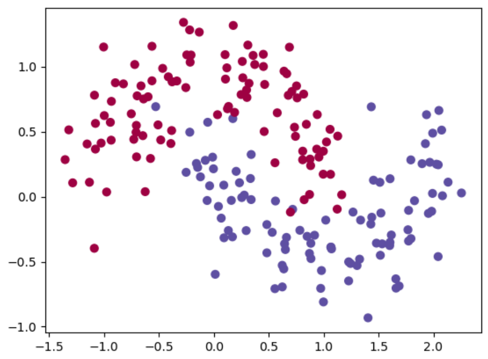
Due 10/10/2024 (11am) ELEC 576

Assignment #1

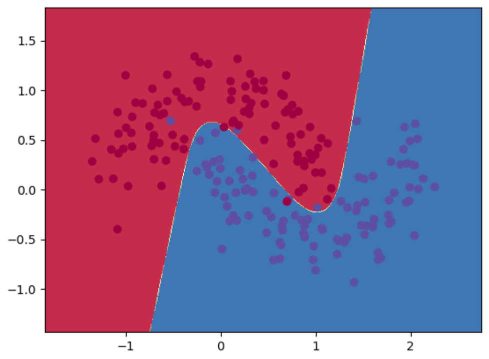
Jiahe Zhang $\|jz185@rice.edu$

1 Backpropagation in a Simple Neural Network

1. Dataset



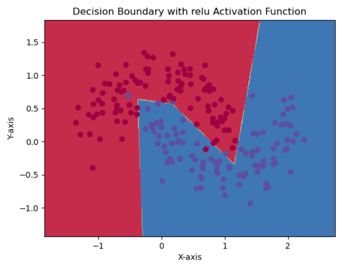
(a) Generated Make-Moons dataset.



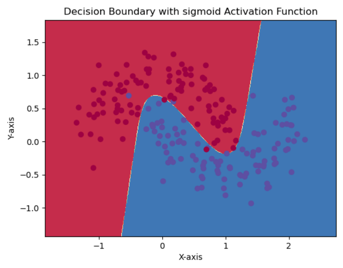
(b) Decision boundary after training the neural network.

Figure 1: Visualizations of the Make-Moons dataset and the trained neural network.

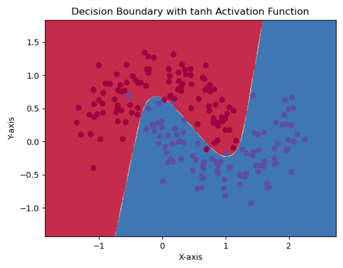
2. Activation Functions



(a) ReLU Activation



(b) Sigmoid Activation



(c) Tanh Activation

Figure 2: Decision boundaries obtained using different activation functions (ReLU, Sigmoid, Tanh).

As shown in Figure 2, different activation functions produce varied decision boundaries for the Make-Moons dataset. The ReLU activation function (2a) tends to produce sharp, straight boundaries due to its linear nature for positive inputs. Sigmoid (2b) and Tanh (2c) activation functions, on the other hand, provide smoother decision boundaries. Tanh gives a more pronounced separation due to its range being symmetric around zero, which helps in centering the data during training.

Derivatives of Activation Functions:

- **Tanh**: The derivative of the Tanh function is given by:

$$f'(z) = 1 - \tanh^2(z) \tag{1}$$

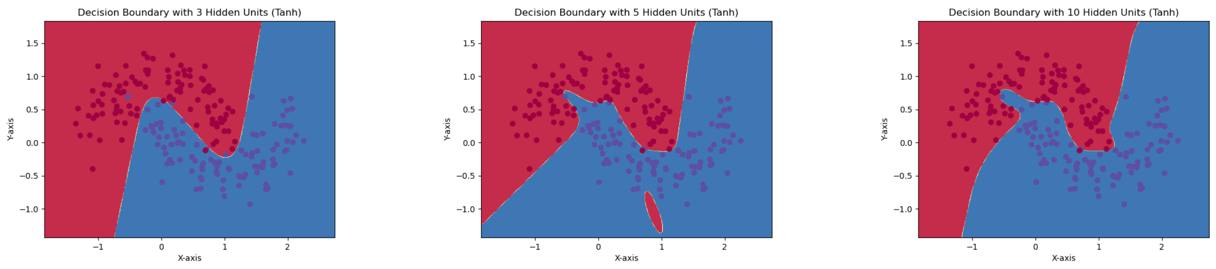
- **Sigmoid**: The derivative of the Sigmoid function is given by:

