Deep Learning Optimization: A Comprehensive Guide

1. Gradient Descent Algorithms 🧮

1.1 Batch Gradient Descent (BGD)

- Definition: Algorithm that uses the entire dataset to compute gradients and update model parameters once per epoch.
- How It Works:
    1. Calculates error for each example in the dataset
    2. Accumulates these errors
    3. Updates weights only after seeing all examples
    4. Repeats this process for each epoch
- Key Characteristics:
    1. Most stable gradient descent variant
    2. Guaranteed to converge to global minimum for convex functions
    3. Very slow for large datasets
    4. High memory requirements
- Real-world Analogy: Imagine you're a chef trying to perfect a recipe. With batch gradient descent, you would:
    1. Cook multiple versions of the dish
    2. Gather ALL feedback from ALL tasters
    3. Make adjustments to the recipe only after receiving everyone's feedback

python
Copy

```
# Batch Gradient Descent Implementation
def batch_gradient_descent(X, y, learning_rate=0.01, epochs=100):
    w = np.zeros(X.shape[1])  # Initialize weights

    for epoch in range(epochs):
        # Calculate gradient using entire dataset
        gradient = np.zeros_like(w)
        for i in range(len(X)):
            prediction = np.dot(X[i], w)
            error = prediction - y[i]
            gradient += error * X[i]

        # Update weights once per epoch
        gradient = gradient / len(X)  # Average gradient
        w -= learning_rate * gradient

    return w
```

- PyTorch Code:

```
loader = DataLoader(dataset, batch_size=len(dataset))  # Load full dataset
```

- optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

1.2 Stochastic Gradient Descent (SGD)

- Definition: Updates model parameters using one randomly selected example at a time.
- How It Works:
    1. Picks one random training example
    2. Calculates error for that example
    3. Immediately updates weights
    4. Moves to next random example
    5. Much more frequent updates than BGD
- Key Characteristics:
    1. Faster updates
    2. Lower memory usage
    3. Noisier updates (can help escape local minima)
    4. May never converge exactly
    5. Better for large datasets
- Real-world Analogy: Like adjusting your cooking based on each individual taster's feedback immediately:
    1. Cook one dish
    2. Get one person's feedback
    3. Adjust recipe immediately
    4. Repeat with next person

python
Copy

```
# Stochastic Gradient Descent Implementation
def sgd(X, y, learning_rate=0.01, epochs=100):
    w = np.zeros(X.shape[1])

    for epoch in range(epochs):
        # Shuffle data
        indices = np.random.permutation(len(X))
        for i in indices:
            # Update weights for each example
            prediction = np.dot(X[i], w)
            error = prediction - y[i]
            w -= learning_rate * error * X[i]

    return w
```

- PyTorch Code:

```
loader = DataLoader(dataset, batch_size=1)  # One sample per batch
```

- optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
- Pros: Fast, avoids local minima.
- Cons: Noisy, unstable convergence.
- Use Case: Large datasets, online learning.

1.3 Mini-Batch Gradient Descent

- Definition: A compromise between BGD and SGD that updates parameters using small batches of data.
- How It Works:
    1. Divides dataset into small batches (typically 32-256 examples)
    2. Calculates error for each batch
    3. Updates weights after each batch
    4. Provides balance between stability and speed
- Key Characteristics:
    1. Most commonly used in practice
    2. Good compromise between computation and stability
    3. Benefits from modern hardware optimization
    4. More stable than SGD but faster than BGD
    5. Allows parallel processing
- Real-world Analogy: Like getting feedback from small focus groups:
    1. Cook several dishes
    2. Get feedback from a small group (e.g., 5 people)
    3. Adjust recipe based on group feedback
    4. Repeat with next group

python
Copy

```
# Mini-batch Gradient Descent Implementation
def mini_batch_gradient_descent(X, y, batch_size=32, learning_rate=0.01, epochs=100):
    w = np.zeros(X.shape[1])

    for epoch in range(epochs):
        # Shuffle data
        indices = np.random.permutation(len(X))

        # Process mini-batches
        for start_idx in range(0, len(X), batch_size):
            batch_idx = indices[start_idx:start_idx + batch_size]
            X_batch = X[batch_idx]
            y_batch = y[batch_idx]

            # Calculate gradient for mini-batch
            gradient = np.zeros_like(w)
```

```
            for i in range(len(X_batch)):
                prediction = np.dot(X_batch[i], w)
                error = prediction - y_batch[i]
                gradient += error * X_batch[i]

            # Update weights after each mini-batch
            gradient = gradient / len(X_batch)
            w -= learning_rate * gradient

    return w
```

- PyTorch Code:

```
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

- optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
- Pros: Industry standard.
- Cons: Batch size tuning required.
- Use Case: Most deep learning tasks.

Comparison of Gradient Descent Variants

Aspect	Batch GD	SGD	Mini-Batch GD
Update Frequency	Once per epoch	Every example	Every batch
Stability	Most stable	Least stable	Moderate
Speed	Slowest	Fastest	Moderate
Memory Usage	Highest	Lowest	Moderate
Convergence	Smooth	Noisy	Moderate
Suitable For	Small datasets	Very large datasets	Most applications

When to Use Each Variant:

1. Use Batch Gradient Descent when:
    - Dataset is small (fits in memory)
    - Need stable and deterministic behavior
    - Computing exact gradient is important
2. Use Stochastic Gradient Descent when:
    - Dataset is very large
    - Need to get quick results
    - Can tolerate noisy updates
3. Use Mini-batch Gradient Descent when:
    - Working with deep learning frameworks
    - Need to balance speed and stability

- Want to leverage modern hardware (GPU/TPU)

2. The Vanishing Gradient Problem 🤯

2.1 Understanding the Problem

- Definition: A situation where gradients become extremely small as they propagate backward through deep neural networks.
- Why It Happens:
    1. Chain Rule Effect:
        - Backpropagation uses the chain rule
        - Each layer multiplies gradients
        - Many small numbers multiplied together become tiny
    2. Activation Function Impact:
        - Traditional activation functions (sigmoid, tanh) have gradients < 1
        - Repeated multiplication makes gradients vanish
- Visual Example:

python
Copy

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def demonstrate_vanishing_gradient():
    # Create a deep network simulation
    layers = 10
    x = np.linspace(-1, 1, 100)
    gradient = np.ones_like(x)

    plt.figure(figsize=(12, 6))
    for i in range(layers):
        gradient = gradient * sigmoid(x) * (1 - sigmoid(x))
        plt.plot(x, gradient, label=f'Layer {i+1}')

    plt.title('Vanishing Gradient Through Layers')
    plt.xlabel('Input')
    plt.ylabel('Gradient')
    plt.legend()
    plt.show()

demonstrate_vanishing_gradient()
```

Vanishing Gradient & Solutions 🤯

Causes:

- Deep networks with sigmoid/tanh activations.
- Gradients become too small during backprop.

Solutions:

1. Weight Initialization
    - Xavier (Glorot): For tanh/sigmoid.
    - python
    - Copy
    - torch.nn.init.xavier_normal_(layer.weight)
    - He Initialization: For ReLU.
    - python
    - Copy
    - torch.nn.init.kaiming_normal_(layer.weight, mode='fan_in', nonlinearity='relu')
2. Batch Normalization
3. python
4. Copy
5. self.bn = nn.BatchNorm1d(64) # Add before activation
    - Pros: Faster training, reduces sensitivity to initialization.

---

3. Early Stopping

- PyTorch Implementation:
- python
- Copy

```
best_loss = float('inf')
patience = 5
for epoch in range(100):
    train()
    val_loss = validate()
    if val_loss < best_loss:
        best_loss = val_loss
        counter = 0
    else:
        counter += 1
```

- if counter >= patience: break
- Use Case: Prevent overfitting.

---

4. Dropout

- PyTorch Code:
- python
- Copy
- self.dropout = nn.Dropout(p=0.5) # Add after activation
- Pros: Simulates ensemble learning.

---

5. Regularization

L1 (Lasso):

python
Copy
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, weight_decay=0)
# Add L1 manually:
l1_loss = torch.abs(model.fc.weight).sum()
loss += 0.01 * l1_loss
```

L2 (Ridge):

python
Copy
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, weight_decay=0.01)
```

L1	L2
Sparse weights	Small weights
Feature selection	Generalization

---

6. Optimizers

SGD with Momentum

python
Copy
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

Adam

python
Copy

optimizer = torch.optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))

- Pros: Combines momentum + adaptive learning rates.

Comparison:

Optimizer	When to Use
SGD + Momentum	Low-resource setups
Adam	Default for most tasks
RMSProp	RNNs, non-stationary targets

---

7. Activation Functions

Function	PyTorch Code	Use Case
ReLU	nn.ReLU()	Hidden layers
Leaky ReLU	nn.LeakyReLU(0.01)	Avoid dead neurons
Sigmoid	nn.Sigmoid()	Binary classification

---

8. Training Tips

1. Learning Rate Scheduling:
2. python
3. Copy

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')

4. scheduler.step(val_loss)
5. Gradient Clipping:
6. python
7. Copy
8. torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

2.2 Impact on Training

- Symptoms:
    - Early layers train very slowly or not at all
    - Model favors shallow patterns over deep ones

    - Training plateaus early
    - Poor performance on complex tasks
- Real-world Analogy: Like playing the telephone game with 100 people:
    - The message gets weaker at each step
    - By the end, the original message is lost
    - Earlier players never get feedback to improve

2.3 Solutions

Better Activation Functions:

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

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

Proper Initialization:

```
python
Copy
# He initialization for ReLU networks
def he_init(shape):
    2.      return np.random.randn(*shape) * np.sqrt(2.0 / shape[0])
```

Skip Connections:

```
python
Copy
class ResidualBlock:
    def forward(self, x):
        # Regular path
        out = self.conv1(x)
        out = self.relu(out)
        out = self.conv2(out)

        # Skip connection
        out += x  # Add input directly to output
    3.          return self.relu(out)
```

Batch Normal
ization:

```
python
Copy
class BatchNorm:
    def __init__(self, num_features):
```

```
        self.gamma = np.ones(num_features)
        self.beta = np.zeros(num_features)
        self.eps = 1e-5

    def forward(self, x):
        mean = np.mean(x, axis=0)
        var = np.var(x, axis=0)
        x_norm = (x - mean) / np.sqrt(var + self.eps)
    4.          return self.gamma * x_norm + self.beta
```

Best Practices to Prevent Vanishing Gradients:

1. Architecture Design:
    - Use ReLU or its variants
    - Keep network depth reasonable
    - Add skip connections for very deep networks
    - Use batch normalization
2. Training Strategy:
    - Start with smaller networks
    - Monitor gradient norms
    - Use gradient clipping if needed
    - Consider pre-training
3. Initialization:
    - Use modern initialization methods
    - Match initialization to activation function
    - Initialize bias terms properly

Batch vs Stochastic

```
import numpy as np
import pandas as pd
import time
```

```
df = pd.read_csv('/content/Social_Network_Ads.csv')
```

```
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
df = df[['Age','EstimatedSalary','Purchased']]
```

```
df.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
X = df.iloc[:,0:2]
y = df.iloc[:,-1]
```

```
X
```

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

```
X_scaled.shape
```

```
(400, 2)
```

```
#from sklearn.model_selection import train_test_split
#X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

```
#X_train.shape
```

```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
```

```
model = Sequential()

model.add(Dense(10,activation='relu',input_dim=2))
model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_42 (Dense)            (None, 10)                30

 dense_43 (Dense)            (None, 10)                110

 dense_44 (Dense)            (None, 1)                 11

=================================================================
Total params: 151
Trainable params: 151
Non-trainable params: 0
_________________________________________________________________
```

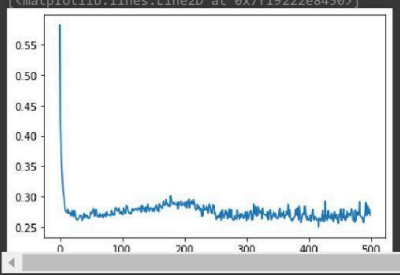
```
model.compile(loss='binary_crossentropy',metrics=['accuracy'])
#start = time.time()
history = model.fit(X_scaled,y,epochs=500,batch_size=1,validation_split=0.2)
#print(time.time() - start)
```

```
Epoch 1/500
320/320 [==============================] - 2s 3ms/step - loss: 0.5813 - accuracy: 0.7688 - val_loss: 0.8103 - val_accuracy: 0.5875
Epoch 2/500
320/320 [==============================] - 1s 2ms/step - loss: 0.4220 - accuracy: 0.8469 - val_loss: 0.8065 - val_accuracy: 0.6000
Epoch 3/500
320/320 [==============================] - 1s 2ms/step - loss: 0.3662 - accuracy: 0.8500 - val_loss: 0.7437 - val_accuracy: 0.6500
Epoch 4/500
320/320 [==============================] - 1s 2ms/step - loss: 0.3427 - accuracy: 0.8562 - val_loss: 0.6665 - val_accuracy: 0.6500
Epoch 5/500
320/320 [==============================] - 1s 2ms/step - loss: 0.3257 - accuracy: 0.8562 - val_loss: 0.5826 - val_accuracy: 0.6625
Epoch 6/500
320/320 [==============================] - 1s 2ms/step - loss: 0.3110 - accuracy: 0.8469 - val_loss: 0.4986 - val_accuracy: 0.6875
Epoch 7/500
320/320 [==============================] - 1s 2ms/step - loss: 0.3025 - accuracy: 0.8531 - val_loss: 0.4282 - val_accuracy: 0.7250
Epoch 8/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2936 - accuracy: 0.8531 - val_loss: 0.3934 - val_accuracy: 0.7500
Epoch 9/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2833 - accuracy: 0.8625 - val_loss: 0.3525 - val_accuracy: 0.7750
Epoch 10/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2763 - accuracy: 0.8656 - val_loss: 0.3115 - val_accuracy: 0.9125
Epoch 11/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2761 - accuracy: 0.8719 - val_loss: 0.2878 - val_accuracy: 0.9625
Epoch 12/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2744 - accuracy: 0.8750 - val_loss: 0.2641 - val_accuracy: 0.9625
Epoch 13/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2740 - accuracy: 0.8813 - val_loss: 0.2551 - val_accuracy: 0.9625
Epoch 14/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2790 - accuracy: 0.8813 - val_loss: 0.2402 - val_accuracy: 0.9625
Epoch 15/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2733 - accuracy: 0.8750 - val_loss: 0.2281 - val_accuracy: 0.9625
Epoch 16/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2745 - accuracy: 0.8813 - val_loss: 0.2155 - val_accuracy: 0.9625
Epoch 17/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2737 - accuracy: 0.8844 - val_loss: 0.2048 - val_accuracy: 0.9625
Epoch 18/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2738 - accuracy: 0.8813 - val_loss: 0.1981 - val_accuracy: 0.9750
Epoch 19/500
320/320 [==============================] - 1s 3ms/step - loss: 0.2685 - accuracy: 0.8938 - val_loss: 0.1971 - val_accuracy: 0.9625
Epoch 20/500
320/320 [==============================] - 1s 3ms/step - loss: 0.2698 - accuracy: 0.8875 - val_loss: 0.1917 - val_accuracy: 0.9625
Epoch 21/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2770 - accuracy: 0.8875 - val_loss: 0.1791 - val_accuracy: 0.9750
Epoch 22/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2702 - accuracy: 0.8906 - val_loss: 0.1726 - val_accuracy: 0.9750
Epoch 23/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2674 - accuracy: 0.8938 - val_loss: 0.1711 - val_accuracy: 0.9750
Epoch 24/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2727 - accuracy: 0.8813 - val_loss: 0.1648 - val_accuracy: 0.9750
Epoch 25/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2770 - accuracy: 0.8906 - val_loss: 0.1602 - val_accuracy: 0.9750
Epoch 26/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2701 - accuracy: 0.9031 - val_loss: 0.1657 - val_accuracy: 0.9750
```

