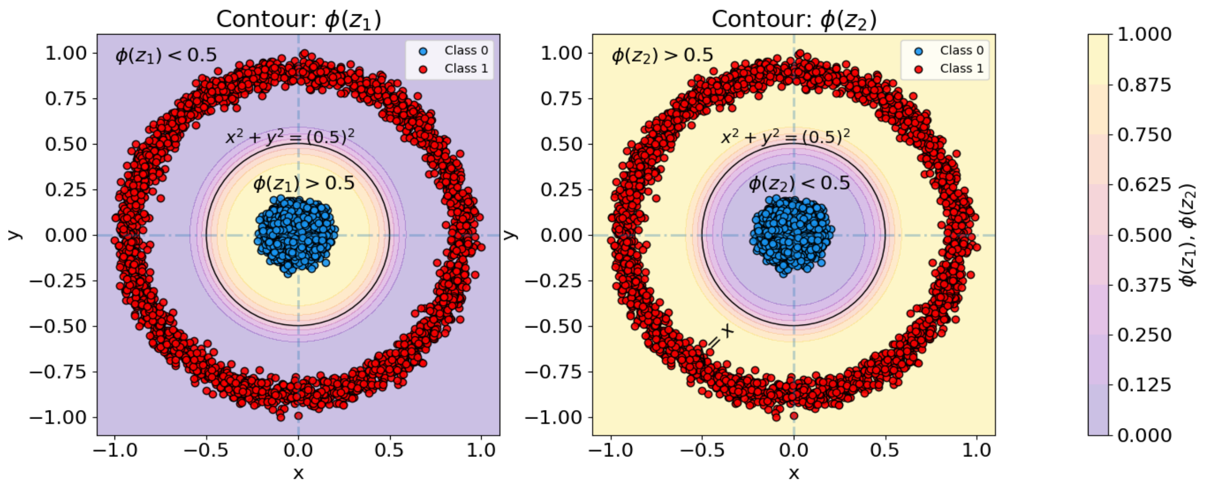


Figure 21: (a) The first plot defines the contour z_1 with softmax function $\phi(z_1)$. (b) The second plot defines the contour z_2 with softmax function $\phi(z_2)$.

- Decision boundary: The plane $x^2 + y^2 = 0.25$ divides the plane into two regions, acting as the decision boundary for classification.
- Softmax values: Softmax(z_1) and softmax(z_2) exhibit an inverse relationship across the boundary, indicating they represent probabilities for opposing classes.

5 Non-linearity with Sigmoid

We have delved in the above problem using some simple activation functions which could be easily understand and there behaviour. We will try to understand how the Sigmoid activation used in these kind of problem or more complex problems. As function is monotonically increasing, It is the linear combination of these functions which makes them universal function approximators.

Sigmoid in one D : We will first explore the one-d examples, which could be generalised to multidimensional problems. For One dimensional problem, lets pick our sigmoid from section 2:

$$g(x) = \frac{1}{1+e^{-x}} \tag{35}$$

$$\text{For our case:} \quad z = ax + b \tag{36}$$

$$g(z) = \frac{1}{1+e^{-(ax+b)}} \tag{37}$$

at $x = -b/a$ the sigmoid function $g(z) = 0.5$, which means, this point separates two regions with $g(z) < 0.5$ and $g(z) > 0.5$. This relation would be a crucial point in our discussions.

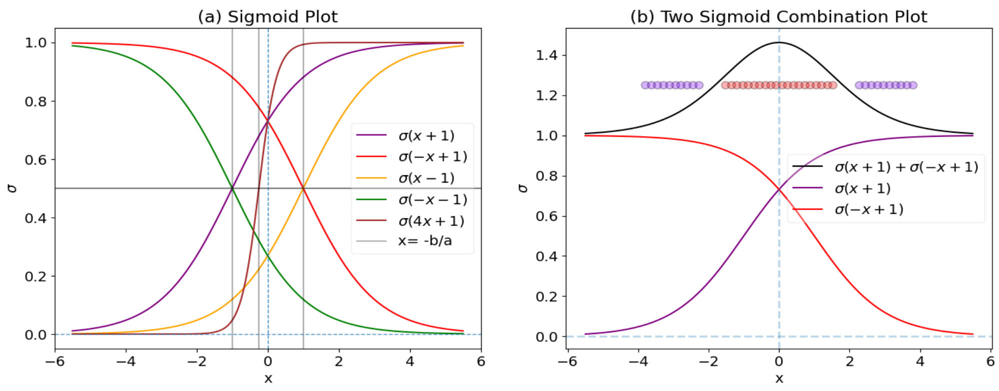


Figure 22: The Sigmoid function with its combination.

