VLBP(X, Y, hidden, eta, gamma, zeta, rho, nu, max_epochs): trains a 2-layers MLP to fit X onto Y using variable learning rate steepest descent backpropagation.

Args:

- X: network inputs
- Y: network targets
- hidden: number of hidden neurons
- η: learning rate
- γ: momentum factor
- ζ : error tolerance
- ρ : decrease factor
- ν : increase factor
- $\mathrm{max\_epochs}$: 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.
- $\mathrm{loss\_hist}$: a history of cost values of different epochs in order of increasing epoch numbers.
- $\mathrm{lr\_hist}$: a history of learning rate changes for different epochs in order of increasing epoch numbers.

In order to calculate the weights we will use the variable learning rate steepest descent backpropagation algorithm as discussed in the textbook. The MSE performance index is assumed.

Regarding the momentum terms, for each weight and bias a running sum V is kept and they are updated at each epoch using:

$$V_{t+1} = \gamma V_t - (1-\gamma)\nabla_t$$

Where ∇_t is the gradient w.r.t. that particular weight or bias at that iteration. Finally, the parameters are updated according to:

$$W_{t+1}\,(\text{or } b_{t+1}) = W_t - V_{t,W}\,(\text{or } b_{t+1} - V_{t,b}).$$

The variable learning rate rules are taken from Page 12-12 of the textbook.

```
function [W1, W2, b1, b2, loss_hist, lr_hist] = VLBP(X, Y, hidden, eta, gamma, zeta, rho, nu, r
    % 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]);
    % learning rate history
    lr_hist = zeros([max_epochs, 1]);
    % weights and biases initialization (random between -0.5 and 0.5)
    W1 = -0.5 + randn(n_1, n_0);
    W2 = -0.5 + randn(n_2, n_1);
```

```
    b1 = -0.5 + randn(n_1, 1);
    b2 = -0.5 + randn(n_2, 1);
    % momentum running sums
    V_W1 = 0;
    V_W2 = 0;
    V_b1 = 0;
    V_b2 = 0;
    [~, Q] = size(X);
    prev_mse = inf;
    gamma_orig = gamma;
    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;
        % BACKWARD PASS
        % Calculating Sensitivities
        S2 = 2*error;
        S1 = a1.*(1-a1).*(W2'*S2);
        % Calculating Weight and Bias Updates
        dW1 = S1*X'*eta;
        db1 = S1*ones(Q,1)*eta;
        dW2 = S2*a1'*eta;
        db2 = S2*ones(Q,1)*eta;
        curr_mse = mean(error.^2, "all");
        lr_hist(i) = eta;
        if curr_mse <= prev_mse
            % ACCEPT UPDATES
            V_W1 = gamma*V_W1 - (1-gamma)*dW1;
            V_W2 = gamma*V_W2 - (1-gamma)*dW2;
            V_b1 = gamma*V_b1 - (1-gamma)*db1;
            V_b2 = gamma*V_b2 - (1-gamma)*db2;
            % Updating Weights and Biases
            W1 = W1 - V_W1;
            W2 = W2 - V_W2;
            b1 = b1 - V_b1;
            b2 = b2 - V_b2;
            eta = eta*nu;
            gamma = gamma_orig;
        elseif curr_mse > (1 + zeta)*prev_mse
            % DISCARD UPDATES
            eta = eta*rho;
            gamma = 0;
        else
            % CAUTIOUSLY ACCEPT UPDATE
            V_W1 = gamma*V_W1 - (1-gamma)*dW1;
            V_W2 = gamma*V_W2 - (1-gamma)*dW2;
            V_b1 = gamma*V_b1 - (1-gamma)*db1;
            V_b2 = gamma*V_b2 - (1-gamma)*db2;
```

```
            % Updating Weights and Biases
            W1 = W1 - V_W1;
            W2 = W2 - V_W2;
            b1 = b1 - V_b1;
            b2 = b2 - V_b2;
            gamma = gamma_orig;
        end
        prev_mse = curr_mse;
        loss_hist(i) = curr_mse;
        if mod(i,5) == 0
            fprintf('Loss at the end of epoch %d: %f\n', i, loss_hist(i));
        end
    end
end
```