Introduction to Artificial Neural Networks

Abhishek
School of Physical Sciences, National Institute of Science Education and Research, HBNI, Jatni-752050, India.
(*abhishek.2019@niser.ac.in)
(Dated: December 15, 2023)

This report presents the study and implementation of an Artificial Neural Network (ANN) using python3. First, it was implemented using the code provided by J. Sargent & John Stachurski (Ref), and then some new methods were tested and improved the overall efficiency and usage of the ANN.

I. INTRODUCTION

Artificial Neural Networks (ANNs) are computing systems inspired by the structure and function of the human brain. Comprising interconnected nodes called neurons, ANNs process information in a way similar to how neurons transmit signals in the brain. These networks learn from data inputs, adjusting their connections (weights) between neurons to recognize patterns, make predictions, or perform tasks such as image or speech recognition. ANNs are a fundamental concept in machine learning, enabling computers to learn and perform tasks by identifying underlying patterns within vast amounts of data.

Artificial neural networks (ANNs) [1] work somewhat like the human brain. Just like our brains have neurons that communicate through connections, ANNs have computing elements that do the same thing. These elements receive input signals and assign a weight to each signal, determining how important it is. These weights can change, adapting as the network learns. The elements then add up all the weighted inputs to produce an output signal, kind of like how neurons in our brains sum up information. There are different types of ANNs, but broadly speaking, they fall into two main categories based on how they learn and adapt over time.

- In "supervised learning", a 'teacher' guides the ANN during the learning phase by providing feedback on its performance or indicating the correct behavior.
- In "unsupervised learning", the ANN independently examines the properties of the dataset and learns to mirror or represent these properties in its output.

II. OBJECTIVES

Primary goals of this project was to achieve the following goals:

- **Estimating Unknown Functions:** This involves finding a way to represent an unknown function using a function that is already known or familiar to us.
- **Using Data for Estimation:** To do this, we gather data on both the left-hand (input) and right-hand (output) variables of the function we're trying to approximate.
- **Applying Known Functions:** With this data, we aim to fit or approximate the unknown function by using a known function that closely matches or represents the relationship between the input and output variables.
- **Refining the Approximation:** The process often involves adjusting parameters or characteristics of the known function to best fit the data, improving the accuracy of the approximation. It might require several iterations or adjustments to refine the estimation until the known function closely resembles the behavior or pattern of the unknown function based on the provided dataset.

III. STRUCTURE OF A SIMPLE ARTIFICIAL NEURAL NETWORK (ANN)

Neuron

A neuron is the fundamental unit of a neural network, analogous to a biological neuron in the human brain. It receives input signals, processes them, and produces an output signal. It integrates these inputs, applies weights to them, and uses an activation function to determine the output. Please see Figure 1.

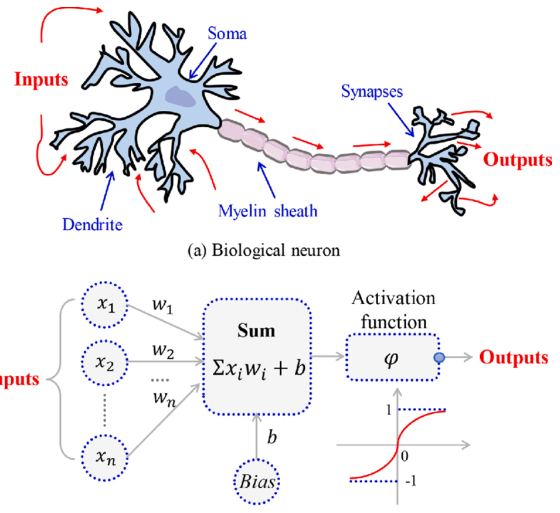

FIG. 1: ANNs are inspired from biological neuron [2].

Activation Function

An activation function is a crucial element of a neural network that introduces non-linearity into the network. It determines whether a neuron should be activated or not based on the weighted sum of its inputs. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), tanh, etc.

