

## Week 8

### XIII (I) - Introduction to Unsupervised Learning

We are provided with unlabeled data and the task of the algorithm is to find some structure in the data. Clustering is one of the examples of unsupervised learning.

### XIII (II) - K-Means Algorithm

Algorithm to find k-clusters in a given set of unlabeled data. During the running of the algorithm it is possible to find out that a particular data does not have k-clusters. Usually, we reject those "invalid" centroids and let the algorithm naturally find suitable clusters. Algorithm also finds clusters in seemingly non-clustered data.

## K-means algorithm

$$\mu_1 \quad \mu_2$$

Randomly initialize  $K$  cluster centroids  $\underline{\mu}_1, \underline{\mu}_2, \dots, \underline{\mu}_K \in \mathbb{R}^n$

Repeat {

*cluster assignment loop*

for  $i = 1$  to  $m$

$\underline{c}^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

$\min_k \|x^{(i)} - \mu_k\|^2$

*new centroid*

for  $k = 1$  to  $K$

$\rightarrow \mu_k :=$  average (mean) of points assigned to cluster  $k$

$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)}$   $\rightarrow c^{(1)}=2, c^{(5)}=2, c^{(6)}=2, c^{(10)}=2$

$\mu_2 = \frac{1}{4} [x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}] \in \mathbb{R}^n$

}

## XIII (III) - Optimization Objective

The basic objective is minimize the sum of square distances of each point to its corresponding centroid. The cost function can be plotted to check for the correctness of k-mean algorithm. The cost function should consistently decrease.

→  $c^{(i)}$  = index of cluster (1,2,...,K) to which example  $x^{(i)}$  is currently assigned

→  $\mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )  $K$   $k \in \{1, 2, \dots, K\}$

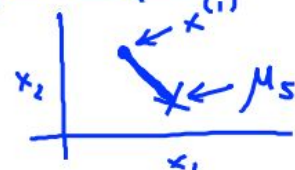
$\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned  
 $x^{(i)} \rightarrow \underline{5}$        $\underline{c^{(i)} = 5}$        $\underline{\mu_{c^{(i)}} = \mu_5}$

Optimization objective:

→  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \boxed{\|x^{(i)} - \mu_{c^{(i)}}\|^2}$  ←

→  $\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

Distortion



### XIII (IV) - Random Initialization

For K clusters out of m training examples:

1. Randomly pick K training examples
2. Set  $\mu_1, \mu_2, \mu_3, \dots$  etc equal to K training examples.

If number of clusters is fairly small, it is better to try 50-1000 random initialization, and pick the one result with the smallest cost function.

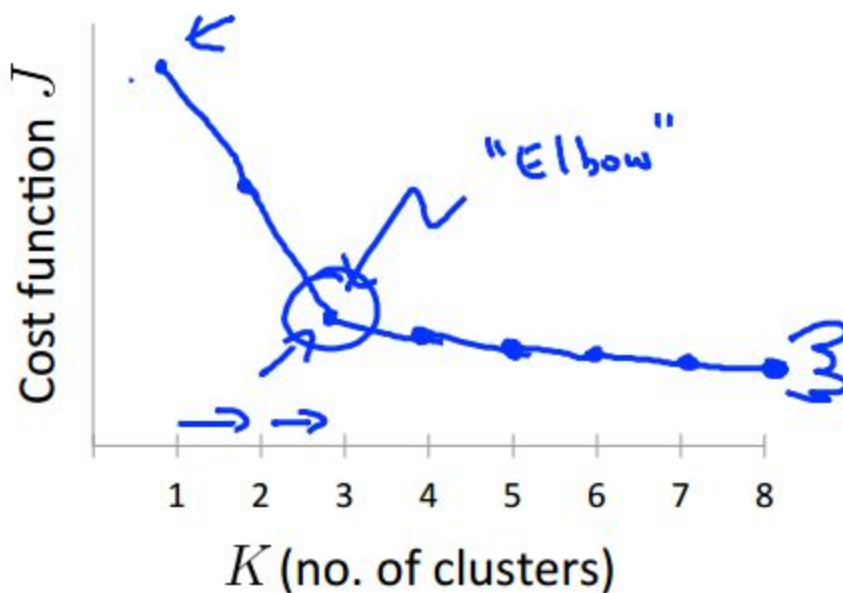
For large value of K, using a single random initialization is usually sufficient.

