LMBP(X, Y, hidden, mu, theta, max_epochs): trains a 2-layers MLP to fit X onto Y using Levenberg-Marquardt backpropagation.

Args:

- X: network inputs
- Y: network targets
- hidden: number of hidden neurons
- μ : jacobian smoothing initial value
- θ : increase/decrease factor
- max_epochs: maximum number of allowed epochs.

Returns:

- W^1, W^2, b^1, b^2 : weights and biases of the connections between the input and the hidden layer and the hidden layer and the output layer.
- loss_hist: a history of cost values of different epochs in order of increasing epoch numbers

In order to calculate the weights we will use the Levenberg-Marquardt algorithm as discussed in the textbook. The MSE performance index is assumed.

Below is a brief description of the algorithm:

- Errors w.r.t. all inputs are calculated.
- Marquardt sensitivities are calculated w.r.t. the error (not the square of the error).
- Total Marquardt sensitivity is constructed from concatenating individual Marquardt sensitivities.
- The Jacobian matrix (derivative of each error w.r.t. each parameter) is calculated with the help of the total Marquardt sensitivity.
- All parameters are put into a row vector x and Δx is obtained from P2-31 eq. 12.32.
- Δx is split into size for each parameter (for instance if W_1, a 4x1 weight vector, was put into x as a 1x4 row vector, we will take the corresponding 4 elements of Δx and transpose them so that ΔW_1, also a 4x1 vector is produced)
- The parameters are updated.
- Performance index is recalculated.
- If it hasn't decreased, μ is multiplied by θ . This slows the learning and allows the algorithm to converge.
- If it has decreased, μ is divided by θ . This speeds up the learning, allowing the algorithm to take larger steps towards the minimum.

```
function [W1, W2, b1, b2, loss_hist] = LMBP(X, Y, hidden, mu, theta, max_epochs)
    % hyperparameter and history initialization
    n_0 = size(X, 1); % number of inputs
    n_1 = hidden;
    n_2 = size(Y, 1);
    % cost history
    loss_hist = zeros([max_epochs, 1]);
    % weights and biases initialization (random between -0.5 and 0.5)
```

```
    W1 = -0.5 + rand(n_1, n_0);
    W2 = -0.5 + rand(n_2, n_1);
    b1 = -0.5 + rand(n_1, 1);
    b2 = -0.5 + rand(n_2, 1);
    for i=1:max_epochs
        % FORWARD PASS
        % 1. First Layer
        n1 = W1*X + b1;
        a1 = logsig(n1);
        % 2. Second Layer
        n2 = W2*a1 + b2;
        a2 = purelin(n2);
        % ERROR CALCULATION
        error = Y - a2;
        loss_hist(i) = mean(error.^2, "all");
        % relaxing mu w.r.t. previous loss
        if i > 1
            if loss_hist(i) >= loss_hist(i-1)
                mu = mu*theta;
            else
                mu = mu/theta;
            end
        end
        % BACKWARD PASS
        % Calculating Marquardt Sensitivities
        S2 = -dpurelin(n2, a2);
        S1 = a1.*(1-a1).*(W2'*S2);
        % Jacobian calculation
        % parameter vector : [W1' b1' W2 b2]
        params_W1 = numel(W1);
        params_b1 = numel(b1);
        params_W2 = numel(W2);
        params_b2 = numel(b2);
        len_params = params_W1 + params_b1 + params_W2 + params_b2;
        jac = zeros([size(X, 2), len_params]);
        % formulae taken from P2-24
        for j=1:size(X,2)
            for k=1:params_W1
                jac(j, k) = S1(k, j)*X(j);
            end
            for k=1:params_b1
                jac(j, k+params_W1) = S1(k, j);
            end
            for k=1:params_W2
                jac(j, k+params_W1+params_b1) = S2(j)*a1(k, j);
            end
            for k=1:params_b2
                jac(j, k+params_W1+params_b1+params_W2) = S2(j);
            end
        end
        d_param = -inv(jac'*jac + mu*eye(len_params))*jac'*error';
        dW1 = d_param(1:params_W1);
        db1 = d_param(1+params_W1:params_W1+params_b1);
        dW2 = d_param(1+params_W1+params_b1:params_W1+params_b1+params_W2)';
```

```
        db2 = d_param(1+params_W1+params_b1+params_W2:end)';
        % Weights and Biases Update
        W1 = W1 + dW1;
        b1 = b1 + db1;
        W2 = W2 + dW2;
        b2 = b2 + db2;
        if mod(i,5) == 0
            fprintf('Loss at the end of epoch %d: %f\n', i, loss_hist(i));
        end
    end
end
```