Exp-5 STUDY OF ACTIVATION FUNCTIONS AND THEIR ROLE

AIM:

To study various activation functions such as Sigmoid, Tanh, ReLU, Leaky ReLU, Leaky ReLU, and ELU and model performance in neural networks.

OBJECTIVES:

→ To understand how activation fun introduce non-linearity into neural networks.

→ To implement and compare commonly used activation functions:

* Sigmoid
* Tanh
* ReLU (Rectified Linear Unit)
* Leaky ReLU
* ELU (Exponential Linear Unit)

→ To observe their effects on gradient flow and training stability

→ To analyze which activation functions are better suited for different types of problems.

PSEUDOCODE:

→ Initialize a range of input values

→ Define each activation functions.

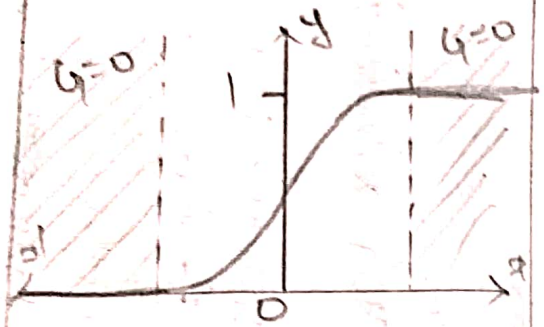
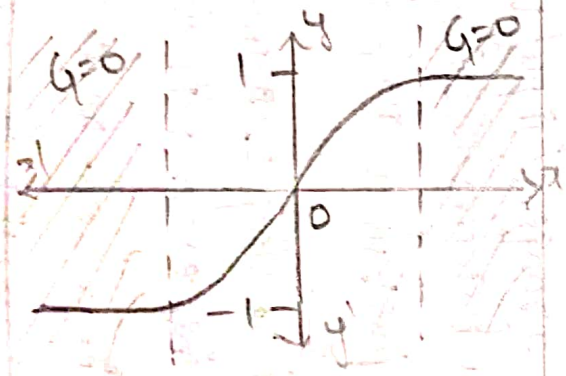
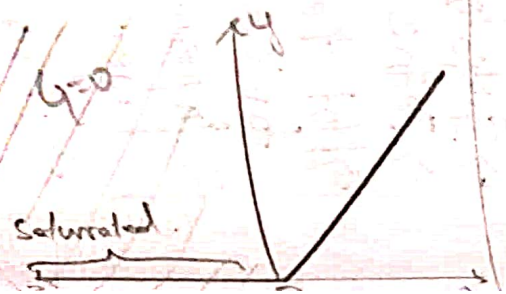
→ Plot the input values against each activation function to visualize.

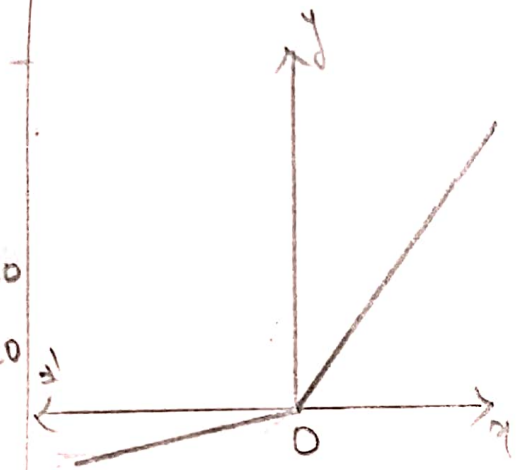
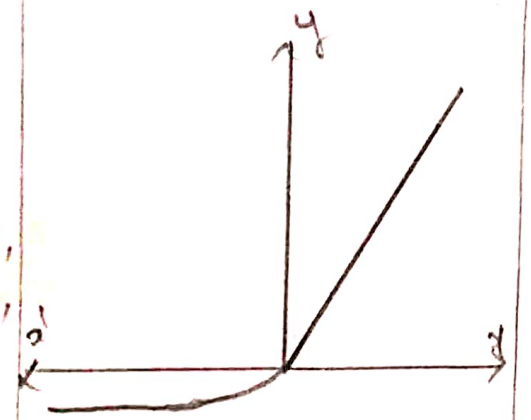
→ Implement a simple neural network with one layer using each activation function.

