CGBP(X, Y, hidden, tol, max_epochs): trains a 2-layers MLP to fit X onto Y using conjugate gradient backpropagation.

Args:

- X: network inputs
- Y: network targets
- hidden: number of hidden neurons
- ϵ : epsilon
- tol: tolerance value used for line search
- 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 calculate the gradients using backpropagaton and the conjugate gradient method for the direction of the next step as discussed in the textbook starting from page 12-14. The MSE performance index is assumed.

Below is a brief description of the algorithm:

- Errors w.r.t. all inputs are calculated and the gradients of each of parameters with respect to these inputs are calculated and averaged.
- At the first step, the first direction is set to the negative of the gradient w.r.t. each parameter.
- In order to find the step size, we will minimize the performance index (here, MSE) in along the line specified by the direction. To that end, we will perform a line search with an initial step size (ϵ), and we will double the step size until we see an increase in performance index. This gives us a search interval for the optimal step size.
- Then we will use the Golden Section Search to reduce this interval and find the step size that minimizes the performance index along the direction specified.
- We will take a step in the direction specified and calculate the next direction with one of the formulae from P12-25 (eq 12.16).
- If we have taken n steps (where n is the dimension of the feature vector, here "1"), reset the direction to the negative of the gradient.

```
function [W1, W2, b1, b2, loss_hist] = CGBP(X, Y, hidden, epsilon, tol, 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);
    [RESET, Q] = size(X);
    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");
        % BACKWARD PASS
        % Calculating Sensitivities
        S2 = 2*error;
        S1 = a1.*(1-a1).*(W2'*S2);
        % Calculating Gradients
        % keep the previous gradient
        dW1 = S1*X';
        db1 = S1*ones(Q,1);
        dW2 = S2*a1';
        db2 = S2*ones(Q,1);
        if i == 1
            prev_dW1 = dW1;
            prev_db1 = db1;
            prev_dW2 = dW2;
            prev_db2 = db2;
        end
        % set search direction
        if mod(i,RESET) == 0
            pW1 = dW1;
            pb1 = db1;
            pW2 = dW2;
            pb2 = db2;
        else
            beta_W1 = (dW1'*dW1)/(prev_dW1'*prev_dW1);
            prev_dW1 = dW1;
            pW1 = dW1 + beta_W1*pW1;
            beta_b1 = (db1'*db1)/(prev_db1'*prev_db1);
            prev_db1 = db1;
            pb1 = db1 + beta_b1*pb1;
            beta_W2 = (dW2'*dW2)/(prev_dW2'*prev_dW2);
            prev_dW2 = dW2;
            pW2 = dW2 + beta_W2*pW2;
            beta_b2 = (db2'*db2)/(prev_db2'*prev_db2);
            prev_db2 = db2;
            pb2 = db2 + beta_b2*pb2;
        end
        % Interval Selection (Line Search)
        % keep a history of epsilon values, initialized with 0
        epsilon_history = [0];
```

```
        % Start from current F(X)
        f_eval_prev = mean(error.^2, "all");
        epsilon_lower = 0;
        epsilon_upper = 0;
        coeff = epsilon;
        pos = 1;
        % Keep increasing epsilon until error is increased
        while true
            epsilon_history = [epsilon_history coeff];
            pos = pos + 1;
            % net output
            a2 = purelin((W2+coeff*pW2)*(logsig((W1+coeff*pW1)*X + (b1+coeff*pb1))) + (b2+coef
            error = Y - a2;
            f_eval_curr = mean(error.^2, "all");
            if f_eval_curr > f_eval_prev
                % set the epsilon upper bound to current coefficient
                epsilon_upper = coeff;
                % set the epsilon lower bound to two positions before in
                % the history
                epsilon_lower = epsilon_history(pos - 2);
                break;
            end
            coeff = 2*coeff;
            f_eval_prev = f_eval_curr;
        end
        % Golden Section Search
        tau = 0.618;
        c = epsilon_lower + (1-tau)*(epsilon_upper - epsilon_lower);
        eval_c = Y - purelin((W2+c*pW2)*(logsig((W1+c*pW1)*X + (b1+c*pb1))) + (b2+c*pb2));
        d = epsilon_upper - (1-tau)*(epsilon_upper - epsilon_lower);
        eval_d = Y - purelin((W2+d*pW2)*(logsig((W1+d*pW1)*X + (b1+d*pb1))) + (b2+d*pb2));
        while epsilon_upper - epsilon_lower > tol
            if eval_c < eval_d
                epsilon_upper = d;
                d = c;
                eval_d = Y - purelin((W2+d*pW2)*(logsig((W1+d*pW1)*X + (b1+d*pb1))) + (b2+d*pb
                c = epsilon_lower + (1-tau)*(epsilon_upper -epsilon_lower);
                eval_c = Y - purelin((W2+c*pW2)*(logsig((W1+c*pW1)*X + (b1+c*pb1))) + (b2+c*pb
            else
                epsilon_lower = c;
                c = d;
                eval_c = Y - purelin((W2+c*pW2)*(logsig((W1+c*pW1)*X + (b1+c*pb1))) + (b2+c*pb
                d = epsilon_upper - (1-tau)*(epsilon_upper - epsilon_lower);
                eval_d = Y - purelin((W2+d*pW2)*(logsig((W1+d*pW1)*X + (b1+d*pb1))) + (b2+d*pb
            end
        end
        step_size = (epsilon_upper + epsilon_lower)/2;
        % Updating Weights and Biases
        W1 = W1 + step_size*dW1;
        W2 = W2 + step_size*dW2;
        b1 = b1 + step_size*db1;
        b2 = b2 + step_size*db2;
        if mod(i,5) == 0
            fprintf('Loss at the end of epoch %d: %f\n', i, loss_hist(i));
```

```
        end
    end
end
```