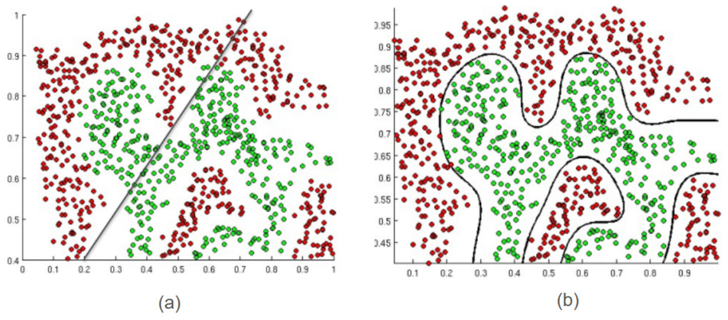

FIG. 2: (a) Without activation function (b) Activation function introducing non-linearity in prediction.

1. **Sigmoid Function:**
    - **Definition:** The sigmoid function is a smooth, S-shaped curve that maps any input to a value between 0 and 1.
    - **Mathematical Expression:** $\sigma(x) = \dfrac{1}{1+e^{-x}}$
    - **Properties:** Useful for binary classification due to its range (0 to 1), but can suffer from vanishing gradients.

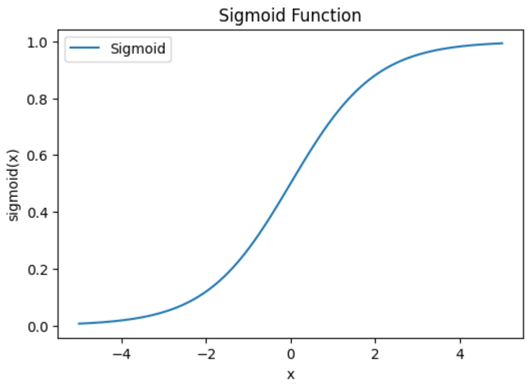

FIG. 3: sigmoid function

2. **ReLU (Rectified Linear Unit):**
    - **Definition:** ReLU outputs the input directly if it's positive, and zero otherwise.
    - **Mathematical Expression:** $f(x) = \max(0, x)$
    - **Properties:** Efficient and helps mitigate vanishing gradients, but can face the "dying ReLU" issue for negative inputs.

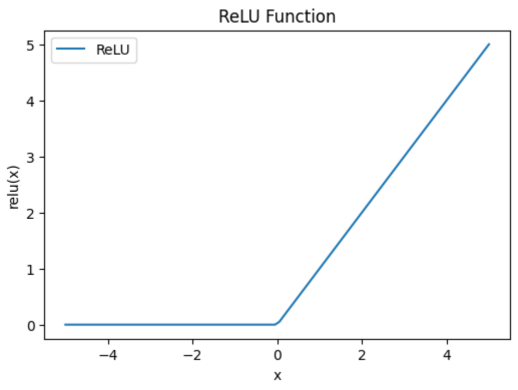

FIG. 4: relu function

3. **Tanh Function:**
    - **Definition:** The hyperbolic tangent function maps values between -1 and 1, centered around zero.
    - **Mathematical Expression:** $\tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
    - **Properties:** Squashes input values to -1 to 1, making it zero-centered, aiding training stability.

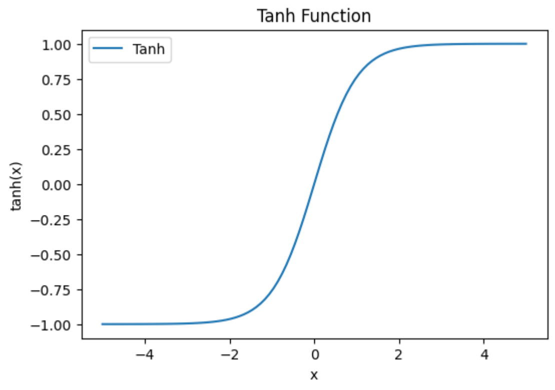

FIG. 5: tanh function

Network of Neurons

A network of neurons, or neural network, is a collection of interconnected neurons organized in layers. Information flows from the input layer through hidden layers to the output layer. The connections between neurons have associated weights that are adjusted during the learning process.

