Neural Networks for Newbies

Jared Orihuela Contreras
Universidad Nacional de Ingeniería
Lima, Perú
jared.orihuela.c@uni.pe

1 What is an Artificial Neural Network?

While you are here reading this paper, it is assumed that you already know what an artificial neural network is, but if not, I will be happy to explain it to you: Imagine you want to teach a computer to perform a specific task, such as distinguishing between dogs and cats. To achieve this, you need a model that can learn to identify the characteristics of a cat and distinguish it from other things. However, there is no algorithm that can determine this, as there are many factors involved. This is where an artificial neural network comes into play. Think of it as a simulation of how the human brain works but on a computer, which instead of performing electrochemical processes uses mathematical elements such as the concept of derivatives and their relationship with optimization, matrix multiplications, and functions that help us determine how accurate our computer is when performing the task we propose. If we look at the functioning of a brain from this mathematical perspective, then we can mimic it using the tools and concepts that we can understand.

2 Key Concepts

2.1 Artificial Neuron

An artificial neuron is a simple processing unit that receives one or more inputs, performs a mathematical operation, and produces an output. Imagine it as a small list of numbers (weights, a concept I will explain later). These stored numbers are the coefficients needed to perform an operation analogous to synapses. Subsequently, the result of this operation will be used as a parameter for an **activation function**.

2.2 Activation Function

An activation function is a mathematical function that takes one or more inputs (usually the weighted sum of inputs and a bias) and returns an output. This output is then used as input for the next **Neural Layer** in a neural network.

2.3 Neural Layer

In a neural network, a layer is a set of neurons that perform operations in parallel. The layers are interconnected, and each layer takes as input the outputs of the previous layer to produce its own output. Formally, a neural network is described as a sequence of layers $\{L_1, L_2, \ldots, L_n\}$, where L_i represents the i-th layer of the network.

2.3.1 Classification of Layers

Layers in a neural network can be classified into three main types:

1. **Input Layer**: It is the first layer of the network and directly receives the features from the dataset.

Preprint. Under review.

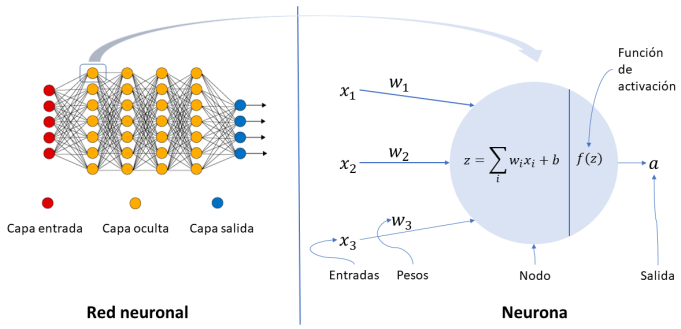

Figure 1: Artificial Neuron Structure.
[4]

2. **Hidden Layers**: These are the layers between the input layer and the output layer. They perform most of the calculations required by the network.
3. **Output Layer**: This is the last layer and produces the output, which is the goal of the network's learning, such as classification or prediction.

2.3.2 Characteristics of a Neural Layer

A neural layer consists of several key characteristics that define its behavior and contribute to the overall functioning of the neural network:

1. **Weights**
   Weights W are trainable parameters of the network that determine the relative importance of each input to a particular neuron. Mathematically, these are represented as a matrix $W^{(l)}$ for layer l.
2. **Biases**
   Biases b are also trainable parameters that allow a neuron to have some flexibility when activating. Formally, these are represented as a vector $b^{(l)}$ for layer l.
3. **Weight Initialization**
   Proper weight initialization is crucial for the effective training of a neural network. There are various methods for weight initialization, such as Xavier initialization and He initialization, which aim to maintain the variances of activations across layers.

2.4 Gradient Descent Algorithm

The gradient descent algorithm is an iterative optimization method used to find the local minimum of a differentiable function. The goal is to iteratively update the parameters θ of a cost function $J(\theta)$ following the equation:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla J(\theta)$$

Here, α is the learning rate, and $\nabla J(\theta)$ is the gradient of the cost function with respect to the parameters θ.