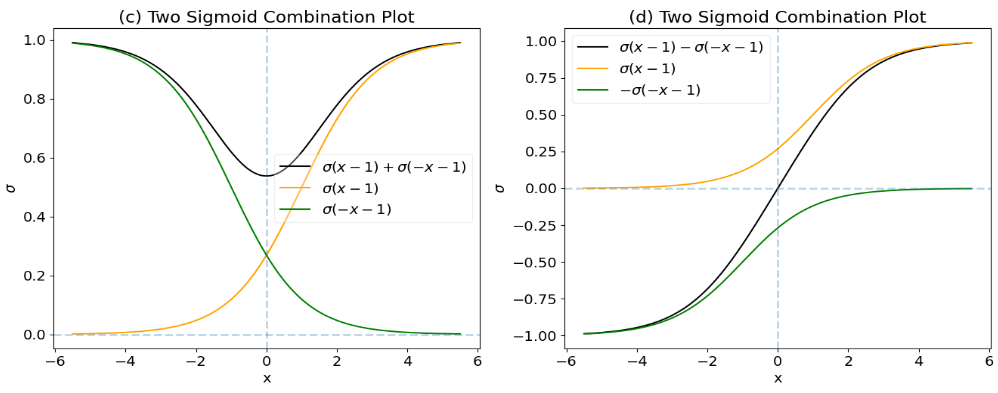


Figure 23: The Sigmoid function with its combination.

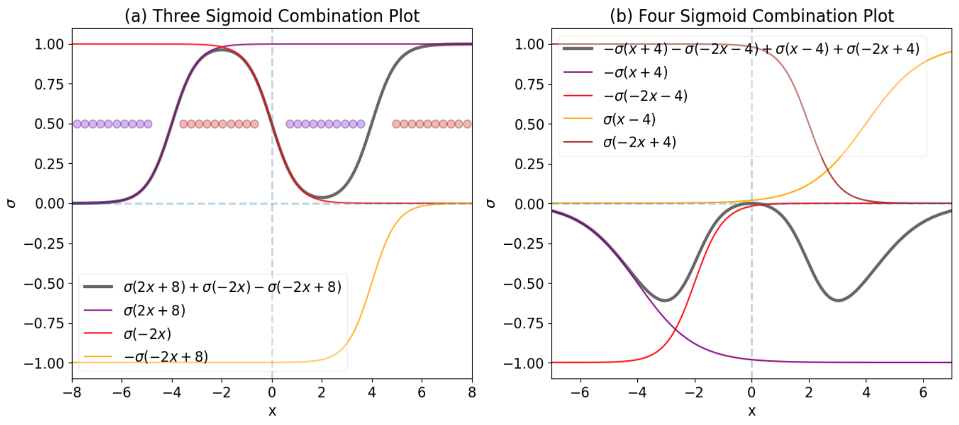


Figure 24: The Sigmoid function with its combination.

Subplot Fig.(22) **(a)** : Sigmoid Plot
The x-axis represents values of $ax + b$, where a and b are constants that determine the shape of the curve. We can understand the how the sigmoid function behaves with respect to parameter a and b.

- Larger values of a result in a steeper curve, leading to a rapid transition from 0 to 1 near the inflection point.
- Smaller values of a produce a gentler, smoother transition along the curve.
- Positive values of a shift the inflection point to the left, causing the curve to rise earlier and reach 1 faster.
- Negative values of a shift the inflection point to the right, resulting in a later and slower rise to 1.
- Positive values of b shift the curve upwards, bringing it closer to 1.
- Negative values of b shift the curve downwards, bringing it closer to 0.

Imagine a horizontal line across the plot at y = 0.5 (the midpoint between 0 and 1). As you decrease the value of a, this line intersects the sigmoid curve later and at a more gradual slope. In contrast, increasing a brings the intersection point closer to the inflection point and makes the transition steeper.

Similarly, shifting b up or down vertically moves the entire curve without changing its shape. So, a positive b value would place the whole curve closer to the top (y = 1), while a negative b would bring it closer to the bottom (y = 0).

Subplot Fig.(22) **(b)** : Two Sigmoid Combination Plot
The blue curve in plot represents the sum of two shifted sigmoid functions:

$$\sigma(x+1)(purplecurve) \tag{38}$$
$$\sigma(-x+1)(orangecurve) \tag{39}$$

Imagine each sigmoid curve starts at 0 and gradually increases to 1. Now, consider the following effects:

- Shifting by 1: Both curves shift one unit to the right (purple) and left (orange) along the x-axis. This means they start rising later compared to the standard sigmoid.
- Negating x-axis: The orange curve also flips horizontally, effectively mirroring it across the y-axis. This inverts its output, so it decreases from 1 to 0 instead of increasing.
- Summing the curves: Adding these shifted and flipped sigmoids creates the blue curve. It starts near 0, rises due to the green curve, reaches a peak, and then gradually decreases due to the orange curve.

Subplot Fig.(23) **(c)** : Two Sigmoid Combination Plot
This plot shows the sum of two sigmoid functions:

- The first sigmoid, sigmoid(x-1), is shifted one unit to the right on the x-axis.
- The second sigmoid, sigmoid(-x-1), is flipped horizontally (mirrored across the y-axis) and shifted one unit to the left on the x-axis.

The resulting combined function (blue curve) starts near 0, rises due to the first sigmoid's contribution, reaches a peak, and then gradually decreases due to the counteracting effect of the flipped and shifted second sigmoid.

Subplot Fig.(23) **(d)** : Two Sigmoid Combination Plot
This plot shows the substraction of two sigmoid functions:
sigmoid(x-1) - sigmoid(-x-1)
The combined function (blue curve) starts near 0, but its behavior depends on the relative strengths of the two sigmoids at different x-values.

