**Group 7**

**Alisha D. Rush,**

**Mohammed Farooq Abdullah,**

**Saima Sano,**

**Zaid Tahir Jamil**

**Patricia McManus**

**ITAI-1378**

**Date: 10/03/2024**

**Introduction:**

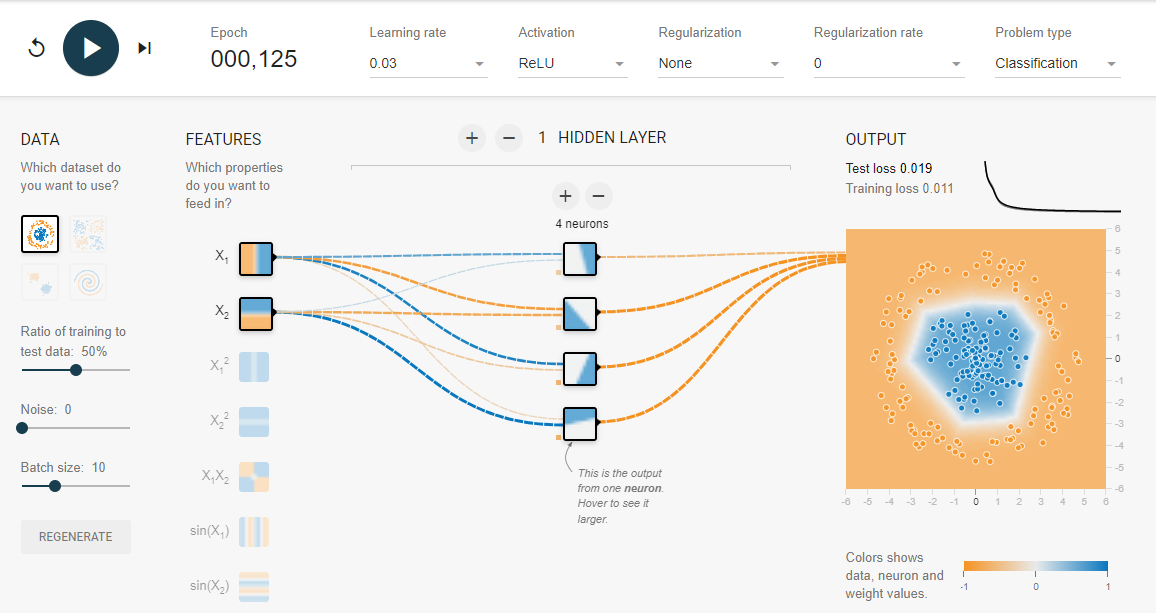
Artificial intelligence is transforming the world, offering unprecedented opportunities to improve various industries. To enhance its capabilities, we can train AI models using diverse approaches. Instead of focusing primarily on programming, we now need to emphasize the development and optimization of algorithms. By training machine learning models with well-designed parameters, we can achieve superior results in real-world applications, making AI more effective and efficient in addressing complex problems. In this assignment, we are working on different parameters to see how generalization and convergence of data is changing by alternating different parameter.

**Task 1 Activation Functions:**

For task 1, we are working on different activation function available in TensorFlow Playground.

**ReLU Activation Function:**

Using the ReLU activation function, neurons are activated only when the output is between 0 and infinity, while outputs less than 0 are deactivated. This behavior often results in faster learning since ReLU does not suffer from gradient saturation, allowing the model to converge more quickly (Sepp, 1998). The accompanying TensorFlow Playground screenshot demonstrates that, with a learning rate of 0.03, ReLU enabled the model to converge within 125 epochs, achieving a test loss of 0.019 and a training loss of 0.011.



**Sigmoid Activation Function:**

When we change the activation function to sigmoid, the model took 333 epochs to converge the data but still the training loss was 0.041 and test loss is 0.052. Since sigmoid suffer from gradient saturation, therefore it took more time to train the model, and the train and test loss number are also greater. This show that this function is not fitting the data well.

A screenshot of a computer

Description automatically generated

**Tanh Activation Function:**

Since, tanh function output values range from -1 to 1, it is very useful in training and learning for smaller networks. The model was able to classify the data in 128 Epochs with train loss of 0.032 and test loss of 0.039. The small gap between training and test loss indicates that tanh function was able to generalize and converge the data efficiently, but the numbers are still higher, so some of the data points were loss during back propagation.

A screenshot of a computer

Description automatically generated

**Interpretation:**

All three activation functions were able to converge the given data and train the model, but ReLU was able to do it quickly with 125 epochs with overall less training and test loss as compared to sigmoid and tanh. Sigmoid activation function took longer to converge the data and was not fitting and generalizing the data as tanh and ReLU. Tanh and ReLU are very close in performance, but tanh had higher training and test loss compared to ReLU. This is due slower gradient updates in areas where tanh has data closer to –1 and 1 as it is not updating weights more effectively in deeper networks. ReLU activation function have high learning speed because it does not consider the neurons with negative output and therefore, it also captures the patterns in the data.

**Conclusion:**

Activation functions play a crucial role in determining the performance of neural networks, and their selection is often influenced by the type of data and the nature of the task. The **Rectified Linear Unit (ReLU)** is widely regarded as a fast and efficient activation function, particularly for linear data or tasks where immediate responses are required. However, for time-dependent tasks, such as those involving sequential or temporal data, **Tanh** and **Sigmoid** functions are often more appropriate.