2.5 Stochastic Gradient Descent

SGD updates parameters using a single training example at a time, offering faster convergence of the cost function.

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla J(\theta; x_i, y_i)$$

In Stochastic Gradient Descent (SGD), the gradient of the cost function $J(\theta)$ is estimated using a random single data point (x_i, y_i) in each iteration. This gradient estimation is denoted as $\nabla J(\theta; x_i, y_i)$.

The property of unbiasedness means that the expected value of this estimation is equal to the true gradient computed over the entire dataset:

$$\mathbb{E}[\nabla J(\theta; x_i, y_i)] = \frac{1}{N} \sum_{i=1}^{N} \nabla J(\theta; x_i, y_i)$$

A More Intuitive Explanation Imagine you are at the top of a hill, and your goal is to reach the lowest point (the valley) as quickly as possible, but there's a rule: you can only take small steps. Moreover, it's very foggy, and you can't see the entire hill at once.

What you do is feel the terrain with your feet to understand in which direction the hill is sloping down most steeply at your current position. Then, you take a small step in that direction. You repeat this process: you feel the terrain and take a step in the direction where you feel the hill is sloping down faster. Eventually, you reach the lowest point of the terrain, which is the local minimum.

This process is analogous to the gradient descent algorithm. The model parameters are like your position on the hill, and the goal is to find the position (set of parameters) that minimizes the cost function (takes you to the lowest point).

2.6 Cost Function

Cost functions, also known as loss functions, measure the discrepancy between the model's predictions and the true labels in the dataset. These functions are fundamental to the optimization process, as they are minimized during training.

2.7 Training Algorithm

2.7.1 Forward Propagation

Forward propagation is the process by which the neural network performs a series of calculations to generate an output. Formally, it computes the activations of all neurons in each layer l, from the input layer L_1 to the output layer L_n.

$$a^{(l)} = f^{(l)}(z^{(l)}), \quad z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

Where $f^{(l)}$ is the activation function, $W^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l, respectively.

2.7.2 Backpropagation

The backpropagation algorithm is the method used to update the parameters of the neural network by minimizing the error. In this process, the gradient of the error with respect to each parameter θ is calculated using the chain rule:

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial \theta}$$

2.7.3 Parameter Update

After calculating the gradients through backpropagation, the parameters are updated using the gradient descent algorithm:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \frac{\partial J}{\partial \theta}$$

3 MNIST Dataset: Handwritten Numbers Recognition

The MNIST dataset is a widely recognized and frequently used dataset in the field of machine learning and computer vision. It serves as a benchmark for testing various image classification algorithms, particularly for handwritten digit recognition. The name "MNIST" stands for Modified National Institute of Standards and Technology database.

3.1 Dataset Overview

The MNIST dataset consists of a collection of 28x28 pixel grayscale images of handwritten digits (0 through 9). It contains a total of 60,000 training images and 10,000 testing images, making it a valuable resource for training and evaluating machine learning models.

Each image in the dataset is associated with a corresponding label, indicating the digit represented in the image. The goal is to develop models that can correctly classify these images into their respective digit classes.

3.2 Importance and Significance

MNIST is considered a fundamental dataset in the field of machine learning for several reasons:

- Simplicity: The dataset is relatively small and straightforward, making it accessible for experimentation and learning.
- Benchmark: It serves as a benchmark for evaluating the performance of new machine learning techniques and algorithms, especially in the early stages of research.
- Education: MNIST is commonly used as a teaching tool for introducing students to image classification and deep learning concepts.
- Baseline: Achieving high accuracy on MNIST is a common starting point for assessing the capabilities of a new machine learning model or architecture.

3.3 Challenges and Variants

While MNIST remains a valuable resource, it is not without limitations. Due to its simplicity, many machine learning models can achieve near-perfect accuracy on this dataset. Researchers have since developed more challenging datasets to push the boundaries of image classification, such as CIFAR-10, CIFAR-100, and ImageNet.