Neural Network as a Composition of Functions

A neural network can be viewed as a composition of functions. Each layer in the network represents a function, transforming the input data through successive layers to produce the final output. This composition allows the network to learn complex relationships between input and output data.

Let $x \in \mathbb{R}$ be a scalar and $y \in \mathbb{R}$ be another scalar. We assume that y is a nonlinear function of x :

$$y = f(x)$$

We want to approximate $f(x)$ with another function that we define recursively. For a network of depth $N \geq 1$, each layer $i = 1, \ldots N$ consists of - an input x_i - an affine function $w_i x_i + bI$, where w_i is a scalar weight placed on the input x_i and b_i is a scalar bias - an activation function h_i that takes $(w_i x_i + b_i)$ as an argument and produces an output x_{i+1}

Back-propagation and the Chain Rule of Differential Calculus

Back-propagation is a key algorithm used in training neural networks. It calculates the gradient of the loss function with respect to the network's weights by applying the chain rule of differential calculus. It propagates the error backwards through the network, adjusting the weights to minimize the difference between predicted and actual outputs.

Loss Function

A loss function measures how well a machine learning model's predictions match the true or target values. It quantifies the model's performance by calculating the difference between predicted values and actual values.

- **Purpose**: The primary goal of a loss function is to provide a single scalar value that represents how well the model is performing on the given dataset.
- **Types of Loss Functions**:
  - **Regression Problems**: Common loss functions include Mean Squared Error (MSE) or Mean Absolute Error (MAE), which measure the difference between predicted and actual numeric values.
  - **Classification Problems**: Cross-entropy-based loss functions like Binary Cross-Entropy or Categorical Cross-Entropy are often used for classification tasks. These penalize the model based on the difference between predicted probabilities and true labels.
  - **Custom Loss Functions**: Depending on the problem at hand, custom loss functions might be designed to address specific requirements of the task.
- **Optimization**: During training, the model adjusts its parameters (weights and biases) to minimize the loss function. This process is typically performed using optimization algorithms like Gradient Descent, where the gradient of the loss function guides the updates to the model's parameters.

Learning Rate

The learning rate is a parameter that controls the step size or rate at which the model parameters are updated during training. It determines the magnitude of adjustments made to the model in response to the calculated gradients of the loss function.

- **Impact on Training**: A higher learning rate allows for larger updates in model parameters in each iteration, potentially leading to faster convergence. However, a very high learning rate might cause the model to overshoot the optimal values and fail to converge.
- **Finding the Right Learning Rate**: The learning rate is a crucial parameter, and finding an appropriate value often involves experimentation and tuning. Techniques like learning rate schedules or adaptive learning rate methods (e.g., Adam, RMSprop) dynamically adjust the learning rate during training based on the observed behaviour of the optimization process.
- **Effects on Optimization**: A low learning rate might lead to slow convergence or get stuck in local minima, while a high learning rate might cause oscillations or divergence in the optimization process.

In summary, the loss function evaluates model performance, while the learning rate governs the rate at which the model parameters are updated during training, affecting the optimization process and the model's ability to converge to an optimal solution.

NEURAL NETWORK PREDICTION PROCESS

1. **Input Data**: The neural network begins with input data that are fed into the network's input layer.
2. **Activation Function**: Each neuron in the network, excluding the input layer, uses an activation function (e.g. ReLU, Sigmoid, Tanh) to introduce non-linearity into the system.
3. **Weights and Bias**: The network assigns weights and biases to connections between neurons, determining the strength and direction of connections.
4. **Forward Propagation**: Input data is multiplied by weights, biased, and passed through activation functions layer by layer, generating an output prediction.
5. **Loss Function**: Predicted output is compared to the actual output using a loss function, which calculates the difference between predicted and actual values.
6. **Backpropagation**: The network adjusts its weights and biases to minimize the loss function by calculating derivatives using the chain rule and updating parameters.

