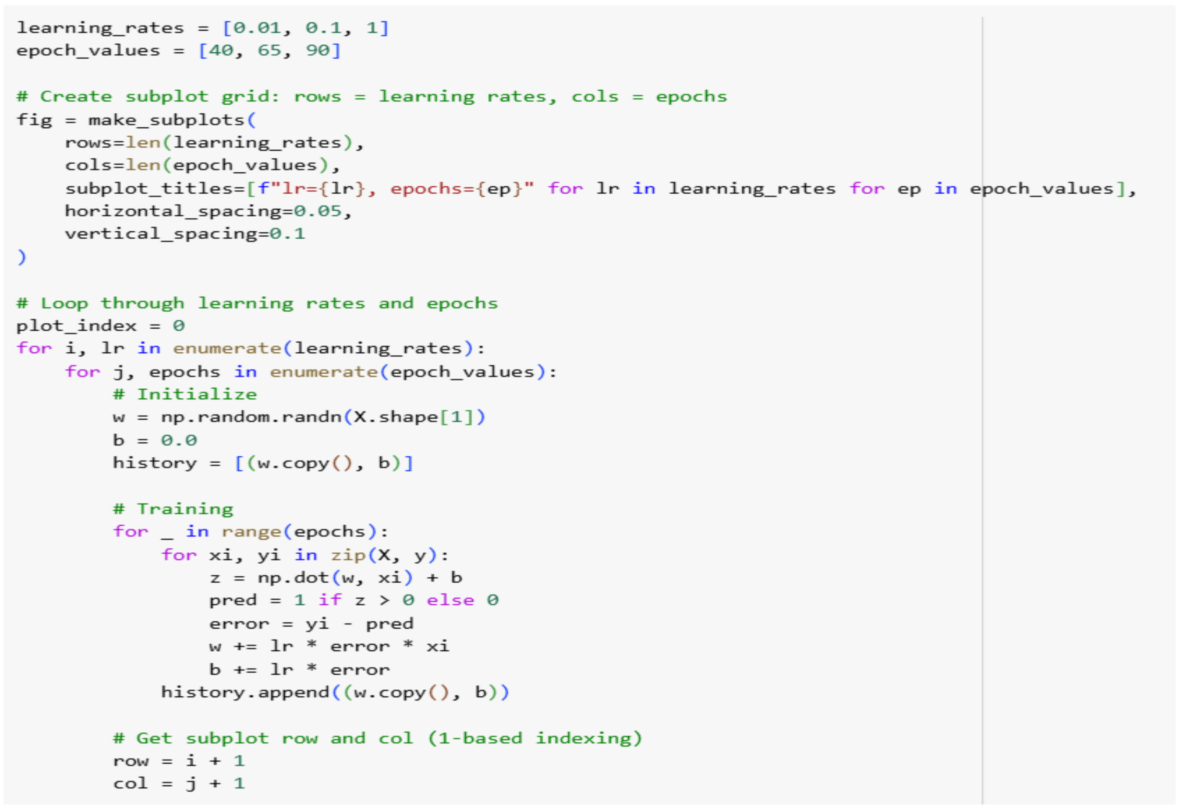
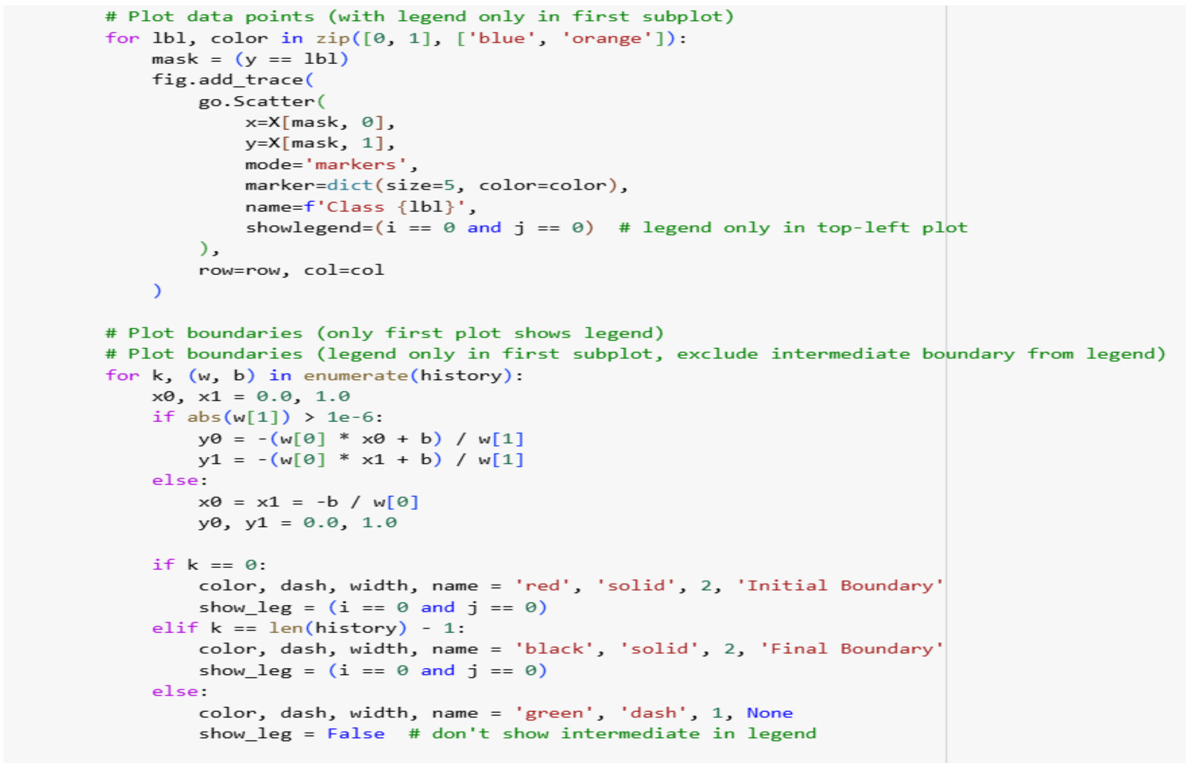
ASSIGNMENT 5