Furthermore, variations of MNIST have been created to address specific challenges, such as the Fashion MNIST dataset, which focuses on classifying clothing items, and the EMNIST dataset, which includes handwritten letters in addition to digits.

3.4 Use Cases

The MNIST dataset[1] has been employed in a wide range of applications, including:

- Digit Recognition: The primary use case is recognizing and classifying handwritten digits, making it relevant for tasks like postal code recognition and automated form processing.
- Educational Tools: MNIST is commonly used in tutorials, courses, and books to teach image classification, machine learning, and deep learning concepts.
- Research and Development: It serves as a starting point for researchers to develop and test new neural network architectures and optimization techniques.

In summary, the MNIST dataset is a fundamental resource that has played a significant role in advancing the field of machine learning and computer vision. While it may be considered basic by today's standards, its simplicity and historical significance make it an essential starting point for many machine learning enthusiasts and researchers.

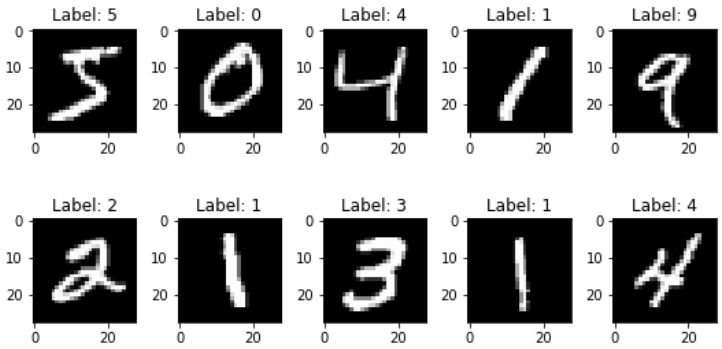

Figure 2: Samples of the data set.
[2]

4 Process

Having grasped the essential concepts required for understanding, it's time to guide the reader through the utilization of functions, algorithms, and data.

Next, we will explain the process of preparing data for training a machine learning model

4.1 Preparing Data

Prior to training a machine learning model, data must be adequately prepared. This involves several crucial steps:

Dataset Acquisition Acquire the dataset, typically from a pre-compiled source or through collection. For instance, the MNIST dataset is commonly used for image classification tasks.

Feature Scaling Normalize the feature values to aid in convergence during training.

$$X \leftarrow \frac{\text{Pixel values}}{255}$$

Label Encoding Convert categorical labels into a one-hot encoded format suitable for the network.

For each label y in labels:
One-hot encode y into y_{encoded}

Data Splitting Divide the dataset into distinct training and validation sets for the training and evaluation phases.

4.2 Model Architecture

With the data ready, the next step is to construct the neural network's architecture.

Layer Initialization Initialize the network with the appropriate number of layers and neurons, using techniques like He initialization [3] for layers with ReLU activations.

Activation Functions Implement non-linear activation functions such as ReLU for hidden layers to allow the model to learn complex patterns.

Loss Function Employ a combination of softmax and cross-entropy loss for classification tasks.

4.3 Training Process

The model is trained iteratively through the following steps:

Forward Propagation For each layer, compute the following:

$$\text{output} \leftarrow \text{activation}(\text{input} \cdot \text{weights} + \text{biases})$$

Loss Computation Utilize the cross-entropy loss function for a multi-class classification:

$$L \leftarrow -\sum(y_{\text{true}} \cdot \log(y_{\text{pred}}))$$

Backpropagation Calculate gradients for the loss function with respect to each parameter:

$$\begin{aligned} \text{error} &\leftarrow y_{\text{pred}} - y_{\text{true}} \\ \text{gradient}_{\text{weights}} &\leftarrow \text{input}^{\text{T}} \cdot \text{error} \\ \text{gradient}_{\text{biases}} &\leftarrow \sum(\text{error}) \end{aligned}$$

Weight Updates Adjust the weights using the gradients and a learning rate:

$$\begin{aligned} \text{weights} &\leftarrow \text{weights} - \text{learning rate} \cdot \text{gradient}_{\text{weights}} \\ \text{biases} &\leftarrow \text{biases} - \text{learning rate} \cdot \text{gradient}_{\text{biases}} \end{aligned}$$

4.4 Model Evaluation

After training is complete, evaluate the model's performance on a separate test set to ensure it generalizes well to unseen data.