Subplot Fig.(24) **(a)** : Three Sigmoid Combination Plot
This plot shows the combination of three sigmoid functions:

- sigmoid(2x + 8): This sigmoid is shifted 8 units to the left on the x-axis and stretched horizontally due to the factor of 2.
- sigmoid(-2x): This sigmoid is flipped horizontally (mirrored across the y-axis) and compressed horizontally due to the factor of -2.
- -sigmoid(-2x + 8): Similar to the second sigmoid, it's flipped and compressed, but also shifted b units to the right on the x-axis and negated (multiplied by -1).

Now, consider how they combine:
–The first sigmoid (purple) rises due to the large positive shift, reaching high values quickly.
–The second sigmoid (orange) starts decreasing immediately due to the flip and compression.
–The third sigmoid (red), shifted positively and negated, initially decreases but eventually starts increasing as it moves further right.
The combined function (blue) reflects the interplay of these individual contributions
Value of b: Shifting the third sigmoid to the right (larger b) can delay its rise and influence the peak location or shape of the combined function.
Scaling factors: The factors 2 and -2 in the first and second sigmoids control their steepness and the strength of their contributions.

Subplot Fig.(24) **(b)** : Four Sigmoid Combination Plot
This plot shows the combination of four sigmoid functions:

- -sigmoid(x+4): Shifted 4 units to the left along the x-axis.
- -sigmoid(-2x-4):Flipped, compressed horizontally due to -2, and shifted 4 units to the left.
- sigmoid(x-4):Shifted 4 units to the right.
- sigmoid(-2x+4):Flipped, compressed, and shifted 4 units to the right.

The first and third sigmoids (purple and red) will rise due to their positive shifts.
The second and fourth sigmoids (orange and blue) will decrease due to flipping and compression.
The combined function (purple) reflects the interplay of these individual contributions:

- It starts near 0 due to the initial dominance of the flipped sigmoids.
- It might exhibit multiple peaks or valleys depending on the balance between the rising and decreasing sigmoids.
- Eventually, it should approach 0 as the flipped sigmoids continue to counteract the rising ones.

Characteristics

- Non-monotonic behavior: Unlike a single sigmoid, this combined function is not always increasing or decreasing. It exhibits a peak point where the green curve's rise is counterbalanced by the orange curve's decline.
- Controllable shape: The values of a and b would influence the steepness, location of the peak, and overall shape of the combined curve.
- Potential applications: This type of non-linearity can be useful for modeling complex relationships in data where a simple increase or decrease wouldn't suffice. For example, it could capture phenomena with a rise and fall pattern, such as sentiment analysis or stock price fluctuations.

Can we think how to solve the first two problems from above section in one-D using sigmoid activation. How many neurons would be required for it. Take it as an exercise...!!!!

Sigmoid in Two-D For Two dimensional problem, lets pick our sigmoid from section 2:

$$g(x) = \frac{1}{1+e^{-x}} \tag{40}$$

$$\text{For our case:} \quad z = ax + by + c \tag{41}$$

$$g(z) = \frac{1}{1+e^{-(ax+by+c)}} \tag{42}$$

at $y = -ax/b - c/b$ the sigmoid function $g(z) = 0.5$, which means, this plain separates two regions with $g(z) < 0.5$

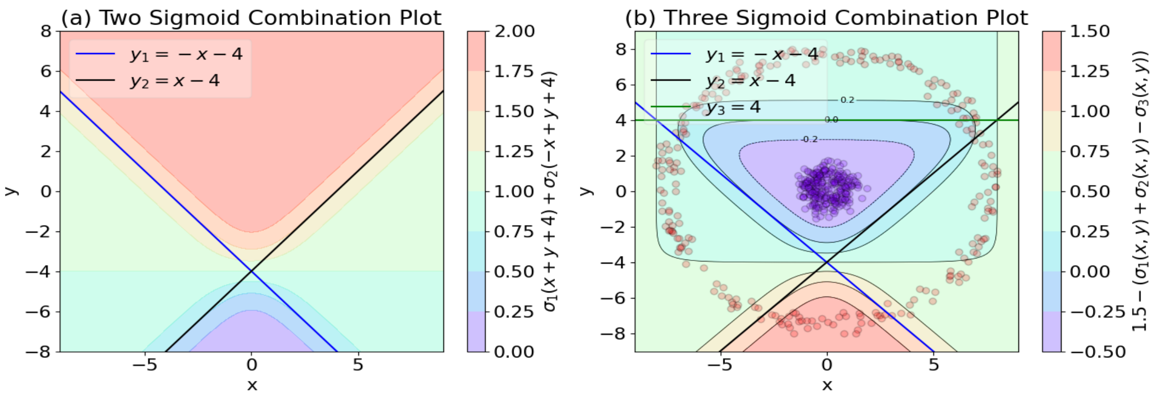


Figure 25: The Sigmoid function with its combination.

and $g(z) > 0.5$. This relation would be a crucial point in our discussions.

Subplot Fig.(25) **(a)** : Two Sigmoid Combination in 2-D Plot

- Each individual sigmoid function creates a curved surface in this space, rising sharply from 0 to 1 near their activation regions.
- For sigmoid(x + y + 4), the activation region shifts along a diagonal plane because both x and y contribute positively.
- For sigmoid(-x + y + 4), the activation region is also diagonal but flipped due to the negative x term.

The combined function (sum of both) creates a complex surface influenced by both diagonal activation regions.