1)

Consider the code snippets:



```python
learning_rates = [0.01, 0.1, 1]
epoch_values = [40, 65, 90]

# Create subplot grid: rows = learning rates, cols = epochs
fig = make_subplots(
    rows=len(learning_rates),
    cols=len(epoch_values),
    subplot_titles=[f"lr={lr}, epochs={ep}" for lr in learning_rates for ep in epoch_values],
    horizontal_spacing=0.05,
    vertical_spacing=0.1
)

# Loop through learning rates and epochs
plot_index = 0
for i, lr in enumerate(learning_rates):
    for j, epochs in enumerate(epoch_values):
        # Initialize
        w = np.random.randn(X.shape[1])
        b = 0.0
        history = [(w.copy(), b)]

        # Training
        for _ in range(epochs):
            for xi, yi in zip(X, y):
                z = np.dot(w, xi) + b
                pred = 1 if z > 0 else 0
                error = yi - pred
                w += lr * error * xi
                b += lr * error
            history.append((w.copy(), b))

        # Get subplot row and col (1-based indexing)
        row = i + 1
        col = j + 1
```

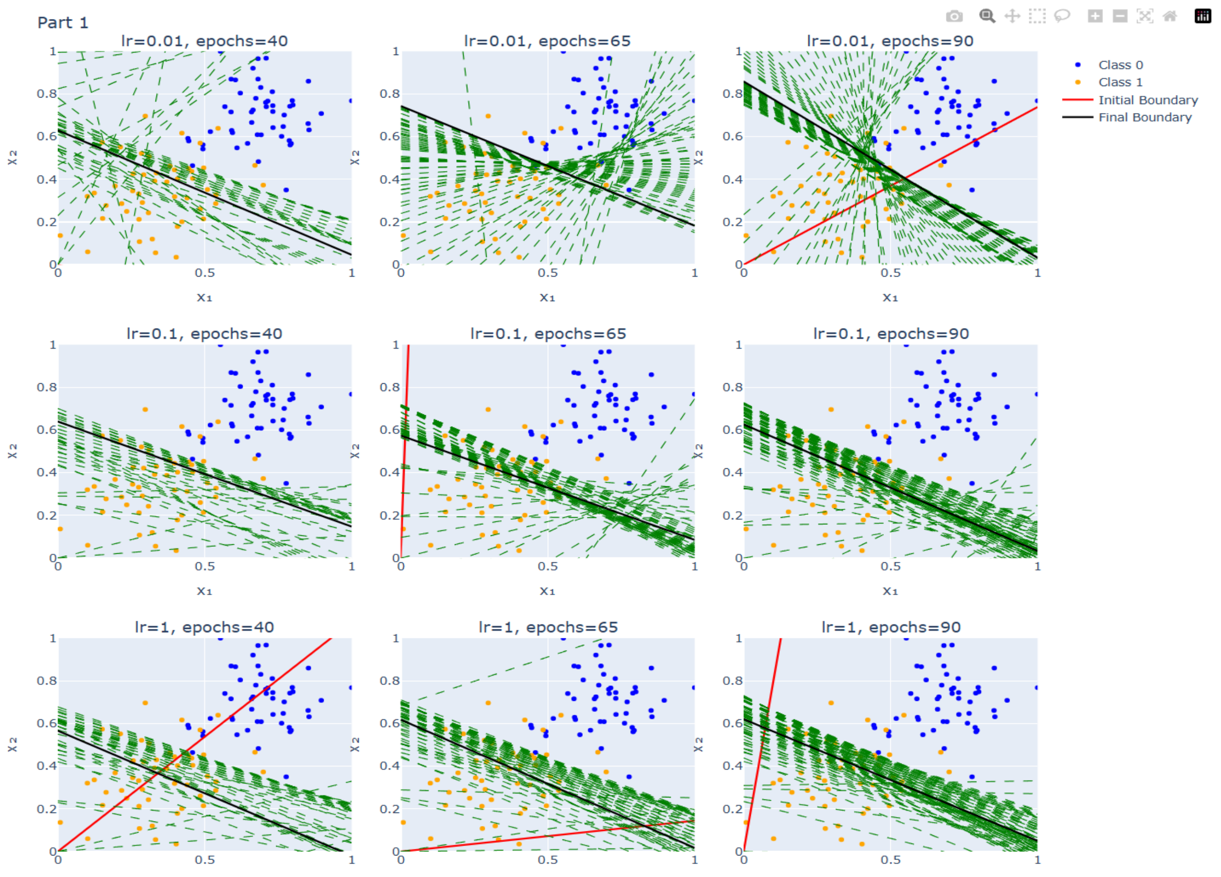


```python
        # Plot data points (with legend only in first subplot)
        for lbl, color in zip([0, 1], ['blue', 'orange']):
            mask = (y == lbl)
            fig.add_trace(
                go.Scatter(
                    x=X[mask, 0],
                    y=X[mask, 1],
                    mode='markers',
                    marker=dict(size=5, color=color),
                    name=f'Class {lbl}',
                    showlegend=(i == 0 and j == 0)  # legend only in top-left plot
                ),
                row=row, col=col
            )

