**Supervised Learning**: given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Regression problem: predict results within a continuous output, meaning that we are trying to map input variables to some continuous function.

Classification problem: predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

**Unsupervised Learning**: little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. With unsupervised learning there is no feedback based on the prediction results.

Clustering: automatically group a collection of 1,000,000 different genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

Non-clustering: The "Cocktail Party Algorithm" -> i.e. identifying individual voices and music from a mesh of sounds at a cocktail party

1. Regularized logistic regression

function [jVal, gradient] = costFunction(theta)

jVal = [...code to compute J(theta)...];

gradient = zeros(n+1,1)

gradient(1) = [...];

gradient(2) = [...];

…

gradient(n+1) = [...];

end

options = optimset(‘GradObj’,’on’,’MaxIter’,100)

initialTheta = zeros(n+1,1)

[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options)

1. Neural network – backpropagation algorithm

% Example: s1=10, s2=10, s3=1

thetaVec = [Theta1(:); Theta2(:); Theta3(:)];

DVec = [D1(:); D2(:); D3(:)];

Theta1 = reshape(thetaVec(1:110),10,11);

Theta2 = reshape(thetaVec(111:220),10,11);

Theta3 = reshape(thetaVec(221:231),1,11);

function [jVal, gradient] = costFunction(thetaVec)

Use forward prop/back prop to compute D1,D2,D3 and J(theta)

Unroll D1,D2,D3 to get gradientVec

% Gradient checking – to verify that backprop is correct

epsilon = 1e-4;

for i = 1:n,

thetaPlus = theta;

thetaPlus(i) += epsilon;

thetaMinus = theta;

thetaMinus(i) -= epsilon;

gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2\*epsilon)

end;

% Once verified that backpropagation algorithm is correct, no need to compute gradApprox again. Otherwise the code to compute gradApprox can be very slow.

% Random initialization (symmetry breaking) -> initialize each theta to a random value in [-epsilon,epsilon]

Theta1 = rand(10,11) \* (2 \* INIT\_EPSILON) - INIT\_EPSILON; %random 10\*11 matrix in [0,1]

Theta2 = rand(10,11) \* (2 \* INIT\_EPSILON) - INIT\_EPSILON;

Theta3 = rand(1,11) \* (2 \* INIT\_EPSILON) - INIT\_EPSILON;

for i = 1:m,

Perform forward propagation and backpropagation using example (x(i),y(i))

(Get activations a(l) and delta terms d(l) for l = 2,...,L

optTheta = fminunc(@costFunction, initialTheta, options)