7. **Gradient Descent**: Derivatives are used in gradient descent algorithms (e.g., SGD, Adam) to update weights and biases, minimizing the loss function.
8. **Iterations**: Steps 4-7 are repeated for multiple iterations or epochs, refining predictions by updating weights and biases.
9. **Convergence**: With iterations, the network learns data patterns, and the loss decreases, ideally converging to a minimum value.
10. **Prediction**: Once trained, the network predicts outputs for new inputs using learned weights and biases through forward propagation.

IV. IMPROVEMENT IN ANN

I experimented with different setups in neural networks, trying out various node counts and hidden layers. I also tested different activation functions and even tried a new loss function. Overall, I worked on expanding the ways we use neural networks, aiming to improve accuracy by exploring different methods.

V. ENHANCEMENTS TO BASIC ANN IMPLEMENTATION

The provided code is the basic implementation of an ANN. It worked fine with the linear functions. However, it failed completely when it comes to predicting the non-linear function. To overcome this limitation, I played with the following parameters to make it more stable and accurate. Obtained results will be shown in the results sections.

- **Addressing Limitations in Predicting Non-Linear Functions**:
  - **Exploration of Network Architectures**:
    - Optimized the number of nodes and layer in the neural network. More numbers of nodes is better but it may introduce the overfitting issue. Similarly, adding more layers can add vanishing gradient [3].Testing various patterns of neural networks by adjusting the number of nodes and hidden layers to improve the model's capability to learn and predict non-linear functions.
  - **Diverse Activation Functions**:
    - In the given code; QuantEcon only used sigmoid which is quite sensitive to the vanishing gradient. Testing and evaluation of different activation functions beyond basic linear functions, such as ReLU, Tanh, and softplus activation functions, to introduce non-linearity and enhance the network's ability to capture complex patterns in data.
  - **Experimentation with Loss Functions**:
    - There are various loss function available discussed in previous section III. However, different loss functions works for different tasks like regression, classification, etc. This report aim to predict the function, which is related to regression. Therefore, ANN is tested with Mean Squared Error (MSE) and Mean Absolute Error (MAE) which significantly improved the accuracy of neural network.
- **Broadening Scope and Testing**:
  - **Increased Model Complexity**:
    - Systematic testing involves adjustments to the number of nodes and hidden layers, exploring a range of network architectures to identify the optimal structure for handling non-linear data representations.
  - **Diverse Function Prediction**:
    - Extensive evaluation across various non-linear functions to validate the model's improved accuracy and performance with the implemented enhancements.
- **Summary of Achievements**:
  - **Enhanced Usage and Accuracy**:
    - Significant improvements are achieved by comprehensively exploring neural network variations, encompassing architecture adjustments, diverse activation functions, and with different loss functions.
    - Increased model versatility and accuracy in predicting non-linear functions, overcoming the limitations of the basic implementation's performance on such functions.

VI. RESULTS OBTAINED FROM MODIFIED CODE

This section presents the latest result obtained with a modified and improved version of ANN available at QuantEcon. From now on, the code given by ANN will be denoted as "ANNQ". The process for getting this improvement is already explained in the previous section.

A. Predicting linear function

When it comes to predict linear function ANNQ is quite accurate.Please see Figure 6.