        # Plot boundaries (only first plot shows legend)
        # Plot boundaries (legend only in first subplot, exclude intermediate boundary from legend)
        for k, (w, b) in enumerate(history):
            x0, x1 = 0.0, 1.0
            if abs(w[1]) > 1e-6:
                y0 = -(w[0] * x0 + b) / w[1]
                y1 = -(w[0] * x1 + b) / w[1]
            else:
                x0 = x1 = -b / w[0]
                y0, y1 = 0.0, 1.0

            if k == 0:
                color, dash, width, name = 'red', 'solid', 2, 'Initial Boundary'
                show_leg = (i == 0 and j == 0)
            elif k == len(history) - 1:
                color, dash, width, name = 'black', 'solid', 2, 'Final Boundary'
                show_leg = (i == 0 and j == 0)
            else:
                color, dash, width, name = 'green', 'dash', 1, None
                show_leg = False  # don't show intermediate in legend
```

[31]

```python
            fig.add_trace(
                go.Scatter(
                    x=[x0, x1],
                    y=[y0, y1],
                    mode='lines',
                    line=dict(color=color, dash=dash, width=width),
                    name=name,
                    showlegend=show_leg
                ),
                row=row, col=col
            )

# Final layout settings
fig.update_layout(
    title='Part 1',
    height=1000,
    width=1200,
    margin=dict(t=50),
    showlegend=True
)

# Set uniform axis ranges
for r in range(1, len(learning_rates) + 1):
    for c in range(1, len(epoch_values) + 1):
        fig.update_xaxes(range=[0, 1], row=r, col=c, title='x₁')
        fig.update_yaxes(range=[0, 1], row=r, col=c, title='x₂')

fig.show()
```

