Furniture Perceptron

September 8, 2024

0.0.1 Owen Kroeger and Teddy Coon

1 Perceptron Optimization for Furniture Placement in a Room

This notebook explores how a simple perceptron can be used to find the optimal placement for furniture within a room.

1.0.1 Videos:

- **Teddy**: https://www.loom.com/share/fb173ad276e4413c881b2ac63e92fa06
- **Owen**: https://www.loom.com/share/1e3ffb0dde9349ab8e4f856b9c35d716?sid=3fd6e910-b5d2-4683-a523-bab3d195cf89

1.1 Problem Statement

The objective is to optimize the placement of furniture in a room using a perceptron model. The room is represented as a 10x10 grid where each cell can either be optimal (1) or not optimal (0). We aim to train the perceptron to learn these optimal placements based on given grid data.

1.2 Algorithm of the Solution

The solution uses a perceptron model with the following components: - **Sigmoid Activation Function**: Used during training to calculate predictions. - **Step Function**: Used for final binary decisions. - **Training**: The perceptron is trained using a simple feedforward approach with error correction. - **Visualization**: Intermediate results and errors are visualized using heatmaps and plots.

1.3 Import Necessary Libraries

We will use NumPy for numerical operations and Matplotlib for plotting and visualizing the perceptron's performance. The following libraries are essential for the implementation:

[ ]:
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from matplotlib.gridspec import GridSpec
```

1.4 Define Activation Functions

The perceptron uses two types of activation functions: - **Sigmoid Activation Function**: Used during the training phase to calculate continuous predictions and update weights. - **Binary Step Function**: Used for making final binary predictions (optimal or not optimal) after training.

[ ]:
```
# Sigmoid activation function for training
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Binary activation function for final predictions
def activation_function(x):
    return 1 if x >= 0 else 0
```

1.5 Perceptron Training Function

The perceptron is trained using a feedforward approach with error correction. The weights are adjusted based on the error between the predicted and actual outputs. The function captures intermediate results to visualize how the perceptron learns over time.

[ ]:
```
# Perceptron training function with intermediate heatmap captures and error␣
↪tracking
def train_perceptron(data, labels, epochs=200, learning_rate=0.2):
    # Initialize weights and bias with small random values for more nuanced␣
↪updates
    weights = np.random.randn(data.shape[1]) * 0.01
    bias = 0.0
    intermediate_weighted_sums = []  # To store intermediate weighted sums
    errors = []  # To track error over epochs

    for epoch in range(epochs):
        total_error = 0
        for i in range(len(data)):
            x = data[i]
            y = labels[i]

            # Calculate the weighted sum and use sigmoid for training
            weighted_sum = np.dot(weights, x) + bias
            prediction = sigmoid(weighted_sum)  # Use sigmoid for more varied␣
↪weight updates

            # Calculate the gradient with respect to the sigmoid output
            error = y - prediction
            total_error += error ** 2  # Accumulate squared error
            weights += learning_rate * error * x * prediction * (1 -␣
↪prediction)  # Derivative of sigmoid
            bias += learning_rate * error * prediction * (1 - prediction)
```

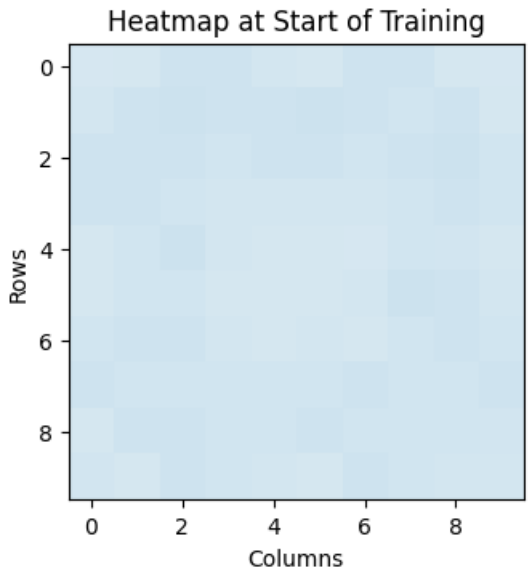
```
        # Store total error for the epoch
        errors.append(total_error)