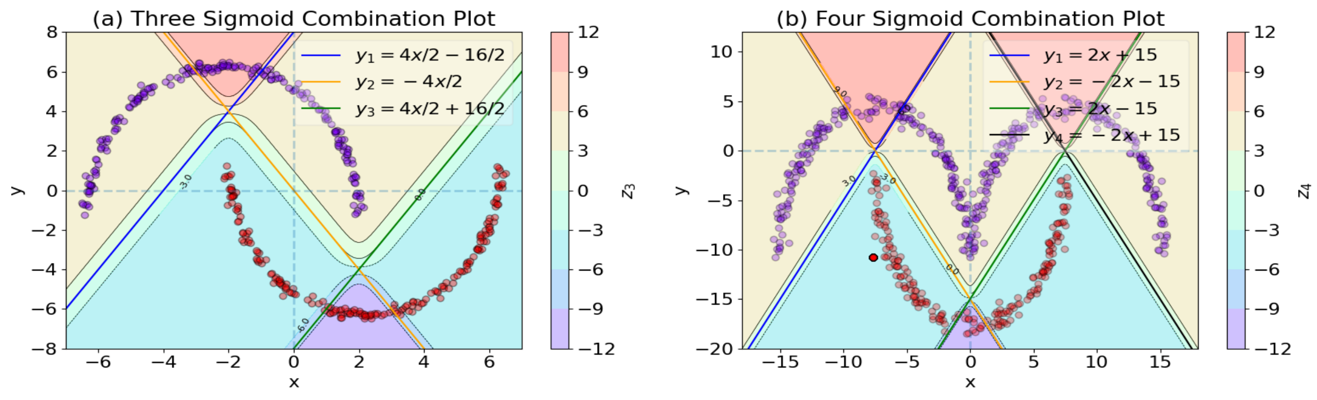


Figure 26: The Sigmoid function with its combination.

Subplot Fig.(25) **(b)** : Three Sigmoid Combination in 2-D Plot
The three sigmoid:

- sigmoid(x + y + 4): Same as in plot (a), combining x and y with an additional positive shift.
- sigmoid(-x + y + 4): Also similar to plot (a), flipping the x-value and adding 4 before applying the sigmoid transformation.
- sigmoid(y - 4): Introduces a new component depending only on the y-value, with a negative shift of 4 before the sigmoid.

These three are combined as:

$$\sigma(x, y) = 1.5 - \sigma(x + y + 4) - \sigma(-x + y + 4) + \sigma(y - 4) \tag{43}$$

We introduce a constant 1.5 to show how a contour is formed with $\sigma < 0$ and $\sigma > 0$ can be seperated. The Concentric circle should remind us of the problem which we encountered earlier. While using sigmoid, it would require three neurons to fit the data.

Subplot Fig.(26) **(a,b)** : Three and Four Sigmoid Combination in 2-D Plot.
From the above discussions, we can now understand how combination of sigmoid functions a complex pattern can be understood. We can also see, how for some problems we approached earlier, only one neurons with specific activation was suffice, however, more neurons are required by using sigmoid activation. This is because sigmoid is monotonically increasing/decreasing function, many neurons are required to generate a curve/complex pattern.

6 Data transformation/ Feature Engineering

In Machine learning Algorithm, our first approach is to transform our dataset, which could be used in the algorithm, as you might remember, we require some kernel approach for classifying concentric circle problem using SVM. These approach are called Feature engineering. However, Deep learning are self sufficient in that. They transform the data from real word dataset to objective defined hyperplane. Considering the objective, the data could be transformed in a peculiar ways, which could make classification or understanding of the data easier.
We would not change the problems, just consider one more layer before passing it to final layer of classification. This layer subdues all the complex information in lower or higher dimension representing in more meaningful way. A Simple architecture of Neural network is presented in Fig. (27). The following code snippet establishes a `Feature Embedding` utilizing PyTorch's neural network module.

```
class Features(nn.Module):
    def __init__(self):
        super(Concentric, self).__init__()
        self.linear1 = nn.Linear(input_num, num)  # Define the weight matrix. Size: Number of features x
        self.m = nn.Sigmoid()
        self.Feature = nn.Linear(num, feature_dim)  # Define the weight matrix. Size: Number of targets
        self.linear3 = nn.Linear(feature_dim, num_class)
```

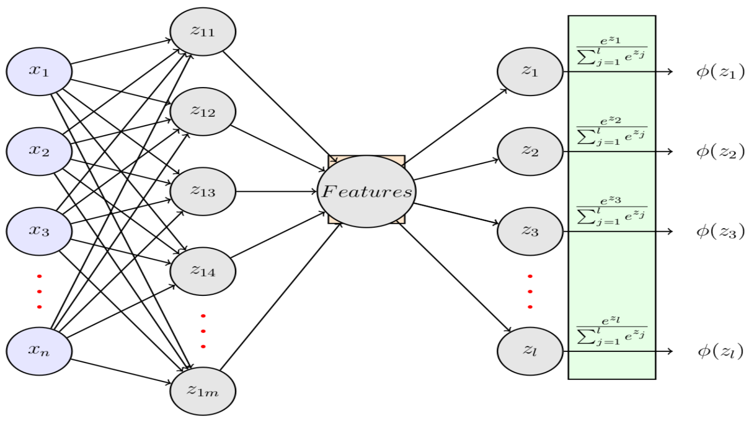

x_1
x_2
x_3
x_n
z_{11}
z_{12}
z_{13}
z_{14}
z_{1m}
$Features$
z_1
z_2
z_3
z_l
$\frac{e^{z_1}}{\sum_{j=1}^{l} e^{z_j}}$
$\frac{e^{z_2}}{\sum_{j=1}^{l} e^{z_j}}$
$\frac{e^{z_3}}{\sum_{j=1}^{l} e^{z_j}}$
$\frac{e^{z_l}}{\sum_{j=1}^{l} e^{z_j}}$
$\phi(z_1)$
$\phi(z_2)$
$\phi(z_3)$
$\phi(z_l)$