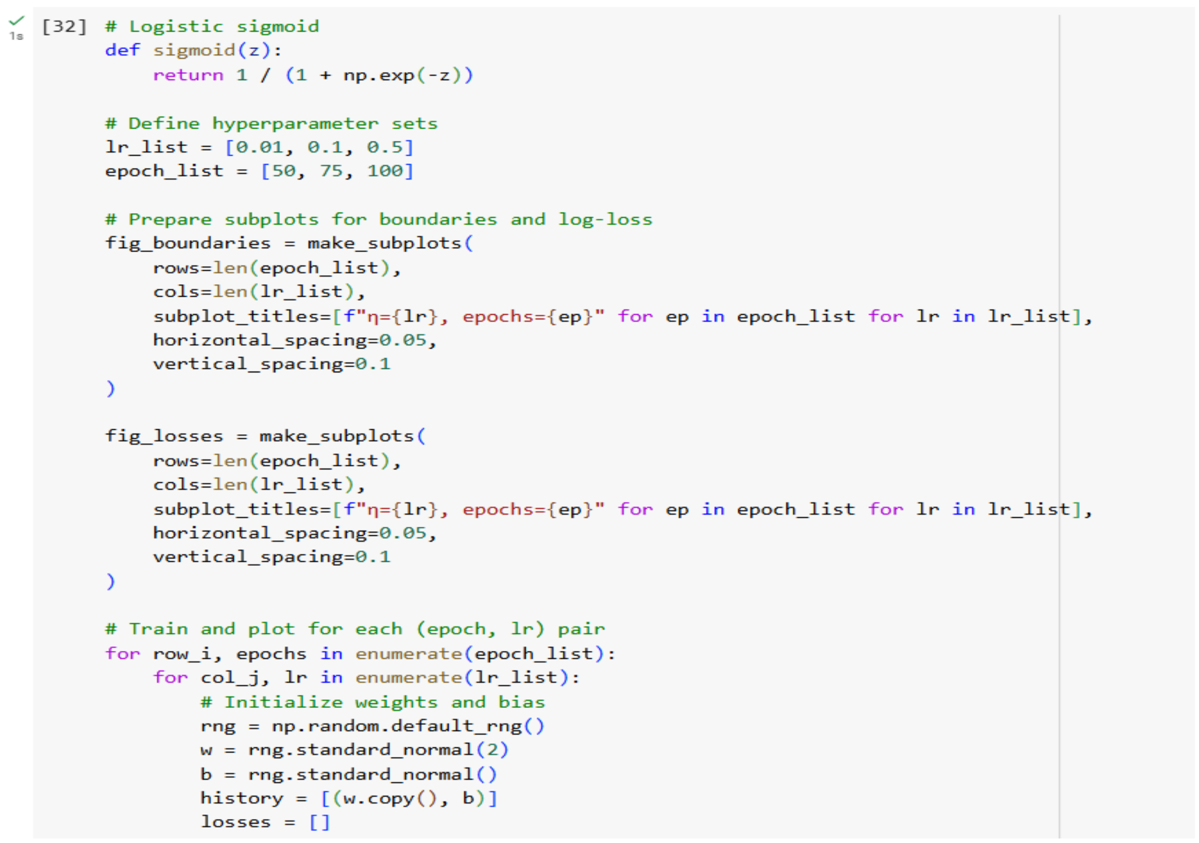
Consider the graphs:



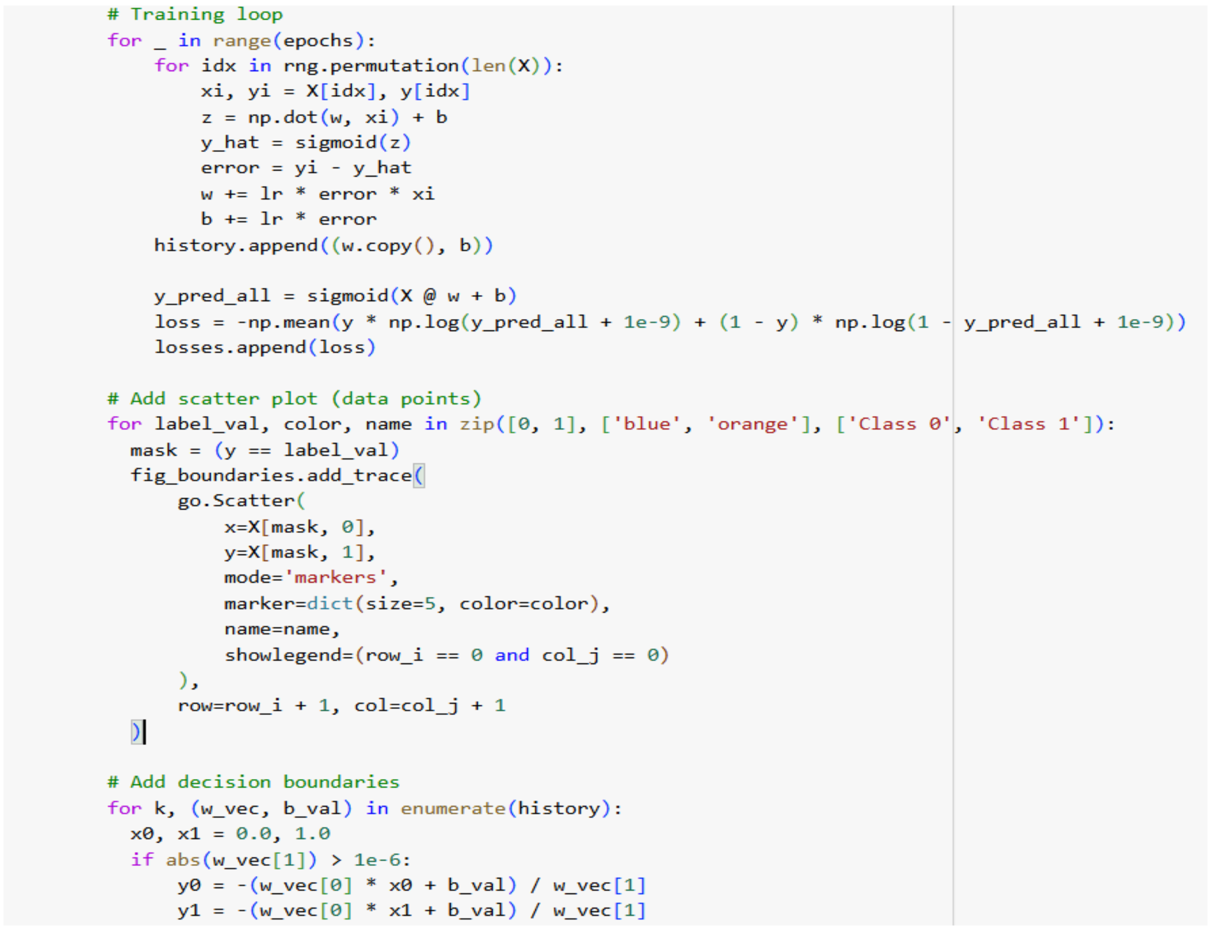
The graph above showcases how the **algorithm** evolves its decision boundary across different **learning rates** (η = 0.01, 0.1, 1) and **epoch values** (40, 65, 90). Each subplot represents a unique combination of learning rate and number of training epochs, where the blue and orange dots indicate **Class 0** and **Class 1** data points, respectively. The red line marks the **initial decision boundary**, while the black line shows the **final boundary** after training. The green dashed lines in between visualise the **intermediate boundaries** learned during each epoch.

Looking across the grid, we can see how the learning rate significantly impacts how fast and smoothly the perceptron converges. For example, at a low learning rate (0.01, top row), the boundary adjusts gradually over epochs and doesn't fully settle by epoch 90—suggesting slow learning. At a moderate rate (0.1, middle row), the model adapts more efficiently, with the final boundaries showing better alignment with class separation. However, at a high learning rate (1.0, bottom row), the learning process becomes unstable. The initial boundary is often erratic, and the intermediate boundaries oscillate heavily—indicating the perceptron is **overshooting** the optimal solution. This visualization powerfully demonstrates the trade-off between convergence speed and stability.

