Lab Work 6 Report
Madiyar Kaliyev

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

These libraries, numpy and matplotlib, facilitate numerical operations, array manipulations, mathematical operations, and data visualization, with numpy used for array manipulation and mathematical operations, and matplotlib employed for creating graphs and histograms.

```
def initialize_weights(method, layer_sizes):
    weights = {f'W{i}': initialization_strategy(method,
layer_sizes[i-1], layer_sizes[i])
               for i in range(1, len(layer_sizes))}
    return weights
```

1. This function initializes weights for all layers using the specified method and layer sizes.
2. It uses a dictionary comprehension to create a dictionary where keys are 'W{i}' for the i-th layer, and values are initialized weights.

```
def initialization_strategy(method, input_connections,
output_connections):
    if method == "zeros":
        return np.zeros((output_connections, input_connections))
    elif method == "small_random":
        return np.random.randn(output_connections, input_connections) *
0.02
    elif method == "large_random":
        return np.random.randn(output_connections, input_connections) *
0.2
    elif method == "xavier":
        scale = np.sqrt(2 / (input_connections + output_connections))
        return np.random.randn(output_connections, input_connections) *
scale
    elif method == "he":
        scale = np.sqrt(2 / input_connections)
        return np.random.randn(output_connections, input_connections) *
scale
```

1. This function takes a weight initialization method (method), input_connections, and output_connections as parameters.
2. It returns the initialized weights based on the specified method.
3. Different strategies include initializing to zeros, small random values, large random values, Xavier initialization, and He initialization.

```
def forward_pass_all_layers(inputs, weights, activation_func):
    activations = [inputs]
    for i in range(1, len(weights) + 1):
        z = np.dot(weights[f'W{i}'], activations[-1])
        a = activation_func(z)
        activations.append(a)
    return activations
```

1. This function performs a forward pass through all layers of the neural network.
2. It takes inputs, weights, and an activation function as parameters.
3. For each layer, it calculates the weighted sum (z) and applies the activation function (a), then appends the result to the list of activations.

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

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

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

These functions define common activation functions used in neural networks: sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

```
# Network configuration
layer_sizes = [500] * 10  # 10 layers, each with 500 neurons
inputs = np.random.randn(500, 1)
```

1. Specifies the network configuration with 10 layers, each containing 500 neurons.
2. Generates random inputs with a unit Gaussian distribution

```
# Weights initialization methods
initialization_methods = ["zeros", "small_random", "large_random",
"xavier", "he"]

# Activation functions
activation_functions = [sigmoid, tanh, relu]
```

Lists of weight initialization methods and activation functions to be used in the experiment.

```
# Plotting
for method in initialization_methods:
    weights = initialize_weights(method, layer_sizes)
    for activation_func in activation_functions:
        activations = forward_pass_all_layers(inputs, weights,
activation_func)
        # Plot histograms of activations for each layer
        num_layers = len(activations) - 1
        num_cols = min(num_layers, 5)
        num_rows = (num_layers - 1) // num_cols + 1

        plt.figure(figsize=(24, 8))

        for i, a in enumerate(activations[1:]):  # to skip the input
layer
            plt.subplot(num_rows, num_cols, i + 1)
            plt.hist(a.ravel(), bins=30, color='green', alpha=0.7)
            plt.title(f'Layer {i+1}')

        plt.suptitle(f'Modified Initialization: {method}, Activation
Function: {activation_func.__name__}')
        plt.show()
```

1. Iterates through different weight initialization methods and activation functions.
2. For each combination, it performs a forward pass through the network and plots histograms of activations for each layer.
3. The histograms are organized in a grid with a specified number of columns and rows.