**Task 2 - Hidden Layer Neurons:**

The task was to experiment with different numbers of neurons and hidden layers while observing their effects on model performance, measured through training and test loss.

**1 hidden layer with 7 neurons**

**Activation: Tanh**

**Learning rate: 0.003**

**Regularization: None**

In this configuration, the network had a single hidden layer with 7 neurons. The performance showed a training loss of 0.002 and a test loss of 0.004, indicating that the network was successfully fitting the data with minimal error. The decision boundary is wrapped closely around the target data points, showing a well-fit model with little to no underfitting. However, without regularization, there was a risk of overfitting as the model complexity was high for the dataset.

A computer screen shot of a diagram

Description automatically generated

**4 hidden layers with neurons (7, 3, 4, 5)**

**Activation: ReLU**

**Learning rate: 0.01**

**Regularization: None**

In this configuration, I increased the number of hidden layers to four, with different numbers of neurons in each. The performance improved dramatically, with both the test and training loss reaching 0.000. While this seems like an ideal outcome, such low losses raise concerns about overfitting. The model was too complex for the data, capturing noise in the training set. The decision boundary became highly intricate, indicating that the network was memorizing the training data instead of generalizing to unseen data. A screenshot of a computer

Description automatically generated

**3 hidden layers with neurons (5, 3, 2)**

**Activation: ReLU**

**Learning rate: 0.01**

**Regularization: L1 with a rate of 0.01**

In this configuration, regularization (L1) was introduced to combat overfitting. The losses were slightly higher, with a test loss of 0.017 and a training loss of 0.010. The decision boundary was smoother and less complex compared to the second configuration, suggesting a better generalization to unseen data. Regularization helped by penalizing large weights and encouraging a simpler model that avoided overfitting. Despite fewer neurons, the network performed well, highlighting the importance of regularization in neural network design.

A screenshot of a computer

Description automatically generated

**Conclusion:**

This exploration with TensorFlow Playground demonstrated the importance of balancing the number of neurons, hidden layers, and the application of regularization to achieve optimal neural network performance. While deeper networks with more neurons can capture complex patterns, they are also prone to overfitting if not regularized properly. Regularization techniques like L1 help ensure the model generalizes well to new data. Through this exercise, I gained insights into how neural network design choices affect both training performance and generalization, which are crucial considerations in real-world machine learning tasks. I learned how different neural network parameters, such as the number of neurons, hidden layers, and activation functions, influence the model's ability to learn and generalize. I also observed firsthand how overfitting can occur when the model becomes too complex and how regularization helps prevent this by simplifying the model. Experimenting with TensorFlow Playground gave me a better understanding of how neural networks balance accuracy and generalization and reinforced the importance of choosing the right architecture and regularization techniques for achieving optimal performance.

### **Task 3: Learning Rate Exploration:**

The learning rate is a fundamental hyperparameter in training neural networks, acting as a determinant for how much the model's weights are adjusted in response to the error during weight updates. The learning rate impacts convergence speed, model performance, and the ability to find an optimal solution. Properly tuning this parameter is essential, as setting it too high can lead to rapid convergence to a suboptimal solution, while a low learning rate can result in excessively slow convergence and prolonged training times (Bengio et al., 2012).

#### **Learning Rate 0.01**

* **Test Loss:** 0.489
* **Training Loss:** 0.465

At a low learning rate of **0.01**, the model demonstrated relatively slow learning progress. The close values of training loss (0.465) and test loss (0.489) suggest that while the model was able to learn, the pace was conservative. The slight difference between these losses indicates a potential degree of overfitting, where the model fits the training data well but struggles to generalize effectively to unseen data.

A screenshot of a computer

Description automatically generatedThe lower learning rate allowed for cautious weight adjustments, mitigating the risk of overshooting optimal weights. However, this advantage comes with the trade-off of requiring more epochs to achieve convergence. This slow learning may be beneficial in scenarios where stability is paramount, yet it can significantly increase training time.

#### **Learning Rate 0.03**

* **Test Loss:** 0.444
* **Training Loss:** 0.390

With a learning rate of **0.03**, the model's performance improved markedly. The test loss dropped to **0.444**, while the training loss decreased significantly to **0.390**. This learning rate facilitated a faster convergence compared to the previous setting, effectively balancing the rate of weight adjustment with accuracy. The smaller gap between training and test losses suggests that the model generalizes better without falling into the trap of overfitting or underfitting.

A screenshot of a computer

Description automatically generatedThis intermediate learning rate likely enabled the model to strike a balance between learning dynamics, optimizing the weights more effectively without compromising on accuracy. It illustrates how a moderate learning rate can enhance model performance, resulting in improved efficiency and generalization capabilities.

#### **Learning Rate 0.1**

* **Test Loss:** 0.484
* **Training Loss:** 0.462

At a higher learning rate of **0.1**, the model reported a test loss of **0.484** and a training loss of **0.462**. These losses were relatively close to those observed with a learning rate of **0.01**, indicating that despite the increase in the learning rate, the model did not achieve significantly improved performance. In fact, the slight deterioration in performance compared to a learning rate of **0.03** suggests potential oscillation around the optimal solution.