        # Capture weighted sums at the start, halfway, and at the end
        if epoch == 0 or epoch == epochs // 2 or epoch == epochs - 1:
            current_weighted_sums = [np.dot(weights, data[i]) + bias for i in␣
 ↪range(len(data))]
            intermediate_weighted_sums.append(np.array(current_weighted_sums).
 ↪reshape(10, 10))

    return weights, bias, intermediate_weighted_sums, errors

# Manually inputting the grid values into training_data and corresponding labels
training_data = np.eye(100)  # 100 vectors, each one representing a grid cell␣
 ↪(one-hot encoded)

# Labels manually inputted based on the grid data (flattened from the grid)
labels = [
    1, 1, 0, 0, 1, 1, 0, 0, 1, 1,  # Row 1
    1, 0, 0, 0, 0, 0, 0, 0, 0, 1,  # Row 2
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  # Row 3
    0, 0, 0, 1, 1, 1, 1, 0, 0, 0,  # Row 4
    1, 0, 0, 1, 1, 1, 1, 0, 0, 1,  # Row 5
    1, 0, 0, 1, 1, 1, 1, 0, 0, 1,  # Row 6
    0, 0, 0, 1, 1, 1, 1, 0, 0, 0,  # Row 7
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  # Row 8
    1, 0, 0, 0, 0, 0, 0, 0, 0, 1,  # Row 9
    1, 1, 0, 0, 1, 1, 0, 0, 1, 1  # Row 10
]

# Train the perceptron using the sigmoid function for training and capture␣
 ↪intermediate states and errors
weights, bias, intermediate_weighted_sums, errors =␣
 ↪train_perceptron(training_data, labels, epochs=200,
                                                                          ␣
 ↪learning_rate=0.2)

# Final testing to capture the final predictions
weighted_sums = [np.dot(weights, training_data[i]) + bias for i in range(100)]
binary_predictions = [activation_function(ws) for ws in weighted_sums]