```
Epoch 27/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2636 - accuracy: 0.8969 - val_loss: 0.1675 - val_accuracy: 0.9750
Epoch 28/500
320/320 [==============================] - 1s 2ms/step - loss: 0.2638 - accuracy: 0.9000 - val_loss: 0.1648 - val_accuracy: 0.9875
Epoch 29/500
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f19222e8450>]
```



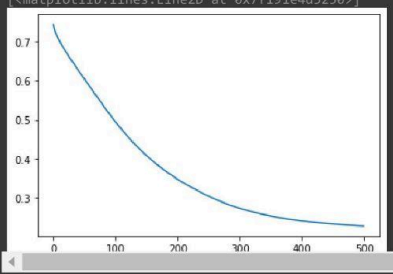

```
model = Sequential()

model.add(Dense(10,activation='relu',input_dim=2))
model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',metrics=['accuracy'])
#start = time.time()
history = model.fit(X_scaled,y,epochs=10,batch_size=250,validation_split=0.2)
#print(time.time() - start)
```

```
Epoch 1/10
2/2 [==============================] - 2s 305ms/step - loss: 0.6440 - accuracy: 0.7125 - val_loss: 0.7794 - val_accuracy: 0.3625
Epoch 2/10
2/2 [==============================] - 0s 45ms/step - loss: 0.6296 - accuracy: 0.7125 - val_loss: 0.7736 - val_accuracy: 0.3625
Epoch 3/10
2/2 [==============================] - 0s 71ms/step - loss: 0.6204 - accuracy: 0.7125 - val_loss: 0.7695 - val_accuracy: 0.3625
Epoch 4/10
2/2 [==============================] - 0s 50ms/step - loss: 0.6125 - accuracy: 0.7125 - val_loss: 0.7659 - val_accuracy: 0.3625
Epoch 5/10
2/2 [==============================] - 0s 47ms/step - loss: 0.6055 - accuracy: 0.7125 - val_loss: 0.7608 - val_accuracy: 0.3625
Epoch 6/10
2/2 [==============================] - 0s 50ms/step - loss: 0.5990 - accuracy: 0.7125 - val_loss: 0.7593 - val_accuracy: 0.3625
Epoch 7/10
2/2 [==============================] - 0s 39ms/step - loss: 0.5928 - accuracy: 0.7125 - val_loss: 0.7558 - val_accuracy: 0.3625
Epoch 8/10
2/2 [==============================] - 0s 43ms/step - loss: 0.5867 - accuracy: 0.7125 - val_loss: 0.7531 - val_accuracy: 0.3625
Epoch 9/10
2/2 [==============================] - 0s 47ms/step - loss: 0.5811 - accuracy: 0.7125 - val_loss: 0.7500 - val_accuracy: 0.3625
Epoch 10/10
2/2 [==============================] - 0s 43ms/step - loss: 0.5758 - accuracy: 0.7125 - val_loss: 0.7487 - val_accuracy: 0.3625
```

```
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f191e4d5250>]
```



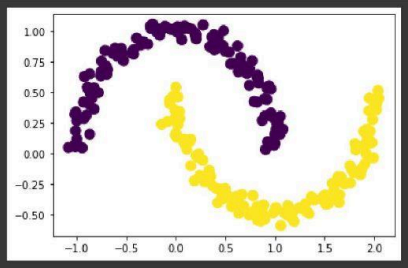

```
Start coding or generate with AI.
```

Vanishing gradient

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from keras.layers import Dense
from keras.models import Sequential
```

```
X,y = make_moons(n_samples=250, noise=0.05, random_state=42)
```

```
plt.scatter(X[:,0],X[:,1], c=y, s=100)
plt.show()
```




```
model = Sequential()

model.add(Dense(10,activation='sigmoid',input_dim=2))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.get_weights()[0]
```

```
array([[-0.33058235,  0.292853  , -0.5027271 , -0.01332176,  0.4663965 ,
        -0.08549386, -0.3767269 ,  0.02521479,  0.17957473,  0.09508276],
       [-0.46751493, -0.5052102 ,  0.618305  , -0.41437078,  0.51485544,
         0.45250946,  0.36829787,  0.66284555,  0.14777309,  0.04143274]],
      dtype=float32)
```

```
old_weights = model.get_weights()[0]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 1/100
7/7 [==============================] - 1s 3ms/step - loss: 0.6966 - accuracy: 0.3700
Epoch 2/100
7/7 [==============================] - 0s 2ms/step - loss: 0.6958 - accuracy: 0.1650
Epoch 3/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6954 - accuracy: 0.3050
Epoch 4/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6950 - accuracy: 0.2800
Epoch 5/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6946 - accuracy: 0.1250
Epoch 6/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6940 - accuracy: 0.5100
Epoch 7/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6936 - accuracy: 0.5100
Epoch 8/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6932 - accuracy: 0.5100
Epoch 9/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6931 - accuracy: 0.5100
Epoch 10/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6926 - accuracy: 0.5100
Epoch 11/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6922 - accuracy: 0.5100
Epoch 12/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6919 - accuracy: 0.5100
Epoch 13/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6917 - accuracy: 0.5100
Epoch 14/100
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.6910 - accuracy: 0.5100
Epoch 15/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6906 - accuracy: 0.5100
Epoch 16/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6902 - accuracy: 0.5100
Epoch 17/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6895 - accuracy: 0.5100
Epoch 18/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6895 - accuracy: 0.5100
Epoch 19/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6894 - accuracy: 0.5100
Epoch 20/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6891 - accuracy: 0.5100
Epoch 21/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6885 - accuracy: 0.5100
Epoch 22/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6880 - accuracy: 0.5100
Epoch 23/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6873 - accuracy: 0.5100
Epoch 24/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6865 - accuracy: 0.5100
Epoch 25/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6861 - accuracy: 0.5100
Epoch 26/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6856 - accuracy: 0.5100
Epoch 27/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6846 - accuracy: 0.5100
Epoch 28/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6836 - accuracy: 0.5100
Epoch 29/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6830 - accuracy: 0.6400
```

```
new_weights = model.get_weights()[0]
```

```
model.optimizer.get_config()["learning_rate"]
```

```
0.001
```

```
gradient = (old_weights - new_weights)/ 0.001
percent_change = abs(100*(old_weights - new_weights)/ old_weights)
```

```
gradient
```

```
array([[ 0.01093745,  0.0115037 ,  0.01388788, -0.01996755, -0.05242321,
         -0.00369549,  0.01478195,  0.03504753,  0.02288818, -0.00494719],
        [-0.02099573, -0.02273917, -0.02834201,  0.04476308,  0.09220838,
          0.00709295, -0.02682209, -0.06717443, -0.05042553,  0.00888109]],
       dtype=float32)
```

```
percent_change
```

```
array([[0.0072271 , 0.00210111, 0.00679461, 0.00370892, 0.36522916,
        0.00057803, 0.00331642, 0.00498917, 0.00324947, 0.00184224],
       [0.01097624, 0.00796773, 0.00666361, 0.01916067, 0.01916059,
        0.00211133, 0.0059982 , 0.01164935, 0.01972343, 0.00127616]],
       dtype=float32)
```

```
old_weights
```

```
array([[-0.33058235,  0.292853  , -0.5027271 , -0.01332176,  0.4663965 ,
         -0.08549386, -0.3767269 ,  0.02521479,  0.17957473,  0.09508276],
        [-0.46751493, -0.5052102 ,  0.618305  , -0.41437078,  0.51485544,
          0.45250946,  0.36829787,  0.66284555,  0.14777309,  0.04143274]],
       dtype=float32)
```

```
new_weights
```

```
array([[ 0.40518048,  0.6488755 , -0.985546  ,  0.6181405 ,  1.034089  ,
         -0.7342173 , -0.5503599 ,  0.18725553,  0.81843793, -0.69256586],
        [-1.5294187 , -1.299696  ,  1.6635121 , -1.4268929 , -0.56944025,
          1.5625011 ,  0.9726272 ,  0.81920713, -0.9686132 ,  1.2659652 ]],
       dtype=float32)
```

```
model = Sequential()

model.add(Dense(10,activation='relu',input_dim=2))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
```

```
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
old_weights = model.get_weights()[0]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 1/100
7/7 [==============================] - 2s 6ms/step - loss: 0.6932 - accuracy: 0.4900
Epoch 2/100
7/7 [==============================] - 0s 6ms/step - loss: 0.6925 - accuracy: 0.5200
Epoch 3/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6914 - accuracy: 0.6950
Epoch 4/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6897 - accuracy: 0.7950
Epoch 5/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6872 - accuracy: 0.8300
Epoch 6/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6835 - accuracy: 0.8350
Epoch 7/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6788 - accuracy: 0.8550
Epoch 8/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6725 - accuracy: 0.8750
Epoch 9/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6635 - accuracy: 0.8850
Epoch 10/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6516 - accuracy: 0.8900
Epoch 11/100
7/7 [==============================] - 0s 3ms/step - loss: 0.6352 - accuracy: 0.9100
Epoch 12/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6122 - accuracy: 0.9100
Epoch 13/100
7/7 [==============================] - 0s 4ms/step - loss: 0.5817 - accuracy: 0.9100
Epoch 14/100
7/7 [==============================] - 0s 4ms/step - loss: 0.5394 - accuracy: 0.9100
Epoch 15/100
7/7 [==============================] - 0s 4ms/step - loss: 0.5003 - accuracy: 0.9100
Epoch 16/100
7/7 [==============================] - 0s 4ms/step - loss: 0.4629 - accuracy: 0.9100
Epoch 17/100
7/7 [==============================] - 0s 4ms/step - loss: 0.4305 - accuracy: 0.9250
Epoch 18/100
7/7 [==============================] - 0s 4ms/step - loss: 0.3991 - accuracy: 0.9300
Epoch 19/100
7/7 [==============================] - 0s 4ms/step - loss: 0.3661 - accuracy: 0.9250
Epoch 20/100
7/7 [==============================] - 0s 4ms/step - loss: 0.3398 - accuracy: 0.9300
Epoch 21/100
7/7 [==============================] - 0s 3ms/step - loss: 0.3108 - accuracy: 0.9300
Epoch 22/100
7/7 [==============================] - 0s 4ms/step - loss: 0.2804 - accuracy: 0.9250
Epoch 23/100
7/7 [==============================] - 0s 4ms/step - loss: 0.2527 - accuracy: 0.9400
Epoch 24/100
7/7 [==============================] - 0s 3ms/step - loss: 0.2123 - accuracy: 0.9450
Epoch 25/100
7/7 [==============================] - 0s 3ms/step - loss: 0.2030 - accuracy: 0.9250
Epoch 26/100
7/7 [==============================] - 0s 4ms/step - loss: 0.1809 - accuracy: 0.9450
Epoch 27/100
7/7 [==============================] - 0s 4ms/step - loss: 0.1642 - accuracy: 0.9500
Epoch 28/100
7/7 [==============================] - 0s 4ms/step - loss: 0.1532 - accuracy: 0.9500
Epoch 29/100
7/7 [==============================] - 0s 4ms/step - loss: 0.1352 - accuracy: 0.9550
```

```
new_weights = model.get_weights()[0]
```

```
model.optimizer.get_config()["learning_rate"]
```

```
0.001
```

```
gradient = (old_weights - new_weights)/ 0.001
percent_change = abs(100*(old_weights - new_weights)/ old_weights)
```

```
gradient
```

```
array([[ 6.685495  , -3.4646986 , -5.3832526 ,  6.741613  , -6.8466363 ,
         5.8019753 , -6.013676  ,  2.311021  ,  0.31654534,  1.327753  ],
       [-5.534008  ,  2.8640625 ,  4.297107  , -6.487876  ,  6.334662  ,
        -5.206227  ,  0.72604764, -4.4065375 , -1.9274204 , -6.5084095 ]],
      dtype=float32)
```

```
percent_change
```

```
array([[ 1.1875801 ,  0.61953056,  1.5359294 ,  1.9122909 ,  1.6018827 ,
         1.2191677 ,  5.3679805 ,  0.7799734 ,  0.15018749,  0.25854072],
       [ 5.258102  ,  0.9604039 ,  1.1871884 ,  1.2909728 ,  1.2937336 ,
         0.87578124, 26.933157  , 12.894475  ,  1.8821074 ,  2.2091355 ]],
      dtype=float32)
```

```
old_weights
```

```
array([[-0.06470567,  0.38120002,  0.2564506 ,  0.4796242 , -0.41284758,
         0.12111282,  0.3640564 , -0.11997336, -0.7036418 , -0.12918228],
       [ 0.4177161 , -0.646973  ,  0.5538413 ,  0.14968556, -0.1020028 ,
        -0.25845426, -0.1095776 ,  0.22031492, -0.01622504, -0.3014395 ]],
      dtype=float32)
```

```
new_weights
```

```
array([[-0.00357266,  0.42682663,  0.36751232,  0.6510883 , -0.47869346,
         0.22171871,  0.4815479 , -0.20610414, -0.73377985, -0.09355866],
       [ 0.4225647 , -0.76475453,  0.56343955,  0.08604728, -0.1019215 ,
        -0.35053518, -0.19012658,  0.28164935,  0.21199906, -0.29813236]],
      dtype=float32)
```

```
Start coding or generate with AI.
```

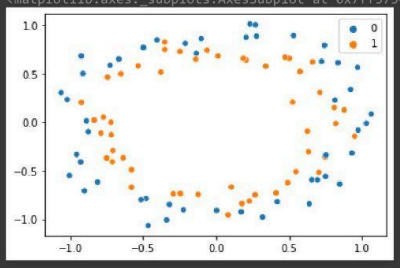
EARLY STOPPING

```
import tensorflow as tf
import numpy as np
import pandas as pd
from pylab import rcParams
import matplotlib.pyplot as plt
import warnings
from mlxtend.plotting import plot_decision_regions
from matplotlib.colors import ListedColormap
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_circles
import seaborn as sns
```

```
X, y = make_circles(n_samples=100, noise=0.1, random_state=1)
```

```
sns.scatterplot(X[:,0],X[:,1],hue=y)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7ff373ca8350>
```



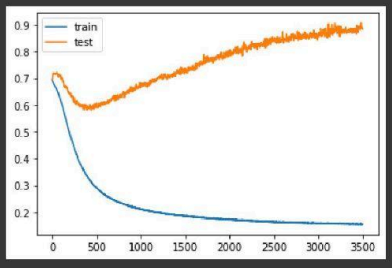

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=2)
```

```
model = Sequential()

model.add(Dense(256, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

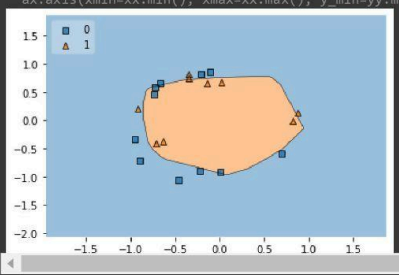
```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3500, verbose=0)
```

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```




```
plot_decision_regions(X_test, y_test.ravel(), clf = model, legend=2)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



Early Stopping

```
model = Sequential()

model.add(Dense(256, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

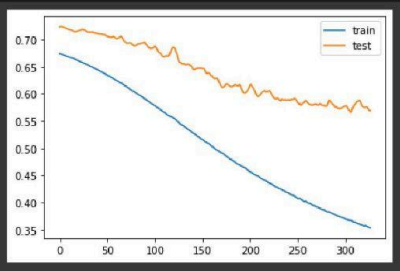
```
callback = EarlyStopping(
    monitor="val_loss",
    min_delta=0.00001,
    patience=20,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=False
)
```

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3500, callbacks=callback)
```