High learning rates can lead to overshooting, resulting in the loss fluctuating rather than consistently decreasing. This phenomenon reflects the instability introduced when the learning rate is too aggressive, leading to ineffective convergence and potentially poorer overall model performance. Consequently, the model fails to reach optimal solutions effectively, confirming the A screenshot of a computer

Description automatically generatedimportance of selecting an appropriate learning rate.

### **Interpretation:**

The choice of learning rate plays a pivotal role in the learning process of neural networks. In this exploration, a learning rate of **0.03** emerged as the optimal balance between convergence speed and accuracy. The findings indicate that:

* A **lower learning rate** of **0.01** resulted in slow learning with higher losses and a tendency towards overfitting.
* A **moderate learning rate** of **0.03** provided the best results, facilitating quicker convergence and effective generalization.
* A **higher learning rate** of **0.1** led to oscillation around the optimal solution, reducing overall performance due to overshooting.

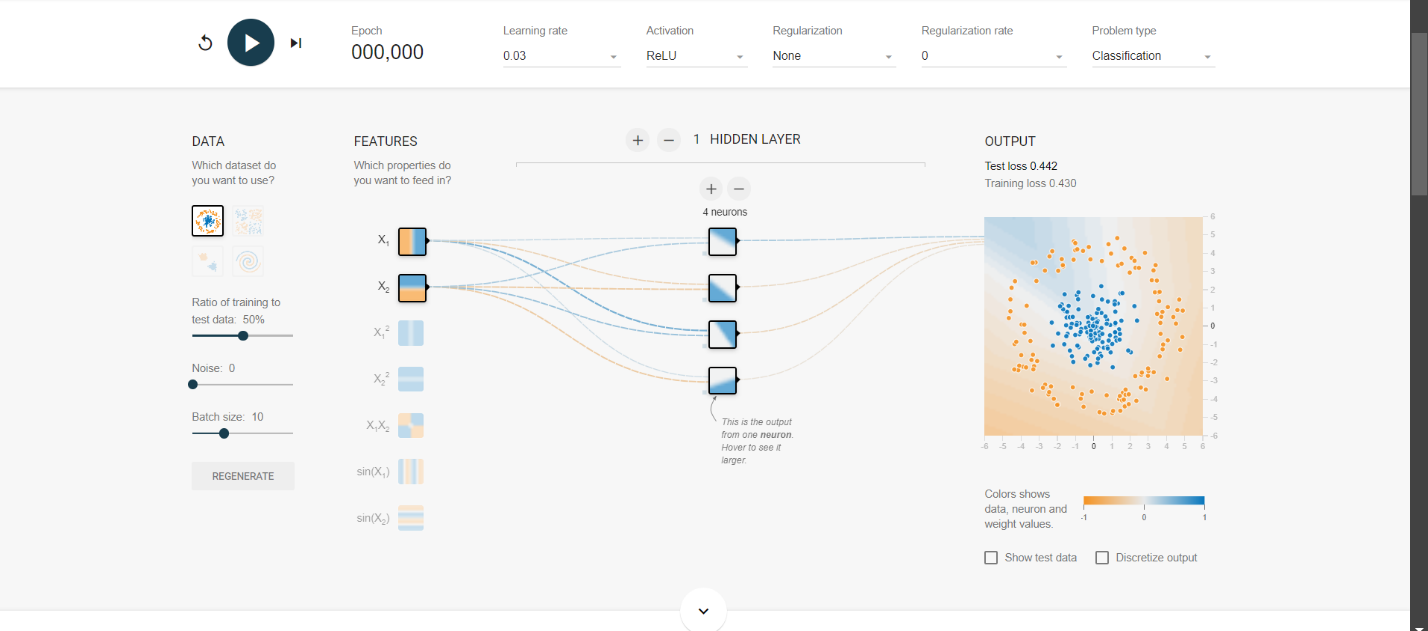
These observations underscore the significance of the learning rate in shaping the learning dynamics of neural networks, influencing both convergence speed and model generalization.