# Reshape weighted sums, binary predictions, and labels to 10x10 grids for␣
 ↪visualization
grid_weighted_sums = np.array(weighted_sums).reshape(10, 10)
grid_predictions = np.array(binary_predictions).reshape(10, 10)
grid_expected = np.array(labels).reshape(10, 10)
```

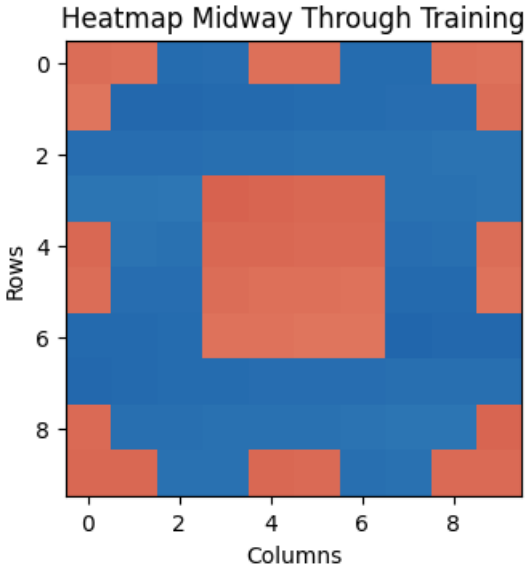
```
# Use symmetric normalization to enhance contrast evenly for both reds and blues
norm = Normalize(vmin=-np.max(np.abs(grid_weighted_sums)), vmax=np.max(np.
  ↪abs(grid_weighted_sums)))
```

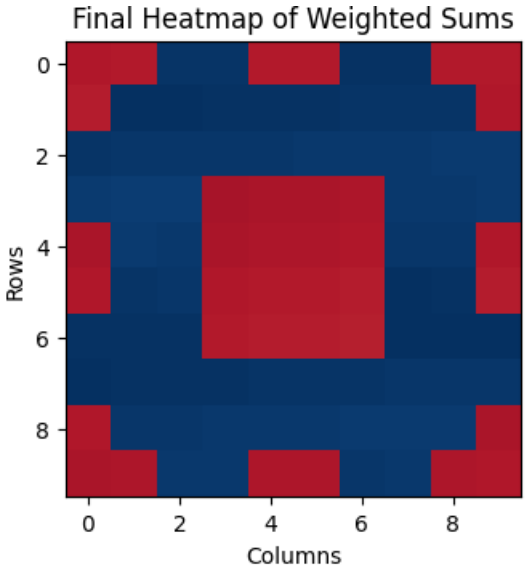
1.6 Output and Analysis

1.6.1 Intermediate Results: Heatmaps of Weighted Sums

The following heatmaps represent the perceptron's confidence (weighted sums) at different stages of training: the start, midway, and end. These visualizations help us understand how the model's predictions evolve over time.





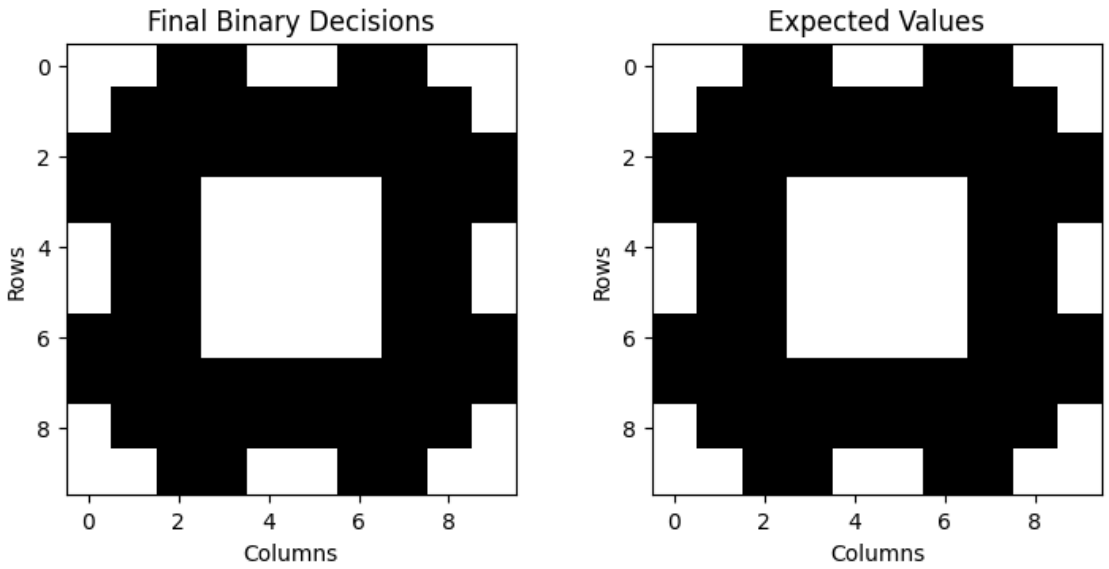


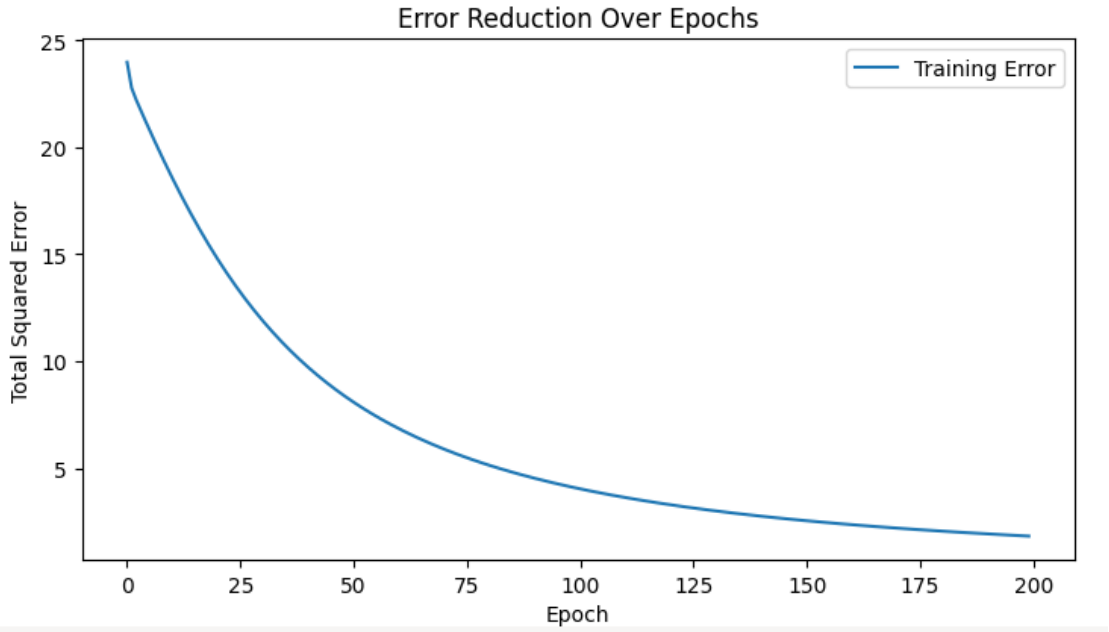
[ ]:
```
# Visualize intermediate heatmaps
fig = plt.figure(figsize=(18, 10))
```