$$f'(z) = \sigma(z) \cdot (1 - \sigma(z)) \tag{2}$$

  where $\sigma(z)$ is the Sigmoid function.
- **ReLU**: The derivative of the ReLU function is:

$$f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \tag{3}$$

3. Hidden Units



(a) 3 Hidden Units (b) 5 Hidden Units (c) 10 Hidden Units

Figure 3: Decision boundaries with increasing hidden units using Tanh activation.

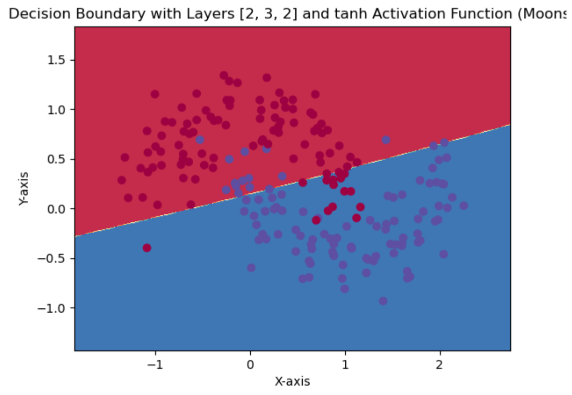
In Figure 3, we see the effect of increasing the number of hidden units in the hidden layer. With 3 hidden units (3a), the decision boundary is relatively simple. As we increase to 5 (3b) and then to 10 hidden units (3c), the model becomes more capable of capturing the complexities in the data, resulting in more intricate decision boundaries. This indicates the model's increasing capacity to overfit to the training data as the number of parameters grows.

4. Training a Deeper Network

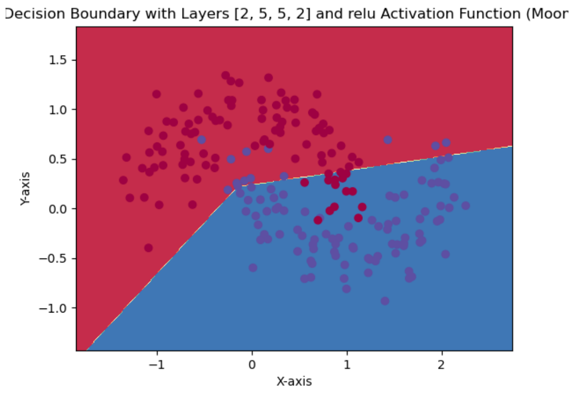
The above figures demonstrate the decision boundaries of deep neural networks trained with different activation functions and configurations.

For the Moons dataset: - **Figure 4a** shows a decision boundary obtained from a 3-layer network using Tanh activation. The boundary is curved and fits well to the distribution of the two classes, showing good non-linearity in the separation. - **Figure 4b** presents the boundary generated with a 4-layer network using ReLU. The boundary is relatively simple with piecewise linear segments, which is consistent with ReLU's behavior in linearly segmenting the input space. - **Figure 4c** demonstrates the decision boundary of a 5-layer network using Sigmoid. The network fits smoothly to the data points, indicating that additional layers provided more representational power, allowing the network to create more complex separations.

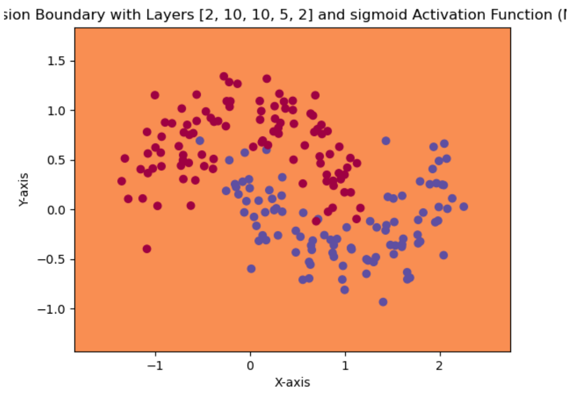
For the Iris dataset: - **Figure 4d** represents the decision boundary from a 3-layer network using Tanh. The boundary shows smoother and more continuous partitioning of the feature space. - **Figure 4e** represents the decision boundary from a 4-layer network using ReLU. The boundary is fairly straightforward, dividing the classes along linear separations. - **Figure 4f** shows the result of using a 5-layer network with Sigmoid. The decision boundary better captures the complexity of the data compared to ReLU, as Sigmoid introduces non-linear separations.