2)



```
[32] # Logistic sigmoid
     def sigmoid(z):
         return 1 / (1 + np.exp(-z))

     # Define hyperparameter sets
     lr_list = [0.01, 0.1, 0.5]
     epoch_list = [50, 75, 100]

     # Prepare subplots for boundaries and log-loss
     fig_boundaries = make_subplots(
         rows=len(epoch_list),
         cols=len(lr_list),
         subplot_titles=[f"η={lr}, epochs={ep}" for ep in epoch_list for lr in lr_list],
         horizontal_spacing=0.05,
         vertical_spacing=0.1
     )

     fig_losses = make_subplots(
         rows=len(epoch_list),
         cols=len(lr_list),
         subplot_titles=[f"η={lr}, epochs={ep}" for ep in epoch_list for lr in lr_list],
         horizontal_spacing=0.05,
         vertical_spacing=0.1
     )

     # Train and plot for each (epoch, lr) pair
     for row_i, epochs in enumerate(epoch_list):
         for col_j, lr in enumerate(lr_list):
             # Initialize weights and bias
             rng = np.random.default_rng()
             w = rng.standard_normal(2)
             b = rng.standard_normal()
             history = [(w.copy(), b)]
             losses = []
```

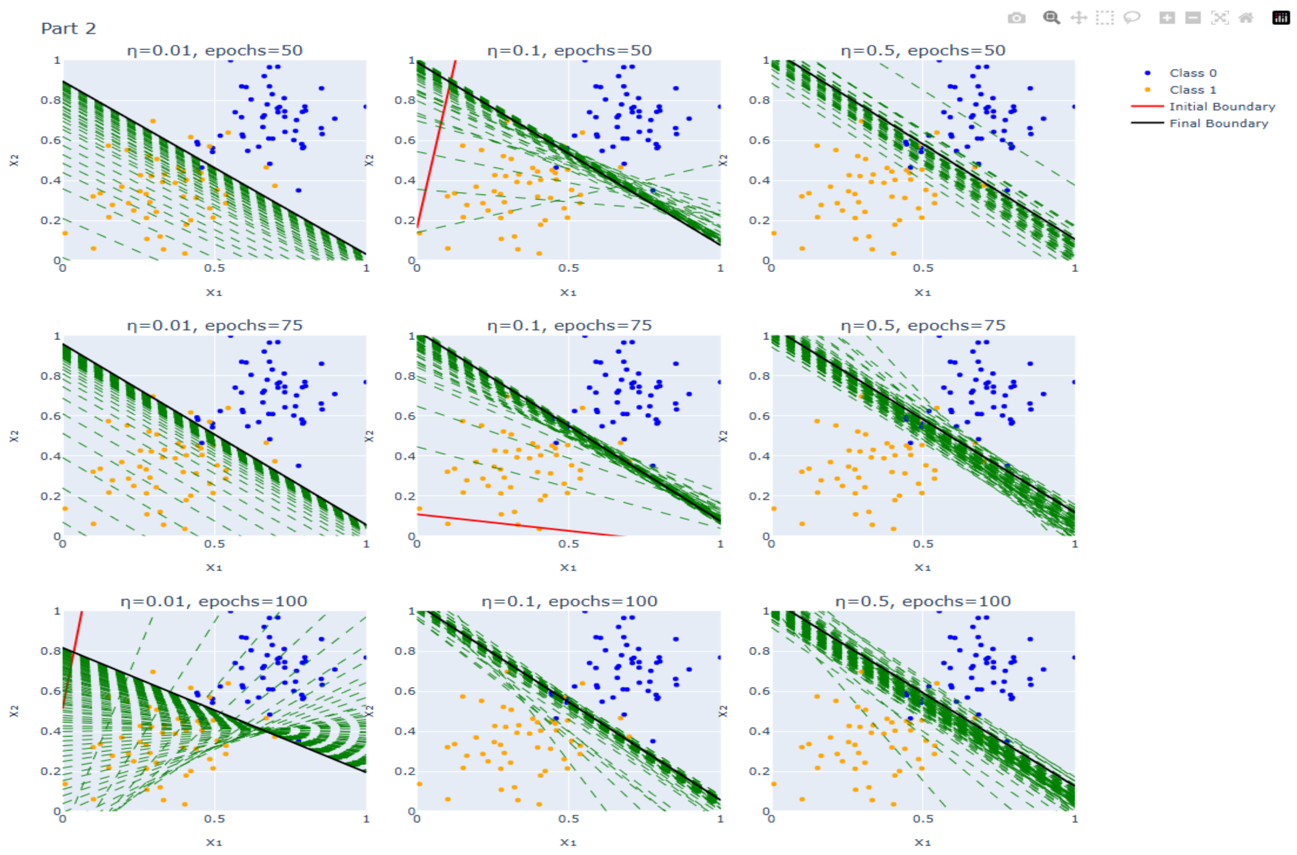


```
# Training loop
for _ in range(epochs):
    for idx in rng.permutation(len(X)):
        xi, yi = X[idx], y[idx]
        z = np.dot(w, xi) + b
        y_hat = sigmoid(z)
        error = yi - y_hat
        w += lr * error * xi
        b += lr * error
    history.append((w.copy(), b))

    y_pred_all = sigmoid(X @ w + b)
    loss = -np.mean(y * np.log(y_pred_all + 1e-9) + (1 - y) * np.log(1 - y_pred_all + 1e-9))
    losses.append(loss)

# Add scatter plot (data points)
for label_val, color, name in zip([0, 1], ['blue', 'orange'], ['Class 0', 'Class 1']):
  mask = (y == label_val)
  fig_boundaries.add_trace(
      go.Scatter(
          x=X[mask, 0],
          y=X[mask, 1],
          mode='markers',
          marker=dict(size=5, color=color),
          name=name,
          showlegend=(row_i == 0 and col_j == 0)
      ),
      row=row_i + 1, col=col_j + 1
  )

# Add decision boundaries
for k, (w_vec, b_val) in enumerate(history):
  x0, x1 = 0.0, 1.0
  if abs(w_vec[1]) > 1e-6:
      y0 = -(w_vec[0] * x0 + b_val) / w_vec[1]
      y1 = -(w_vec[0] * x1 + b_val) / w_vec[1]
```