### XIII (V) - Choosing K

Elbow method :

- Plot cost( $J$ ) against number of clusters( $K$ ). The value where an "elbow" appears, is a good choice for  $K$
- The method would not work in most cases, as "elbow" may not be very clear in each case. This may be tried however.

Elbow method:



The best way to choose  $K$ , is to evaluate how well it suits our purpose, for which we are using clustering.

=====

## XIV (I & II) - Dimensionality Reduction

It is often possible that out of hundreds/thousands of features of training examples, strong correlation may exist among some features. In such a case, it is better to project those features into a smaller subset of features.

Dimensionality reduction is reducing dimension of feature vectors for each training example. This helps in reducing computational complexity.

## XIV (III) - Principal Component Analysis

- Algorithm to reduce dimensionality of features.
- It is important to use feature scaling and mean normalization before using PCA algorithm.

Find projection of  $n$  dimensional vector on the linear subspace spanned by these  $k$  vectors :  $u^{(1)}, u^{(2)}, u^{(3)} \dots u^{(k)}$

Reduce from  $n$ -dimension to  $k$ -dimension: Find  $k$  vectors  $u^{(1)}, u^{(2)}, \dots, u^{(k)}$  ←  
onto which to project the data, so as to minimize the projection error.

## XIV (IV) - PCA Algorithm

The idea is to find a  $k$ -dimensional subspace ( $k < n$ ) such that the projections of training examples on these lines have minimum lengths.

We are doing this in the algorithm below:

1. Compute covariance matrix ( $n \times n$  sigma matrix)
2. Compute eigenvectors of the matrix sigma
3.  $U$  matrix will be  $n \times n$  matrix. We take the first  $k$  vectors (columns) of this matrix. This gives us  $u^{(1)}, u^{(2)}, u^{(3)} \dots u^{(k)}$  vectors of our linear subspace (Ureduce).
4.  $Ureduce^T * x$  gives us a  $k \times 1$  vector. So we will get projection of training example  $x$  on this linear subspace of  $k$ -dimension spanned by  $u^{(1)}, u^{(2)}, u^{(3)} \dots u^{(k)}$

## Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

→  $[U, S, V] = \text{svd}(\text{Sigma})$ ;

→  $Ureduce = U(:, 1:k)$ ;

→  $z = Ureduce' * x$ ;

↑

↑

$x \in \mathbb{R}^n$

~~$x_0 = 1$~~

$$X = \begin{bmatrix} - & x^{(1)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix}$$

→  $\text{Sigma} = (1/m) * X' * X$

## XIV (V) - Choosing Number K

The objective is to choose a  $k$  so that the total squared error is minimized while the variation in the data is below 1%.

Average squared projection error:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$   
 Total variation in the data:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

Typically, choose  $k$  to be smallest value so that

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq \frac{0.01}{0.05} \quad \frac{(1\%)}{5\%} \quad (10\%)$$

> "~~99%~~ of variance is retained"  
~~95%~~ to 90%.

Matrix  $S$  is a diagonal matrix with all non-diagonal entries as zeros.  
 The above computation is much more simplified if we use matrix  $S$

Check if

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01?$$

$k = 17$

For given  $k$

$$1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \leftarrow$$

Procedure to compute  $k$  :



- We start with assumption  $k = 1$
- Imp:  $S$  is computed only once. The value of  $k$  is incremented using the same  $S$  until we obtain our ideal value of  $k$ .