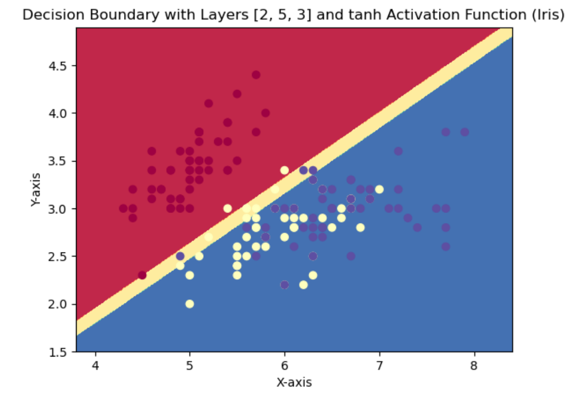
(a) 3-layer network using Tanh on Moons dataset.



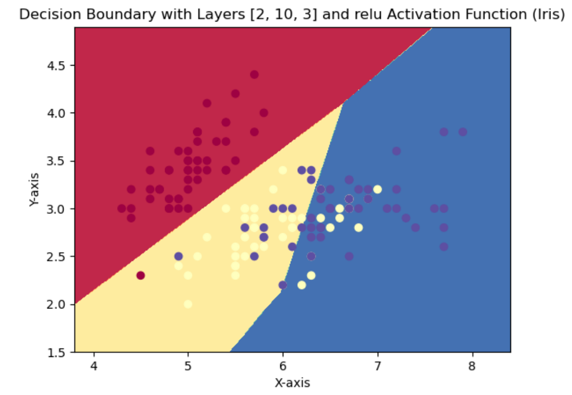
(b) 4-layer network using ReLU on Moons dataset.



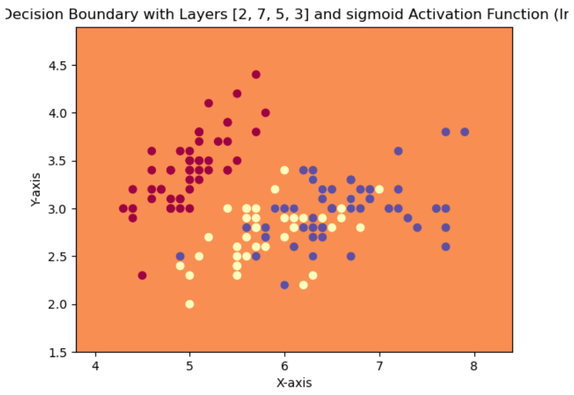
(c) 5-layer network using Sigmoid on Moons dataset.



(d) 3-layer network using Tanh on Iris dataset.



(e) 4-layer network using ReLU on Iris dataset.



(f) 5-layer network using Sigmoid on Iris dataset.

Figure 4: Comparison of decision boundaries obtained by training deep networks with different configurations and activation functions on Moons and Iris datasets.

Overall, the deeper networks and different activation functions provide insights into how network depth and non-linearities affect the model's ability to learn complex decision boundaries. We observe that the Sigmoid and Tanh activations often produce smoother transitions between classes, while ReLU is prone to creating more abrupt, piecewise-linear boundaries. This behavior is consistent across both the Moons and Iris datasets.

5. Derivation of Gradients for Backpropagation For backpropagation, we need to derive the following gradients: $\frac{\partial L}{\partial W_2}$, $\frac{\partial L}{\partial b_2}$, $\frac{\partial L}{\partial W_1}$, and $\frac{\partial L}{\partial b_1}$.