Figure 27: A Simple Architecture to present Feature Embedding of Data.

```
    def forward(self, x):
        x = self.linear1(x)
        x = self.m(x)
        x2 = self.Feature(x)
        x = self.linear3(x2)

        return x,x2
Loss = nn.CrossEntropyLoss()
```

Once we train the model for few Epoch, the matrix "self.Feature" layer would present the features in another space with simplicity. As we have seen from the previous sections, we can easily understand for One-D problem in Fig.(28), it would require Three neurons or $num = 3$ get the proper classification (24)(a). We have used $feature\_dim = 1$ and 2 in Fig.(28) for One dimensional Feature and Two dimensional Feature respectively.
Similarly for Two-D problem in Fig.(29), it would require Three neurons or $num = 3$ get the proper classification (25)(b). We have used $feature\_dim = 1$ and 2 in Fig.(29) for One dimensional Feature and Two dimensional Feature respectively.

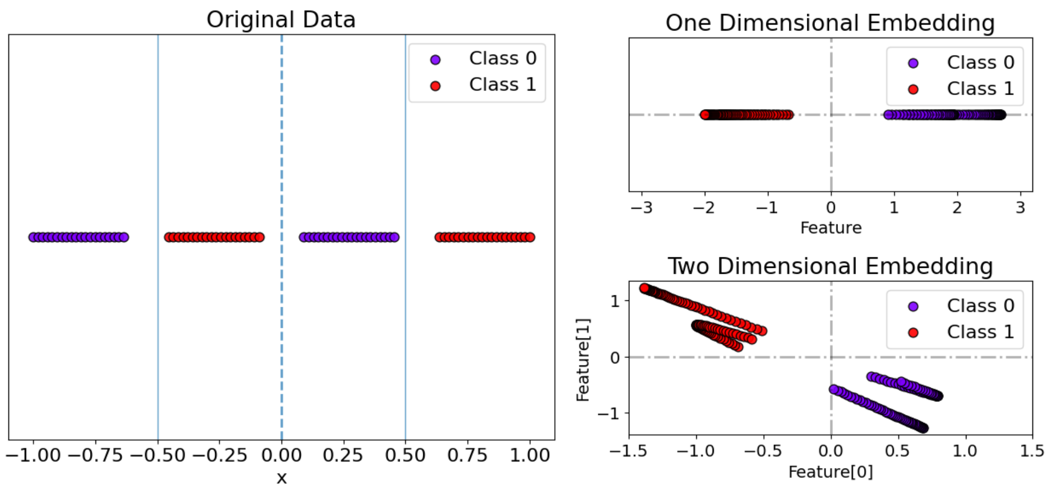


Figure 28: The Sigmoid function with its combination.

Understanding that the original data is difficult to classify helps contextualize the plots you described. Here are some insights we can infer:
Challenges in Classification:

- It is quite evident from Fig. (28) and (29) original data points, it would be challenging to seperate in the initial feature space.
- This could be due to: Non-linear relationships: The underlying relationship between features and class labels might be non-linear, requiring more complex models than simple linear classifiers.

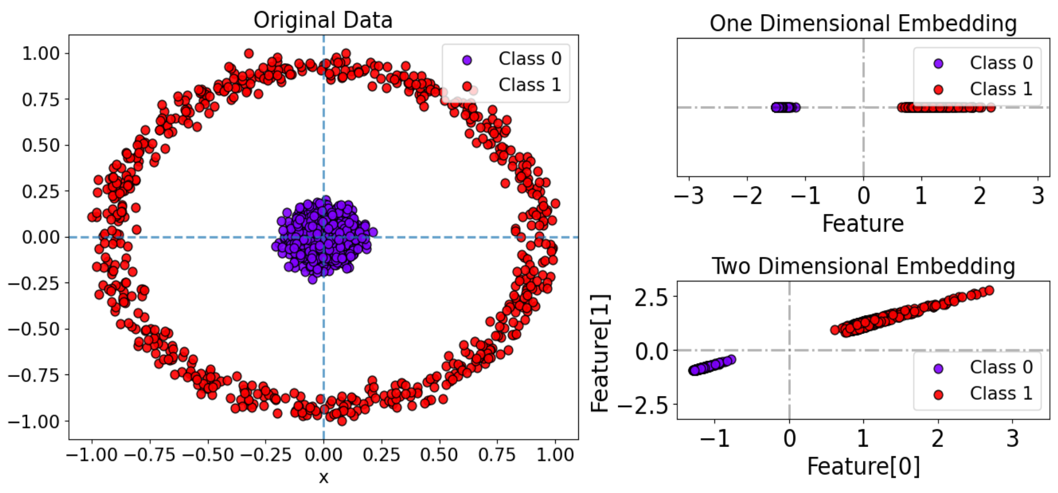


Figure 29: The Sigmoid function with its combination.