4.5 Didactic Additions

Forward Propagation Formula For a given input vector x_i and layer l with weights W_l and biases b_l, the layer output a_l is given by:

$$a_l \leftarrow \sigma(W_l \cdot x_i + b_l)$$

where σ represents the activation function, such as ReLU or softmax.

Backpropagation Error Formula The error attributed to each neuron (imputed error) in the output layer is the derivative of the loss function with respect to the neuron's activation:

$$\delta^L \leftarrow \frac{\partial L}{\partial a^L} \odot \sigma'(z^L)$$

For preceding layers, the error is propagated backwards using the chain rule:

$$\delta^l \leftarrow ((W^{l+1})^{\text{T}} \cdot \delta^{l+1}) \odot \sigma'(z^l)$$

where L denotes the last layer, δ^l is the error of layer l, a^L is the activation of the last layer, z^L is the weighted input to the last layer, and σ' denotes the derivative of the activation function o new, unseen data. function. These formulas represent the essence of the learning process in neural networks, allowing the network to adjust its weights to minimize the loss function, hence improving its predictions over time.

Updating Weights and Biases The updates for weights and biases using the gradients and the learning rate are mathematically represented as follows:

$$\begin{aligned} W_l &\leftarrow W_l - \eta \cdot \frac{\partial L}{\partial W_l} \\ b_l &\leftarrow b_l - \eta \cdot \frac{\partial L}{\partial b_l} \end{aligned}$$

where η is the learning rate, W_l and b_l are the weights and biases of layer l, respectively, and $\frac{\partial L}{\partial W_l}$ and $\frac{\partial L}{\partial b_l}$ are the gradients of the loss function with respect to the weights and biases.

Learning Rate The learning rate η is a hyperparameter that controls the size of the step the algorithm takes in the direction of the local gradient. A suitable value for η is critical for the convergence of the algorithm and is often determined through experimentation.

[1]

5 Appendix

5.1 Activation Functions

5.1.1 ReLU (Rectified Linear Unit)

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

5.1.2 Leaky ReLU

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

Where α is a small constant, like 0.01.

5.1.3 Sigmoid

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

5.1.4 Hyperbolic Tangent (Tanh)

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

5.1.5 Softmax

Given a vector $\mathbf{x}$ of dimension K, the Softmax function is defined as:

$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

for $i = 1, \ldots, K$.

5.1.6 Swish

$$f(x) = x \cdot \sigma(x)$$

Where $\sigma(x)$ is the sigmoid function.

5.2 Cost Function

5.2.1 Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Where N is the number of samples, y_i are the true labels, and $\hat{y}_i$ are the model predictions.

5.2.2 Categorical Cross-Entropy

$$\text{CCE} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Where C is the number of classes, y_i is the true label for class i, and $\hat{y}_i$ is the predicted probability for class i.

[1] If you want to see how get this formulas, I recommend that readers visit `https://towardsdatascience.com/deriving-the-backpropagation-equations-from-scratch-part-1-343b300c585a`, a friendly and visual explanation awaits you.

5.2.3 Binary Cross-Entropy

$$\text{BCE} = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

This is a special case of Categorical Cross-Entropy for binary classification problems.

5.2.4 Hinge Loss

$$\text{Hinge Loss} = \max(0, 1 - y \cdot \hat{y})$$

Primarily used in Support Vector Machines (SVM) but can also be used in neural networks for classification problems.

5.2.5 Kullback-Leibler Divergence

$$\text{KL Divergence} = \sum_{i=1}^{C} y_i \log\left(\frac{y_i}{\hat{y}_i}\right)$$

This function measures how one probability distribution differs from a second reference distribution.

5.2.6 Calculating Imputed Error for Hidden and Output Layers

In backpropagation, the imputed error for the output and hidden layers is calculated to update the weights and biases in the network.