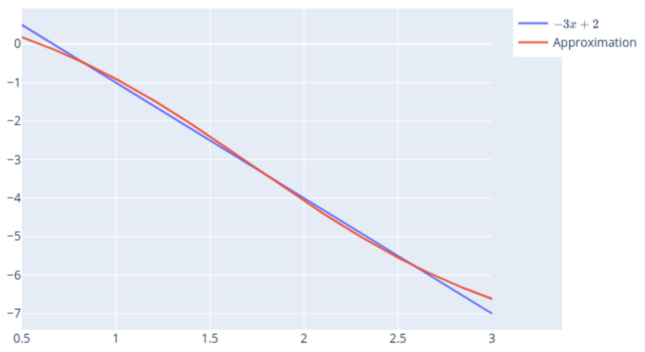

FIG. 6: $f(x) = -3x+2$ predicted by ANNQ

B. Predicting non-linear functions

However, when it comes to predict the non-linear function it failed. See Figure 7.

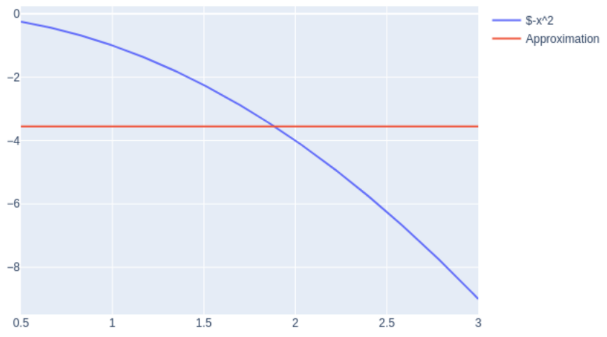

FIG. 7: $f(x) = x^2$ predicted by ANNQ

Therefore, the activation function was changed from sigmoid to **softmax** in ANNQ.

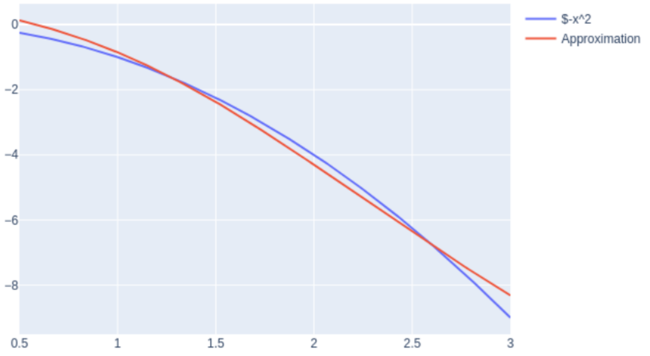

FIG. 8: Modified ANN predicting $f(x) = -x^2$ in the interval of [0.5,3]

1. *Predicting sin(x) with softmax as activation function*

I tried to approximate the periodic function by changing different numbers of layers and learning rates. Different numbers of layers are tested with 0.001 as the learning rate. We can see that the best-obtained result while using a number of layers is four. After adding more layers, it faces a vanishing gradient. Please see Figure 9.

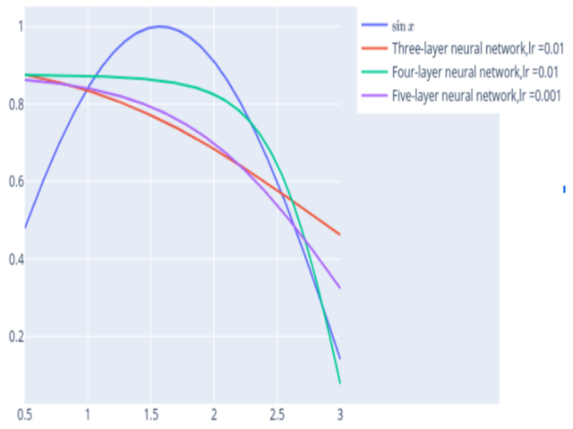

FIG. 9: Prediction of $f(x) = sin(x)$

2. *Predicting binary addition(+)*

Using the same NN, an adding operation was trained and successfully predicted the add(+) operation; see Figure 10.