Role of Feature Transformation : The purpose of the models in Fig. (28) and (29) One dimensional Embedding and Two dimensional Embedding is to learn new feature representations that are more suitable for classification. They project the original data into new feature spaces designed to separate the classes more effectively.
By analyzing the plots, you can assess how well these transformations address the challenges of the original data:

- Improved class separation: Look for clearer boundaries between red and blue points in the transformed feature spaces compared to the original data.
- Dimensionality reduction: If the new feature space has fewer dimensions, consider if it captures the essential information for classification without losing important details.

While the one-dimensional representations might not provide complete information, they still offer valuable insights:

Fig. (28) **and** (29) **One dimensional Embedding and Two dimensional Embedding** : Compare the class separation achieved by different models or transformations. Which one results in a clearer distinction between red and blue points?

Decision boundaries : Observe the shape and location of the decision boundaries within each plot. Do they capture meaningful relationships between features and classes?

Information loss : Consider how well the one-dimensional projections represent the actual relationships in the original data. Would higher-dimensional visualizations be more informative?

7 NLP: An Approach

We often ponder the meaning of words. Do they truly convey sense? What if we swapped "Positive" with "Negative" and vice versa? If we inform everyone about this change, could we collectively alter our understanding of words? It turns out that context plays a crucial role in shaping meaning, not just the individual words themselves.
In the realm of Natural Language Processing (NLP), representing words as embedding vectors is pivotal. These vectors serve as compact numerical representations of words. Why is this important?

- Tasks: Word embeddings are essential for various NLP tasks, including sentiment analysis, machine translation, and text classification.
- Individual Meaning: Each word has its unique meaning, but it's also related to other words. For instance, "apple" and "fruit" share semantic similarity. Vector Representation: To make sense to machines, we associate each word with a numeric value. These values form vectors that capture the word's semantic meaning relative to other words.
- Similarity Metrics: We measure similarity using techniques like dot product, cosine similarity, or Euclidean distance. The choice depends on the specific context and task.

How do we approach this Embedding. We create an one hot encoding for each words and pass to a network, which would be of $vocab(v) \times Embedding(N)$ dimensional matrix. The one hot encoding would ensure to ignite the specific entry from the matrix in vocab. That N dimensional vector would represent the embedding of that word as shown in Fig. (30) Here the Wi_{th} word is one hot encoded. Only V_i element of matrix would be triggered,

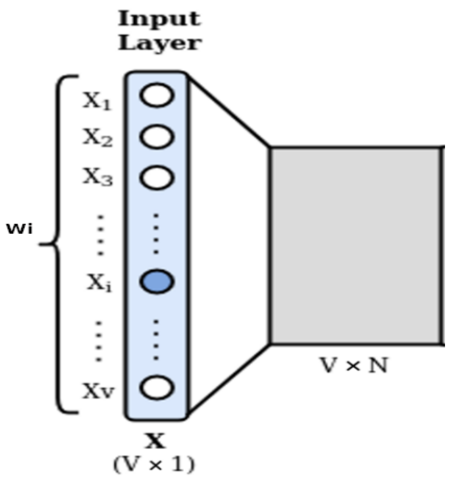


Figure 30: Architecture to present Embedding of words.

with embedding dimension of N. Let's consider a problem where we have some poistive and negative sentiment words. We can create vocabulary for above given words. Lets Assume there are 100 words, with 50-50 positive and

Positive	**Negative**
Affirmative	Negative
Auspicious	Inauspicious
Authentic	Fake
Calm	Agitated
Candid	Deceitful
Captivating	Repellent
....	.....

Table 2: Positive and Negative Sentiment words Table

Negative Sentiment. Hence; The vocubulary would be something like:

$$vocab = \{0 : Affirmative, 1 : Negative, 2 : Auspicious, 3 : Inauspicious, ...\} \tag{44}$$

The One hot encoding would be like :

$$\begin{aligned} Affirmative &= [1, 0, 0, 0, ...] \\ Negative &= [0, 1, 0, 0, 0, ...] \\ Auspicious &= [0, 0, 1, 0, 0, ...] \quad (45) \\ &..... \quad (46) \end{aligned}$$

For our case Let's Consider Embedding Dimension $N = 2$. We create an architecture where The output is taken last layer with classification process to positive (0) and Negative (1) Class. The Architecture is as follows in Pytorch Framework:

```
class Sent_Embedding(nn.Module):
    def __init__(self):
        super(Concentric, self).__init__()
        self.Embedding = nn.Embedding(vocabulary_size,Embed)  # Define the weight matrix. Size: (Vocab x
        self.linear2 = nn.Linear(Embed, num_class)  # Define the weight matrix. Size: Number of Embed x N


    def forward(self, x):
        x = self.Embedding(x)
        x = self.linear2(x)
        return x
Loss = nn.CrossEntropyLoss()
```

Once we train the model for few Epoch, the matrix "self.linear1" layer would learn Sentiment clustering, shown in Fig.(31).