For the output layer, the imputed error is the difference between the predicted output $\hat{y}$ and the actual target y:

$$\delta^{\text{out}} = \hat{y} - y$$

This error is then used to calculate the gradient for the weights of the output layer.

For hidden layers, the imputed error is a bit more complex as it involves the errors propagated from the layers ahead:

$$\delta^l = (W^{l+1})^{\text{T}} \delta^{l+1} \odot \sigma'(z^l)$$

where δ^l is the error of the l-th layer, δ^{l+1} is the error of the layer ahead, W^{l+1} are the weights of the layer ahead, σ' is the derivative of the activation function, and z^l is the input to the l-th layer's activation function.

5.2.7 Imputed Error for Biases

The error for biases in each layer can be directly taken from the imputed error of that layer as biases are considered as weights with an input of 1:

$$\delta^{\text{bias},l} = \delta^l$$

This error is then used to update the biases similarly to how weights are updated.

5.2.8 Gradient Descent Update Rules

The weights and biases are updated in the opposite direction of the gradient:

$$W_l = W_l - \eta \cdot \frac{\partial L}{\partial W_l} = W_l - \eta \cdot \delta^l \cdot a^{l-1}$$

$$b_l = b_l - \eta \cdot \frac{\partial L}{\partial b_l} = b_l - \eta \cdot \delta^{\text{bias},l}$$

Here, η is the learning rate, δ^l is the error of the l-th layer, and a^{l-1} is the activation from the previous layer.

5.2.9 Variations of the Gradient Descent Algorithm

There are several variations of the gradient descent algorithm designed to improve the efficiency and accuracy of the optimization process:

1. **Batch Gradient Descent**
   In this method, each parameter update is made after computing the gradient of the error over the entire dataset. It is computationally expensive for large datasets.

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \sum_{i=1}^{N} \nabla J(\theta; x_i, y_i)$$

2. **Stochastic Gradient Descent (SGD)**
   Unlike batch gradient descent, SGD updates parameters using a single training example at a time, offering faster convergence of the cost function.

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla J(\theta; x_i, y_i)$$

   In Stochastic Gradient Descent (SGD), the gradient of the cost function $J(\theta)$ is estimated using a random single data point (x_i, y_i) in each iteration. This gradient estimation is denoted as $\nabla J(\theta; x_i, y_i)$.

   The property of unbiasedness means that the expected value of this estimation is equal to the true gradient computed over the entire dataset:

$$\mathbb{E}[\nabla J(\theta; x_i, y_i)] = \frac{1}{N} \sum_{i=1}^{N} \nabla J(\theta; x_i, y_i)$$

3. **Mini-Batch Gradient Descent**
   This variant is a middle ground between batch gradient descent and SGD. It updates parameters using a mini-batch of m training examples.

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \sum_{i=1}^{m} \nabla J(\theta; x_i, y_i)$$

4. **Adaptive Methods**
   There are methods like Adagrad, Adadelta, RMSprop, and Adam that adjust the learning rate during training. These methods can be more efficient and converge faster than traditional methods.

5.2.10 Regularization Terms

Regularization can be added to the loss function to prevent overfitting by penalizing large weights. Common regularization terms include L1 and L2 norms:

$$L1 : \text{Regularization} = \lambda \sum_{i} |w_i|$$

$$L2 : \text{Regularization} = \frac{1}{2} \lambda \sum_{i} w_i^2$$

where λ is the regularization parameter, and w_i represents the weights in the network.

5.2.11 Momentum Update

Momentum can be used to accelerate the training process and to help navigate the parameter space more effectively:

$$v_t = \gamma v_{t-1} + \eta \cdot \frac{\partial L}{\partial W}$$

$$W = W - v_t$$

where v_t is the current velocity, γ is the momentum coefficient, η is the learning rate, and $\frac{\partial L}{\partial W}$ is the gradient of the loss function with respect to the weights.

These formulas constitute a fundamental part of the algorithmic structure of training neural networks and allow for the practical application of the theoretical principles outlined earlier in the document.

5.3 Results

Now that we have obtained all the necessary concepts and ideas, it is essential to see the fruits of our work. The author has implemented two codes that contain some clever ideas and different parameters, which will yield distinct levels of accuracy and precision.

