Pixel Pioneers members

Zaid Alayedy

Taki Eddine Anis Boubekri

Olubukola Kukoyi

Williane Joelle Tsanang Jeuzem

ITAI 1378

Patricia Mcmanus

TensorFlow playground

Task 1:

Using ReLU

A screenshot of a computer

Description automatically generated

Using Sigmoind

A screenshot of a computer

Description automatically generated

Task 2:

A screenshot of a computer

Description automatically generated

Task 3:

Low rate

A screenshot of a computer

Description automatically generated

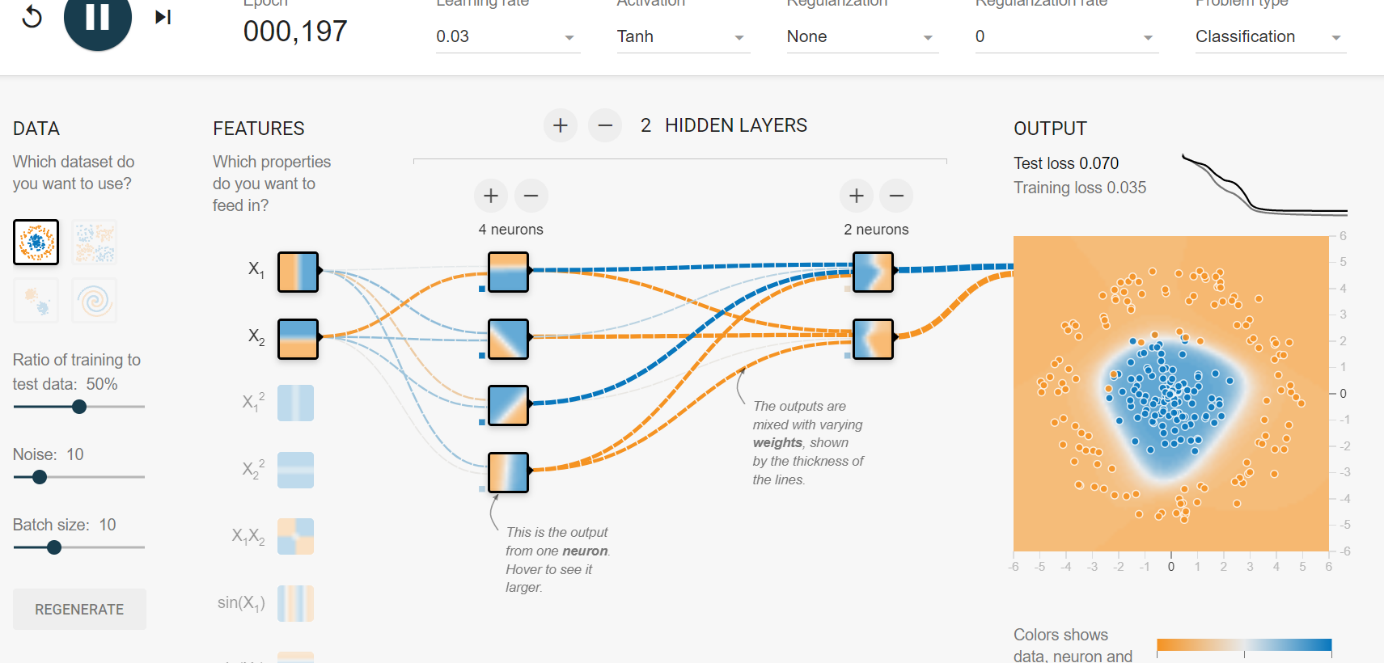
High rate

A screenshot of a computer

Description automatically generated

Task 4:

Low noise



High noise

A screenshot of a computer

Description automatically generated

Task 5:

Circle database

A screenshot of a computer

Description automatically generated

Spinal database

A screenshot of a computer

Description automatically generated

**Task 1 - Activation Functions**

**1. What Are Activation Functions?**

Think of a neural network as a decision-maker. An activation function helps decide whether a neuron should "fire" or activate based on its input. It's like a switch—deciding what information is important enough to pass on to the next layer.