```
Epoch 1/3500
3/3 [==============================] - 0s 27ms/step - loss: 0.6737 - accuracy: 0.5750 - val_loss: 0.7236 - val_accuracy: 0.4000
Epoch 2/3500
3/3 [==============================] - 0s 16ms/step - loss: 0.6733 - accuracy: 0.5750 - val_loss: 0.7238 - val_accuracy: 0.4000
Epoch 3/3500
3/3 [==============================] - 0s 12ms/step - loss: 0.6728 - accuracy: 0.5750 - val_loss: 0.7240 - val_accuracy: 0.4000
Epoch 4/3500
3/3 [==============================] - 0s 16ms/step - loss: 0.6720 - accuracy: 0.5750 - val_loss: 0.7234 - val_accuracy: 0.4000
Epoch 5/3500
3/3 [==============================] - 0s 14ms/step - loss: 0.6712 - accuracy: 0.5625 - val_loss: 0.7230 - val_accuracy: 0.4000
Epoch 6/3500
3/3 [==============================] - 0s 22ms/step - loss: 0.6706 - accuracy: 0.5750 - val_loss: 0.7221 - val_accuracy: 0.4000
Epoch 7/3500
3/3 [==============================] - 0s 14ms/step - loss: 0.6701 - accuracy: 0.5750 - val_loss: 0.7213 - val_accuracy: 0.4000
Epoch 8/3500
3/3 [==============================] - 0s 15ms/step - loss: 0.6693 - accuracy: 0.5750 - val_loss: 0.7205 - val_accuracy: 0.4000
Epoch 9/3500
3/3 [==============================] - 0s 14ms/step - loss: 0.6687 - accuracy: 0.5750 - val_loss: 0.7198 - val_accuracy: 0.4000
Epoch 10/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6681 - accuracy: 0.5875 - val_loss: 0.7187 - val_accuracy: 0.4000
Epoch 11/3500
3/3 [==============================] - 0s 21ms/step - loss: 0.6678 - accuracy: 0.5875 - val_loss: 0.7180 - val_accuracy: 0.4000
Epoch 12/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6669 - accuracy: 0.5875 - val_loss: 0.7178 - val_accuracy: 0.4000
Epoch 13/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6663 - accuracy: 0.6000 - val_loss: 0.7175 - val_accuracy: 0.4000
Epoch 14/3500
3/3 [==============================] - 0s 15ms/step - loss: 0.6659 - accuracy: 0.5875 - val_loss: 0.7166 - val_accuracy: 0.4000
Epoch 15/3500
3/3 [==============================] - 0s 15ms/step - loss: 0.6649 - accuracy: 0.6125 - val_loss: 0.7155 - val_accuracy: 0.4500
Epoch 16/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6645 - accuracy: 0.6125 - val_loss: 0.7141 - val_accuracy: 0.4500
Epoch 17/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6634 - accuracy: 0.6250 - val_loss: 0.7140 - val_accuracy: 0.4500
Epoch 18/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6628 - accuracy: 0.6250 - val_loss: 0.7143 - val_accuracy: 0.4500
Epoch 19/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6620 - accuracy: 0.6250 - val_loss: 0.7147 - val_accuracy: 0.4500
Epoch 20/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6614 - accuracy: 0.6250 - val_loss: 0.7157 - val_accuracy: 0.4500
Epoch 21/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6605 - accuracy: 0.6250 - val_loss: 0.7168 - val_accuracy: 0.4500
Epoch 22/3500
```

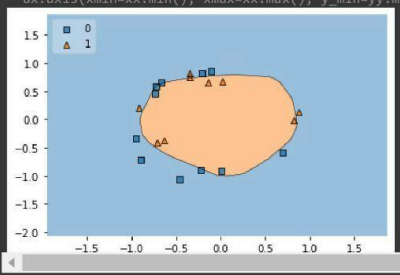
```
3/3 [==============================] - 0s 13ms/step - loss: 0.6596 - accuracy: 0.6250 - val_loss: 0.7165 - val_accuracy: 0.4500
Epoch 23/3500
3/3 [==============================] - 0s 14ms/step - loss: 0.6587 - accuracy: 0.6125 - val_loss: 0.7168 - val_accuracy: 0.4500
Epoch 24/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6584 - accuracy: 0.6125 - val_loss: 0.7183 - val_accuracy: 0.4500
Epoch 25/3500
3/3 [==============================] - 0s 14ms/step - loss: 0.6575 - accuracy: 0.6125 - val_loss: 0.7183 - val_accuracy: 0.4000
Epoch 26/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6564 - accuracy: 0.6125 - val_loss: 0.7181 - val_accuracy: 0.4000
Epoch 27/3500
3/3 [==============================] - 0s 13ms/step - loss: 0.6558 - accuracy: 0.6000 - val_loss: 0.7169 - val_accuracy: 0.4000
Epoch 28/3500
3/3 [==============================] - 0s 17ms/step - loss: 0.6549 - accuracy: 0.6125 - val_loss: 0.7165 - val_accuracy: 0.4000
Epoch 29/3500
```

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```




```
plot_decision_regions(X_test, y_test.ravel(), clf = model, legend=2)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```




```
Start coding or generate with AI.
```

Feature Scaling

```
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('/content/Social_Network_Ads.csv')
```

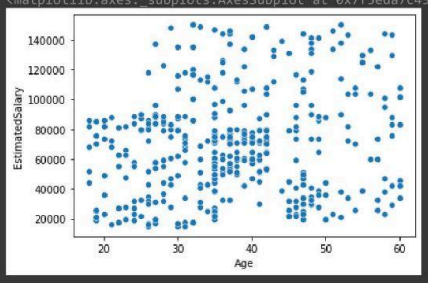
```
df = df.iloc[:,2:]
df.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
import seaborn as sns
```

```
sns.scatterplot(df.iloc[:,0],df.iloc[:,1])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f5eda7c43d0>
```




```
X = df.iloc[:,0:2]
y = df.iloc[:,-1]
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
```

```
model = Sequential()

model.add(Dense(128,activation='relu',input_dim=2))
model.add(Dense(1,activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 128)               384

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 513
Trainable params: 513
Non-trainable params: 0
_________________________________________________________________
```

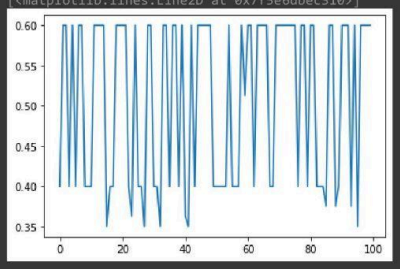
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
history = model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100)
```

```
Epoch 1/100
10/10 [==============================] - 1s 42ms/step - loss: 6689.0693 - accuracy: 0.3469 - val_loss: 3588.6782 - val_accuracy: 0.4000
Epoch 2/100
10/10 [==============================] - 0s 9ms/step - loss: 1788.8799 - accuracy: 0.3719 - val_loss: 828.2932 - val_accuracy: 0.6000
Epoch 3/100
10/10 [==============================] - 0s 9ms/step - loss: 961.7184 - accuracy: 0.6531 - val_loss: 824.4434 - val_accuracy: 0.6000
Epoch 4/100
10/10 [==============================] - 0s 8ms/step - loss: 325.9465 - accuracy: 0.4344 - val_loss: 68.8772 - val_accuracy: 0.4000
Epoch 5/100
10/10 [==============================] - 0s 9ms/step - loss: 105.6520 - accuracy: 0.5188 - val_loss: 5.7623 - val_accuracy: 0.6000
Epoch 6/100
10/10 [==============================] - 0s 8ms/step - loss: 19.1263 - accuracy: 0.5406 - val_loss: 17.4829 - val_accuracy: 0.4000
Epoch 7/100
10/10 [==============================] - 0s 8ms/step - loss: 20.7027 - accuracy: 0.5312 - val_loss: 33.1314 - val_accuracy: 0.6000
Epoch 8/100
10/10 [==============================] - 0s 8ms/step - loss: 43.7668 - accuracy: 0.4406 - val_loss: 26.5325 - val_accuracy: 0.6000
Epoch 9/100
10/10 [==============================] - 0s 9ms/step - loss: 53.5641 - accuracy: 0.5719 - val_loss: 91.2861 - val_accuracy: 0.4000
Epoch 10/100
10/10 [==============================] - 0s 8ms/step - loss: 87.6180 - accuracy: 0.4969 - val_loss: 126.8609 - val_accuracy: 0.4000
Epoch 11/100
10/10 [==============================] - 0s 8ms/step - loss: 83.3905 - accuracy: 0.5094 - val_loss: 112.9638 - val_accuracy: 0.4000
Epoch 12/100
10/10 [==============================] - 0s 9ms/step - loss: 54.7899 - accuracy: 0.5094 - val_loss: 10.9042 - val_accuracy: 0.6000
Epoch 13/100
10/10 [==============================] - 0s 9ms/step - loss: 40.3692 - accuracy: 0.5281 - val_loss: 37.2375 - val_accuracy: 0.6000
Epoch 14/100
10/10 [==============================] - 0s 9ms/step - loss: 67.3726 - accuracy: 0.5281 - val_loss: 176.2580 - val_accuracy: 0.6000
Epoch 15/100
10/10 [==============================] - 0s 8ms/step - loss: 96.3750 - accuracy: 0.5312 - val_loss: 206.0263 - val_accuracy: 0.6000
Epoch 16/100
10/10 [==============================] - 0s 9ms/step - loss: 145.4039 - accuracy: 0.5469 - val_loss: 9.8404 - val_accuracy: 0.3500
Epoch 17/100
10/10 [==============================] - 0s 8ms/step - loss: 121.6079 - accuracy: 0.5125 - val_loss: 114.8439 - val_accuracy: 0.4000
Epoch 18/100
10/10 [==============================] - 0s 9ms/step - loss: 144.6243 - accuracy: 0.6062 - val_loss: 116.7214 - val_accuracy: 0.4000
Epoch 19/100
10/10 [==============================] - 0s 8ms/step - loss: 152.1929 - accuracy: 0.4969 - val_loss: 225.4776 - val_accuracy: 0.6000
Epoch 20/100
10/10 [==============================] - 0s 8ms/step - loss: 131.7005 - accuracy: 0.4844 - val_loss: 193.4258 - val_accuracy: 0.6000
Epoch 21/100
10/10 [==============================] - 0s 9ms/step - loss: 110.8412 - accuracy: 0.5469 - val_loss: 25.9139 - val_accuracy: 0.6000
Epoch 22/100
10/10 [==============================] - 0s 9ms/step - loss: 59.6000 - accuracy: 0.5219 - val_loss: 49.9986 - val_accuracy: 0.6000
Epoch 23/100
10/10 [==============================] - 0s 11ms/step - loss: 58.9580 - accuracy: 0.4688 - val_loss: 15.9721 - val_accuracy: 0.4000
Epoch 24/100
10/10 [==============================] - 0s 9ms/step - loss: 44.8860 - accuracy: 0.5188 - val_loss: 7.2842 - val_accuracy: 0.3625
Epoch 25/100
10/10 [==============================] - 0s 11ms/step - loss: 41.7748 - accuracy: 0.5000 - val_loss: 77.9447 - val_accuracy: 0.6000
Epoch 26/100
10/10 [==============================] - 0s 9ms/step - loss: 60.8529 - accuracy: 0.5219 - val_loss: 32.6075 - val_accuracy: 0.4000
Epoch 27/100
10/10 [==============================] - 0s 8ms/step - loss: 23.4272 - accuracy: 0.5469 - val_loss: 20.3319 - val_accuracy: 0.4000
Epoch 28/100
10/10 [==============================] - 0s 8ms/step - loss: 28.0190 - accuracy: 0.5125 - val_loss: 11.2252 - val_accuracy: 0.3500
Epoch 29/100
10/10 [==============================] - 0s 9ms/step - loss: 35.0810 - accuracy: 0.5688 - val_loss: 8.0743 - val_accuracy: 0.6000
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7f5e6dbec310>]
```




```
# Applying scaling
```

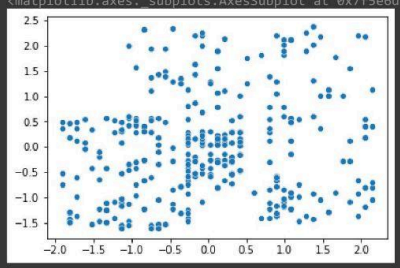
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled
```

```
array([[ 0.88928823, -0.65924556],
       [-0.17254846,  0.87392651],
       [-1.04132394, -0.36440478],
       [ 0.98581884,  0.6085698 ],
       [-0.94479333,  0.57908572],
       [ 0.40663519,  0.01888824],
       [ 0.98581884,  2.11225779],
       [ 0.31010458, -0.30543662],
       [ 1.7580637 , -0.27595254],
       [-0.17254846,  2.20071003],
       [ 1.7580637 ,  1.0213469 ],
       [-1.33091576, -1.48479975],
       [ 2.04765553,  0.54960165],
       [ 1.27541066,  1.90586924],
       [-1.13785454,  0.31372902],
       [-0.36560968, -0.77718187],
       [-1.71703819,  0.49063349],
       [-0.5586709 , -1.51428383],
       [ 0.31010458, -0.71821372],
       [ 0.02051275, -0.57079333],
       [ 0.02051275,  0.04837232],
       [-0.07601785, -0.51182517],
       [-0.6552015 , -1.51428383],
       [ 0.02051275,  0.31372902],
       [ 0.31010458,  0.07785639],
       [-0.46214029, -1.13099081],
       [-0.75173211, -1.54376791],
       [-0.26907907, -0.65924556],
       [-1.13785454,  0.49063349],
       [-0.07601785,  2.20071003],
       [ 0.02051275,  0.04837232],
       [-1.13785454, -1.57325199],
       [ 1.08234944,  0.54960165],
       [-0.26907907, -1.24892713],
       [ 1.37194127, -0.92460227],
       [-1.42744637, -1.21944305],
       [-0.94479333, -0.95408634],
       [ 1.95112492, -0.65924556],
       [ 0.88928823, -0.57079333],
       [-1.13785454,  0.31372902],
       [ 0.02051275, -0.24646847],
       [ 0.79275762, -1.39634752],
       [-0.26907907, -0.36440478],
       [ 0.88928823,  1.2867036 ],
       [ 0.31010458, -0.18750031],
       [-0.26907907, -0.57079333],
       [-0.26907907, -1.39634752],
       [ 1.46847188, -1.04253858],
       [-0.07601785,  0.13682455],
       [-0.84826272, -0.65924556],
       [-0.07601785,  0.01888824],
       [-0.26907907,  0.10734047],
       [ 0.21357397, -0.30543662],
       [-0.26907907,  0.28424494],
       [ 0.11704336,  0.04837232],
       [ 1.95112492,  2.20071003],
       [-1.04132394, -1.45531567],
       [ 0.31010458,  0.31372902],
```

```
sns.scatterplot(X_train_scaled[:,0],X_train_scaled[:,1])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e6d91d950>
```



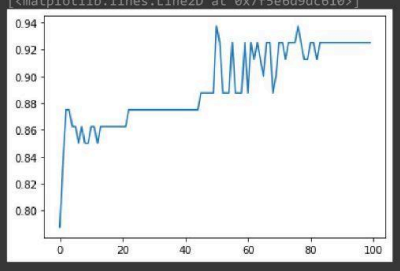

```
model = Sequential()

model.add(Dense(128,activation='relu',input_dim=2))
model.add(Dense(1,activation='sigmoid'))
```

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

history = model.fit(X_train_scaled,y_train,validation_data=(X_test_scaled,y_test),epochs=100)
```