5.4 Using Classic Gradient Descent

5.4.1 Characteristics

- **Layer Initialization:** Utilizes Kaiming (He) initialization[3] for weights, suitable for ReLU activations.
- **Activation Functions:** Employs ReLU for hidden layers and Softmax for the output layer.
- **Data Processing:** Implements data normalization and one-hot encoding for labels.
- **Training Parameters:**
  - *Epochs*: Defines the number of complete passes through the dataset.
  - *Learning Rate*: A crucial value affecting convergence speed and quality.
- **Loss Calculation:** Uses cross-entropy loss for evaluating model predictions.
- **Backpropagation:** Adjusts weights and biases based on the loss gradient.

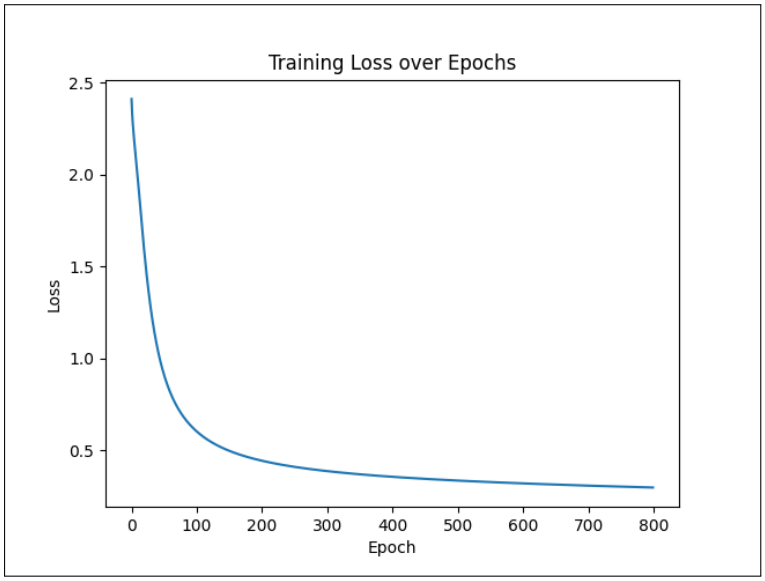

Figure 3: Loss VS Epoch figure.

History	
Parameter	Value
Number of epochs	800
Learning rate	10^{-6}
Input dimension	(10^6 , 784)

5.5 Using Stochastic Gradient Descent

To use the Stochastic method of Gradient Descent, we have to fix and modify some main parameters.

5.5.1 Characteristics

- **Learning Rate:** Set to 0.01, crucial for determining the step size in parameter updates.
- **Number of Epochs:** Set to 1, indicating the model processes each training sample once.
- **Data Normalization:** Inputs are normalized by dividing by 255.
- **One-Hot Encoding:** Labels are converted to one-hot representations for classification.
- **Activation Functions:** ReLU for hidden layers and Softmax for output layer.
- **Loss Function:** Cross-entropy used to compute error for each training example.
- **Training Approach:**
  - Forward propagation and loss calculation are performed for each individual training sample.
  - Backpropagation and parameter updates are conducted after each sample.