→ Train the network on sample data.

→ Record accuracy, loss, and convergence rate for each function.

→ Analyze and compare the behavior based on the observations.

ACTIVATION FUNCTION	SHAPE CHARACTERISTICS	VANISHING GRADIENT	COMPUTATIONAL COST	OUTPUT RANGE	SHAPE	BEST USE CASES
Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$		Yes	Moderate	(0,1)	S-shape curve	Binary classification output layer
Tanh $\tanh(z) = g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$		Yes	Moderate	(-1,1)	Steeper S-curve	Hidden layers centered data
ReLU $f(z) = \begin{cases} 0, & z<0 \\ z & z \geq 0 \end{cases}$		No	low	$[0, \infty)$	linear for positive	" sparse activation

Leaky ReLU $f(x) = \begin{cases} \alpha x & x<0 \\ x & x \geq 0 \end{cases}$		Reduced	Low	$(-\infty, \infty)$	Allows small slope.	Avoid dying ReLU problem
ELU $\max(w_1^T x + b_1, w_2^T x + b_2, \ldots)$		Reduced	Moderate.	$(-\alpha, \infty)$	Smooth for negative.	Improve learning faster convergence

OBSERVATION:

1) Sigmoid
- The training loss decreases very slowly compared to other suffers from slower convergence.
- Its starts around 0.72 and ends slightly above 0.6 after 1000 epochs.

2) Tanh
- The loss decreases faster than Sigmoid.
- It shows a consistant downward trend. ends at 0.32

3) ReLU
- ReLU starts with a similar inital loss but decreases steadily and converges better than Sigmoid. ends at 0.37

4) Leaky ReLU:
- It has a consistent decline in loss and ends near 0.34, indication better learning capability than ReLU.

5) ELU
- It shows a steady and significant decrease, reaching the lowest loss around 0.31.
- converge faster and better compared.

RESULT:

The experiment was implemented and obtained result sucessfully.

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
# Activation functions and their derivatives
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```python
def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)
```

```python
def tanh(x):
    return np.tanh(x)
```

```python
def tanh_derivative(x):
    return 1 - np.tanh(x)**2
```

```python
def relu(x):
    return np.maximum(0, x)
```

```python
def relu_derivative(x):
    return (x > 0).astype(float)
```

Variables Terminal

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```
def tanh_derivative(x):
    return 1 - np.tanh(x)**2
```

```
def relu(x):
    return np.maximum(0, x)
```

```
def relu_derivative(x):
    return (x > 0).astype(float)
```

```
def leaky_relu(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)
```

```
def leaky_relu_derivative(x, alpha=0.01):
    dx = np.ones_like(x)
    dx[x < 0] = alpha
    return dx
```

```
def elu(x, alpha=1.0):
    return np.where(x >= 0, x, alpha * (np.exp(x) - 1))
```

```
def elu_derivative(x, alpha=1.0):
    return np.where(x >= 0, 1, elu(x, alpha) + alpha)
```

```
# Simple 1-hidden-layer Neural Network class
class SimpleNN:
```

Variables Terminal

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```python
def elu(x, alpha=1.0):
    return np.where(x >= 0, x, alpha * (np.exp(x) - 1))
```

```python
def elu_derivative(x, alpha=1.0):
    return np.where(x >= 0, 1, elu(x, alpha) + alpha)
```

```python
# Simple 1-hidden-layer Neural Network class
class SimpleNN:
    def __init__(self, input_size, hidden_size, output_size, activation, activation_deriv):
        self.activation = activation
        self.activation_deriv = activation_deriv