   The loss function is given by:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{i \in C} y_{n,i} \log \hat{y}_{n,i} \tag{4}$$

   The gradients are derived as follows:

   - **Gradient with respect to W_2**:
$$\frac{\partial L}{\partial W_2} = a_1 \cdot (\hat{y} - y)^T \tag{5}$$
   - **Gradient with respect to b_2**:
$$\frac{\partial L}{\partial b_2} = \hat{y} - y \tag{6}$$
   - **Gradient with respect to W_1**:
$$\frac{\partial L}{\partial W_1} = X \cdot \delta_1^T \tag{7}$$
     where δ_1 is the error term for the hidden layer, computed as:
$$\delta_1 = (W_2^T \cdot (\hat{y} - y)) \odot f'(z_1) \tag{8}$$
   - **Gradient with respect to b_1**:
$$\frac{\partial L}{\partial b_1} = \delta_1 \tag{9}$$

   These gradients are used in the backpropagation algorithm to update the weights and biases of the network to minimize the loss function.

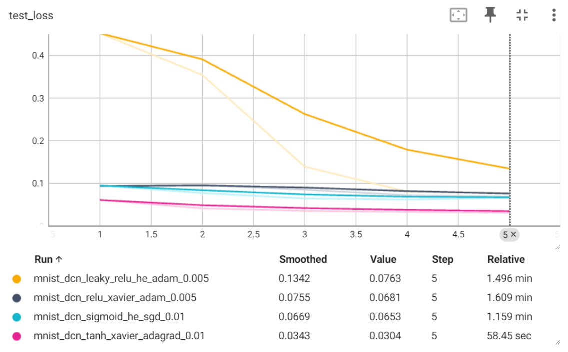
2 Training a Simple Deep Convolutional Network on MNIST

1. Performance Analysis of Training

   In this section, we analyzed the performance of training a simple deep convolutional network (DCN) on the MNIST dataset with four different configurations, including different activation functions, initialization methods, and optimizers. The graphs below represent the test loss, training loss, and test accuracy metrics for each configuration.

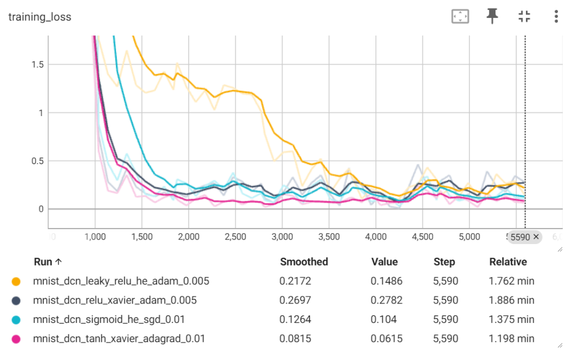
   The configurations tested were:

   - **ReLU with Xavier Initialization and Adam Optimizer (Learning rate: 0.005)**
   - **Sigmoid with He Initialization and SGD Optimizer (Learning rate: 0.01)**
   - **Tanh with Xavier Initialization and Adagrad Optimizer (Learning rate: 0.01)**
   - **Leaky ReLU with He Initialization and Adam Optimizer (Learning rate: 0.005)**



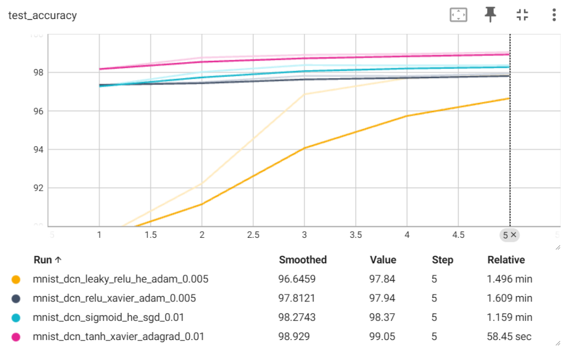
Run ↑	Smoothed	Value	Step	Relative
● mnist_dcn_leaky_relu_he_adam_0.005	0.1342	0.0763	5	1.496 min
● mnist_dcn_relu_xavier_adam_0.005	0.0755	0.0681	5	1.609 min
● mnist_dcn_sigmoid_he_sgd_0.01	0.0669	0.0653	5	1.159 min
● mnist_dcn_tanh_xavier_adagrad_0.01	0.0343	0.0304	5	58.45 sec

(a) Test loss for different configurations.