```
Epoch 1/100
10/10 [==============================] - 2s 78ms/step - loss: 0.6879 - accuracy: 0.5437 - val_loss: 0.6411 - val_accuracy: 0.7875
Epoch 2/100
10/10 [==============================] - 0s 9ms/step - loss: 0.6250 - accuracy: 0.7875 - val_loss: 0.5815 - val_accuracy: 0.8375
Epoch 3/100
10/10 [==============================] - 0s 8ms/step - loss: 0.5747 - accuracy: 0.8375 - val_loss: 0.5304 - val_accuracy: 0.8750
Epoch 4/100
10/10 [==============================] - 0s 11ms/step - loss: 0.5297 - accuracy: 0.8469 - val_loss: 0.4881 - val_accuracy: 0.8750
Epoch 5/100
10/10 [==============================] - 0s 8ms/step - loss: 0.4913 - accuracy: 0.8562 - val_loss: 0.4532 - val_accuracy: 0.8625
Epoch 6/100
10/10 [==============================] - 0s 7ms/step - loss: 0.4592 - accuracy: 0.8625 - val_loss: 0.4235 - val_accuracy: 0.8625
Epoch 7/100
10/10 [==============================] - 0s 9ms/step - loss: 0.4317 - accuracy: 0.8719 - val_loss: 0.3989 - val_accuracy: 0.8500
Epoch 8/100
10/10 [==============================] - 0s 10ms/step - loss: 0.4070 - accuracy: 0.8844 - val_loss: 0.3787 - val_accuracy: 0.8625
Epoch 9/100
10/10 [==============================] - 0s 9ms/step - loss: 0.3862 - accuracy: 0.8906 - val_loss: 0.3618 - val_accuracy: 0.8500
Epoch 10/100
10/10 [==============================] - 0s 9ms/step - loss: 0.3681 - accuracy: 0.8906 - val_loss: 0.3471 - val_accuracy: 0.8500
Epoch 11/100
10/10 [==============================] - 0s 8ms/step - loss: 0.3530 - accuracy: 0.8938 - val_loss: 0.3349 - val_accuracy: 0.8625
Epoch 12/100
10/10 [==============================] - 0s 11ms/step - loss: 0.3398 - accuracy: 0.8938 - val_loss: 0.3250 - val_accuracy: 0.8625
Epoch 13/100
10/10 [==============================] - 0s 8ms/step - loss: 0.3280 - accuracy: 0.8969 - val_loss: 0.3175 - val_accuracy: 0.8500
Epoch 14/100
10/10 [==============================] - 0s 8ms/step - loss: 0.3178 - accuracy: 0.8969 - val_loss: 0.3095 - val_accuracy: 0.8625
Epoch 15/100
10/10 [==============================] - 0s 9ms/step - loss: 0.3089 - accuracy: 0.9000 - val_loss: 0.3047 - val_accuracy: 0.8625
Epoch 16/100
10/10 [==============================] - 0s 8ms/step - loss: 0.3014 - accuracy: 0.8969 - val_loss: 0.2987 - val_accuracy: 0.8625
Epoch 17/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2945 - accuracy: 0.9000 - val_loss: 0.2947 - val_accuracy: 0.8625
Epoch 18/100
10/10 [==============================] - 0s 8ms/step - loss: 0.2889 - accuracy: 0.9000 - val_loss: 0.2893 - val_accuracy: 0.8625
Epoch 19/100
10/10 [==============================] - 0s 8ms/step - loss: 0.2831 - accuracy: 0.9000 - val_loss: 0.2868 - val_accuracy: 0.8625
Epoch 20/100
10/10 [==============================] - 0s 10ms/step - loss: 0.2793 - accuracy: 0.9000 - val_loss: 0.2875 - val_accuracy: 0.8625
Epoch 21/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2747 - accuracy: 0.9000 - val_loss: 0.2846 - val_accuracy: 0.8625
Epoch 22/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2711 - accuracy: 0.9031 - val_loss: 0.2819 - val_accuracy: 0.8625
Epoch 23/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2683 - accuracy: 0.8938 - val_loss: 0.2801 - val_accuracy: 0.8750
Epoch 24/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2663 - accuracy: 0.8969 - val_loss: 0.2757 - val_accuracy: 0.8750
Epoch 25/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2634 - accuracy: 0.8969 - val_loss: 0.2752 - val_accuracy: 0.8750
Epoch 26/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2609 - accuracy: 0.8969 - val_loss: 0.2746 - val_accuracy: 0.8750
Epoch 27/100
10/10 [==============================] - 0s 8ms/step - loss: 0.2589 - accuracy: 0.8969 - val_loss: 0.2737 - val_accuracy: 0.8750
Epoch 28/100
10/10 [==============================] - 0s 9ms/step - loss: 0.2574 - accuracy: 0.8969 - val_loss: 0.2734 - val_accuracy: 0.8750
Epoch 29/100
10/10 [==============================] - 0s 8ms/step - loss: 0.2558 - accuracy: 0.9000 - val_loss: 0.2711 - val_accuracy: 0.8750
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7f5e6d9dc610>]
```




```
Start coding or generate with AI.
```

Dropout

```
import numpy as np
import matplotlib.pyplot as plt
```

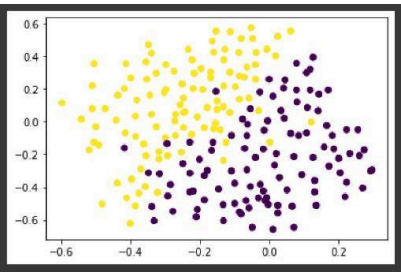
```
X = np.array([[-1.58986e-01,  4.23977e-01],
       [-3.47926e-01,  4.70760e-01],
       [-5.04608e-01,  3.53801e-01],
       [-5.96774e-01,  1.14035e-01],
       [-5.18433e-01, -1.72515e-01],
       [-2.92627e-01, -2.07602e-01],
       [-1.58986e-01, -4.38596e-02],
       [-5.76037e-02,  1.43275e-01],
       [-7.14286e-02,  2.71930e-01],
       [-2.97235e-01,  3.47953e-01],
       [-4.17051e-01,  2.01754e-01],
       [-4.40092e-01,  8.77193e-03],
       [-3.24885e-01, -3.21637e-02],
       [-2.46544e-01,  5.55556e-02],
       [-2.18894e-01,  2.01754e-01],
       [-3.43318e-01,  1.60819e-01],
       [-5.09217e-01,  7.89474e-02],
       [-3.84793e-01, -9.06433e-02],
       [-1.49770e-01,  1.25731e-01],
       [-1.95853e-01,  3.24561e-01],
       [-3.91705e-02, -2.19298e-01],
       [-1.08295e-01, -3.01170e-01],
       [-1.86636e-01, -3.30409e-01],
       [-2.18894e-01, -4.23977e-01],
       [-8.06452e-02, -5.64327e-01],
       [ 6.68203e-02, -5.17544e-01],
       [ 9.44700e-02, -3.24561e-01],
       [ 1.86636e-01, -1.66667e-01],
       [ 6.22120e-02, -7.30994e-02],
       [ 2.07373e-02, -1.95906e-01],
       [ 2.99539e-02, -3.42105e-01],
       [-9.90783e-02, -3.77193e-01],
       [-6.91244e-03, -4.64912e-01],
       [ 1.31336e-01, -4.29825e-01],
       [ 2.32719e-01, -1.95906e-01],
       [ 8.52535e-02, -8.47953e-02],
       [-1.31336e-01, -2.36842e-01],
       [ 2.30415e-03, -1.25731e-01],
       [ 1.22120e-01, -2.92398e-03],
       [-3.47926e-01, -3.12865e-01],
       [-2.28111e-01, -1.25731e-01],
       [-7.60369e-02,  1.46199e-02],
       [ 4.37788e-02,  2.04678e-02],
       [ 1.15207e-02,  1.54971e-01],
       [-4.17051e-01, -1.60819e-01],
       [-3.15668e-01, -3.18713e-01],
       [ 1.26728e-01, -2.19298e-01],
       [ 2.05069e-01, -3.12865e-01],
       [ 2.18894e-01, -4.59064e-01],
       [ 7.14286e-02, -6.46199e-01],
       [-1.31336e-01, -6.05263e-01],
       [-2.09677e-01, -5.81871e-01],
       [-2.28111e-01, -4.29825e-01],
       [-1.45161e-01, -4.12281e-01],
       [-6.68203e-02, -4.82456e-01],
       [ 1.35945e-01, -5.11696e-01],
       [ 2.69585e-01, -4.06433e-01],
       [ 2.97235e-01, -2.95322e-01],
       [ 2.74194e-01, -1.72515e-01],
       [ 2.55760e-01, -4.97076e-02],
       [ 2.23502e-01, -4.97076e-02],
       [ 1.82028e-01, -8.47953e-02],
       [ 1.58986e-01, -1.54971e-01],
       [ 7.14286e-02, -2.13450e-01],
       [ 1.61290e-02, -2.66082e-01],
       [-2.53456e-02, -3.83041e-01],
       [-1.15207e-02, -4.82456e-01],
       [-2.30415e-03, -5.05848e-01],
       [ 2.53456e-02, -5.11696e-01],
       [ 2.53456e-02, -5.58480e-01],
       [ 1.15207e-02, -6.57895e-01],
       [-4.83871e-02, -6.46199e-01],
       [-8.52535e-02, -5.52632e-01],
       [-9.90783e-02, -5.00000e-01],
       [-1.61290e-02, -4.23977e-01],
       [ 1.31336e-01, -3.59649e-01],
       [ 2.23502e-01, -3.71345e-01],
```

```
[ 2.92627e-01, -3.01170e-01],
[ 2.60369e-01, -2.07602e-01],
[ 2.00461e-01, -2.25146e-01],
[ 1.72811e-01, -2.71930e-01],
[-1.31336e-01,  9.06433e-02],
[-1.49770e-01,  7.30994e-02],
[-2.41935e-01,  6.14035e-02],
[-3.01843e-01,  1.78363e-01],
[-2.97235e-01,  1.95906e-01],
[-2.74194e-01,  3.07018e-01],
[-3.24885e-01,  2.95322e-01],
[-3.98618e-01,  2.66082e-01],
[-4.35484e-01,  1.60819e-01],
[-4.72350e-01,  7.89474e-02],
[-3.38710e-01,  4.38596e-02],
[-2.69585e-01,  4.38596e-02],
[-2.55760e-01,  1.02339e-01],
[-1.68203e-01,  2.66082e-01],
[-1.12903e-01,  3.01170e-01],
[-3.91705e-02,  3.47953e-01],
[-1.26728e-01,  4.41520e-01],
[-2.32719e-01,  4.41520e-01],
[-3.38710e-01,  4.18129e-01],
[-4.12442e-01,  3.53801e-01],
[-5.09217e-01,  2.19298e-01],
[-5.41475e-01,  1.46199e-02],
[-5.04608e-01, -1.25731e-01],
[-4.90783e-01, -1.43275e-01],
[-3.61751e-01, -1.37427e-01],
[-2.69585e-01, -8.47953e-02],
[-2.23502e-01, -7.89474e-02],
[-1.86636e-01, -3.80117e-02],
[-1.54378e-01, -8.77193e-03],
[-1.12903e-01,  5.55556e-02],
[-8.52535e-02,  1.37427e-01],
[-8.52535e-02,  2.77778e-01],
[-1.68203e-01,  3.01170e-01],
[-1.91244e-01,  1.95906e-01],
[-1.40553e-01, -4.97076e-02],
[-2.99539e-02,  6.72515e-02],
[-2.00461e-01, -2.30994e-01],
[-1.08295e-01, -8.47953e-02],
[ 3.45622e-02,  6.72515e-02],
[ 8.06452e-02,  1.19883e-01],
[-3.85369e-01,  3.30409e-02],
[-3.81221e-01,  1.31287e-01],
[-3.52189e-01,  2.58187e-01],
[-3.54263e-01,  3.64620e-01],
[-4.14401e-01, -6.92982e-02],
[-4.99424e-01, -3.24561e-02],
[-2.98272e-01, -9.79532e-02],
[-3.16935e-01, -1.83918e-01],
[-3.68779e-01, -2.90351e-01],
[-3.56336e-01, -3.96784e-01],
[-2.71313e-01,  4.38596e-03],
[-1.77995e-01,  8.62573e-02],
[-2.46429e-01,  1.43567e-01],
[-2.50576e-01,  2.29532e-01],
[-2.21544e-01,  3.76901e-01],
[-2.15323e-01,  2.95029e-01],
[-1.30300e-01,  2.17251e-01],
[-2.07028e-01,  2.89474e-02],
[-9.71198e-02,  2.13158e-01],
[-3.90553e-02,  2.58187e-01],
[ 1.90092e-02,  4.01462e-01],
[-3.69816e-02,  4.21930e-01],
[-6.39401e-02,  3.31871e-01],
[-1.30300e-01,  3.76901e-01],
[-3.75000e-01, -4.37719e-01],
[-3.95737e-01, -3.51754e-01],
[-3.54263e-01, -2.08480e-01],
[-4.37212e-01, -3.76316e-01],
[-4.80760e-01, -5.03216e-01],
[-4.10253e-01, -4.66374e-01],
[-2.48502e-01, -2.57602e-01],
[-2.27765e-01, -3.14912e-01],
[-2.83756e-01, -3.84503e-01],
[-2.92051e-01, -4.54094e-01],
[-3.37673e-01, -5.19591e-01],
[-2.77535e-01, -5.48246e-01],
[-2.07028e-01, -5.35965e-01],
[-1.86290e-01, -4.78655e-01],
[-1.32373e-01, -5.07310e-01],
[-1.77995e-01, -2.98538e-01]
```

```
       [-1.65553e-01, -1.75731e-01],
       [-1.61406e-01, -1.26608e-01],
       [ 3.45622e-04,  2.58187e-01],
       [ 7.91475e-02,  3.56433e-01],
       [-2.66129e-02,  1.80409e-01],
       [-5.35714e-02,  7.80702e-02],
       [-1.41705e-02, -5.29240e-02],
       [-7.01613e-02, -1.63450e-01],
       [-6.39401e-02, -2.94444e-01],
       [-3.07604e-02, -4.66374e-01],
       [-5.77189e-02, -5.27778e-01],
       [-5.35714e-02, -3.96784e-01],
       [ 5.21889e-02, -4.17251e-01],
       [-1.62442e-02, -1.67544e-01],
       [-6.39401e-02, -8.56725e-02],
       [-6.18664e-02, -1.60819e-02],
       [-3.80184e-03,  4.38596e-03],
       [ 4.18203e-02,  2.04971e-01],
       [ 7.91475e-02,  1.92690e-01],
       [ 4.59677e-02,  2.54094e-01],
       [ 1.18548e-01,  1.92690e-01],
       [ 1.10253e-01,  8.62573e-02],
       [ 1.08180e-01, -6.92982e-02],
       [ 1.66244e-01, -2.42690e-02],
       [ 1.41359e-01,  6.57895e-02],
       [ 1.43433e-01,  1.68129e-01],
       [ 1.70392e-01,  1.92690e-01],
       [ 1.08180e-01,  2.99123e-01],
       [ 1.18548e-01,  3.19591e-01],
       [ 1.26843e-01,  3.93275e-01],
       [-8.67512e-02,  4.21930e-01],
       [-4.73502e-02,  5.07895e-01],
       [ 2.52304e-02,  5.20175e-01],
       [ 6.25576e-02,  5.52924e-01],
       [-5.87558e-03,  4.42398e-01],
       [-5.14977e-02,  5.73392e-01],
       [-8.05300e-02,  5.07895e-01],
       [-1.53111e-01,  5.52924e-01],
       [-1.11636e-01,  5.48830e-01],
       [-1.63479e-01,  4.91520e-01],
       [-2.52650e-01, -1.88012e-01],
       [-2.46429e-01, -3.65497e-02],
       [-3.21083e-01, -4.33626e-01],
       [-3.31452e-01, -6.05556e-01],
       [-3.85369e-01, -5.15497e-01],
       [-3.99885e-01, -6.21930e-01],
       [-1.24078e-01, -1.26608e-01],
       [-3.16935e-01, -2.28947e-01],
       [-2.94124e-01, -1.34795e-01],
       [-1.53111e-01,  1.84503e-01]])
```

```
y = np.array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 1., 1., 1., 1., 0., 0.])
```

```
import matplotlib.pyplot as plt
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
```