### **Conclusion:**

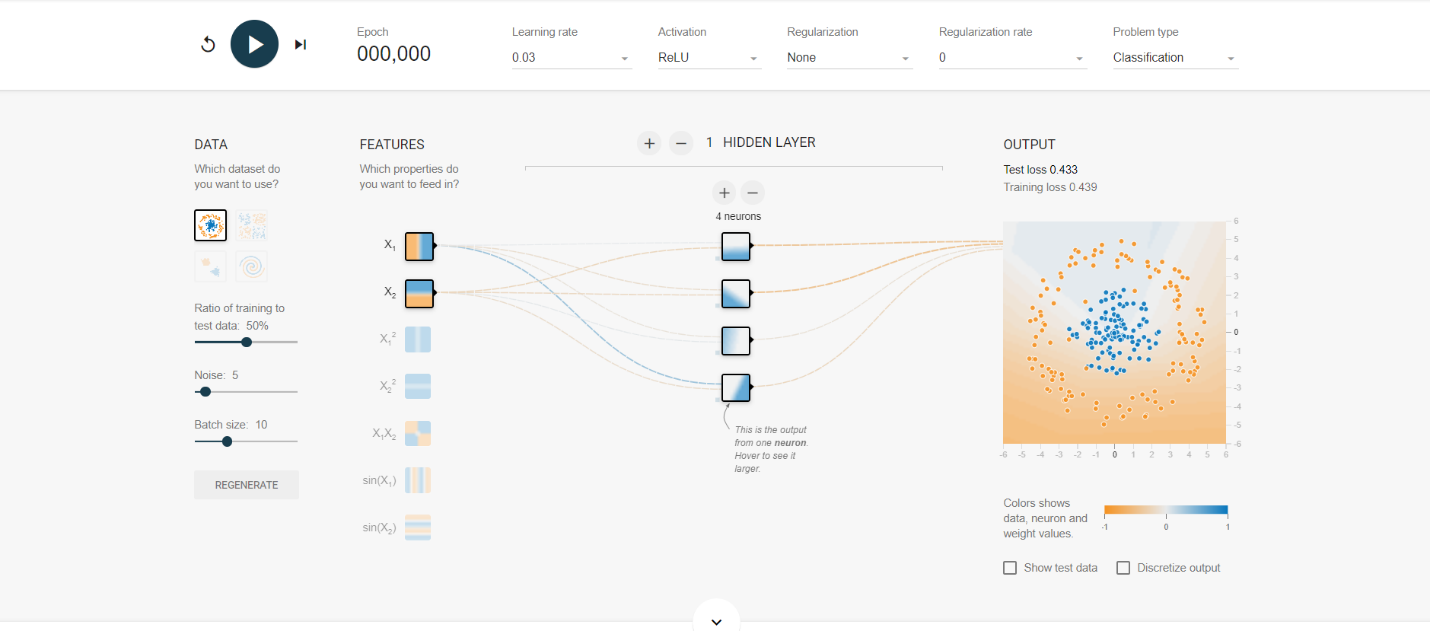
This investigation highlights the critical importance of selecting an appropriate learning rate for training neural networks. An ideal learning rate enables efficient convergence while minimizing the risk of oscillation or excessively prolonged training. Based on the conducted experiments, a learning rate of **0.03** yielded the best overall results, reinforcing its practical significance in optimizing neural network training strategies.

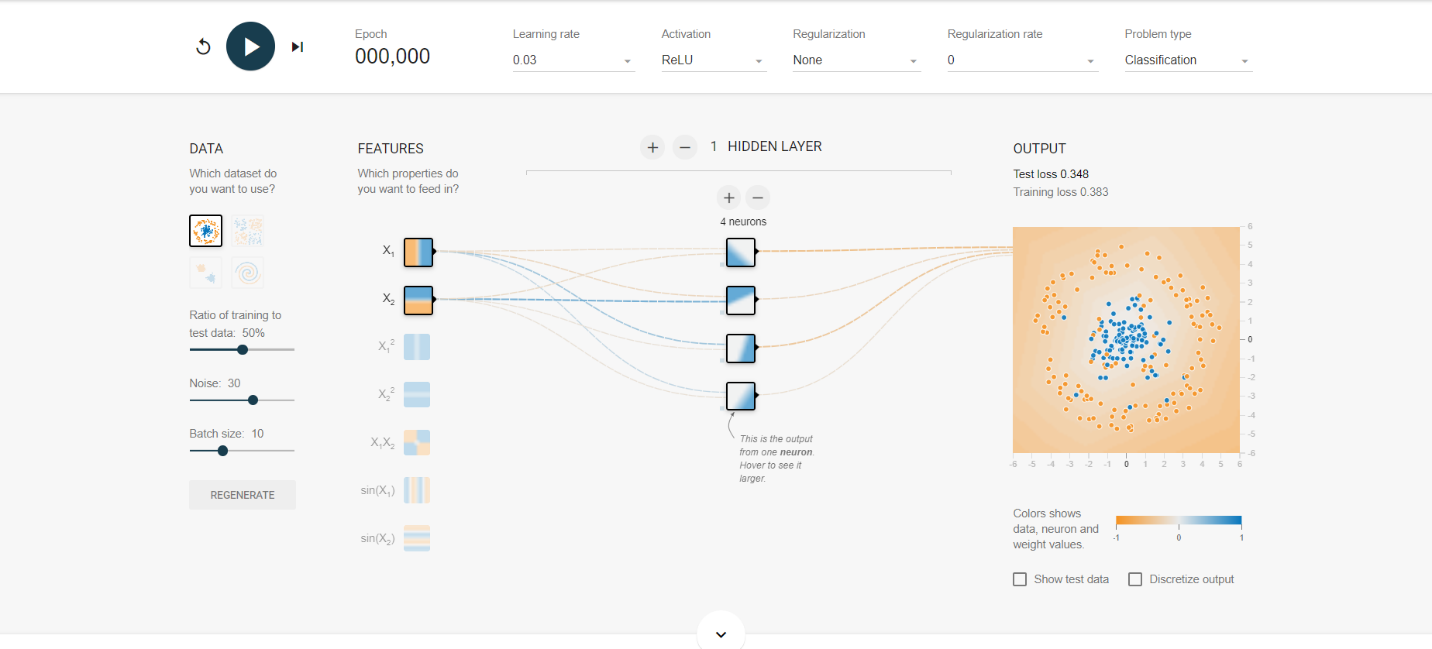
### **Task 4: Data Noise**

In this task, we introduced noise into the data by changing the "Noise" slider in TensorFlow Playground. We looked at noise levels of 0, 5, and 30 to see how they affected the network's ability to generalize.