```
gs = GridSpec(2, 3, figure=fig)

ax1 = fig.add_subplot(gs[0, 0])
im1 = ax1.imshow(intermediate_weighted_sums[0], cmap='RdBu_r',␣
 ↪interpolation='nearest', norm=norm)
ax1.set_title('Heatmap at Start of Training')
ax1.set_xlabel('Columns')
ax1.set_ylabel('Rows')

ax2 = fig.add_subplot(gs[0, 1])
im2 = ax2.imshow(intermediate_weighted_sums[1], cmap='RdBu_r',␣
 ↪interpolation='nearest', norm=norm)
ax2.set_title('Heatmap Midway Through Training')
ax2.set_xlabel('Columns')
ax2.set_ylabel('Rows')

ax3 = fig.add_subplot(gs[0, 2])
im3 = ax3.imshow(intermediate_weighted_sums[2], cmap='RdBu_r',␣
 ↪interpolation='nearest', norm=norm)
ax3.set_title('Final Heatmap of Weighted Sums')
ax3.set_xlabel('Columns')
ax3.set_ylabel('Rows')

# Add a colorbar to the right of the heatmaps
cbar = fig.colorbar(im3, ax=[ax1, ax2, ax3], fraction=0.015, pad=0.04,␣
 ↪orientation='vertical')
cbar.set_label('Weighted Sum (Gradient of Optimality)')

plt.show()
```

1.6.2 Final Outputs: Binary Decisions and Error Reduction

Below, we see the perceptron's final binary decisions (optimal vs. not optimal) compared against the expected values. Additionally, the error plot illustrates the reduction of training error over epochs, showing how the model improves with training.





[ ]:
```
# Visualize final binary decisions, expected values, and error plot
fig = plt.figure(figsize=(18, 10))
gs = GridSpec(2, 3, figure=fig)

ax4 = fig.add_subplot(gs[1, 0])
im4 = ax4.imshow(grid_predictions, cmap='gray', interpolation='nearest')
ax4.set_title('Final Binary Decisions (1 = Optimal, 0 = Not Optimal)')
ax4.set_xlabel('Columns')
ax4.set_ylabel('Rows')

ax5 = fig.add_subplot(gs[1, 1])
im5 = ax5.imshow(grid_expected, cmap='gray', interpolation='nearest')
ax5.set_title('Expected Values (1 = Optimal, 0 = Not Optimal)')
```

```
ax5.set_xlabel('Columns')
ax5.set_ylabel('Rows')

ax6 = fig.add_subplot(gs[1, 2])
ax6.plot(errors, label='Training Error')
ax6.set_title('Error Reduction Over Epochs')
ax6.set_xlabel('Epoch')
ax6.set_ylabel('Total Squared Error')
ax6.legend()

plt.tight_layout()
plt.show()
```

1.6.3 Analysis of the Findings

- **Intermediate Heatmaps**: The heatmaps show a progression of the perceptron's confidence from random guesses at the start to more refined predictions by the end. Midway heatmaps provide insight into the learning process.
- **Final Binary Decisions**: Comparing the final binary decisions with the expected values reveals areas where the perceptron correctly identifies optimal placements and where it fails.
- **Error Plot**: The error plot indicates a clear reduction in error over epochs, demonstrating the perceptron's learning capability. A consistent decrease suggests effective training.