```
model = Sequential()

model.add(Dense(128, input_dim=2, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

adam = Adam(learning_rate=0.01)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history = model.fit(X, y, epochs=500, validation_split = 0.2,verbose=1)
```

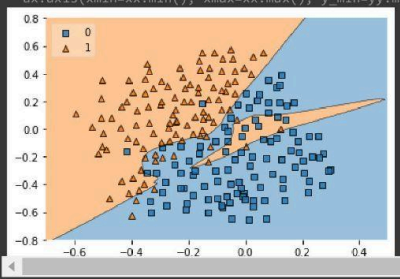
```
Epoch 1/500
6/6 [==============================] - 1s 36ms/step - loss: 0.5667 - accuracy: 0.8393 - val_loss: 0.6492 - val_accuracy: 0.6279
Epoch 2/500
6/6 [==============================] - 0s 9ms/step - loss: 0.3443 - accuracy: 0.8393 - val_loss: 0.5098 - val_accuracy: 0.6977
Epoch 3/500
6/6 [==============================] - 0s 12ms/step - loss: 0.2406 - accuracy: 0.8929 - val_loss: 0.3613 - val_accuracy: 0.8605
Epoch 4/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2168 - accuracy: 0.9167 - val_loss: 0.3645 - val_accuracy: 0.8605
Epoch 5/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2301 - accuracy: 0.8988 - val_loss: 0.4342 - val_accuracy: 0.8605
Epoch 6/500
6/6 [==============================] - 0s 9ms/step - loss: 0.2191 - accuracy: 0.9167 - val_loss: 0.4084 - val_accuracy: 0.8605
Epoch 7/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2567 - accuracy: 0.8869 - val_loss: 0.3422 - val_accuracy: 0.8605
Epoch 8/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2513 - accuracy: 0.8988 - val_loss: 0.4096 - val_accuracy: 0.8372
Epoch 9/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2139 - accuracy: 0.9107 - val_loss: 0.4799 - val_accuracy: 0.7209
Epoch 10/500
6/6 [==============================] - 0s 12ms/step - loss: 0.2231 - accuracy: 0.9107 - val_loss: 0.3972 - val_accuracy: 0.8605
Epoch 11/500
6/6 [==============================] - 0s 13ms/step - loss: 0.2033 - accuracy: 0.9226 - val_loss: 0.3831 - val_accuracy: 0.8605
Epoch 12/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1916 - accuracy: 0.9405 - val_loss: 0.3639 - val_accuracy: 0.8605
Epoch 13/500
6/6 [==============================] - 0s 12ms/step - loss: 0.2005 - accuracy: 0.9405 - val_loss: 0.3492 - val_accuracy: 0.8605
Epoch 14/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1945 - accuracy: 0.9345 - val_loss: 0.4050 - val_accuracy: 0.8605
Epoch 15/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1898 - accuracy: 0.9286 - val_loss: 0.4337 - val_accuracy: 0.8605
Epoch 16/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1908 - accuracy: 0.9226 - val_loss: 0.4427 - val_accuracy: 0.8605
Epoch 17/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1897 - accuracy: 0.9286 - val_loss: 0.3872 - val_accuracy: 0.8605
Epoch 18/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1871 - accuracy: 0.9524 - val_loss: 0.3717 - val_accuracy: 0.8605
Epoch 19/500
6/6 [==============================] - 0s 14ms/step - loss: 0.1961 - accuracy: 0.9345 - val_loss: 0.4220 - val_accuracy: 0.8605
Epoch 20/500
6/6 [==============================] - 0s 13ms/step - loss: 0.1949 - accuracy: 0.9286 - val_loss: 0.3772 - val_accuracy: 0.8605
Epoch 21/500
6/6 [==============================] - 0s 14ms/step - loss: 0.1796 - accuracy: 0.9524 - val_loss: 0.4178 - val_accuracy: 0.8605
Epoch 22/500
6/6 [==============================] - 0s 12ms/step - loss: 0.1835 - accuracy: 0.9345 - val_loss: 0.4119 - val_accuracy: 0.8605
Epoch 23/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1781 - accuracy: 0.9405 - val_loss: 0.4477 - val_accuracy: 0.8605
Epoch 24/500
6/6 [==============================] - 0s 19ms/step - loss: 0.1796 - accuracy: 0.9345 - val_loss: 0.4339 - val_accuracy: 0.8605
Epoch 25/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1763 - accuracy: 0.9464 - val_loss: 0.3802 - val_accuracy: 0.8605
Epoch 26/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1839 - accuracy: 0.9345 - val_loss: 0.4218 - val_accuracy: 0.8605
Epoch 27/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1782 - accuracy: 0.9286 - val_loss: 0.4185 - val_accuracy: 0.8605
Epoch 28/500
6/6 [==============================] - 0s 12ms/step - loss: 0.1849 - accuracy: 0.9226 - val_loss: 0.3679 - val_accuracy: 0.8605
```

```
Epoch 29/500
6/6 [==============================] - 0s 14ms/step - loss: 0.1710 - accuracy: 0.9524 - val_loss: 0.4132 - val_accuracy: 0.8605
```

```
from mlxtend.plotting import plot_decision_regions
```

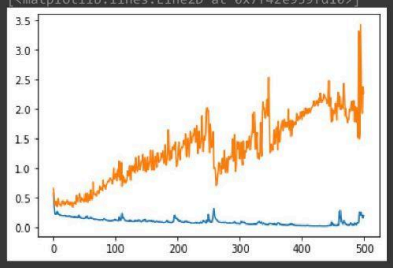
```
plot_decision_regions(X, y.astype('int'), clf=model, legend=2)
plt.xlim(-0.7,0.5)
plt.ylim(-0.8,0.8)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



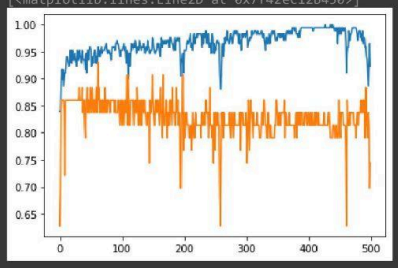

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f42e959fd10>]
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7f42ec12b450>]
```



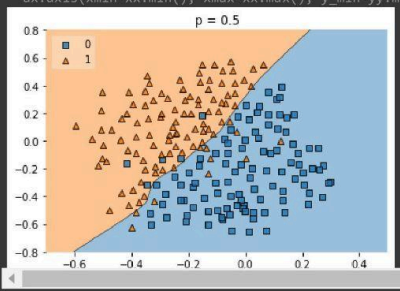

```
model = Sequential()

model.add(Dense(128, input_dim=2, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(1, activation="sigmoid"))

adam = Adam(learning_rate=0.01)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history = model.fit(X, y, epochs=500, validation_split = 0.2,verbose=1)
```

```
Epoch 1/500
6/6 [==============================] - 1s 78ms/step - loss: 0.5846 - accuracy: 0.7738 - val_loss: 0.6484 - val_accuracy: 0.5814
Epoch 2/500
6/6 [==============================] - 0s 6ms/step - loss: 0.3438 - accuracy: 0.8571 - val_loss: 0.9476 - val_accuracy: 0.5814
Epoch 3/500
```

```
6/6 [==============================] - 0s 7ms/step - loss: 0.2886 - accuracy: 0.8631 - val_loss: 0.6692 - val_accuracy: 0.6977
Epoch 4/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2981 - accuracy: 0.8810 - val_loss: 0.4833 - val_accuracy: 0.7442
Epoch 5/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2408 - accuracy: 0.9167 - val_loss: 0.3649 - val_accuracy: 0.8140
Epoch 6/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2541 - accuracy: 0.9107 - val_loss: 0.3456 - val_accuracy: 0.8605
Epoch 7/500
6/6 [==============================] - 0s 7ms/step - loss: 0.2637 - accuracy: 0.8988 - val_loss: 0.3714 - val_accuracy: 0.8372
Epoch 8/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2515 - accuracy: 0.8810 - val_loss: 0.4228 - val_accuracy: 0.8140
Epoch 9/500
6/6 [==============================] - 0s 11ms/step - loss: 0.2342 - accuracy: 0.8929 - val_loss: 0.4598 - val_accuracy: 0.7907
Epoch 10/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2535 - accuracy: 0.8929 - val_loss: 0.4082 - val_accuracy: 0.8372
Epoch 11/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2290 - accuracy: 0.9286 - val_loss: 0.3471 - val_accuracy: 0.8605
Epoch 12/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2100 - accuracy: 0.9226 - val_loss: 0.3436 - val_accuracy: 0.8605
Epoch 13/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2578 - accuracy: 0.8929 - val_loss: 0.3524 - val_accuracy: 0.8605
Epoch 14/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2351 - accuracy: 0.8929 - val_loss: 0.3733 - val_accuracy: 0.8372
Epoch 15/500
6/6 [==============================] - 0s 9ms/step - loss: 0.2520 - accuracy: 0.8988 - val_loss: 0.3792 - val_accuracy: 0.8605
Epoch 16/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2449 - accuracy: 0.8869 - val_loss: 0.4230 - val_accuracy: 0.7907
Epoch 17/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2230 - accuracy: 0.9107 - val_loss: 0.4280 - val_accuracy: 0.8372
Epoch 18/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2269 - accuracy: 0.9167 - val_loss: 0.4135 - val_accuracy: 0.8372
Epoch 19/500
6/6 [==============================] - 0s 10ms/step - loss: 0.2146 - accuracy: 0.9167 - val_loss: 0.3629 - val_accuracy: 0.8605
Epoch 20/500
6/6 [==============================] - 0s 5ms/step - loss: 0.2195 - accuracy: 0.8988 - val_loss: 0.3568 - val_accuracy: 0.8605
Epoch 21/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2201 - accuracy: 0.9107 - val_loss: 0.4109 - val_accuracy: 0.8372
Epoch 22/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2246 - accuracy: 0.9286 - val_loss: 0.3891 - val_accuracy: 0.8605
Epoch 23/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2233 - accuracy: 0.9226 - val_loss: 0.4098 - val_accuracy: 0.8605
Epoch 24/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2355 - accuracy: 0.8988 - val_loss: 0.4258 - val_accuracy: 0.8605
Epoch 25/500
6/6 [==============================] - 0s 9ms/step - loss: 0.2185 - accuracy: 0.9167 - val_loss: 0.4370 - val_accuracy: 0.8372
Epoch 26/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2336 - accuracy: 0.9226 - val_loss: 0.4027 - val_accuracy: 0.8605
Epoch 27/500
6/6 [==============================] - 0s 10ms/step - loss: 0.2089 - accuracy: 0.9167 - val_loss: 0.3827 - val_accuracy: 0.8605
Epoch 28/500
6/6 [==============================] - 0s 7ms/step - loss: 0.2192 - accuracy: 0.9226 - val_loss: 0.3848 - val_accuracy: 0.8605
Epoch 29/500
6/6 [==============================] - 0s 12ms/step - loss: 0.2108 - accuracy: 0.9286 - val_loss: 0.3867 - val_accuracy: 0.8605
```

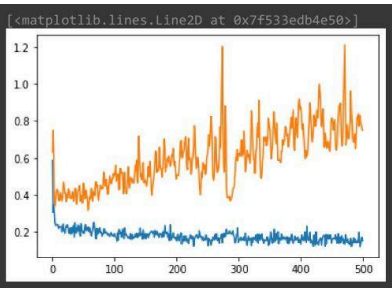
```
plot_decision_regions(X, y.astype('int'), clf=model, legend=2)
plt.xlim(-0.7,0.5)
plt.ylim(-0.8,0.8)
plt.title('p = 0.5')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



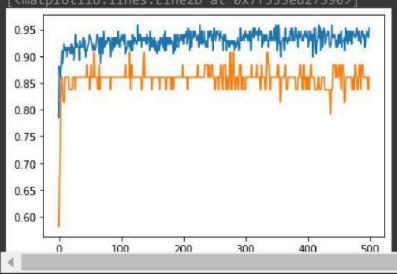

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f533edb4e50>]
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7f533ed27390>]
```



Start coding or generate with AI.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Generate Data

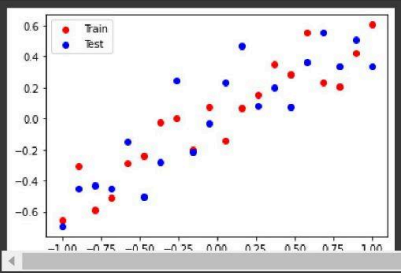
```
X_train = np.linspace(-1, 1, 20)
```

```
y_train = np.array([-0.6561 , -0.3099 , -0.59035, -0.50855, -0.285  ,
                    -0.2443 , -0.02445,  0.00135, -0.2006 ,  0.07475,
                    -0.1422 ,  0.06515,  0.15265,  0.3521 ,  0.28415,
                    0.5524 ,  0.23115,  0.20835, 0.4211,  0.60485])
```

```
X_test = np.linspace(-1, 1, 20)
```

```
y_test = np.array([-0.69415, -0.451  , -0.43005, -0.4484 , -0.1475 ,
                   -0.5019 , -0.28055,  0.24595, -0.21425, -0.0286 ,
                   0.23415,  0.46575, 0.07955,  0.1973 ,  0.0719 ,
                   0.3639 ,  0.5536 ,  0.3365 , 0.50705,  0.33435])
```

```
plt.scatter(X_train, y_train, c='red', label='Train')
plt.scatter(X_test, y_test, c='blue', label='Test')
plt.legend()
plt.show()
```



Regression Model

```
model_1 = Sequential()
model_1.add(Dense(128, input_dim=1, activation="relu"))
model_1.add(Dense(128, activation="relu"))
model_1.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_1.compile(loss='mse', optimizer=adam, metrics=['mse'])
history = model_1.fit(X_train, y_train, epochs=500,
                      validation_data = (X_test, y_test),
                      verbose=False)
```

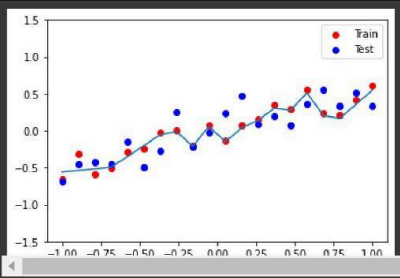
```
# evaluate the model
_, train_mse = model_1.evaluate(X_train, y_train, verbose=0)
_, test_mse = model_1.evaluate(X_test, y_test, verbose=0)
print('Train: {}, Test: {}'.format(train_mse, test_mse))
```

```
Train: 0.004562994930893183, Test: 0.04608117789030075
```

```
y_pred_1 = model_1.predict(X_test)
```

```
plt.figure()
plt.scatter(X_train, y_train, c='red', label='Train')
plt.scatter(X_test, y_test, c='blue', label='Test')
plt.plot(X_test, y_pred_1)
plt.legend()
```

```
plt.ylim((-1.5, 1.5))
plt.show()
```



Dropout Model

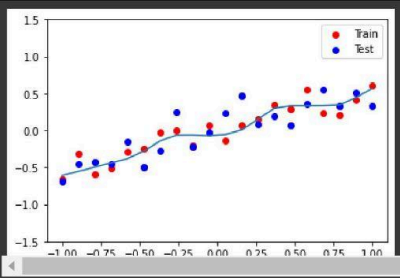
```
model_2 = Sequential()
model_2.add(Dense(128, input_dim=1, activation="relu"))
model_2.add(Dropout(0.2))
model_2.add(Dense(128, activation="relu"))
model_2.add(Dropout(0.2))
model_2.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_2.compile(loss='mse', optimizer=adam, metrics=['mse'])

drop_out_history = model_2.fit(X_train, y_train, epochs=500,
                               validation_data = (X_test, y_test),
                               verbose=False)
```

```
# evaluate the model
_, train_mse = model_2.evaluate(X_train, y_train, verbose=0)
_, test_mse = model_2.evaluate(X_test, y_test, verbose=0)
print('Train: {}, Test: {}'.format(train_mse, test_mse))
```

```
Train: 0.011907287873327732, Test: 0.03752660006284714
```

```
y_pred_2 = model_2.predict(X_test)
```

```
plt.figure()
plt.scatter(X_train, y_train, c='red', label='Train')
plt.scatter(X_test, y_test, c='blue', label='Test')
plt.plot(X_test, y_pred_2)
plt.legend()
plt.ylim((-1.5, 1.5))
plt.show()
```



Start coding or generate with AI.

Zero initialization

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

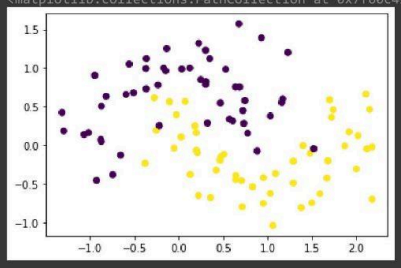
```
df = pd.read_csv('/content/ushape.csv')
```

```
df.head()
```

	X	Y	class
0	0.0316	0.9870	0.0
1	2.1200	-0.0462	1.0
2	0.8820	-0.0758	0.0
3	-0.0551	-0.0373	1.0
4	0.8300	-0.5390	1.0

