**ISM6561 Deep Learning Research Assignment**

**Exploring Initialization Methods and Cost Functions**

**By Guardians Of The Galaxy**

**Research Members:** Kathryn Burkhart, Debjani Sarma, Prince Praveen

Date: 02/22/2025

**1 Experiment Setup**

In this experiment, we are investigating the impact of different weight initialization methods and cost functions on the training performance of a simple neural network. We will conduct the study with various weight initialization strategies and assess their effectiveness using two non-trivial cost functions. The primary goal of this experiment is to understand how different initialization techniques influence the convergence behavior, training stability, and final performance metrics of the model.

**1.1 Dataset**

For this experiment, we use the well-known **Iris dataset** from the UCI Machine Learning Repository. The dataset consists of 150 samples of iris flowers, with each sample having 4 features:

* Sepal Length
* Sepal Width
* Petal Length
* Petal Width

Each sample also has a corresponding target class label (species). In this experiment, we limit our dataset to a binary classification task, where we select only two classes of the Iris species: **Setosa** and **Versicolor**. These two classes will be used for the classification task, resulting in a binary classification problem with **two output classes**.

**1.2 Neural Network Model and Architecture**

For this experiment, we utilize a **simple neural network architecture** with the following layers:

* **Input Layer:** The input layer consists of 4 neurons, one for each feature of the iris flowers (sepal length, sepal width, petal length, and petal width).
* **Hidden Layer:** A single hidden layer with 10 neurons. This hidden layer is followed by a **ReLU activation function** to introduce non-linearity into the model.
* **Output Layer:** The output layer consists of 2 neurons corresponding to the two classes of the binary classification task (Setosa and Versicolor). A **softmax function** is used in the output layer to convert the raw scores into probabilities.

The choice of this architecture is based on its simplicity and efficiency for solving small-scale binary classification problems. The **ReLU activation function** is used in the hidden layer as it is commonly known to help with avoiding the vanishing gradient problem and works well in practice for most tasks. Additionally, the network is small, which is appropriate for the Iris dataset due to the limited number of features and relatively simple decision boundary between the two classes. For training, we use **Adam optimizer** with a learning rate of 0.01 and experiment with two different loss functions: **Cross Entropy Loss** and **Negative Log Likelihood Loss**.

**2 Weight Initialization Techniques**

In this experiment, we implemented and evaluated three distinct **weight initialization strategies** to understand their impact on the performance of the neural network. Weight initialization plays a crucial role in the training of deep neural networks, as it helps ensure that the network starts from a good point in the parameter space and converges effectively during training

**2.1 Constant Initialization**

In **Constant Initialization**, all weights in the neural network are initialized to the same constant value, often a small value. This method is simple and ensures uniform starting values for all weights, but it might lead to poor performance in practice due to the lack of diversity in initial weight values.

**Mathematical Formulation:** Let the weight matrix W of layer l be initialized to a constant value c:



**Parameters Used:**

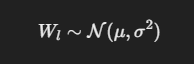
* **Constant value**: In this experiment, we initialized the weights to c=0.5 (arbitrary choice) to avoid starting with zeros which would lead to symmetry problems where all neurons in a layer learn the same features.
* **Biases**: We initialized all biases to 0.

The constant initialization method is easy to implement, and while it may not always be the best choice for training deep networks, it provides a baseline for comparison. It is useful to check whether more advanced initialization strategies improve training performance or not

**2.2 Normal (Gaussian) Initialization**

In **Normal Initialization**, the weights are initialized by sampling from a **Gaussian distribution** with a specified mean and standard deviation. This initialization technique helps break symmetry and provides more diversity in the starting weights. The key idea is to provide randomness in the weights, which can encourage the network to explore a broader solution space.

**Mathematical Formulation:** The weights are initialized from a normal distribution with mean **μ** and standard deviation **σ**:



**Parameters Used:**

* **Mean** μ=0: A zero-centered distribution, so the weights will be distributed symmetrically around 0.
* **Standard Deviation** σ=0.01: A small value for the standard deviation ensures that the weights are not too large at the start of training, which could lead to large gradients.