Data noise refers to random variations in the data that can obscure important patterns. Its impact on neural networks is pretty significant. As noise levels go up, the model might struggle to tell the difference between useful data and random fluctuations. At noise level 0, the model did a great job identifying the decision boundary and achieved high accuracy in classifying the data.

When we adjusted the noise to level 5, we noticed some irregularities in the decision boundary. The performance dropped a bit as the network started fitting some noise into its predictions.

At noise level 30, the model really struggled. There was a big drop in performance, and it couldn’t find any meaningful patterns in the data.

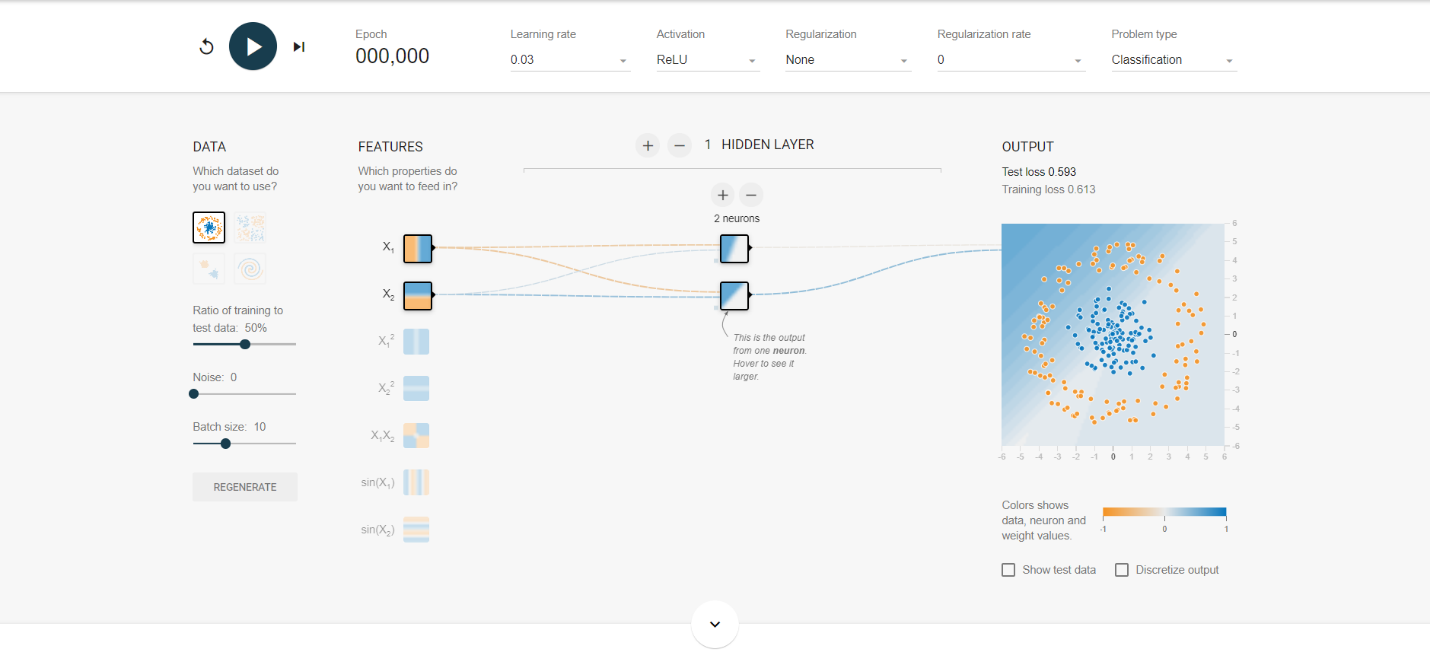


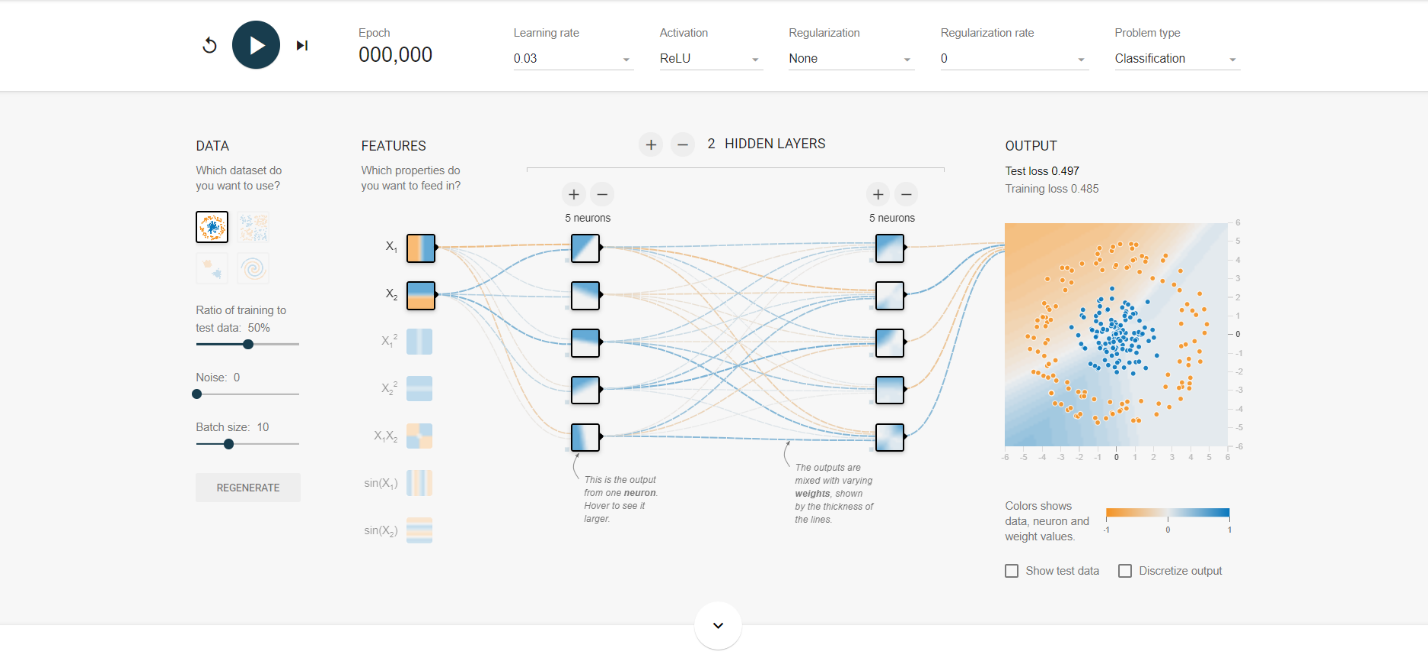
These observations show that higher noise levels can hurt the model's learning ability, leading to overfitting and lower accuracy. Understanding the effects of data noise is important for building neural networks that work well in real-life situations.

### **Task 5: Dataset Exploration**

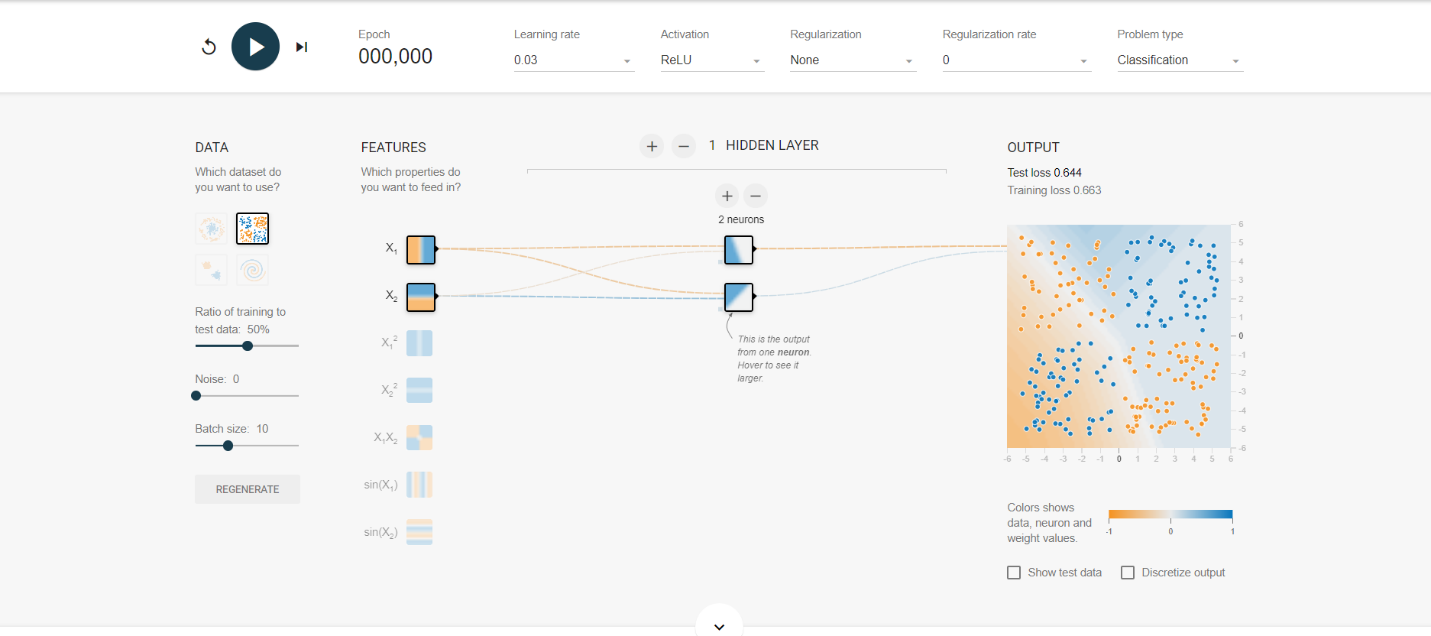
In this task, we explored different datasets available in TensorFlow Playground to analyze how the network performs on each. We focused primarily on the circular dataset and the XOR problem.