```
plt.scatter(df['X'],df['Y'],c=df['class'])
```

```
<matplotlib.collections.PathCollection at 0x7f60c42f09d0>
```




```
X = df.iloc[:,0:2].values
y = df.iloc[:,-1].values
```

```
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
```

```
model = Sequential()

model.add(Dense(2,activation='sigmoid',input_dim=2))
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 2)                 6

 dense_13 (Dense)            (None, 1)                 3

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

```
# Set parameters to 0
model.get_weights()
```

```
[array([[ 0.9283565 ,  0.5740107 ],
        [-0.23292029, -0.09779382]], dtype=float32),
 array([0., 0.], dtype=float32),
 array([[-0.46729255],
        [-0.8852598 ]], dtype=float32),
 array([0.], dtype=float32)]
```

```
initial_weights = model.get_weights()
```

```
initial_weights[0] = np.ones(model.get_weights()[0].shape)*0.5
initial_weights[1] = np.ones(model.get_weights()[1].shape)*0.5
initial_weights[2] = np.ones(model.get_weights()[2].shape)*0.5
initial_weights[3] = np.ones(model.get_weights()[3].shape)*0.5
```

```
model.set_weights(initial_weights)
```

```
model.get_weights()
```

```
[array([[0.5, 0.5],
        [0.5, 0.5]], dtype=float32),
 array([0.5, 0.5], dtype=float32),
 array([[0.5],
        [0.5]], dtype=float32),
 array([0.5], dtype=float32)]
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
history = model.fit(X,y,epochs=100,validation_split=0.2)
```

```
Epoch 1/100
3/3 [==============================] - 1s 97ms/step - loss: 0.8635 - accuracy: 0.5000 - val_loss: 0.8362 - val_accuracy: 0.5000
Epoch 2/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8613 - accuracy: 0.5000 - val_loss: 0.8340 - val_accuracy: 0.5000
Epoch 3/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8590 - accuracy: 0.5000 - val_loss: 0.8319 - val_accuracy: 0.5000
Epoch 4/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8568 - accuracy: 0.5000 - val_loss: 0.8299 - val_accuracy: 0.5000
Epoch 5/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8547 - accuracy: 0.5000 - val_loss: 0.8278 - val_accuracy: 0.5000
Epoch 6/100
3/3 [==============================] - 0s 19ms/step - loss: 0.8525 - accuracy: 0.5000 - val_loss: 0.8257 - val_accuracy: 0.5000
Epoch 7/100
3/3 [==============================] - 0s 24ms/step - loss: 0.8503 - accuracy: 0.5000 - val_loss: 0.8237 - val_accuracy: 0.5000
Epoch 8/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8482 - accuracy: 0.5000 - val_loss: 0.8217 - val_accuracy: 0.5000
Epoch 9/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8460 - accuracy: 0.5000 - val_loss: 0.8197 - val_accuracy: 0.5000
Epoch 10/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8439 - accuracy: 0.5000 - val_loss: 0.8177 - val_accuracy: 0.5000
Epoch 11/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8419 - accuracy: 0.5000 - val_loss: 0.8158 - val_accuracy: 0.5000
Epoch 12/100
3/3 [==============================] - 0s 17ms/step - loss: 0.8398 - accuracy: 0.5000 - val_loss: 0.8138 - val_accuracy: 0.5000
Epoch 13/100
3/3 [==============================] - 0s 19ms/step - loss: 0.8377 - accuracy: 0.5000 - val_loss: 0.8119 - val_accuracy: 0.5000
Epoch 14/100
3/3 [==============================] - 0s 19ms/step - loss: 0.8359 - accuracy: 0.5000 - val_loss: 0.8100 - val_accuracy: 0.5000
Epoch 15/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8337 - accuracy: 0.5000 - val_loss: 0.8082 - val_accuracy: 0.5000
Epoch 16/100
3/3 [==============================] - 0s 19ms/step - loss: 0.8319 - accuracy: 0.5000 - val_loss: 0.8063 - val_accuracy: 0.5000
Epoch 17/100
3/3 [==============================] - 0s 18ms/step - loss: 0.8298 - accuracy: 0.5000 - val_loss: 0.8045 - val_accuracy: 0.5000
Epoch 18/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8280 - accuracy: 0.5000 - val_loss: 0.8027 - val_accuracy: 0.5000
Epoch 19/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8259 - accuracy: 0.5000 - val_loss: 0.8009 - val_accuracy: 0.5000
Epoch 20/100
3/3 [==============================] - 0s 24ms/step - loss: 0.8240 - accuracy: 0.5000 - val_loss: 0.7991 - val_accuracy: 0.5000
Epoch 21/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8224 - accuracy: 0.5000 - val_loss: 0.7973 - val_accuracy: 0.5000
Epoch 22/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8202 - accuracy: 0.5000 - val_loss: 0.7955 - val_accuracy: 0.5000
Epoch 23/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8184 - accuracy: 0.5000 - val_loss: 0.7938 - val_accuracy: 0.5000
Epoch 24/100
3/3 [==============================] - 0s 17ms/step - loss: 0.8167 - accuracy: 0.5000 - val_loss: 0.7920 - val_accuracy: 0.5000
Epoch 25/100
3/3 [==============================] - 0s 17ms/step - loss: 0.8148 - accuracy: 0.5000 - val_loss: 0.7904 - val_accuracy: 0.5000
Epoch 26/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8130 - accuracy: 0.5000 - val_loss: 0.7888 - val_accuracy: 0.5000
Epoch 27/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8114 - accuracy: 0.5000 - val_loss: 0.7872 - val_accuracy: 0.5000
Epoch 28/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8096 - accuracy: 0.5000 - val_loss: 0.7856 - val_accuracy: 0.5000
Epoch 29/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8081 - accuracy: 0.5000 - val_loss: 0.7841 - val_accuracy: 0.5000
```

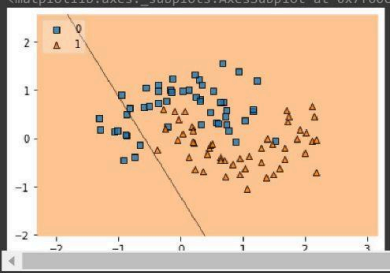
```
model.get_weights()
```

```
[array([[0.7145846, 0.7145846],
        [0.2454365, 0.2454365]], dtype=float32),
 array([0.27247417, 0.27247417], dtype=float32),
 array([[0.26374042],
```

```
           [0.26374042]], dtype=float32),
     array([0.24974917], dtype=float32)]
```

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X,y.astype('int'), clf=model, legend=2)
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:242: UserWarning: No contour levels were found within the data range.
  antialiased=True)
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
<matplotlib.axes._subplots.AxesSubplot at 0x7f60c413bdd0>
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('/content/ushape.csv')
```

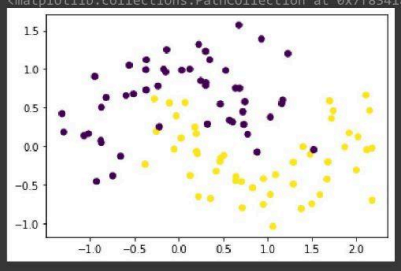
```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-a7eef0a23fe4> in <module>()
----> 1 df = pd.read_csv('/content/ushape.csv')

                                  7 frames
/usr/local/lib/python3.7/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors,
storage_options)
    705                 encoding=ioargs.encoding,
    706                 errors=errors,
--> 707                 newline="",
    708             )
    709         else:

FileNotFoundError: [Errno 2] No such file or directory: '/content/ushape.csv'
```

```
df.head()
```

	X	Y	class
0	0.0316	0.9870	0.0
1	2.1200	-0.0462	1.0
2	0.8820	-0.0758	0.0
3	-0.0551	-0.0373	1.0
4	0.8300	-0.5390	1.0

```
plt.scatter(df['X'],df['Y'],c=df['class'])
```

```
<matplotlib.collections.PathCollection at 0x7f8341a81d50>
```




```
X = df.iloc[:,0:2].values
y = df.iloc[:,-1].values
```

```
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
```

```
model = Sequential()

model.add(Dense(10,activation='sigmoid',input_dim=2))
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 10)                30

 dense_15 (Dense)            (None, 1)                 11

=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

```
# Set parameters to 0
model.get_weights()
```

```
[array([[ 0.20610076, -0.4179246 ,  0.33154517,  0.09572625,  0.58029276,
          0.49422508,  0.3457225 ,  0.45362252,  0.55885166,  0.6358045 ],
        [ 0.3447644 ,  0.23532373, -0.24924499,  0.38395554,  0.5557688 ,
         -0.34025267, -0.42193514,  0.53247017, -0.5466664 ,  0.15849692]],
       dtype=float32),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
 array([[-0.32940134],
        [-0.67858446],
        [-0.18315494],
        [ 0.29897088],
        [ 0.00572836],
        [-0.21986455],
        [ 0.70618635],
        [-0.60569775],
        [-0.64798903],
        [ 0.11328101]], dtype=float32),
 array([0.], dtype=float32)]
```

```
initial_weights = model.get_weights()
```

```
initial_weights[0] = np.zeros(model.get_weights()[0].shape)
initial_weights[1] = np.zeros(model.get_weights()[1].shape)
initial_weights[2] = np.zeros(model.get_weights()[2].shape)
initial_weights[3] = np.zeros(model.get_weights()[3].shape)
```

```
model.set_weights(initial_weights)
```

```
model.get_weights()
```

```
[array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
 array([[0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.]], dtype=float32),
 array([0.], dtype=float32)]
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
history = model.fit(X,y,epochs=100,validation_split=0.2)
```

```
Epoch 1/100
3/3 [==============================] - 1s 247ms/step - loss: 0.6934 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2/100
3/3 [==============================] - 0s 59ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/100
3/3 [==============================] - 0s 35ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/100
3/3 [==============================] - 0s 31ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/100
3/3 [==============================] - 0s 35ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/100
3/3 [==============================] - 0s 20ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7/100
3/3 [==============================] - 0s 17ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8/100
3/3 [==============================] - 0s 18ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9/100
3/3 [==============================] - 0s 18ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10/100
3/3 [==============================] - 0s 17ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 11/100
3/3 [==============================] - 0s 17ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 12/100
3/3 [==============================] - 0s 16ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 13/100
3/3 [==============================] - 0s 28ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 14/100
3/3 [==============================] - 0s 57ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 15/100
3/3 [==============================] - 0s 65ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 16/100
3/3 [==============================] - 0s 57ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.5000
```

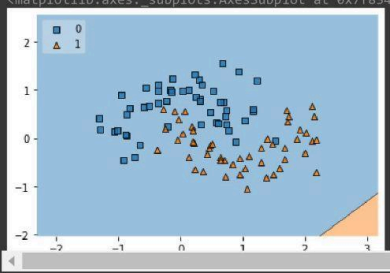
```
Epoch 17/100
3/3 [==============================] - 0s 64ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6929 - val_accuracy: 0.5000
Epoch 18/100
3/3 [==============================] - 0s 57ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6929 - val_accuracy: 0.5000
Epoch 19/100
3/3 [==============================] - 0s 23ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6929 - val_accuracy: 0.5000
Epoch 20/100
3/3 [==============================] - 0s 33ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6928 - val_accuracy: 0.5000
Epoch 21/100
3/3 [==============================] - 0s 37ms/step - loss: 0.6929 - accuracy: 0.5000 - val_loss: 0.6928 - val_accuracy: 0.5000
Epoch 22/100
3/3 [==============================] - 0s 13ms/step - loss: 0.6928 - accuracy: 0.5000 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 23/100
3/3 [==============================] - 0s 14ms/step - loss: 0.6928 - accuracy: 0.5000 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 24/100
3/3 [==============================] - 0s 14ms/step - loss: 0.6927 - accuracy: 0.5000 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 25/100
3/3 [==============================] - 0s 17ms/step - loss: 0.6927 - accuracy: 0.5000 - val_loss: 0.6925 - val_accuracy: 0.5000
Epoch 26/100
3/3 [==============================] - 0s 15ms/step - loss: 0.6928 - accuracy: 0.5000 - val_loss: 0.6924 - val_accuracy: 0.5000
Epoch 27/100
3/3 [==============================] - 0s 15ms/step - loss: 0.6926 - accuracy: 0.5000 - val_loss: 0.6923 - val_accuracy: 0.5000
Epoch 28/100
3/3 [==============================] - 0s 14ms/step - loss: 0.6926 - accuracy: 0.5000 - val_loss: 0.6922 - val_accuracy: 0.5000
Epoch 29/100
3/3 [==============================] - 0s 15ms/step - loss: 0.6925 - accuracy: 0.5000 - val_loss: 0.6921 - val_accuracy: 0.5000
```

```
model.get_weights()
```

```
[array([[-0.4567254 , -0.4567254 , -0.4567254 , -0.4567254 , -0.4567254 ,
         -0.4567254 , -0.4567254 , -0.4567254 , -0.45672542, -0.45672542],
        [ 0.47703952,  0.47703952,  0.47703952,  0.47703952,  0.47703952,
          0.47703952,  0.47703952,  0.47703952,  0.47703955,  0.47703955]],
       dtype=float32),
 array([-0.10217472, -0.10217472, -0.10217472, -0.10217472, -0.10217472,
        -0.10217472, -0.10217472, -0.10217472, -0.10217471, -0.10217471],
       dtype=float32),
 array([[-0.06693693],
        [-0.06693693],
        [-0.06693693],
        [-0.06693693],
        [-0.06693693],
        [-0.06693693],
        [-0.06693693],
        [-0.06693693],
        [-0.06693694],
        [-0.06693694]], dtype=float32),
 array([0.07306698], dtype=float32)]
```

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X,y.astype('int'), clf=model, legend=2)
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
<matplotlib.axes._subplots.AxesSubplot at 0x7f8343028e90>
```

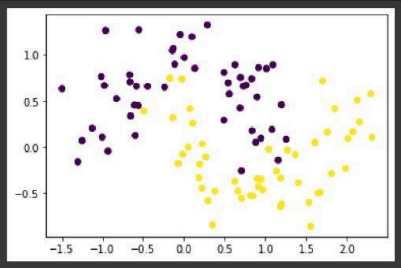



```
Start coding or generate with AI.
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
import seaborn as sns
from mlxtend.plotting import plot_decision_regions

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
```

```
X, y = make_moons(100, noise=0.25,random_state=2)
```

```
import matplotlib.pyplot as plt
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```




```
model1 = Sequential()

model1.add(Dense(128,input_dim=2, activation="relu"))
model1.add(Dense(128, activation="relu"))
model1.add(Dense(1,activation='sigmoid'))

model1.summary()
```

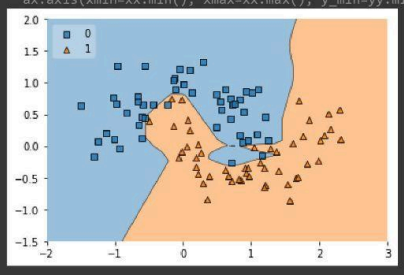
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 128)               384

 dense_7 (Dense)             (None, 128)               16512

 dense_8 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,025
Trainable params: 17,025
Non-trainable params: 0
_________________________________________________________________
```

```
adam = Adam(learning_rate=0.01)
model1.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history1 = model1.fit(X, y, epochs=2000, validation_split = 0.2,verbose=0)
```

```
plot_decision_regions(X, y.astype('int'), clf=model1, legend=2)
plt.xlim(-2,3)
plt.ylim(-1.5,2)
plt.show()
```

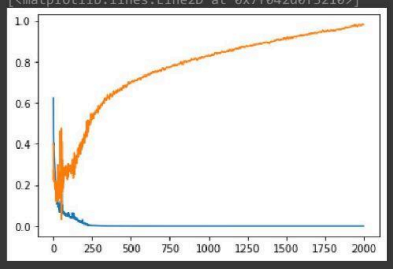
Regularization

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```




```
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f042d0f3210>]
```



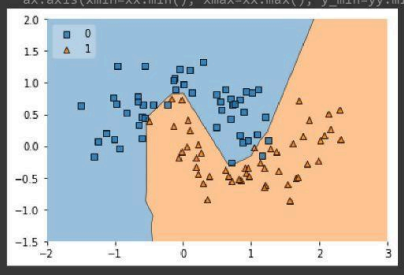

```
model2 = Sequential()

model2.add(Dense(128,input_dim=2, activation="relu",kernel_regularizer=tensorflow.keras.regularizers.l1(0.001)))
model2.add(Dense(128, activation="relu",kernel_regularizer=tensorflow.keras.regularizers.l1(0.001)))
model2.add(Dense(1,activation='sigmoid'))

model2.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 128)               384

 dense_22 (Dense)            (None, 128)               16512

 dense_23 (Dense)            (None, 1)                 129

=================================================================
Total params: 17,025
Trainable params: 17,025
Non-trainable params: 0
_________________________________________________________________
```