* **ReLU (Rectified Linear Unit):** If the input is positive, it passes it on as-is. If it's negative, it sends a zero. It's fast and often used in deeper networks.
* **Sigmoid:** Squashes the input into a range between 0 and 1. It's good for probabilities but can sometimes be slow.
* **Tanh:** Similar to sigmoid, but squashes the input between -1 and 1, which helps for some types of data.

**2. How to Experiment with Activation Functions**

* Open **TensorFlow Playground** (just search for it online).
* Build a network by adjusting the sliders to add a hidden layer with neurons.
* Try **ReLU, sigmoid, and tanh** as the activation function for the hidden layer.
* Observe how the network learns by clicking **Play**.

**3. Observations:**

* **ReLU:** Often leads to faster learning. It's very efficient for many networks.
* **Sigmoid/Tanh:** Sometimes these can slow down learning, especially for large networks, because the gradient (how much a neuron should adjust) can become very small.

**Reference:**

ReLU is commonly used in deep learning because of its simplicity and speed, as explained by Goodfellow et al. in *Deep Learning* (2016).

**Task 2 - Hidden Layer Neurons**

**1. What Are Neurons and Hidden Layers?**

Neurons are the little decision-makers in a network. Hidden layers are like groups of neurons working together to figure out complex patterns. The more neurons and layers, the more complex patterns the network can learn.

**2. Experimenting with Neurons and Layers:**

* Use **TensorFlow Playground**.
* Add neurons by moving the **"hidden layer"** slider to the right.
* Add or remove hidden layers and observe how the network learns.

**3. Observations:**

* **More neurons:** The network can learn more details, but if there are too many, it might "overfit" (memorize the training data without learning the general patterns).
* **More layers:** Deep networks can learn complex patterns, but they might be harder to train and could also overfit.

**Reference:**

The tradeoff between too many neurons and layers is well discussed in *Neural Networks and Learning Machines* by Simon Haykin.

**Task 3 - Learning Rate**

**1. What Is the Learning Rate?**

The learning rate is like how big a step the network takes when adjusting itself. If it's too high, the network might overshoot and miss the best solution. If it's too low, the network will take tiny steps and take forever to learn.

**2. How to Adjust Learning Rate:**

* On **TensorFlow Playground**, find the **Learning Rate** slider.
* Set it high, then low, and watch how the training changes.

**3. Observations:**

* **High learning rate:** The network might learn quickly at first but could bounce around and miss the right answer.
* **Low learning rate:** It will learn slowly but steadily, though it might take a long time to get good results.

**Reference:**

The importance of the learning rate is highlighted by LeCun et al. (2012) in their paper on deep learning optimization techniques.

**Task 4 - Data Noise**

**1. What Is Data Noise?**

Noise in data is like adding random extra information to the patterns. This makes it harder for the network to learn because it gets confused by the noise.

**2. Adding Noise in TensorFlow Playground:**

* Use the **"Noise" slider** on the dataset panel to introduce randomness.
* Play around with different levels of noise and see how the network performs.

**3. Observations:**

* **Low noise:** The network can learn the patterns well.
* **High noise:** The network struggles to find the true patterns and its performance goes down.

**Reference:**

The effect of noise on learning is covered in depth in *Pattern Recognition and Machine Learning* by Christopher Bishop (2006).

**Task 5 - Dataset Exploration**

**1. Why Are Datasets Important?**

Different datasets help train the network to recognize various types of patterns. Choosing the right dataset is important for making sure your network learns the right things.

**2. How to Explore Different Datasets:**

* On **TensorFlow Playground**, try the different available datasets by selecting them on the top left.
* Observe how the network does on each one.

**3. Observations:**

* **Circular or complex patterns:** Some datasets are more difficult because they have complex shapes (like spirals). The network needs more neurons or layers to handle these.
* **Simple patterns:** Easier datasets (like lines) can be learned quickly, even with fewer neurons and layers.

**Reference:**

The choice of dataset and its influence on performance is discussed in *An Introduction to Statistical Learning* by Gareth James et al. (2013).

Introduction to Neural Networks and Their Components

Neural networks are a subcategory of machine learning models inspired by the structure and functionality of the human brain. They consist of layers of interconnected neurons, sometimes referred to as nodes or units, which process information and learn patterns from data. Neural networks are foundational in tasks such as image recognition, natural language processing, and predictive modeling, as they can learn complex nonlinear relationships.