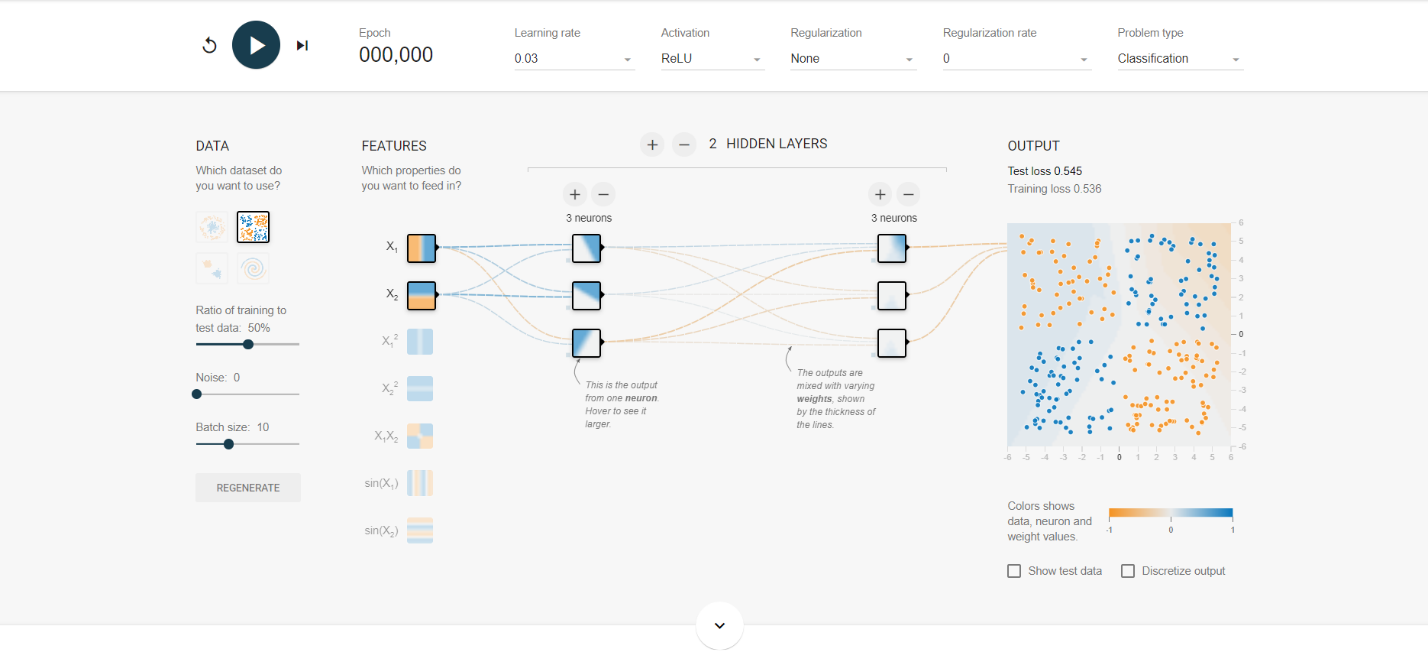
For the **circular dataset**, we created two classes of points arranged in concentric circles. Initially, we set up a simple neural network with one hidden layer and 2 neurons. As expected, this configuration struggled to classify the data accurately, resulting in a linear decision boundary that failed to separate the classes effectively.



To improve performance, we increased the complexity of the model. We added another hidden layer and increased the number of neurons to 5 in both layers. With these adjustments, the model was able to capture the non-linear relationships present in the data and created a much more accurate decision boundary, demonstrating how a more complex model can learn intricate patterns.

Next, we examined the **XOR problem** dataset, which is a classic example of a non-linear classification problem. We began with a simple model featuring one hidden layer and 2 neurons. This configuration could not accurately classify the XOR points, as it also produced a linear boundary.



To address this issue, we modified the model by adding a second hidden layer, each containing 3 neurons. This change allowed the network to create the necessary non-linear decision boundary required to accurately classify the points. This highlights the importance of having at least one hidden layer in models dealing with non-linearly separable datasets like the XOR problem.

Overall, exploring these datasets emphasized the significance of selecting appropriate model architectures for different types of problems. More complex models are essential for datasets like the circular dataset and XOR problem, as they require the ability to capture non-linear relationships. Understanding these characteristics helps in designing effective neural networks that perform well in real-world applications.

### **Key Concepts Learned:**

**Task 1: Activation Functions**

1. **Neural Network:**

Neural Network is inspired by the biological neural network of human brain. It is a computational model where each connection has a weight that adjusts as the model learns.

1. **Convolutional Neural Network:**

CNN or Convolutional Neural Network is a specialized type of neural network to process grid-like data such as images and video frames. It is good in identifying spatial hierarchies in the data, like edges, textures, and object in an image.

1. **Recurrent Neural Network:**

This type of neural network is designed handle sequential data. RNNs have feedback loop that allows information to persist, making them compatible for time-dependent tasks.

