1. $P(z_1 = L) \times P(6|L) \times P(F|L) \times P(3|F) \times P(F|F) \times P(1|F) \times P(L|F)$

$\times P(2|L) \times P(L|L) \times P(4|L)$

$= \frac{1}{2} \times \frac{1}{2} \times 0.4 \times \frac{1}{6} \times 0.6 \times \frac{1}{6} \times 0.4 \times \frac{1}{10} \times 0.6 \times \frac{1}{10}$

$= 4 \times 10^{-6}$

2.

$V_k(t+1) = (e_{k',x_{t+1}}) \max_{k'}(a_{k,k'} \cdot V(t))$

The V Matrix

	t = 1	t = 2	t = 3
k = 1 (Fair)	$\frac{1}{6} * \frac{1}{2}$	$\frac{1}{6} \times 0.6 \times \frac{1}{12} = \frac{1}{120}$	$\frac{1}{6} \times 0.4 \times \frac{1}{60} = \frac{1}{900}$
k = 2 (Loaded)	$\frac{1}{10} * \frac{1}{2}$	$\frac{1}{2} \times 0.4 \times \frac{1}{12} = \frac{1}{60}$	$\frac{1}{10} \times 0.6 \times \frac{1}{60} = \frac{1}{1000}$
observation	"4"	"6"	"3"

The Ptr Matrix

	t = 1	t = 2	t = 3
k = 1 (Fair)	0	1	2
k = 2 (Loaded)	0	1	2
observation	"4"	"6"	"3"

$$\max V_K(n) = \frac{1}{6} \times \frac{1}{2} \times 0.4 \times \frac{1}{2} \times 0.4 \times \frac{1}{6} = \frac{1}{900}$$

$$\begin{array}{ccc} \boxed{1} & 1 & \boxed{1} \\ \\ 2 & \boxed{2} & 2 \end{array}$$

$$P(x, z^*) = \frac{1}{900}$$

$$z^* = (1.\ 2.\ 1)$$

3 Feed Forward Neural Networks

%% activation tanh.m

```
function y = activation_tanh(alpha)

    y = (exp(alpha) - exp(-alpha)) ./ (exp(alpha) + exp(-
alpha));
end
```

%% activation tanh gradient.m

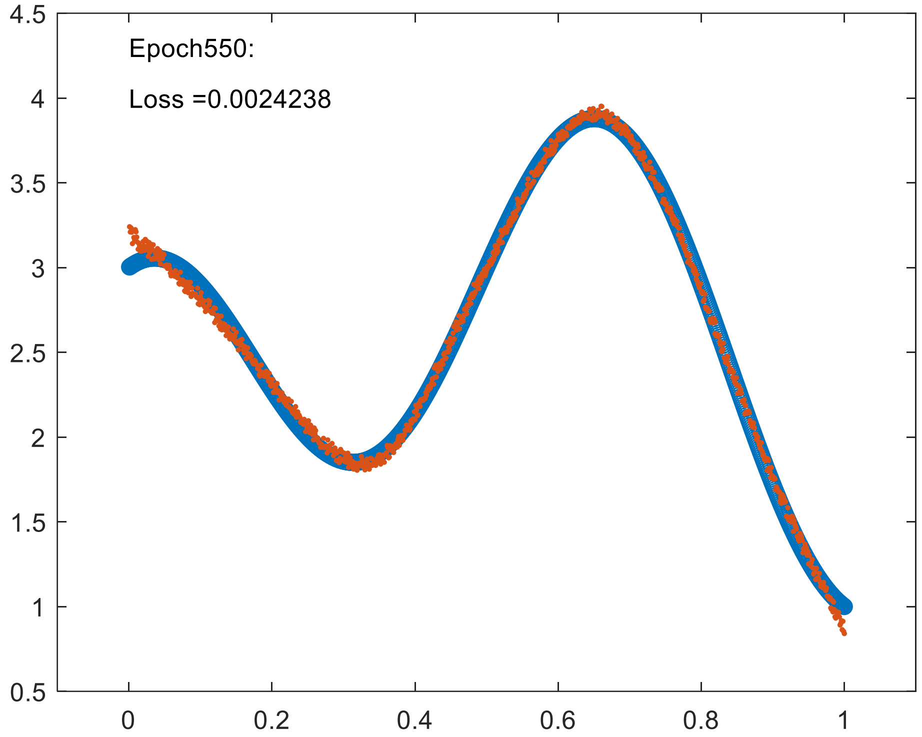
```
function gradient = activation_tanh_gradient(y)

    gradient = 1 - y .^ 2;
end
```