        # Initialize weights with small random values
        self.W1 = np.random.randn(input_size, hidden_size) * 0.1
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size) * 0.1
        self.b2 = np.zeros((1, output_size))

    def forward(self, X):
        self.z1 = X.dot(self.W1) + self.b1
        self.a1 = self.activation(self.z1)
        self.z2 = self.a1.dot(self.W2) + self.b2
        # Output layer - sigmoid for binary classification
        self.a2 = sigmoid(self.z2)
        return self.a2

    def backward(self, X, y, output, lr=0.01):
        m = y.shape[0]
```

Variables Terminal

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```python
        return self.a2

    def backward(self, X, y, output, lr=0.01):
        m = y.shape[0]

        # Compute output error
        delta2 = (output - y) * (output * (1 - output))  # derivative of sigmoid output layer

        # Compute error for hidden layer
        delta1 = delta2.dot(self.W2.T) * self.activation_deriv(self.z1)

        # Update weights and biases
        self.W2 -= lr * self.a1.T.dot(delta2) / m
        self.b2 -= lr * np.sum(delta2, axis=0, keepdims=True) / m
        self.W1 -= lr * X.T.dot(delta1) / m
        self.b1 -= lr * np.sum(delta1, axis=0, keepdims=True) / m

    def train(self, X, y, epochs=1000, lr=0.01):
        losses = []
        for epoch in range(epochs):
            output = self.forward(X)
            loss = np.mean(-(y * np.log(output + 1e-9) + (1 - y) * np.log(1 - output + 1e-9)))
            losses.append(loss)
            self.backward(X, y, output, lr)
        return losses
```

```python
# Generate a toy dataset
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
y = y.reshape(-1, 1)
```

Variables Terminal

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```
            loss = np.mean(-(y * np.log(output + 1e-9) + (1 - y) * np.log(1 - output + 1e-9)))
            losses.append(loss)
            self.backward(X, y, output, lr)
        return losses
```

```
# Generate a toy dataset
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
y = y.reshape(-1, 1)
```

```
# Scale features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
# Split train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Dictionary of activation functions and derivatives to test
activations = {
    'Sigmoid': (sigmoid, sigmoid_derivative),
    'Tanh': (tanh, tanh_derivative),
    'ReLU': (relu, relu_derivative),
    'Leaky ReLU': (leaky_relu, leaky_relu_derivative),
    'ELU': (elu, elu_derivative)
}
```

```
# Train networks with different activations and store losses
losses_dict = {}
```

Variables Terminal

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```
        'Sigmoid' : (sigmoid, sigmoid_derivative),
        'Tanh': (tanh, tanh_derivative),
        'ReLU': (relu, relu_derivative),
        'Leaky ReLU': (leaky_relu, leaky_relu_derivative),
        'ELU': (elu, elu_derivative)
    }
```

```
# Train networks with different activations and store losses
losses_dict = {}
```

```
for name, (act, act_deriv) in activations.items():
    print(f"Training with {name} activation...")
    nn = SimpleNN(input_size=2, hidden_size=10, output_size=1, activation=act, activation_deriv=act_deriv)
    losses = nn.train(X_train, y_train, epochs=1000, lr=0.05)
    losses_dict[name] = losses
```

```
Training with Sigmoid activation...
Training with Tanh activation...
Training with ReLU activation...
Training with Leaky ReLU activation...
Training with ELU activation...
```

```
# Plot the loss curves for comparison
plt.figure(figsize=(12, 8))
for name, losses in losses_dict.items():
    plt.plot(losses, label=name)
plt.title("Training Loss Curves for Different Activation Functions")
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.legend()
```