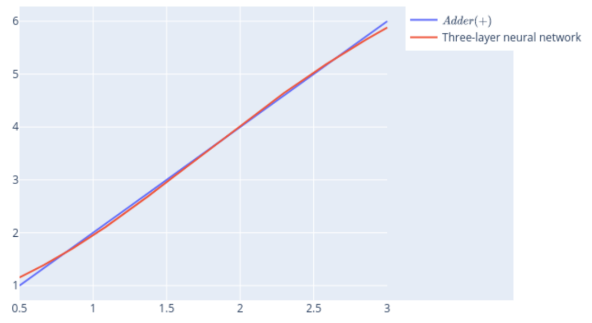

FIG. 10: Adder operation predicted by modified ANN

VII. POSSIBILITY OF INCLUSION OF ANN IN REINFORCEMENT LEARNING (RL)

I have already answered this section in the Q&A report as following.

A. What is RL ?

Reinforcement Learning is a whole different regime in machine learning. I will try to explain it to the best of my knowledge and relate to the project.
Before combining the RL and ANN. Let's first understand what exactly RL is and terms related to RL.
Reinforcement Learning (RL) is like teaching NN to learn from its output. In each step of a sequence, NN explores an environment and gets rewards or penalties based on its outputs. It figures out the best path to get more rewards over time. For example, it's how a robot learns to walk, or a game-playing AI learns to win by playing and getting better each time it makes a move. In addition, recently, it is possible to solve differential equations and prove theorems using RL.

- **Control Optimization Problem**:
  - RL is applied to recognize the best action in each state encountered by a system to optimize an objective function.
  - Objective functions could include metrics like average reward per unit time or total discounted reward within a specified time horizon.
- **Usage Context of RL**:
  - RL is preferred when systems exhibit an extensive range of states, often exceeding a thousand, making closed-form analysis impractical.
  - Systems with intricate and complex stochastic structures find RL beneficial due to its adaptability in handling non-trivial dynamics.
- **Alternatives Based on Problem Complexity**:
  - For problems with a limited number of states and simpler underlying stochastic structures, dynamic programming approaches are viable.
  - Dynamic programming techniques are suitable when the system's state space is relatively small and the underlying randomness is less complex.
- **Q-Networks**:
  - In reinforcement learning, a Q-network refers to an artificial neural network (ANN) used to approximate the action-value function (Q-function). The Q-function, denoted as Q(s, a), represents the expected cumulative future rewards of taking a specific action (a) in a given state (s) and following a particular policy thereafter.

As we have seen, there are multiple parts in RL. So, where can ANN be used in this method? The key idea is that saving each Q-value separately becomes impractical when there are many s-a (state-action) pairs. So, it's smarter to group Q-values for a single action within an **Artificial Neural Network**. When you need a Q-value, you can get it from ANN. To update a Q-value, the new value changes the neural network directly. This is how we can employ ANN in RL.

VIII. CONCLUSION

In this report, an ANN was developed from scratch with only minimal use of the Jax library. From this project, I gained basic knowledge about neural networks. In addition, all the exercises done in this project could be done pretty easily with advanced machine-learning libraries like Pytorch and TensorFlow. However, the objective of this report was to gain a deeper insight into artificial neural networks. In the end, we explored the possibility of RL using ANN.

IX. ACKNOWLEDGMENT

I would like to express my sincere gratitude to the course instructor **Dr. Colin Benjamin** for his invaluable guidance and teaching throughout the semester. His dedication to teaching and explaining the intricacies of numerical Python has been instrumental in shaping my understanding of computational physics.

[1] S. Lek and Y.S. Park. *Artificial Neural Networks*, page 237–245. Elsevier, 2008.

[2] Xianlin Wang, Yuqing Liu, and Haohui Xin. Bond strength prediction of concrete-encased steel structures using hybrid machine learning method. *Structures*, 32:2279–2292, August 2021.

[3] Amanatulla. Vanishing gradient problem in deep learning: Understanding intuition and solutions. *Medium*, 2023.