## Choosing $k$ (number of principal components)

➤  $[U, S, V] = \text{svd}(\text{Sigma})$

Pick smallest value of  $k$  for which

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$$

$k=100$

(99% of variance retained)

## XIV (VI) - Reconstruction from Compressed Representation

An approximation of the training example  $x^{(i)}$  can be reconstructed using simple formula :

$$x^{(i)} = U_{\text{reduce}} * z^{(i)}$$

## XIV (VII) - Advice For Applying PCA



For a supervised learning example :

1. Extract inputs ( $x$ 's from training examples) setting aside labels ( $y$ 's in training example)
2. Apply PCA to get projections of inputs in a linear subspace of smaller dimensions represented as  $z(1), z(2), \dots, z(m)$
3. We will use the new training set  $(z(1), y(1)), (z(2), y(2)), \dots$  for training our algorithm

### Supervised learning speedup

$(\underline{x^{(1)}}, y^{(1)}), (\underline{x^{(2)}}, y^{(2)}), \dots, (\underline{x^{(m)}}, y^{(m)})$

Extract inputs:

Unlabeled dataset:  $\underline{x^{(1)}}, \underline{x^{(2)}}, \dots, \underline{x^{(m)}} \in \mathbb{R}^{10000} \leftarrow$

$\downarrow PCA$

$\underline{z^{(1)}}, \underline{z^{(2)}}, \dots, \underline{z^{(m)}} \in \mathbb{R}^{1000} \leftarrow$

New training set:

$(\underline{z^{(1)}}, y^{(1)}), (\underline{z^{(2)}}, y^{(2)}), \dots, (\underline{z^{(m)}}, y^{(m)})$

$$h_{\theta}(z) = \frac{1}{1 + e^{-\theta^T z}}$$

Note: Mapping  $x^{(i)} \rightarrow z^{(i)}$  should be defined by running PCA only on the training set. This mapping can be applied as well to the examples  $x_{cv}^{(i)}$  and  $x_{test}^{(i)}$  in the cross validation and test sets.

- PCA should not be used to prevent overfitting ! Overfitting should be handled with regularization.
- PCA should not be applied straightaway. It is better to apply algorithm without PCA. Only if this does not work and there is some reason to believe that PCA would help in computation, it should be applied.

## XIV (I) - Anomaly Detection: Problem Motivation

Anomaly detection is simply identifying data points in our dataset which are lying outside general range (identical to the concept of outliers) over a certain feature(s).

→ Fraud detection:

→  $x^{(i)}$  = features of user  $i$ 's activities

→ Model  $p(x)$  from data.

→ Identify unusual users by checking which have  $p(x) < \varepsilon$

$x_2$   
 $x_3$   
 $x_4$        $p(x)$

## XIV (II) - Gaussian Distribution

It is also called normal distribution. General representation of a Gaussian distribution is  $N(\mu, \sigma^2)$

**Properties of the Standard Normal Curve (Z):**

1. The highest point occurs at  $\mu=0$ .
2. It is a bell-shaped curve that is symmetric about the mean,  $\mu=0$ . One half of the curve is a mirror image of the other half, i.e., the area under the curve to the right of  $\mu=0$  is equal to the area under the curve to the left of  $\mu=0$  equals  $\frac{1}{2}$ .
3. It has inflection points at  $\mu-\sigma = 0-1 = -1$  and  $\mu+\sigma = 0+1 = +1$ .
4. The curve is asymptotic to the horizontal axis at the extremes.
5. The total area under the curve equals one.
6. Empirical Rule:
  - Approximately 68% of the area under the curve is between -1 and +1.
  - Approximately 95% of the area under the curve is between -2 and +2.
  - Approximately 99.7% of the area under the curve is between -3 and +3.

Probability Distribution Function for a Gaussian distribution :

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

### XIV (III) - Algorithm for Anomaly detection

Based on estimating probability distribution of features.

## Anomaly detection algorithm

→ 1. Choose features  $x_i$  that you think might be indicative of anomalous examples.  $\{x^{(1)}, \dots, x^{(n)}\}$

→ 2. Fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\begin{aligned} \rightarrow \mu_j &= \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \\ \rightarrow \sigma_j^2 &= \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2 \end{aligned}$$

Handwritten notes:  $p(x_j; \mu_j, \sigma_j^2)$ ,  $\mu_1, \mu_2, \dots, \mu_n$ ,  $\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$

→ 3. Given new example  $x$ , compute  $p(x)$ :

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if  $p(x) < \varepsilon$

## XIV (IV) - Developing an Anomaly Detection System

This involves using supervised learning concept for evaluation of algorithm for detecting anomalies.