```
adam = Adam(learning_rate=0.01)
model2.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history2 = model2.fit(X, y, epochs=2000, validation_split = 0.2,verbose=0)
```

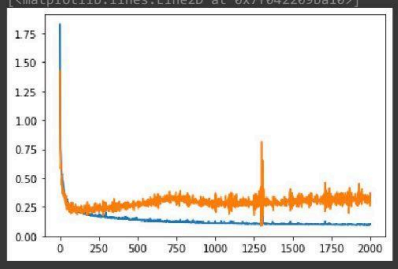
```
plot_decision_regions(X, y.astype('int'), clf=model2, legend=2)
plt.xlim(-2,3)
plt.ylim(-1.5,2)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```




```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
```

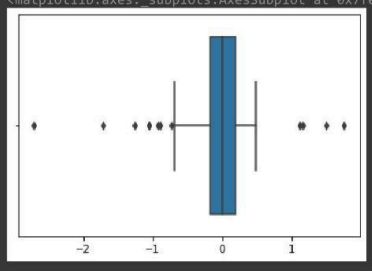
```
[<matplotlib.lines.Line2D at 0x7f042209ba10>]
```



```
model1_weight_layer1 = model1.get_weights()[0].reshape(256)
model2_weight_layer1 = model2.get_weights()[0].reshape(256)
```

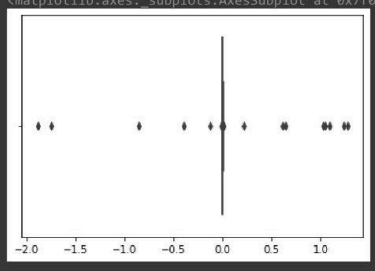
```
sns.boxplot(model1_weight_layer1)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f04220b8b10>
```



```
sns.boxplot(model2_weight_layer1)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f042202c390>
```



```
model1_weight_layer1.min()
```

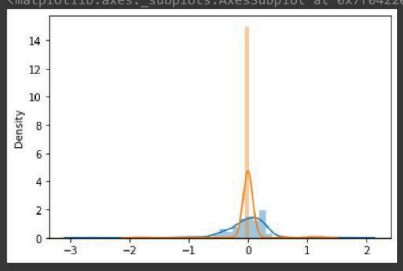
```
-2.7185578
```

```
model2_weight_layer1.min()
```

```
-1.8934479
```

```
sns.distplot(model1_weight_layer1)
sns.distplot(model2_weight_layer1)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a futu
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a futu
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0422018610>
```




```
model1.get_weights()[0].reshape(256)
```

```
array([ 8.36581886e-02,  3.62687036e-02, -4.17765856e-01, -3.93163204e-01,
       -3.96005869e-01,  2.13018522e-01,  2.28988439e-01, -3.22141826e-01,
        2.23829538e-01,  1.97904125e-01, -5.38365006e-01,  1.70224056e-01,
       -1.77298978e-01,  4.41261902e-02,  6.83471262e-02,  2.14902475e-01,
        4.95364517e-02,  8.23249444e-02, -5.02141356e-01, -1.56526896e-03,
        9.14724991e-02, -2.30933651e-02,  2.13529035e-01,  9.54458341e-02,
       -2.91170686e-01,  1.57895297e-01, -5.04917026e-01,  2.12428153e-01,
       -2.12982274e-03,  2.03352317e-01,  1.95124984e-01, -5.25625683e-02,
       -3.26485872e-01,  2.97886580e-01, -6.85776353e-01,  2.16803566e-01,
       -2.26653010e-01,  2.15190932e-01,  2.18419120e-01,  9.44057107e-02,
       -3.32020015e-01,  2.72148997e-01, -4.05541629e-01,  2.10304454e-01,
       -3.53123620e-02,  2.39017248e-01,  1.97818264e-01,  4.17221524e-02,
        2.47686148e-01,  2.60426432e-01,  7.42920935e-02, -1.97996497e-02,
        2.49996199e-03,  7.39424825e-02, -3.76863003e-01, -3.45605701e-01,
        2.64541715e-01,  1.32041499e-01,  2.02703670e-01, -5.18711984e-01,
       -1.18189253e-01,  1.92909852e-01,  6.19973317e-02,  1.69551060e-01,
        1.87287569e-01, -4.96824086e-01,  6.72358125e-02, -4.61391211e-01,
        2.53824741e-01, -4.05834556e-01,  1.26976356e-01, -5.35438359e-01,
        2.17906699e-01, -4.61450815e-01,  1.40626235e-02,  8.39711912e-03,
        1.33992940e-01,  2.37961203e-01, -3.23832154e-01,  2.20816463e-01,
        1.54505417e-01,  2.60821521e-01, -4.87969033e-02, -1.69064537e-01,
       -2.45658420e-02,  2.68627226e-01, -3.08435112e-02, -5.11454821e-01,
        1.59141541e-01, -3.70742589e-01,  7.88443834e-02,  4.04660106e-02,
        1.14555068e-01,  6.63752928e-02,  9.02682170e-03, -2.60784894e-01,
       -4.40374583e-01, -1.14047769e-02, -3.74770425e-02,  1.68620914e-01,
        1.78412095e-01,  3.17960113e-01, -1.22944184e-01,  2.22067803e-01,
        2.16269627e-01, -1.55968731e-02, -1.53711125e-01,  1.20375909e-01,
       -2.82058775e-01, -4.21607405e-01, -2.38318786e-01, -3.87454391e-01,
       -5.39283343e-02, -5.19582815e-02,  2.20146533e-02,  2.19618052e-01,
       -3.93210024e-01,  5.31659797e-02,  1.20239288e-01, -1.07759438e-01,
        6.92361444e-02,  2.54981816e-01,  2.05702752e-01, -4.69989777e-01,
       -2.02217568e-02,  2.18708590e-01, -3.57157469e-01, -4.57049876e-01,
        1.50024498e+00,  4.97291982e-01,  1.08143896e-01, -2.13980023e-02,
       -3.68858159e-01, -1.55548185e-01, -2.20214128e-01, -3.67496133e-01,
        3.76471758e-01, -1.12438962e-01, -1.42594591e-01,  2.63052106e-01,
        2.86761671e-01, -1.75136283e-01,  1.42654404e-01, -1.64357230e-01,
       -2.05702230e-01, -3.31739932e-01, -6.89846743e-03,  1.20316848e-01,
        2.39299849e-01,  2.02913046e-01, -1.23616964e-01,  1.96781605e-02,
        1.68995097e-01,  8.50129426e-02, -7.56526217e-02, -1.47868231e-01,
       -1.22441379e-02,  1.24331288e-01,  8.93589929e-02, -3.18250328e-01,
        1.45956382e-01,  1.50910780e-01, -1.24292299e-01, -1.20857731e-01,
        3.02440643e-01, -1.56325176e-01, -8.48699063e-02,  1.12520742e+00,
        2.29321763e-01, -1.60620332e-01,  2.70279080e-01, -1.20651469e-01,
        2.99278051e-01,  4.02042896e-01, -1.20495699e-01, -2.19048098e-01,
       -1.40762851e-01, -2.05702767e-01, -4.23372239e-02, -8.96272540e-01,
       -1.72349977e+00, -3.21702920e-02, -4.25565869e-01,  1.55981630e-01,
       -2.06218705e-01,  2.18112305e-01,  9.11024734e-02, -1.53775085e-02,
        2.21733779e-01, -1.11189082e-01, -5.94162345e-02, -7.17675462e-02,
       -1.28930464e-01,  1.42400444e-04,  1.16128314e+00,  1.24267630e-01,
       -1.96727023e-01,  2.12383211e-01,  3.87428761e-01, -1.30678445e-01,
        3.66538137e-01,  2.10566714e-01, -9.19768572e-01,  1.75965977e+00,
        2.43212238e-01, -1.89635113e-01,  1.17535554e-01, -1.66406915e-01,
       -7.35416561e-02,  4.38676953e-01, -1.25603628e+00,  2.58704990e-01,
       -1.68005824e-01, -1.97434500e-01,  1.37823373e-01,  2.18255166e-02,
       -6.31580055e-02, -3.56804490e-01, -1.04981947e+00, -4.25514160e-03,
```

Weight initialization

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

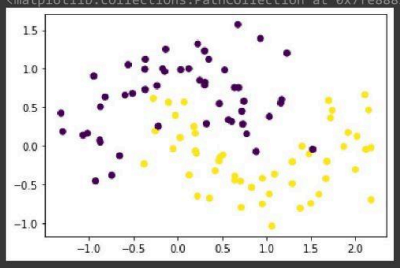
```
df = pd.read_csv('/content/ushape.csv')
```

```
df.head()
```