%% feedforward network tanh.m

```
%clear memory
clear;

%% prepare data and initialize the network
% Generate 1000 training data points
X_all = [0.001 * (1 : 1000)' 0.001*(1000-(1:1000))']; % shape
[1000, 2]
y_all = cos(3 * pi * X_all(:, 1)) + sin(pi* X_all(:, 2)) +
2; % shape [1000, 1]
n_sample = length(X_all(:, 1));

% Initialization network parameters.
% The training data are split to many "batches".
% In this lab, each batch has 50 data points
% Model training (the forward-backward propagation)
% is performed on batch-by-batch.
batch_size = 50;
W1 = 0.01*randn(2, 256); % shape [2, 256]
W2 = 0.01*randn(256, 256); % shape [256, 256]
W3 = 0.01*randn(256, 1); % shape [256, 1]

% variables for optimization algorithm: AdamSGD
% for this lab, you do not have to know them
% just regard optimizer as a function (blackbox).
m_W1 = 0; v_W1 = 0;
m_W2 = 0; v_W2 = 0;
m_W3 = 0; v_W3 = 0;
step = 0;

%% start training
%prepare to visulize the results
figure = gcf();
%An epoch means an iteration over all 1000 training data
points.
%we let the program to finish after 550 epoches.
for epoch = 1:550
```

```
    % permute the training dataset for each epoch
    perm = randperm(1000);
    Xp = X_all(perm,:);
    yp = y_all(perm);

    total_error = 0;
    y_pred = zeros(n_sample, 1); % variable that saves the
prediction

    for i = 1:(n_sample/batch_size)
        %% Prepare data for the current batch
        d_start = batch_size * (i - 1);
        X_batch = Xp(d_start+(1:batch_size), :);
        y_batch = yp(d_start+(1:batch_size));

        %% ---------------------FORWARD PROPAGATION-----------
--------
        X = X_batch;  % dense1

        layer1_alpha = weighted_sum(X, W1);
        layer1_h = activation_tanh(layer1_alpha);

        layer2_alpha = weighted_sum(layer1_h, W2);
        layer2_h = activation_tanh(layer2_alpha);

        output_layer_alpha = weighted_sum(layer2_h, W3);
        output_layer = output_layer_alpha;

        error = mean((output_layer-y_batch).^2);

        %% -------------------BACKPROP-----------------------
        output_layer_gradient = 2*(output_layer-
y_batch)/batch_size;

        % calculate gradients w.r.t. W3 and h2 (see defination
of W3 and h2 in the figure of the handout)
        [W3_gradient, layer2_h_gradient] =
compute_gradient_for_weights_and_one_layer_below(output_layer_
gradient, W3, layer2_h);

        % add some code below
        % to calculate gradients of error w.r.t. alpha_2 (see
defination of alpha_2 in the figure of the handout)
        layer2_alpha_gradient = layer2_h_gradient .*
activation_tanh_gradient(layer2_h);

        % calculate gradients w.r.t. W2 and h1 (see defination
of W3 and h2 in the figure of the handout)
        [W2_gradient, layer1_h_gradient] =
compute_gradient_for_weights_and_one_layer_below(layer2_alpha_
gradient, W2, layer1_h);

        % add some code below
        % to calculate gradients of error w.r.t. alpha_1 (see
defination of alpha_1 in the figure of the handout)
```

```
        layer1_alpha_gradient = layer1_h_gradient .*
activation_tanh_gradient(layer1_h);

        % calculate gradients w.r.t. W1 and X (see defination
of W1 in the figure of the handout)
        [W1_gradient, ~] =
compute_gradient_for_weights_and_one_layer_below(layer1_alpha_
gradient, W1, X);

        % update error and prediction
        total_error = total_error + error;
        y_pred(d_start+(1:batch_size)) = output_layer;


        %% ---------------------Update-----------------------
-
        % Using optimizer: Adam SGD
        % For reference see: https://arxiv.org/abs/1412.6980
        % for this lab, you do not have to know it
        % just regard optimizer as a function (blackbox).
        step = step + 1;
        m_W1 = (0.9 * m_W1 + 0.1 * W1_gradient);
        v_W1 = (0.999 * v_W1 + 0.001 * W1_gradient.^2);
        W1 = W1 - 0.01 * (m_W1/(1-0.9^step)) ./ sqrt(v_W1/(1-
0.999^step) + 1e-8);

        m_W2 = (0.9 * m_W2 + 0.1 * W2_gradient);
        v_W2 = (0.999 * v_W2 + 0.001 * W2_gradient.^2)/(1-
0.999^step);
        W2 = W2 - 0.01 * (m_W2/(1-0.9^step)) ./ sqrt(v_W2/(1-
0.999^step) + 1e-8);

        m_W3 = (0.9 * m_W3 + 0.1 * W3_gradient);
        v_W3 = (0.999 * v_W3 + 0.001 * W3_gradient.^2);
        W3 = W3 - 0.01 * (m_W3/(1-0.9^step)) ./ sqrt(v_W3/(1-
0.999^step) + 1e-8);

    end

    % visulize the results
    fprintf('Epoch %d ...\n', epoch);
    fprintf('Squared Loss: %.4f\n\n',
total_error*(batch_size/n_sample))
    hold off

    plot(X_all(:,1), y_all, 'o')
    hold on
    plot(Xp(:,1), y_pred, '.')
    axis([-0.1, 1.1, 0.5, 4.5])
    text(0, 4.3, strcat('Epoch   ', num2str(epoch), ':'))
    text(0, 4.0, strcat('Loss = ' ,
num2str(total_error*(batch_size/n_sample))));
    drawnow

end
```



Epoch550:
Loss =0.0024238
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0.2
0.4
0.6
0.8
1