- In the training set (usually 60% of data), we assume there are no anomalous cases (it is ok if a few such cases creep in)
- Cross validation set and Test set both containing about 20% of non-anomalous cases. Anomalous examples are equally assigned to both sets. (There are very few anomalous cases here compared to anomalous cases)

## Algorithm evaluation

- Fit model  $p(x)$  on training set  $\{x^{(1)}, \dots, x^{(m)}\}$
- On a cross validation/test example  $x$ , predict

$(x_{\text{test}}^{(i)}, y_{\text{test}}^{(i)})$   
↑

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases} \quad \underline{y=0}$$

Possible evaluation metrics:

- - True positive, false positive, false negative, true negative
- - Precision/Recall
- -  $F_1$ -score ←

CV

Test set

Can also use cross validation set to choose parameter  $\varepsilon$  ←

## XIV (V) - Anomaly Detection Vs. Supervised Learning

The difference basically lies in number of positive examples. In anomaly detection, we have very few anomaly examples and a new anomaly may be of entirely different type, of what we have encountered so far.

Anomaly detection	vs.	Supervised learning
<ul style="list-style-type: none"> <li>→ Very small number of positive examples (<math>y = 1</math>). (0-20 is common).</li> <li>→ Large number of negative (<math>y = 0</math>) examples. <math>p(x)</math></li> <li>→ Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like;</li> <li>→ future anomalies may look nothing like any of the anomalous examples we've seen so far.</li> </ul>		<ul style="list-style-type: none"> <li>Large number of positive and negative examples. ←</li> <li>Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set. ←</li> <li>Spam ←</li> </ul>

Anomaly detection	vs.	Supervised learning
<ul style="list-style-type: none"> <li>→ • <u>Fraud detection</u> <math>y=1</math></li> <li>→ • Manufacturing (e.g. aircraft engines)</li> <li>→ • Monitoring machines in a data center</li> </ul>		<ul style="list-style-type: none"> <li>• Email spam classification ←</li> <li>• Weather prediction (sunny/rainy/etc).</li> <li>• Cancer classification ←</li> </ul>
:		:

#### XIV (VI) - What Features to Chose

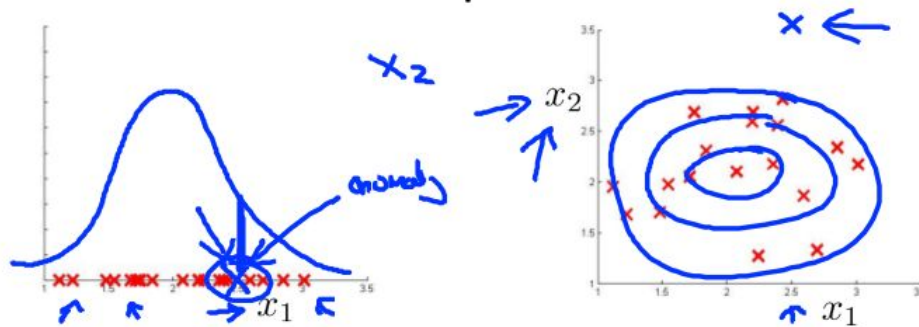
The anomaly detection algorithm described above often works well with Non-Gaussian features. However, non-Gaussian features may be transformed using several types of operators :



- $\log(x + c)$  : we can play with  $c$  to make the feature more normal-like
- $x^{1/p}$  :  $p$  can be changed

Most common problem that we face in anomaly detection is that the probability distribution does not single out anomalous examples from non-anomalous examples.

$p(x)$  is comparable (say, both large) for normal and anomalous examples



In case of failure of algorithm, we have to think about and choose features which are most likely to cause failures :

→ Choose features that might take on unusually large or small values in the event of an anomaly.