Normal initialization introduces randomness in a controlled manner, helping to break symmetry between neurons in the network.

**2.3 Xavier (Glorot) Initialization**

**Xavier Initialization**, also known as **Glorot Initialization**, is designed to maintain the variance of activations across layers during forward and backward passes. This method adjusts the weight distribution based on the number of input and output neurons of each layer. Xavier initialization is particularly useful for layers using the **tanh** or **sigmoid** activation functions, as it helps avoid the vanishing gradient problem by keeping the variance of the gradients from becoming too small.

**Mathematical Formulation:** For Xavier normal initialization, the weights are sampled from a normal distribution:



**Parameters Used:**

* **Mean** μ=0 (centered around 0).
* **Variance** is determined by the formula above, ensuring the weights are scaled according to the number of input and output neurons.
* **Biases** are initialized to 0.

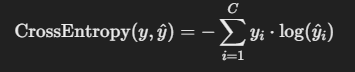
Xavier initialization is widely used for deep networks, especially when using activation functions like **tanh** or **sigmoid**, as it ensures the variance of the activations is controlled. This method helps in avoiding issues like vanishing or exploding gradients, which are particularly problematic when training deep networks.

**3 Cost Functions**

In this experiment, we implemented two different cost functions to evaluate the performance of our neural network models: **Cross-Entropy Loss** and **Negative Log-Likelihood (NLL) Loss**.

**3.1 Cross Entropy Loss**

Cross-Entropy Loss is commonly used for classification problems where the output of the network is a probability distribution across multiple classes. For a binary or multi-class classification problem, the Cross-Entropy Loss between the true labels **y** and the predicted probabilities **y^**​ is calculated as:



Where:

* C is the number of classes.
* yi​ is the true probability distribution (usually one-hot encoded), where yi​=1 for the true class and 0 for others.
* y^i is the predicted probability for class i.
* The sum is taken over all classes.

**Cross-Entropy Loss** is ideal for classification problems, especially when the output is a probability distribution. It measures the dissimilarity between the true distribution **y** and the predicted distribution **y^**​. Minimizing this loss encourages the model to assign high probability to the true class and low probability to the other classes.

**3.2 Negative Log-Likelihood (NLL) Loss**

The Negative Log-Likelihood (NLL) loss is another widely used loss function in classification tasks, especially when the output of the model is a probability distribution. The NLL loss is defined as:



Where:

* y is the true class label (assumed to be a one-hot encoded vector).
* y^​k​ is the predicted probability for the true class k.

**NLL Loss** is useful when the model's output is a log-probability, which can occur when we use the softmax activation combined with a logarithmic transformation of the predictions. It is directly related to maximizing the likelihood of the true class and is widely used in classification problems where the output is categorical.

**4 Experimentation & Analysis**

**4.1 Experimental Design**

The network architecture used in each experiment consists of two fully connected layers. The first layer has 4 input features and 10 hidden neurons, while the second layer has 10 hidden neurons and 2 output neurons (corresponding to binary classification). We evaluate the network performance under each of the following combinations:

1. **Constant Initialization + Cross-Entropy Loss**
2. **Constant Initialization + NLL Loss**
3. **Uniform Initialization + Cross-Entropy Loss**
4. **Uniform Initialization + NLL Loss**
5. **Xavier Initialization + Cross-Entropy Loss**
6. **Xavier Initialization + NLL Loss**

We train the network using the **Adam optimizer** and a **learning rate of 0.01**. The number of epochs for each experiment is set to **100** to ensure that the model has enough time to converge. For each combination, we record the **training loss**, **accuracy**, **precision**, **recall**, and **F1 score** as the primary metrics for evaluating performance.

**TO-DOs**

1. Copy Paste the performance graphs and talk about it.
2. Talk about the advantages & potential disadvantages of the selected choices of weight initialization & cost functions.

…(W.I.P)