1. **ReLU Activation Function:**

The ReLU (Rectified Linear Unit) is the most widely used activation function in deep learning models. It output value range from 0 to 1. Therefore, for all negative value becomes zero and those neurons get inactive. This reduces the number of neurons contributing to the output. This technique made ReLU model to better generalization. (Nair et al., 2010)

1. **Sigmoid Activation Function:**

This function maps input values from 0 to 1. It is often used in binary classification tasks, such as logistic regression. When the input value is higher or lower, Sigmoid function gradient become very small during backpropagation, which slow down the weight updates and delaying convergence. The sigmoid function converges the data at slow rate as compared to Tanh and ReLU leading to longer training times and difficulties in learning complex patterns. (Glorot, et al., 2010)

1. **Tanh Activation Function:**

The tanh (Hyperbolic Tangent) function has output range from -1 to 1 resulting in improve learning dynamics. Tanh function also suffer from vanishing gradient problem like sigmoid function, but its gradients are steeper near 0, allowing for better gradient propagation and faster convergence. (LeCun et al., 1998)

**Task 2: Hidden Layer Neurons**

1. **Neurons**:

Neurons in a hidden layer serve as feature detectors, transforming the input data into more abstract representations. In the first experiment, the use of 7 neurons in a single layer allowed the network to map the non-linear decision boundary effectively. However, adding too many neurons (as in the second experiment) can lead to overfitting, where the model becomes overly specific to the training data.

1. **Hidden Layers:**

Adding hidden layers increases the network's capacity to learn more complex patterns in the data. In the second configuration, having four hidden layers allowed the network to capture intricate decision boundaries, but without regularization, the model became too complex, leading to overfitting. The third configuration balanced complexity and generalization with fewer neurons and layers, aided by regularization.

1. **L1 Regularization:**

L1 regularization penalizes large weights, promoting sparsity in the model. This means that only the most relevant neurons are heavily weighted, reducing the risk of overfitting. In the third configuration, the regularized model generalized better to the test data, even with fewer neurons and layers.

**Task 3: Learning Rate Exploration:**

**Learning rate: A** key hyperparameter dictating how much to adjust model weights in response to error during training. Selecting an appropriate learning rate is vital; a rate that is too low can slow down convergence, while a rate that is too high can lead to unstable training.

1. **Low Learning Rate**: A low learning rate may lead to very slow convergence, resulting in extended training times. This can prevent the model from effectively capturing the underlying patterns in the data, as it takes smaller steps towards the minimum loss function. As a consequence, the model might become stuck in local minima, hindering its overall performance.
2. **Medium Learning Rate**: A medium learning rate often strikes a balance between convergence speed and stability. It allows the model to learn effectively by making adequate weight adjustments without overshooting the optimal values. This approach generally results in faster training and better performance on validation data, making it a common starting point in hyperparameter tuning.
3. **High Learning Rate**: A high learning rate can cause erratic training behavior, leading to overshooting the optimal weight adjustments and resulting in divergence. This instability can manifest as fluctuating loss values and hinder the model's ability to converge to a solution. Consequently, training may become ineffective, and the model may fail to generalize well to new data.

**Task 4: Batch Size Impact**

1. **Batch Size:**

Batch size in neural network or machine learning refers to the number of training example processed in one forward and backward pass through the network. It is a key hypermeter to train the neural network. It determines the number of samples processed before model’s parameters are weighted.

1. **Epochs:**

Epochs is one complete pass of the entire dataset through the network.

**Task 5: Dropout Regularization**

1. **Regularization:**

The technique to prevent overfitting of data by adding constraint is called regularization. Overfitting is when model performs well on training data but poorly on test data due to complexity.

1. **Dropout:**

A regularization technique in which randomly selected neurons are dropped out or ignored in each iteration.

**Real World Applications:**

The choice of activation function, learning parameter, number of epochs and monitoring training vs test loss data is necessary for the better performance of machine learning models. ReLU is widely used in deep learning model because it is faster and computationally efficient. From image recognition and processing to natural language processing to self-driving car model, ReLU is used to train Convolutional Neural Networks (CNNs) for better image and language analysis.

Epochs is very helpful in determining whether how many times, we must drain deep learning for accurate diagnosis of disease. The loss function help detect fraudulent transactions in financial. The mean squared error (MSE) and Cross-Entropy functions are loss function are used to create accurate fraud detection system. Test loss function is used by Amazon, Netflix or Spotify to recommend relevant content to the user. Therefore, all these parameters are training deep learning to perform better in healthcare, education and finance sector.

**References:**

* Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (1998). DOI: 10.1142/S0218488598000094
* Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010.
* LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
* Nair, Vinod, and Geoffrey Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." *Proceedings of the 27th International Conference on Machine Learning (ICML)*. 2010.
* Xu, Bing, et al. "Empirical Evaluation of Rectified Activations in Convolutional Network." *arXiv preprint arXiv:1505.00853* (2015).
* Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
* [Nielsen, M. A. (2015). \*Neural Networks and Deep Learning\*. Determination Press.](https://scholar.google.com/scholar?q=Nielsen,+M.+A.+(2015).+*Neural+Networks+and+Deep+Learning*.+Determination+Press.&hl=en&as_sdt=0&as_vis=1&oi=scholart)
* TensorFlow Playground. (n.d.). Neural Network Visualizer. Retrieved from https://playground.tensorflow.org.
* Bengio, Y., Courville, A., & Vincent, P. (2012). *Representation Learning: A Review and New Perspectives.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), 1798-1828.
* Kingma, D. P., & Ba, J. (2015). *Adam: A Method for Stochastic Optimization.* In Proceedings of the 3rd International Conference on Learning Representations (ICLR).
* Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In Advances in Neural Information Processing Systems (NIPS)