- $x_1$  = memory use of computer
- $x_2$  = number of disk accesses/sec
- $x_3$  = CPU load ←
- $x_4$  = network traffic ←

$$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$$

$$x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$



#### XIV (VII) - Multivariate Gaussian Distribution

Multivariate Gaussian Distribution at times may capture anomalies which univariate Gaussian Distribution may not. This is specially useful when we expect the features to be correlated (positive or negative)

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$|\Sigma|$  = determinant of  $\Sigma$       $\det(\text{Sigma})$

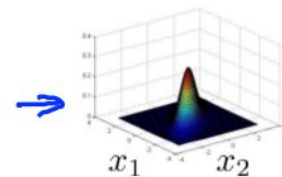
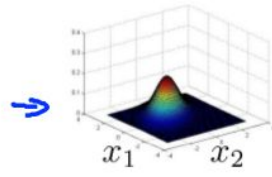
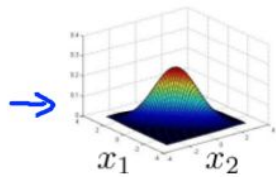
#### XIV (VIII) - Anomaly Detection Using Multivariate Gaussian Distribution

Step 1 : compute mu and sigma for multivariate Gaussian Distribution

Parameters  $\mu, \Sigma$

$$\mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$$

$$\rightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$



Parameter fitting:

Given training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

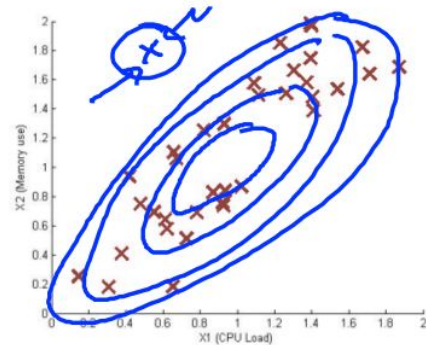
$$x \in \mathbb{R}^n$$

$$\rightarrow \boxed{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad \rightarrow \boxed{\Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

Step 2 : Fit the model in the given data

1. Fit model  $p(x)$  by setting

$$\left[ \begin{aligned} \mu &= \frac{1}{m} \sum_{i=1}^m x^{(i)} \\ \Sigma &= \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T \end{aligned} \right.$$



2. Given a new example  $x$ , compute

$$\left[ p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right) \right.$$

Flag an anomaly if  $p(x) < \underline{\varepsilon}$

The original model is a special case of Multivariate Gaussian Distribution where the matrix sigma is a pure "diagonal matrix"

## → Original model

$$p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$$

Manually create features to capture anomalies where  $x_1, x_2$  take unusual combinations of values.

$$\rightarrow x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$$

Computationally cheaper (alternatively, scales better to large  $n=10,000, n=100,000$ )

OK even if  $m$  (training set size) is small

## vs. → Multivariate Gaussian

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

→ Automatically captures correlations between features

$$\Sigma \in \mathbb{R}^{n \times n}$$

$$\Sigma^{-1}$$

Computationally more expensive

$$\rightarrow \Sigma \sim \frac{n^2}{2}$$

Must have  $m > n$  or else  $\Sigma$  is non-invertible.  $\rightarrow m \geq 10n$

$$\left. \begin{array}{l} \rightarrow x_1 = x_2 \\ x_3 = x_4 + x_5 \end{array} \right\}$$

## Important Point:

- If sigma for multivariate Gaussian Distribution turns out to be non-invertible despite  $m > 10n$ , it is very likely that we have redundant features (features which are linearly dependent), which naturally makes matrix non-invertible.

## XV(I) - Recommender System (Problem Formulation)

Given certain feedback(ratings) by the user, our objective is to develop a recommender system. Notations used for movie ratings:

$\rightarrow n_u = \text{no. users}$   
 $\rightarrow n_m = \text{no. movies}$   
 $r(i, j) = 1$  if user  $j$  has rated movie  $i$   
 $y^{(i,j)}$  = rating given by user  $j$  to movie  $i$  (defined only if  $r(i, j) = 1$ )

## XV(II) - Content Based Recommendations

Given some ratings of the user, we use linear regression algorithm to learn parameters for each user : The vector  $x$  contains feature vector for each movie.