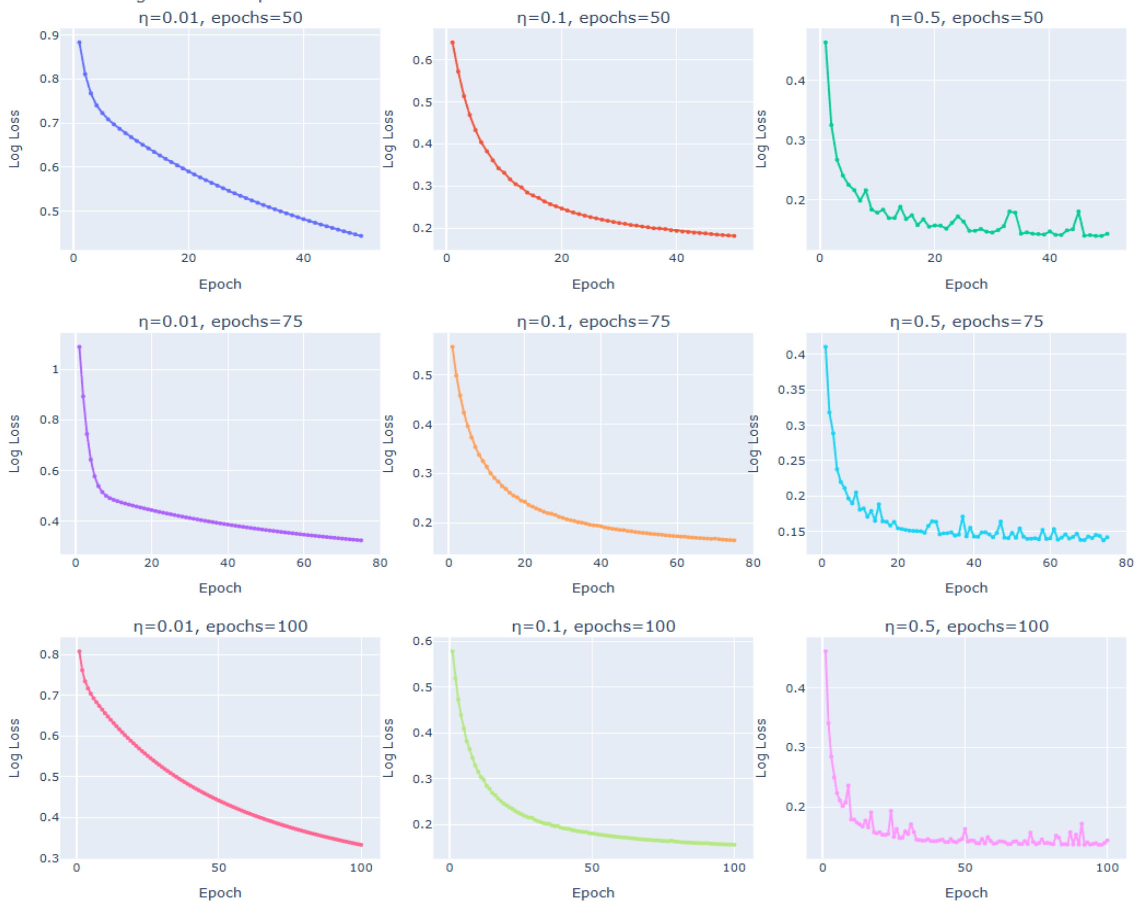
```python
          else:
              x0 = x1 = -b_val / w_vec[0]
              y0, y1 = 0.0, 1.0

          # Line style + legend control
          if k == 0:
              line_style = dict(color='red', dash='solid', width=2)
              name = "Initial Boundary"
              show_leg = (row_i == 0 and col_j == 0)
          elif k == len(history) - 1:
              line_style = dict(color='black', dash='solid', width=2)
              name = "Final Boundary"
              show_leg = (row_i == 0 and col_j == 0)
          else:
              line_style = dict(color='green', dash='dash', width=1)
              name = None
              show_leg = False

          fig_boundaries.add_trace(
              go.Scatter(
                  x=[x0, x1],
                  y=[y0, y1],
                  mode='lines',
                  line=line_style,
                  name=name,
                  showlegend=show_leg
              ),
              row=row_i + 1,
              col=col_j + 1
          )
        # Add log-loss plot
        fig_losses.add_trace(
            go.Scatter(
                x=list(range(1, len(losses) + 1)),
                y=losses,
                mode='lines+markers',
                marker=dict(size=4),
                showlegend=False
            ),
            row=row_i + 1,
            col=col_j + 1
        )
```

```python
# Final layout for boundaries
fig_boundaries.update_layout(
    title="Part 2",
    height=1000,
    width=1200,
    margin=dict(t=60),
    legend=dict(x=1.05, y=1.0),  # Move legend outside top right
    showlegend=True
)

# Final layout for log-loss curves
fig_losses.update_layout(
    title="Part 2 – Log-Loss Over Epochs",
    height=1000,
    width=1200,
    margin=dict(t=60)
)

# Uniform axis ranges
for r in range(1, len(epoch_list) + 1):
    for c in range(1, len(lr_list) + 1):
        fig_boundaries.update_xaxes(range=[0, 1], title="x₁", row=r, col=c)
        fig_boundaries.update_yaxes(range=[0, 1], title="x₂", row=r, col=c)
        fig_losses.update_xaxes(title="Epoch", row=r, col=c)
        fig_losses.update_yaxes(title="Log Loss", row=r, col=c)

# Show both plots
fig_boundaries.show()
fig_losses.show()
```