	X	Y	class
0	0.0316	0.9870	0.0
1	2.1200	-0.0462	1.0
2	0.8820	-0.0758	0.0
3	-0.0551	-0.0373	1.0
4	0.8300	-0.5390	1.0

```
plt.scatter(df['X'],df['Y'],c=df['class'])
```

```
<matplotlib.collections.PathCollection at 0x7fe8883fae50>
```




```
X = df.iloc[:,0:2].values
y = df.iloc[:,-1].values
```

```
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
```

```
model = Sequential()

model.add(Dense(10,activation='relu',input_dim=2,kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu',kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu',kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu',kernel_initializer='he_normal'))
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 10)                30

 dense_23 (Dense)            (None, 10)                110

 dense_24 (Dense)            (None, 10)                110

 dense_25 (Dense)            (None, 10)                110

 dense_26 (Dense)            (None, 1)                 11

=================================================================
Total params: 371
Trainable params: 371
Non-trainable params: 0
_________________________________________________________________
```

```
model.get_weights()
```

```
[array([[ 0.9634355 ,  2.008911  , -0.34264505,  0.6884312 ,  0.7782312 ,
         -0.6561154 ,  0.14167677,  0.9779885 ,  0.9325577 ,  1.1402048 ],
        [-0.16728629,  1.4618363 , -0.8058679 , -0.50964737, -0.866167  ,
```

```
              0.5496935 ,  0.13310157,  1.0779827 , -0.6639141 , -0.01463179]],
            dtype=float32),
       array([0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
       array([[-0.10539851,  0.48526895,  0.714316  , -0.18866718, -0.7250839 ,
               0.22085175,  0.7534904 ,  0.36095998, -0.6742738 ,  0.5922916 ],
              [-0.6299539 ,  0.10253704,  0.30383942, -0.45836744, -0.09067469,
               -0.10327455,  0.11008824, -0.01055317, -0.16426785,  0.10706678],
              [-0.01778925,  0.6664191 ,  0.6245424 , -0.03912661,  0.0080572 ,
               0.6449524 , -0.5234461 ,  0.30333588,  0.33092707,  0.3393051 ],
              [-0.00530986, -0.9603113 ,  0.5049274 ,  0.22863166, -0.45042127,
               -0.2503643 ,  0.1029162 ,  0.25492725, -0.29852968,  0.48624536],
              [-0.8647297 ,  0.6059265 , -0.03488478, -0.9845163 , -0.2521756 ,
               -0.6724048 , -0.15583909,  0.8362818 , -0.7816974 ,  0.41586933],
              [-0.0256603 , -0.7476501 ,  0.4419385 , -0.42284152,  0.13786316,
               0.33295807,  0.29497185, -0.39135888,  0.3087249 ,  0.5474781 ],
              [-0.2170632 , -0.89181143,  0.66846764,  0.2825762 ,  0.14591902,
               -0.15910214, -0.30370116,  0.08618522, -0.26557195,  0.4293157 ],
              [ 0.47798464, -0.3429556 ,  0.38501078,  0.77214956, -0.04640924,
               0.59088236, -0.689049  ,  0.27817264,  0.3736175 , -0.2784376 ],
              [ 0.42301384, -0.03015674,  0.8002327 , -0.611186  , -0.59336454,
               -0.22364625, -0.22954208,  0.36390936,  0.01810098,  0.1830766 ],
              [ 0.01415879, -0.51591563, -0.4304976 ,  0.83690506, -0.14185967,
               -0.47761172, -0.13387348, -0.23567227,  0.286203  ,  0.08208084]],
            dtype=float32),
       array([0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
       array([[-0.22719581,  0.25135857,  0.11783469,  0.25465578, -0.8545871 ,
               -0.545164  ,  0.2919587 ,  0.02047073,  0.0474746 , -0.22990222],
              [ 0.20630676,  0.45647845, -0.1643038 , -0.12552026,  0.02140639,
               0.18163018,  0.87371415, -0.45510593,  0.03604161, -0.72112197],
              [ 0.05588618,  0.33604667,  0.02206722,  0.40988854,  0.36898085,
               -0.39405695,  0.39123395, -0.8767327 , -0.14077851, -0.27199763],
              [-0.2094524 ,  0.16879848,  0.1295846 ,  0.1220797 , -0.06777103,
               -0.46362278,  0.17815213, -0.75938493, -0.10657755, -0.4833386 ],
              [-0.10479715, -0.12263593,  0.37804526, -0.16613133, -0.06238799,
               -0.13015227,  0.15683806,  0.72975564, -0.542468  , -0.82793933],
              [-0.15535268, -0.5324612 , -0.20927174, -0.51257825,  0.6215454 ,
               0.15115292, -0.39339757,  0.5876791 , -0.14108567,  0.54076993],
              [ 0.16276093,  0.86959326, -0.15111575,  0.6687028 ,  0.32407516,
               0.1618845 ,  0.12549894, -0.07277837, -0.06342666,  0.7858933 ],
              [-0.19009963,  0.11054919,  0.9038928 ,  0.7062179 ,  0.8278545 ,
               -0.5386862 , -0.21597224,  0.13052948, -0.05630735, -0.24903378],
              [ 0.23637347, -0.5177935 , -0.06894839,  0.34669626, -0.21427627,
               -0.22265106, -0.8586252 ,  0.22244304,  0.7417311 , -0.48316273],
              [ 0.22706413,  0.30637008, -0.48278323,  0.66773885, -0.68563974,
               0.29843712,  0.41248435, -0.9194688 , -0.30100805, -0.14725935]],
            dtype=float32),
       array([0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
       array([[-0.12719171,  0.32572147, -0.19186367,  0.1080595 ,  0.89044815,
               -0.11340923, -0.84662694,  0.14842165, -0.58024925, -0.3221063 ],
              [ 0.33102384,  0.6551966 ,  0.9588741 ,  0.61069065,  0.30429554,
               0.08153346,  0.30817097,  0.5135824 , -0.07844207,  0.3508088 ],
              [ 0.6542758 , -0.35206947, -0.00892634, -0.5251519 ,  0.44953102,
               0.47949585,  0.15924747,  0.12385951,  0.10013837,  0.55241907],
              [-0.42704555,  0.26947352, -0.22298317, -0.01166455,  0.00808559,
               -0.38047653,  0.6130559 ,  0.47299102,  0.02962644, -0.45622355],
```

```
initial_weights = model.get_weights()
```

```
initial_weights[0] = np.random.randn(2,10)*np.sqrt(1/2)
initial_weights[1] = np.zeros(model.get_weights()[1].shape)
initial_weights[2] = np.random.randn(10,10)*np.sqrt(1/10)
initial_weights[3] = np.zeros(model.get_weights()[3].shape)
initial_weights[4] = np.random.randn(10,10)*np.sqrt(1/10)
initial_weights[5] = np.zeros(model.get_weights()[5].shape)
initial_weights[6] = np.random.randn(10,10)*np.sqrt(1/10)
initial_weights[7] = np.zeros(model.get_weights()[7].shape)
initial_weights[8] = np.random.randn(10,1)*np.sqrt(1/10)
initial_weights[9] = np.zeros(model.get_weights()[9].shape)
```

```
model.set_weights(initial_weights)
```

```
model.get_weights()
```

```
[array([[-0.9083814 , -0.42979184, -0.53292066, -0.8351218 , -0.16821018,
            -0.17595254,  0.44825128, -0.3543974 , -0.26178998, -1.1923412 ],
           [ 0.91741747,  0.3588762 ,  0.50364774, -0.10173513,  0.2106472 ,
            -0.21627867, -0.18277432, -0.6875152 ,  0.35624236,  0.4325114 ]],
          dtype=float32),
     array([0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
     array([[ 0.14801227,  0.2561245 ,  0.10928258, -0.20176922,  0.00727606,
              0.42879048,  0.13790296, -0.4391284 , -0.06173414, -0.5698351 ],
            [-0.7840016 , -0.05161778,  0.06571397, -0.3170529 ,  0.41050068,
             -0.03985342, -0.580036  ,  0.294646  , -0.01483387, -0.06064747],
            [-0.07505108, -0.05698978,  0.1829229 , -0.13824858,  0.41852456,
             -0.663167  , -0.08177052,  0.23831266, -0.21426578, -0.30900267],
            [-0.26688266, -0.0863719 ,  0.41794643, -0.03646072,  0.02351558,
             -0.28430468,  0.2584214 , -0.1209321 ,  0.4445468 ,  0.05717354],
```

```
            [ 0.34896645, -0.00430898,  0.23457436, -0.3618909 ,  0.2587986 ,
             -0.39212444,  0.08126691, -0.08995533, -0.10002058,  0.08600869],
            [-0.04275794, -0.09703516,  0.3815181 ,  0.3097038 ,  0.11292977,
              0.117372  , -0.02248203,  0.4889994 ,  0.57115424,  0.43073615],
            [-0.04957008, -0.44252124, -0.17205828,  0.41337928, -0.3724524 ,
              0.2732053 , -0.4093258 ,  0.0459554 ,  0.19809112,  0.25311488],
            [-0.31412154, -0.01881288,  0.56514144,  0.38490713, -0.25869706,
             -0.21606022, -0.7732147 , -0.6392791 ,  0.0509569 , -0.20062056],
            [ 0.51777196,  0.08096474,  0.21223843, -0.22728097,  0.02850839,
             -0.02672394, -0.39952034,  0.08997314,  0.1420588 ,  0.24861185],
            [-0.16972117,  0.2020552 , -0.45849884,  0.07805384, -0.10639963,
              0.30771437, -0.13902055, -0.48376253,  0.32917154, -0.0343847 ]],
           dtype=float32),
     array([0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
     array([[ 0.351531  ,  0.04825034, -0.09516362, -0.02325017, -0.07679025,
             -0.0077562 ,  0.1440094 , -0.01102947,  0.2874337 ,  0.46714061],
            [ 0.1952239 , -0.07405984,  0.09022895,  0.4952026 , -0.41903275,
             -0.06319922,  0.4024669 , -0.33745408,  0.28385013, -0.3387091 ],
            [ 0.09652926, -0.02516844, -0.5685633 ,  0.01987785, -0.28808635,
              0.40259665,  0.21185209, -0.5279558 ,  0.48293707, -0.02610426],
            [ 0.45866492, -0.09240092, -0.15243736,  0.576126  ,  0.37718555,
             -0.08485635,  0.14718197, -0.11515211,  0.24846207, -0.11716742],
            [ 0.43098387, -0.25120708, -0.04714221,  0.01326465, -0.19509016,
              0.0197177 , -0.10951312,  0.45644504,  0.12346061,  0.30889845],
            [ 0.15061665,  0.406988  , -0.11505055,  0.19250084,  0.03419141,
              0.24471171,  0.17944743,  0.02243084,  0.10530423,  0.2657897 ],
            [ 0.13679977, -0.07137353,  0.19502199, -0.11566745, -0.19738303,
              0.08590439,  0.1332807 , -0.35735178,  0.4210001 ,  0.34009323],
            [ 0.4538984 , -0.1178948 ,  0.49594876, -0.33116475, -0.06217633,
              0.5180906 , -0.0145489 , -0.02893704,  0.16750787, -0.11778154],
            [-0.12415569, -0.5834298 ,  0.36892793, -0.24332199, -0.3879341 ,
             -0.07469143,  0.04734641, -0.48414105,  0.27407247, -0.05363356],
            [-0.12103921, -0.14562546, -0.07509393,  0.06635666, -0.18832582,
             -0.12808347, -0.38067216, -0.06058634, -0.5804807 , -0.07056971]],
           dtype=float32),
     array([0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
     array([[-0.08597732,  0.18572108, -0.13289165,  0.08838793, -0.4285278 ,
             -0.15164156, -0.3983799 ,  0.2767031 , -0.29137242, -0.17500088],
            [ 0.3288246 , -0.20354667, -0.00204105, -0.44299152, -0.03474264,
             -0.39402092, -0.10728984, -0.814813  ,  0.37740347, -0.33862337],
            [ 0.3411839 , -0.558053  ,  0.01472748, -0.26107407,  0.51546216,
              0.00784547,  0.10155795, -0.07124441, -0.21167699,  0.1757577 ],
            [-0.08078285, -0.2714235 ,  0.20375411,  0.03531255,  0.10036895,
              0.379184  ,  0.15216552,  0.13386655, -0.8858865 , -0.08510572],
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
history = model.fit(X,y,epochs=100,validation_split=0.2)
```

```
Epoch 1/100
3/3 [==============================] - 1s 85ms/step - loss: 1.7058 - accuracy: 0.5000 - val_loss: 1.6510 - val_accuracy: 0.5000
Epoch 2/100
3/3 [==============================] - 0s 15ms/step - loss: 1.6300 - accuracy: 0.5125 - val_loss: 1.5720 - val_accuracy: 0.5000
Epoch 3/100
3/3 [==============================] - 0s 15ms/step - loss: 1.5416 - accuracy: 0.5125 - val_loss: 1.4990 - val_accuracy: 0.5000
Epoch 4/100
3/3 [==============================] - 0s 14ms/step - loss: 1.4703 - accuracy: 0.5125 - val_loss: 1.4281 - val_accuracy: 0.5500
Epoch 5/100
3/3 [==============================] - 0s 15ms/step - loss: 1.3973 - accuracy: 0.5125 - val_loss: 1.3605 - val_accuracy: 0.5500
Epoch 6/100
3/3 [==============================] - 0s 15ms/step - loss: 1.3355 - accuracy: 0.4875 - val_loss: 1.2960 - val_accuracy: 0.5000
Epoch 7/100
3/3 [==============================] - 0s 23ms/step - loss: 1.2629 - accuracy: 0.5000 - val_loss: 1.2362 - val_accuracy: 0.5000
Epoch 8/100
3/3 [==============================] - 0s 14ms/step - loss: 1.2154 - accuracy: 0.4875 - val_loss: 1.1789 - val_accuracy: 0.5000
Epoch 9/100
3/3 [==============================] - 0s 15ms/step - loss: 1.1565 - accuracy: 0.4750 - val_loss: 1.1273 - val_accuracy: 0.4500
Epoch 10/100
3/3 [==============================] - 0s 15ms/step - loss: 1.1053 - accuracy: 0.4875 - val_loss: 1.0798 - val_accuracy: 0.4500
Epoch 11/100
3/3 [==============================] - 0s 15ms/step - loss: 1.0585 - accuracy: 0.4625 - val_loss: 1.0363 - val_accuracy: 0.4500
Epoch 12/100
3/3 [==============================] - 0s 14ms/step - loss: 1.0107 - accuracy: 0.4500 - val_loss: 0.9977 - val_accuracy: 0.4500
Epoch 13/100
3/3 [==============================] - 0s 14ms/step - loss: 0.9755 - accuracy: 0.4625 - val_loss: 0.9607 - val_accuracy: 0.4500
Epoch 14/100
3/3 [==============================] - 0s 15ms/step - loss: 0.9358 - accuracy: 0.4625 - val_loss: 0.9269 - val_accuracy: 0.4000
Epoch 15/100
3/3 [==============================] - 0s 15ms/step - loss: 0.9045 - accuracy: 0.4625 - val_loss: 0.8953 - val_accuracy: 0.4500
Epoch 16/100
3/3 [==============================] - 0s 16ms/step - loss: 0.8716 - accuracy: 0.4875 - val_loss: 0.8674 - val_accuracy: 0.4500
Epoch 17/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8443 - accuracy: 0.4875 - val_loss: 0.8419 - val_accuracy: 0.3500
Epoch 18/100
3/3 [==============================] - 0s 15ms/step - loss: 0.8210 - accuracy: 0.4875 - val_loss: 0.8181 - val_accuracy: 0.3000
Epoch 19/100
3/3 [==============================] - 0s 14ms/step - loss: 0.7964 - accuracy: 0.4875 - val_loss: 0.7965 - val_accuracy: 0.3000
Epoch 20/100
3/3 [==============================] - 0s 15ms/step - loss: 0.7738 - accuracy: 0.4875 - val_loss: 0.7775 - val_accuracy: 0.3000
Epoch 21/100
```

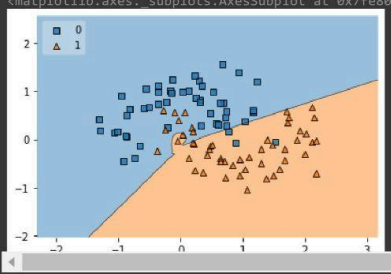
```
3/3 [==============================] - 0s 17ms/step - loss: 0.7562 - accuracy: 0.4875 - val_loss: 0.7609 - val_accuracy: 0.3500
Epoch 22/100
3/3 [==============================] - 0s 15ms/step - loss: 0.7401 - accuracy: 0.5000 - val_loss: 0.7458 - val_accuracy: 0.4000
Epoch 23/100
3/3 [==============================] - 0s 15ms/step - loss: 0.7232 - accuracy: 0.5125 - val_loss: 0.7322 - val_accuracy: 0.4000
Epoch 24/100
3/3 [==============================] - 0s 19ms/step - loss: 0.7096 - accuracy: 0.5125 - val_loss: 0.7190 - val_accuracy: 0.4000
Epoch 25/100
3/3 [==============================] - 0s 14ms/step - loss: 0.6961 - accuracy: 0.5250 - val_loss: 0.7067 - val_accuracy: 0.4500
Epoch 26/100
3/3 [==============================] - 0s 15ms/step - loss: 0.6841 - accuracy: 0.5500 - val_loss: 0.6951 - val_accuracy: 0.4500
Epoch 27/100
3/3 [==============================] - 0s 15ms/step - loss: 0.6720 - accuracy: 0.5625 - val_loss: 0.6839 - val_accuracy: 0.5000
Epoch 28/100
3/3 [==============================] - 0s 15ms/step - loss: 0.6622 - accuracy: 0.5750 - val_loss: 0.6729 - val_accuracy: 0.5500
Epoch 29/100
3/3 [==============================] - 0s 14ms/step - loss: 0.6520 - accuracy: 0.6250 - val_loss: 0.6624 - val_accuracy: 0.6500
```

```
model.get_weights()
```

```
[array([[-0.7534644 , -0.40642726, -0.46373174, -0.7715832 , -0.1277438 ,
         -0.2729551 ,  0.34282172, -0.2771892 , -0.26632193, -1.429772  ],
        [ 1.0737787 ,  0.47488663,  0.62465155, -0.12547861,  0.20325062,
         -0.28902182, -0.28489602, -0.7651134 ,  0.32014093,  0.38592413]],
       dtype=float32),
 array([ 0.17320989,  0.00114872,  0.0397424 , -0.13545433,  0.00454802,
        -0.22851905, -0.04516467,  0.15110366, -0.06830908,  0.17423655],
       dtype=float32),
 array([[ 0.07947282,  0.26585355,  0.15415847, -0.25968108, -0.01067472,
          0.37633535,  0.03868366, -0.38048545, -0.19025889, -0.65158546],
        [-0.85772514, -0.04266584,  0.12410757, -0.38518146,  0.4134989 ,
         -0.11743478, -0.66751945,  0.37822312, -0.16036926, -0.14447264],
        [-0.14926817, -0.04317714,  0.24179652, -0.20616755,  0.4172752 ,
         -0.737155  , -0.17425217,  0.3156445 , -0.3552739 , -0.395115  ],
        [-0.26271275, -0.16797927,  0.30563584, -0.03270569, -0.01150104,
         -0.24627677,  0.24142104, -0.07621646,  0.3339557 ,  0.10243275],
        [ 0.26255026,  0.02363358,  0.3236393 , -0.43547204,  0.26811856,
         -0.49417892, -0.0136685 ,  0.00086602, -0.24076873, -0.00815725],
        [ 0.06651404, -0.16314201,  0.26959333,  0.3236881 ,  0.11817549,
          0.17660452,  0.11021394,  0.47018412,  0.5029724 ,  0.54021436],
        [ 0.01232329, -0.43329835, -0.21424858,  0.48738435, -0.38471356,
          0.35022077, -0.33246592, -0.03527692,  0.34334227,  0.33408895],
        [-0.17499602, -0.08846112,  0.44534263,  0.41706678, -0.25442937,
         -0.13242589, -0.5917319 , -0.75003874,  0.0550186 , -0.08872192],
        [ 0.4335519 ,  0.11488257,  0.29251856, -0.29791567,  0.03763288,
         -0.11979282, -0.4849098 ,  0.18739605,  0.00637927,  0.16282775],
        [-0.17443036,  0.08167496, -0.5419055 ,  0.12802073, -0.22356339,
          0.38808206, -0.21330813, -0.49614394,  0.22101277, -0.00226254]],
       dtype=float32),
 array([ 0.049227  ,  0.02721086,  0.03504166, -0.01238636, -0.01706586,
         0.01988176,  0.00092993, -0.07296985,  0.04619956, -0.05349142],
       dtype=float32),
 array([[ 0.29350972,  0.01495209, -0.16467233,  0.02042359, -0.14378741,
         -0.05700863,  0.11989471, -0.05637817,  0.27254355,  0.47402745],
        [ 0.25167063, -0.14288695,  0.17566955,  0.41847163, -0.3347687 ,
          0.00793358,  0.48080194, -0.25084803,  0.34036502, -0.39949074],
        [ 0.2644032 , -0.0856568 , -0.4119659 , -0.09630942, -0.17804305,
          0.52713746,  0.3026423 , -0.40907007,  0.5909925 , -0.0976069 ],
        [ 0.36357093, -0.00488952, -0.26088867,  0.67006505,  0.28054923,
         -0.17542177,  0.05486431, -0.21949461,  0.16455914, -0.04049552],
        [ 0.55641574, -0.34682623,  0.07759377, -0.08662193, -0.09695615,
          0.11891202, -0.01521198,  0.56446314,  0.23075795,  0.23031878],
        [-0.09142579,  0.5548033 , -0.32469928,  0.3647462 , -0.12595163,
          0.07089631,  0.03449333, -0.14888313, -0.05808999,  0.39150393],
        [ 0.07599428, -0.12699255,  0.13093065, -0.07967905, -0.2547059 ,
          0.04206409,  0.12510847, -0.39710048,  0.40889964,  0.32673928],
        [ 0.5740713 , -0.2281679 ,  0.58825105, -0.40184653, -0.00410547,
          0.58658844,  0.05537734,  0.03298928,  0.31169656, -0.19233175],
        [-0.12104338, -0.5014624 ,  0.3113903 , -0.18781507, -0.46274775,
         -0.11499538, -0.02999777, -0.53375983,  0.18876345,  0.00505815],
        [-0.19417681, -0.06788933, -0.17673816,  0.1532216 , -0.28136235,
         -0.2114139 , -0.47005594, -0.15971005, -0.65872353, -0.00340652]],
       dtype=float32),
 array([-0.10150597, -0.03123332, -0.0282005 ,  0.00438787, -0.0004478 ,
        -0.01638761,  0.02175754, -0.0104647 , -0.0085741 ,  0.0146223 ],
       dtype=float32),
 array([[-0.22318298,  0.07007452, -0.2524823 , -0.05630882, -0.28767478,
         -0.30032828, -0.53940725,  0.39169684, -0.15365812, -0.0530921 ],
```

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X,y.astype('int'), clf=model, legend=2)
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments t
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
<matplotlib.axes._subplots.AxesSubplot at 0x7fe801e4ef50>
```




```
(np.random.randn(10,10)*0.01).min()
```

```
-0.01538420487066039
```

```
(np.random.randn(10,10)*0.01).max()
```

```
0.02825126600648443
```

```
Start coding or generate with AI.
```