To learn  $\theta^{(j)}$  (parameter for user  $j$ ):

$$\rightarrow \min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2$$

To learn  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$\theta^{(1)}, \dots, \theta^{(n_u)}$

Gradient descent update:

$$J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

$$\frac{1}{m_j} \quad \frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

### XV(III) - Collaborative Filtering

Collaborative filtering deals with the problem of finding features for recommender system problem. We assume that we are given thetas (parameters) for each user:

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(i)}$ :

$$\rightarrow \min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^{(i)})^2 \leftarrow$$

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

## XV(IV) - Collaborative Filtering Algorithm

We can think of two types of cost function :

- First sums over all users  $j$  and then sum over all movie rated by them

Given  $x^{(1)}, \dots, x^{(n_m)}$ , estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \left[ \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2 \right]$$

Handwritten notes:  $x \in \mathbb{R}^n$ ,  $\theta \in \mathbb{R}^n$ ,  $x_i$

- Second sums over all the movies  $i$  and then over all the users who rated that movie

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , estimate  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \left[ \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 \right]$$

We can combine the two sums as in :

Minimizing  $x^{(1)}, \dots, x^{(n_m)}$  and  $\theta^{(1)}, \dots, \theta^{(n_u)}$  simultaneously:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

Handwritten notes:  $\theta \rightarrow x \rightarrow \theta \rightarrow x \rightarrow \dots$

Andrew N

Implementation of Collaborative Filtering:



- 1. Initialize  $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$  to small random values.
- 2. Minimize  $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$  using gradient descent (or an advanced optimization algorithm). E.g. for every  $j = 1, \dots, n_u, i = 1, \dots, n_m$  :

$$x_k^{(i)} := x_k^{(i)} - \alpha \left( \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right) \leftarrow \frac{\partial J(\dots)}{\partial x_k^{(i)}}$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \leftarrow \frac{\partial J(\dots)}{\partial \theta_k^{(j)}}$$

- 3. For a user with parameters  $\theta$  and a movie with (learned) features  $x$ , predict a star rating of  $\theta^T x$ .

$$(\theta^{(j)})^T (x^{(i)})$$

### XV(V) - Low Rank Matrix Factorization

After deducing thetas and x's for each user and movies respectively we can get a big matrix  $Y$  ( $n_m \times n_u$ ) (jth column representing user j's predicted rating for movie i)

Predicted ratings:  $(i,j)$

$$\begin{bmatrix} (\theta^{(1)})^T (x^{(1)}) & (\theta^{(2)})^T (x^{(1)}) & \dots & (\theta^{(n_u)})^T (x^{(1)}) \\ (\theta^{(1)})^T (x^{(2)}) & (\theta^{(2)})^T (x^{(2)}) & \dots & (\theta^{(n_u)})^T (x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T (x^{(n_m)}) & (\theta^{(2)})^T (x^{(n_m)}) & \dots & (\theta^{(n_u)})^T (x^{(n_m)}) \end{bmatrix}$$

$Y$  can be seen as a matrix resulting from product of two matrices  $X$  and  $\Theta$ . Each row of matrix  $X$ , contains  $n$  features of movie. Each row of  $\Theta$  contains  $n$  parameters of a user.



$$\rightarrow X = \begin{bmatrix} -(x^{(1)})^T \\ -(x^{(2)})^T \\ \vdots \\ -(x^{(n_m)})^T \end{bmatrix} \quad \rightarrow \textcircled{L} = \begin{bmatrix} -(\theta^{(1)})^T \\ -(\theta^{(2)})^T \\ \vdots \\ -(\theta^{(n_u)})^T \end{bmatrix}$$

$\rightarrow$  Low rank matrix factorization

Finding related movie is similar to finding closest points in n-dimensional space to our chosen points :

How to find movies  $j$  related to movie  $i$ ?

$$\text{small } \|x^{(i)} - x^{(j)}\| \rightarrow \text{movie } j \text{ and } i \text{ are "similar"}$$

5 most similar movies to movie  $i$ :

Find the 5 movies  $j$  with the smallest  $\|x^{(i)} - x^{(j)}\|$ .