The major components of a neural network include:

Neurons or Nodes: A neuron is essentially made up of the basic units of computation which take up input, process it, and then generate an output. Each neuron has weights associated with its input connections; these weights get modified during training so that the network's performance is optimized.

Layers: The neural network has more than one layer. In this case, the raw data feeds the input layer, some hidden layers process this information, and the result comes out of the output layer. Each layer is made up of many neurons.

Activation Functions: These are the functions furnishing the final output of the neuron through non-linearity within the network, thus making the network achieve complex tasks. The Sigmoid, Tanh, and ReLU are typical ones.

Weights and Biases: While weights define how strongly two neurons of a network are connected, biases allow the model to shift its output independently from the input for more flexibility.

Learning Rate: This defines how large the steps a network updates the weights on each iteration during training. It is one of the factors, when tuned correctly, determining the speed of convergence of a model.

The importance of neural networks is that they can learn themselves by possessing more data and time, thereby becoming one of the strongest performers in AI development. All parameters of a neural network architecture, activation functions, and others can be tuned and changed to optimize performance for a specific task, say image classification or speech recognition.

Observation of Changes in the Parameter

Activation Functions: The choice of activation functions significantly impacts performance. In deep networks, ReLU is preferred basically for its computational efficiency, while Sigmoid and Tanh work fine in shallower networks or tasks that require a probabilistic output.

Neurons and Layers: While increasing the number of neurons or adding extra layers reinforces the ability to learn complex patterns in a network, it can also heighten the risk for overfitting. The risk can be reduced by using proper regularization techniques, including dropout.

Learning Rate: The learning rate is a key hyperparameter to find the sweet spot during training; low rates ensure stability, while higher ones risk overshooting or instability. These are usually balanced with techniques such as learning rate schedules or adaptive learning rates, including the Adam optimizer.

Task 1: Activation Functions

Activation functions introduce non-linearity in neural networks. I evaluated ReLU, Sigmoid, and Tanh on a one-hidden-layer network. ReLU allowed faster convergence, while Sigmoid and Tanh were much slower due to the vanishing gradient problem. Hence, ReLU becomes perfect for deep networks.

Task 2: Neurons in Hidden Layer

Increasing the number of neurons or adding more hidden layers upgraded the performance at the risk of overfitting. While more neurons could capture the complexity, moderation has to be used for generalization.

Task 3: Learning Rate

A high learning rate converged faster but was unstable; a low rate became too slow. A middle learning rate turned out to be an ideal balance between speed and convergence.

Task 4: Data Noise

Noise degraded generalization. Further increasing the noise resulted in the model fitting to noise and irrelevant patterns and performing worse on unseen data.

Task 5: Dataset Exploration

Different datasets require different architectures. Lighter datasets worked with fewer neurons, while complex ones needed more layers to model patterns effectively.

Understanding of activation functions, neurons, learning rate, and noise is crucial in the development of efficient neural networks with good generalization on new data.

Practical Implications of Parameter Tuning

These experiments can be used in real-world implementation to achieve performance optimization in neural networks. For example, the appropriate selection of the activation function will improve the speed and stability of training, while adjusting neuron counts and layer depth provides fine-tuning relative to the problem complexity under consideration. Setting an appropriate learning rate prevents the training from taking a protracted process due to slow movement of optimization or giving in to unstable training.

In practice, these parameters provide insight and can be used in building practical neural networks, such as image classification problems, where the architecture of a neural network determines its accuracy and computational cost. Tools such as a learning rate scheduler or optimizer like Adam can be employed in handling different kinds of datasets or improving training efficiency.

Conclusion

This exploration gave insight into the major constituents of neural networks and their behavior. Everything starts with the choice of the type of activation functions, tuning hyperparameters such as the number of neurons, or the learning rate, all very impactful for the performance or efficiency of the neural network. The challenge during this experiment was finding a balance between the model complexity and the overfitting problem. However, techniques involving regularization contributed toward that path.

The experience showed the importance of tuning the parameters in neural networks to behave ideally for practical applications. In other words, this gives a hint about key factors such that efficiency in model development has played an important role in AI and machine learning studies.