This is a simple way to understand a word embedding. However, we would like to understand, how to relate different words with respect to semantics. To approach this problem we would first understand the key concepts of Common Embedding Techniques:

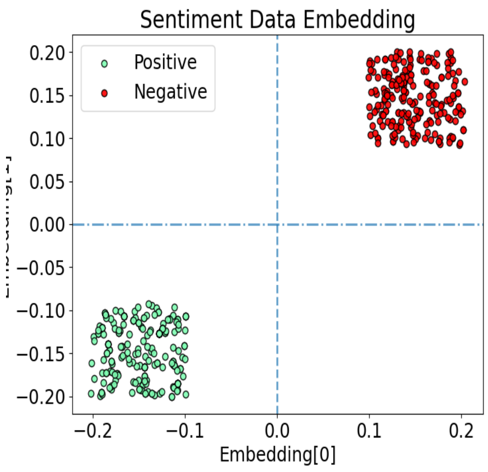


Figure 31: The Embedding Layer of model with sentiment training.

- Word2Vec: This popular method learns word embeddings by analyzing word co-occurrence in a large corpus. It has two main algorithms:
  –CBOW (Continuous Bag-of-Words): Predicts a current word based on its surrounding context words.
  –Skip-gram: Predicts surrounding words based on a given word.
- GloVe (Global Vectors for Word Representation): Combines statistical information from word co-occurrence and word analogy relationships.
- Reduced dimensionality: Enables efficient processing of large text datasets.
- Capture semantic relationships: Enhances tasks like sentiment analysis and text classification by considering word similarities.

Embedding Techniques:

Word2Vec : Uses a shallow neural network with two architectures:

- CBOW: Employs a one-hot encoded context window to predict the target word. Loss function minimises the distance between predicted and actual word embedding vectors.
- Skip-gram: Uses the target word embedding to predict words in its context window. Loss function minimises the distance between predicted and actual context word embedding vectors.

Learns word embedding in a low-dimensional space (typically 300-500) while preserving semantic relationships. We would initiate it by formulating a problem which is easy to understand. Lets consider numbers between 1 to 9, I want the model to understand the relation between each numbers such that the next adjacent numbers are nearby. We propose a problem where we choose 5 random numbers in ascending order, pass 4 numbers and predict the number missing from dataset as shown in Table. (3)

We would first create a vocabulary:

$$vocab = \{0:0, 1:1, 2:2, 3:3, ...\} \tag{47}$$

Numbers	Input	Output
$[1,3,5,6,7]$	$[1,3,0,6,7]$	$[5]$
$[1,1,2,4,7]$	$[1,0,2,4,7]$	$[1]$
$[2,3,5,6,7]$	$[2,3,5,0,7]$	$[6]$
$[1,5,5,6,7]$	$[1,5,5,0,7]$	$[6]$
⋮	⋮	⋮

Table 3: Ascending order Data Table

The One hot encoding would be like :

$$0 = [1,0,0,0, ...]$$
$$1 = [0,1,0,0,0, ...]$$
$$2 = [0,0,1,0,0, ...] \tag{48}$$
$$..... \tag{49}$$

We need to define an Architecture which could satisfy our requirements. We would use CBOW approach in pytorch

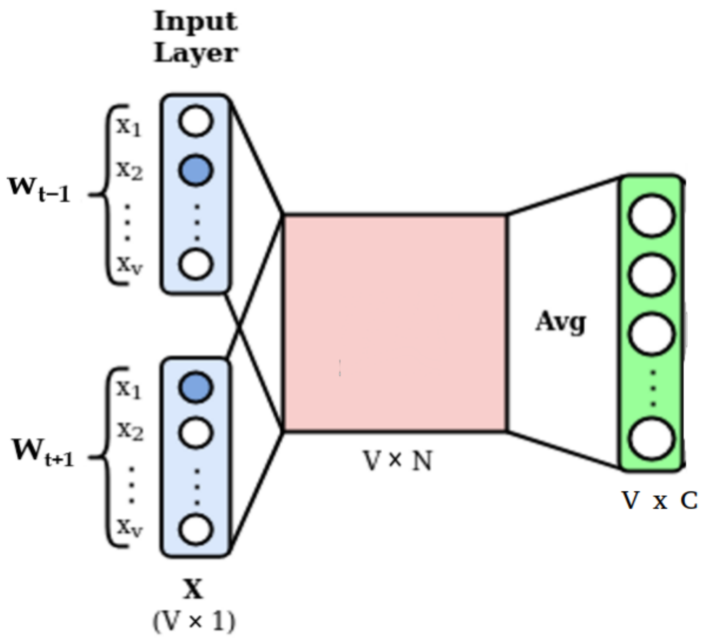


Figure 32: The Architecture for Embedding of Numbers.

framework wehere mean of output from embedding vector is taken. The architecture is given in Fig. (32)

```python
class Number_Embedding(nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super(CBOW, self).__init__()

        self.Embedding = nn.Embedding(vocab_size, embedding_dim)
        self.linear3 = nn.Linear(embedding_dim, vocab_size)
    def forward(self, inputs):
        out = torch.mean(self.Embedding(inputs),dim=1)
        out = self.linear3(out)
        return out
Loss = nn.CrossEntropyLoss()
```

Once we train the model for few Epoch, the matrix ”self.Embedding” layer would learn Sentiment clustering, shown in Fig.(33). It is amazing to see how a simple problem with appropriate loss function could give such a great insights.
The interesting aspect is that the numbers, arranged in ascending order, with almost equal distance from centre it form a curve with good cosine similarity, which suggests.