### XV(VI) - Mean Normalization

Mean normalization means finding mean rating for each movie and then using it for making prediction for entirely new user :

## Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

$$\rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:

$$\rightarrow (\theta^{(j)})^T (x^{(i)}) + \mu_i$$

$$\downarrow \text{learn } \underline{\theta^{(j)}}, \underline{x^{(i)}}$$

User 5 (Eve):

$$\underline{\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}}$$

$$\underbrace{(\theta^{(5)})^T (x^{(i)})}_{\rightarrow 0} + \boxed{\mu_i}$$

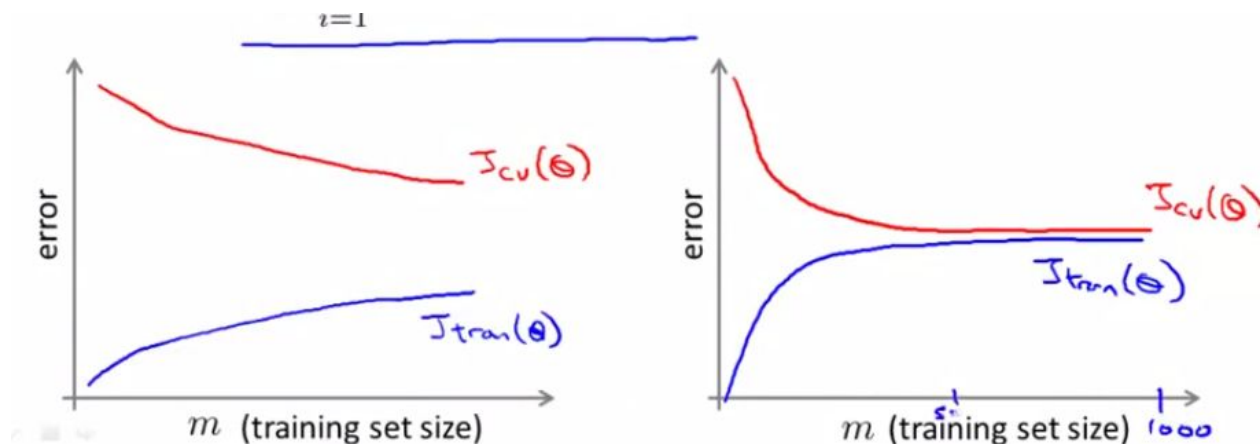
=====

## Week 10

### XVI (I) - Machine Learning with Large Datasets

In a typical machine learning problem of today, we have dataset with millions of training examples. Using so many training examples throws computational complexity at us.

It is advisable to first analyze, whether using so many training examples is useful or not. We should randomly select a small set of training examples (say 1000 or 1%) and plot training error and cross validation error to evaluate our learning algorithm.



- The graph on left indicates high variance in our algo. So using more training examples may be useful.
- The graph on right indicates high bias in our algo. Using more training examples won't help. We need to add more features, or try some other algorithms.

## XVI (II) - Stochastic Gradient Descent

Batch Gradient Descent :

$$\Rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\rightarrow \theta_j := \theta_j - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}}_{\frac{\partial}{\partial \theta_j} J_{train}(\theta)}$$

(for every  $j = 0, \dots, n$ )

}

For large dataset applying batch gradient descent is computationally very expensive. Stochastic Gradient Descent is a modification of it, where the hypothesis is improved over one training example at a time.

The cost function for each training example is defined as:

$$cost(\theta, \underbrace{(x^{(i)}, y^{(i)})}_m) = \frac{1}{2} \underbrace{(h_{\theta}(x^{(i)}) - y^{(i)})^2}$$

Step One : Randomly shuffle dataset

Step two : Stochastic Gradient Descent is run in a loop

$$\begin{aligned} &2. \text{ Repeat } \{ \\ &\quad \text{for } i=1, \dots, m \{ \\ &\quad \quad \theta_j := \theta_j - \alpha \underbrace{(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}_{\text{(for } j=0, \dots, n)} \\ &\quad \} \end{aligned}$$

- Within the loop only one training example is used, making it much more computationally efficient than Batch Gradient Descent
- It may not converge to global minimum, but for a large number of training examples, the algorithm converges to a point much closer to global minimum.

### XVI (III) - Mini Gradient Descent

It is a slight modification of Stochastic Gradient Descent.

- Sometimes may run faster than Stochastic Gradient Descent.
- Feasible and efficient if we use vectorized implementation

Stochastic Gradient Descent uses 1 example in each iteration. Mini Gradient Descent uses  $b$  examples in each iteration.

Say  $b = 10$ ,  $m = 1000$ .

