MLP(X, Y, hidden, eta): trains a 2-layers MLP to fit X onto Y.

Args:

- X: network inputs
- Y: network targets
- η: learning rate
- $\mathrm{max\_epochs}$: maximum number of allowed epochs.
- $W^1_{\mathrm{init}}, W^2_{\mathrm{init}}, b^1_{\mathrm{init}}, b^2_{\mathrm{init}}$:initial weights and biases

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.
- $\mathrm{cost\_hist}$: a history of cost values of different epochs in order of increasing epoch numbers

In order to calculate the weights we will use the backpropagation algorithm as discussed in the textbook. The MSE performance index is assumed and the algorithm trains for a 1000 epochs without checking for convergence. Note that the number of hidden layers is derived from initial weights and biases. We will also use momentum in the gradient descent step of the algorithm by adding a portion of the previous weights to the current update as follows:

$$V_t = \beta V_{t-1} + \eta \frac{\partial L}{\partial W}, V_0 = 0$$

$$W_{t+1} = W_t - V_t$$

Additionally, in order to obtain stable convergence, gradients updates are applied at the end of each epoch after averaging the gradients w.r.t. the entire dataset.

```
function [W1, W2, b1, b2, cost_hist] = MLP(X, Y, eta, max_epochs, W1_init, W2_init, b1_init, b2
    % cost history
    cost_hist = zeros([max_epochs, 1]);
    % momentum hyperparameter
    beta = 0.9;
    V_W1 = 0;
    V_W2 = 0;
    V_b1 = 0;
    V_b2 = 0;
    W1 = W1_init;
    W2 = W2_init;
    b1 = b1_init;
    b2 = b2_init;
    for i=1:max_epochs
        cost = 0;
        W1_grads = zeros(size(W1));
        W2_grads = zeros(size(W2));
        b1_grads = zeros(size(b1));
        b2_grads = zeros(size(b2));
        for k=1:length(X)
            p = X(:,k);
```

```
            % target
            t = Y(:,k);
            % FORWARD PASS
            % 1. Input Layer
            n1 = W1*p + b1;
            a1 = logsig(n1);

            % 2. Hidden Layer
            n2 = W2*a1 + b2;
            a2 = logsig(n2);

            % 3. Output and Error Calculation
            loss = (t-a2)'*(t-a2); % squared error
            cost = cost + loss;

            % BACKWARD PASS
            % 1. Calculating Sensitivities
            % S2 = -2F_dot2(n2)*error
            % S1 = F_dot1(n1)*W2'*S2
            S2 = -2*diag(dlogsig(n2, a2), 0)*(t-a2);
            S1 = diag(dlogsig(n1, a1), 0)*W2'*S2;

            % 2. Weight Updates

            W1_grads = W1_grads + S1*p';
            W2_grads = W2_grads + S2*a1';

            % 3. Bias Updates
            b1_grads = b1_grads + S1;
            b2_grads = b2_grads + S2;
        end
        V_W1 = beta*V_W1 + eta*W1_grads/length(X);
        V_W2 = beta*V_W2 + eta*W2_grads/length(X);
        V_b1 = beta*V_b1 + eta*b1_grads/length(X);
        V_b2 = beta*V_b2 + eta*b2_grads/length(X);
        W1 = W1 - V_W1;
        W2 = W2 - V_W2;
        b1 = b1 - V_b1;
        b2 = b2 - V_b2;
        if mod(i,100) == 0
            fprintf('Loss at the end of epoch %d: %f\n', i, cost/length(X));
        end
        cost_hist(i) = cost/length(X);
    end
end
```