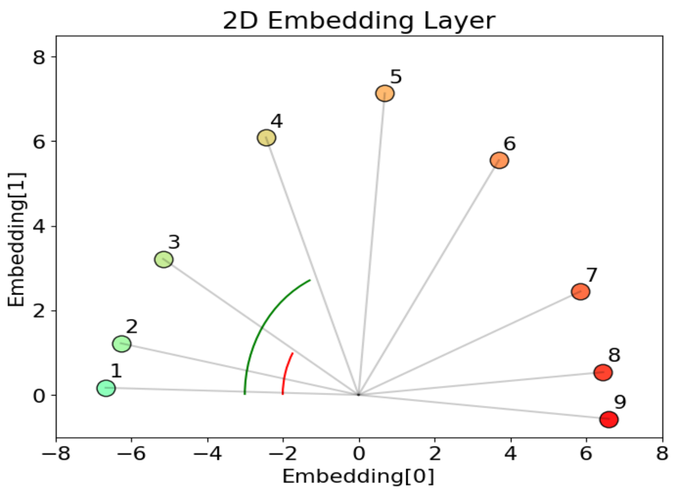


Figure 33: The Embedding of Numbers.

1. The model captures semantic relationships: Numbers inherently have a sequential relationship – their meaning progressively changes with increasing value. The model seems to learn and represent this relationship in the embedding space, where numbers closer in sequence have more similar embeddings.
2. Cosine similarity as a meaningful metric: Since your curve shows good cosine similarity, this implies that the embedding space effectively captures the relative differences between numbers. Cosine similarity measures the angle between vectors, so similar vectors (close numbers) have smaller angles and higher cosine values.

8 Conclusion

Deep learning's hidden magic lies in its ability to capture complex relationships beyond the reach of basic models. We saw how two neurons, armed with non-linearity, can bend their output to fit intricate curves, unlocking secrets hidden within data. The question now burns: can this magic extend to even higher-order relationships, like cubic curves? Exploring this frontier requires delving deeper into weight adjustments and data normalisation, pushing the boundaries of our understanding. Our journey further unfurled as we delved into the realm of classification problems, daring to tackle thought-provoking challenges and confronting them with unconventional activation functions. Building upon this foundation, we ventured into the real-world dilemmas, employing the art of feature engineering to navigate the intricate landscape of natural language processing. We also studied how combination of sigmoid functions allows us to comprehend complex patterns. While for certain problems, a single neuron with a specific activation function suffices, the sigmoid's monotonically increasing/decreasing nature demands multiple neurons to generate intricate patterns.This journey has ignited a spark of curiosity, urging us to unveil the profound potential of deep learning, not just for solving problems, but for gaining a deeper grasp of the intricate tapestry of our world.

A Appendix

A.A Activation tanh function

We just explored the most common activation functions, and showed how it provides the appropriate non-linearity to our model. Is this an ultimate activation function ?? Does anything else comes into picture while selecting activation function other than vanishing and exploding grading issues (a computational challenge) ? In the subsequent section we would explore some simple still powerful activation functions. Even polynomial of x^2 or x^3 could play significant role in an appropriate model and data distribution.
Lets Explore another interesting one tanh activation function.

$$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{50}$$

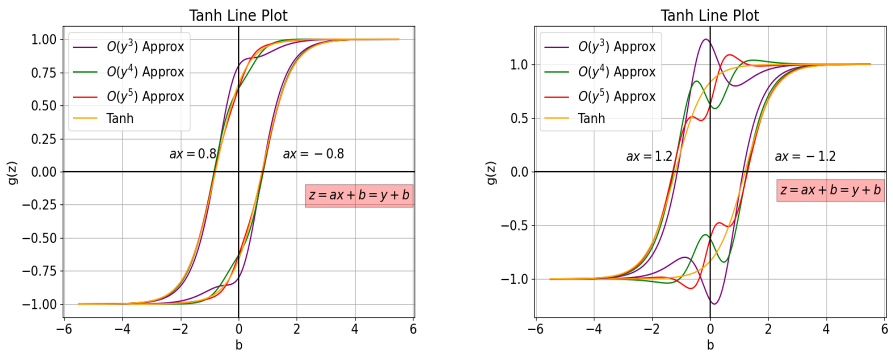
$$\text{For our case:} \quad z = y + b \quad ; \text{where } y = ax \tag{51}$$

$$g(z) = \tanh(y+b) = \frac{e^{(y+b)} - e^{-(y+b)}}{e^{(y+b)} + e^{-(y+b)}} \tag{52}$$

Approximate Tanh Activation Function

$$g(z) \approx \tanh(b) + y\,\mathrm{sech}^2(b) - y^2 \tanh(b)\,\mathrm{sech}(b)^2 + \frac{1}{3}y^3(\cosh(2b) - 2)\,\mathrm{sech}(b)^4 \tag{53}$$

$$+ y^4\left(\tanh^5(b) - \frac{5\tanh^3(b)}{3} + \frac{2\tanh(b)}{3}\right) + \dots \tag{54}$$



(a) (b)

Figure 34: (a) The approximated tanh function works pretty good for $|y| < 1$, for all the values of b. (b) The Sigmoid approximation for $|y| > 1$, is only satisfied for range of b values where $|y + b| < 1$.

This Expression also remain pretty good for $|y| < 1$.