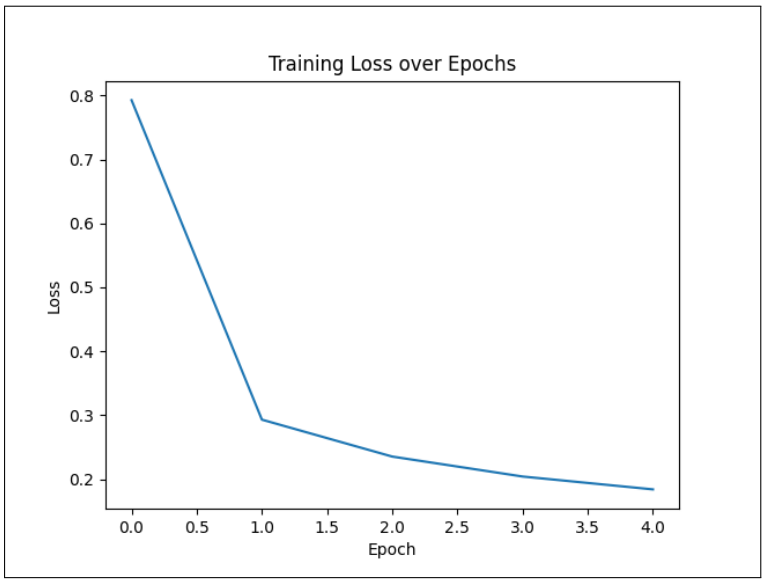

Figure 4: Loss VS Epoch figure.

History	
Parameter	Value
Number of epochs	5
Learning rate	10^{-2}
Input dimension	(1 , 784)

6 Comparisson with Frameworks

6.0.1 Characteristics

- **Learning Rate:** Set to 0.01, crucial for determining the step size in parameter updates.
- **Number of Epochs:** Set to 1, indicating the model processes each training sample once.
- **Data Normalization:** Inputs are normalized by dividing by 255.

- **One-Hot Encoding:** Labels are converted to one-hot representations for classification.
- **Activation Functions:** ReLU for hidden layers and Softmax for output layer.
- **Loss Function:** Cross-entropy used to compute error for each training example.
- **Training Approach:**
  - Forward propagation and loss calculation are performed for each individual training sample.
  - Backpropagation and parameter updates are conducted after each sample.

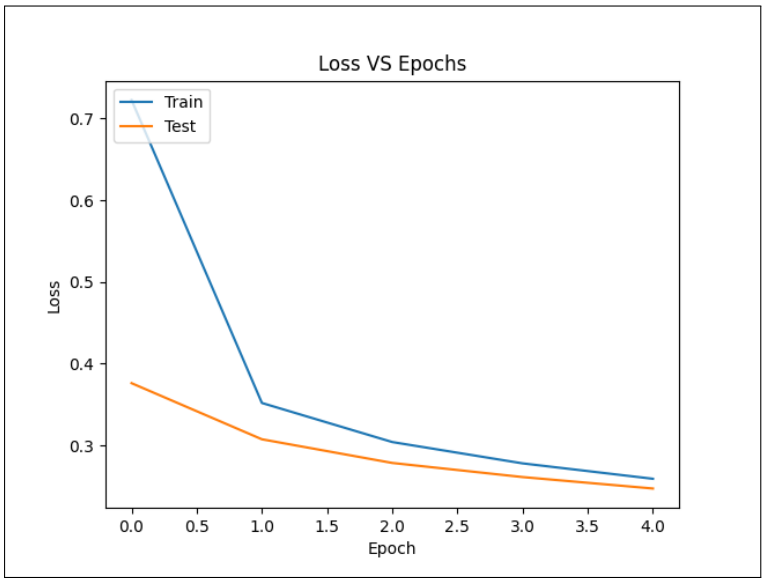

Figure 5: Loss VS Epoch figure.

History	
Parameter	Value
Number of epochs	5
Learning rate	10^{-2}
Input dimension	(1 , 784)

7 Conclusion

The above steps, when implemented in a systematic manner, form the pipeline for training a neural network. The process starts with pre-processing the data, involves iterative training cycles (epochs) of forward and backpropagation, and concludes with evaluating the model's performance. The model's efficacy is typically gauged by its accuracy on the validation and test sets, which reflect its ability to generalize from the training data to unseen data.

References

[1] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[2] Daniel Etzold. *MNIST Dataset of Handwritten Digits*. https://medium.com/mlearning-ai/mnist-dataset-of-handwritten-digits-f8cf28edafe. Last accessed: Jan 12, 2022.

[3] Siddharth Krishna Kumar. “On weight initialization in deep neural networks”. In: *arXiv preprint arXiv:1704.08863* (2017).

[4] Jahaziel Ponce. *Conoce qué son las funciones de activación y cómo puedes crear tu función de activación usando Python, R y Tensorflow*. Blog personal sobre Inteligencia Artificial. 2021. URL: https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tensorflow/ (visited on 11/12/2023).