Variables Terminal

https://colab.research.google.com/drive/1CX6pArhhDo7azAKmE98BbLXS140Sxa0e?authuser=2

lab_5.ipynb

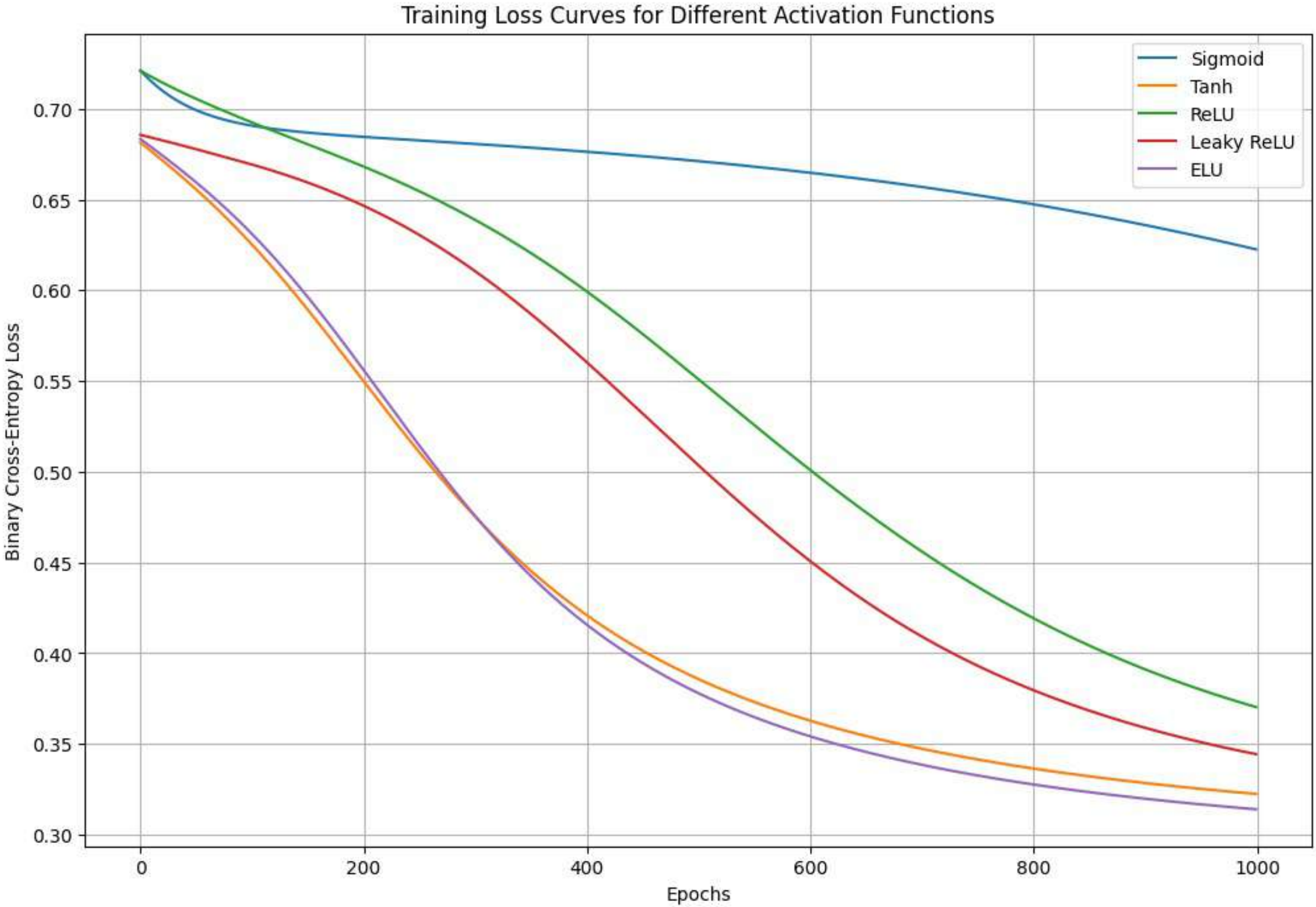
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Share Gemini Connect

```
# Plot the loss curves for comparison
plt.figure(figsize=(12, 8))
for name, losses in losses_dict.items():
    plt.plot(losses, label=name)
plt.title("Training Loss Curves for Different Activation Functions")
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.legend()
plt.grid(True)
plt.show()
```

Variables Terminal


Training Loss Curves for Different Activation Functions
Sigmoid
Tanh
ReLU
Leaky ReLU
ELU
Binary Cross-Entropy Loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
200
400
600
800
1000
Epochs