Run ↑	Smoothed	Value	Step	Relative
● mnist_dcn_leaky_relu_he_adam_0.005	0.2172	0.1486	5,590	1.762 min
● mnist_dcn_relu_xavier_adam_0.005	0.2697	0.2782	5,590	1.886 min
● mnist_dcn_sigmoid_he_sgd_0.01	0.1264	0.104	5,590	1.375 min
● mnist_dcn_tanh_xavier_adagrad_0.01	0.0815	0.0615	5,590	1.198 min

(b) Training loss for different configurations.



Run ↑	Smoothed	Value	Step	Relative
● mnist_dcn_leaky_relu_he_adam_0.005	96.6459	97.84	5	1.496 min
● mnist_dcn_relu_xavier_adam_0.005	97.8121	97.94	5	1.609 min
● mnist_dcn_sigmoid_he_sgd_0.01	98.2743	98.37	5	1.159 min
● mnist_dcn_tanh_xavier_adagrad_0.01	98.929	99.05	5	58.45 sec

(c) Test accuracy for different configurations.

Figure 5: Performance metrics of DCN on MNIST for different activation functions, initialization methods, and optimizers.

The results for each configuration are summarized below:

Test Loss (Figure 5a): The `tanh-xavier-adagrad` configuration (magenta line) showed the lowest final test loss, demonstrating effective generalization. In contrast, `leaky_relu-he-adam` (orange line) started with a significantly higher loss and converged slowly, indicating that leaky ReLU with this learning rate and optimizer was less effective for this particular network architecture and dataset.

Training Loss (Figure 5b): Similar to the test loss, the training loss was also effectively minimized by the `tanh-xavier-adagrad` configuration. ReLU and Sigmoid configurations showed comparable performance in terms of convergence rates, with the training loss stabilizing earlier. The `leaky_relu-he-adam` configuration displayed a slow decrease, with noticeable oscillations. This could imply sensitivity to hyperparameters, or the complexity of learning was not well handled by this particular combination of activation function, initialization, and optimizer.

Test Accuracy (Figure 5c): The `tanh-xavier-adagrad` achieved the highest final test accuracy, reaching around 99%. The sigmoid and ReLU configurations also reached high accuracy levels, slightly above 98%. However, the `leaky_relu-he-adam` configuration started from a much lower accuracy value and exhibited a slower convergence rate compared to the other configurations.

2. Insights from Different Configurations

   - **Activation Functions**: Activation functions play a crucial role in the convergence of a deep neural network. From the results, Tanh and Sigmoid demonstrated smoother learning curves with lower test losses and higher accuracies, suggesting that these activation functions helped the network learn more effectively from the MNIST data. ReLU also performed well, but leaky ReLU exhibited suboptimal convergence behavior. This could be due to insufficient tuning of hyperparameters, such as learning rate, that would suit the behavior of leaky ReLU.

- **Initialization Methods**: Xavier initialization seemed to work well for Tanh and ReLU. In contrast, He initialization performed better with ReLU, which aligns with the common practice, but the performance of He initialization with leaky ReLU was less promising, as observed in the slow convergence in the test and training losses.
- **Optimizers**: The Adagrad optimizer showed consistent results in minimizing both test and training losses while achieving high accuracy. Adam also performed well with ReLU and Sigmoid, though leaky ReLU struggled with Adam in terms of convergence. SGD, in combination with He initialization and Sigmoid activation, showed satisfactory performance, but the convergence was slightly slower compared to Adagrad and Adam.

3. Conclusion

From the above analysis, it is evident that different configurations of activation functions, initialization methods, and optimizers significantly impact the performance of training a deep convolutional network on the MNIST dataset. The `tanh-xavier-adagrad` configuration emerged as the best performing one, showing stable and fast convergence with the highest test accuracy and lowest test loss. However, the right choice of these hyperparameters is highly dependent on the nature of the dataset and the complexity of the task at hand.

It is worth noting that other activation functions (such as ReLU and Sigmoid) also provided good results but required careful tuning of learning rates and optimizers to reach optimal convergence. Additionally, the leaky ReLU configuration, although theoretically helpful in mitigating dying ReLU problems, underperformed due to its sensitivity to learning rate and optimizer settings. This highlights the importance of hyperparameter tuning and testing various configurations to achieve optimal network performance.