Part 2



Part 2 – Log-Loss Over Epochs



The first set of visualizations illustrates the evolution of decision boundaries for the perceptron trained using gradient descent, under various learning rates (η = 0.01, 0.1, 0.5) and epoch settings (50, 75, 100). Each subplot provides a clear view of how the decision boundary progresses from its initial state (red line), through intermediate stages (green dashed lines), to its final position (black line) after training.

From these plots, it is evident that a low learning rate (η = 0.01) results in very gradual and stable updates. Even after 100 epochs, the boundary shifts incrementally, indicating a slow convergence. Conversely, a moderate learning rate (η = 0.1) demonstrates a more optimal learning behavior — the decision boundary converges efficiently and aligns well with the data distribution, often within 50 to 75 epochs. However, with a high learning rate (η = 0.5), the model shows signs of instability. The boundary updates become erratic, with frequent directional changes between epochs, suggesting that the learning process may be overshooting the optimal solution. While the final boundary may still be acceptable, the path to convergence is less consistent.

The second figure presents the corresponding log-loss curves over epochs for each learning rate and epoch combination. For η = 0.01, the loss decreases smoothly but at a slower pace, consistent with the observed gradual boundary shifts. The η = 0.1 configuration again shows favorable results, achieving a rapid and stable decline in log-loss, indicating effective learning. In contrast, η = 0.5 yields a steep initial drop in loss, but is followed by noticeable oscillations. These fluctuations point to potential instability in the model's learning trajectory, reflecting the same pattern seen in the decision boundary plots.

In summary, the visualizations highlight the trade-offs between stability and speed in gradient descent optimization. A moderate learning rate such as η = 0.1 offers a balanced approach, achieving faster convergence while maintaining reliable learning dynamics. Lower rates ensure stability at the cost of speed, while higher rates risk overshooting and inconsistent convergence.

Link: https://github.com/Surya516/CSCI580_AI_ASSIGNMENT5