Repeat {

→ for  $i = 1, 11, 21, 31, \dots, 991$  {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every  $j = 0, \dots, n$ )

}

}

#### XVI (IV) - Stochastic Gradient Descent Convergence

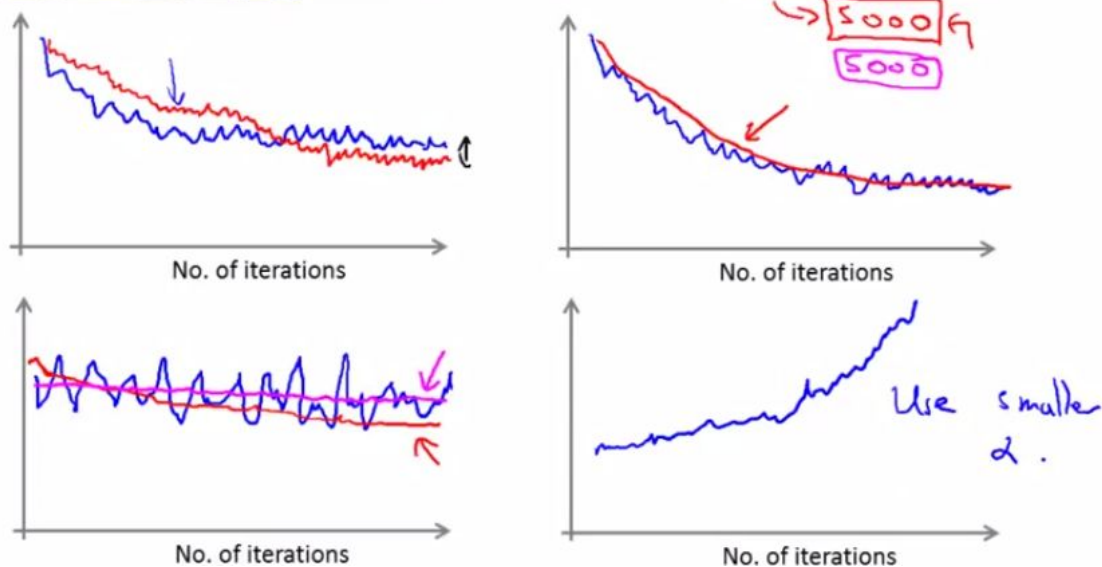
For checking whether Stochastic Gradient Descent is working or not, we can do this :

During learning, compute  $cost(\theta, (x^{(i)}, y^{(i)}))$  before updating  $\theta$   
using  $(x^{(i)}, y^{(i)})$ . ↗ ↗

- Every 1000 iterations (say), plot  $cost(\theta, (x^{(i)}, y^{(i)}))$  averaged over the last 1000 examples processed by algorithm.

## Checking for convergence

Plot  $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ , averaged over the last 1000 (say) examples



The last graph is an example of diverging algorithm. Decreasing the learning rate  $\alpha$  means that each iteration of stochastic gradient descent will take a smaller step, thus it will likely converge instead of diverging.

In most learning algorithms we keep  $\alpha$  constant. But, the stochastic learning algorithm may converge better if we decrease  $\alpha$  in the iteration :

Learning rate  $\alpha$  is typically held constant. Can slowly decrease  $\alpha$  over time if we want  $\theta$  to converge. (E.g.  $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$  )



- But changing alpha using this method needs us to come up with two constants. Tinkering with these two constants may make implementation more complex.
- Second reason is that often without this we still get good parameters for our learning algorithm keeping alpha constant.

## XVI (IV) - Online Learning

If we are getting a continuous stream of data (shopping website) we need to continuously update our learning algorithm one at a time.

In this approach we repeatedly get a single training example, take one step of stochastic gradient descent using that example, and then move on to the next example (Online learning algorithms throw away old examples, incorporating them only once when they are first seen.).

Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ( $y = 1$ ), sometimes not ( $y = 0$ ).

Features  $x$  capture properties of user, of origin/destination and asking price. We want to learn  $p(y = 1|x; \theta)$  to optimize price.

Repeat forever {  
 Get  $(x, y)$  corresponding to user. price logistic regression  
 Update  $\theta$  using  $(x, y)$ :  

$$\theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j \quad (j=0)$$

This helps algorithm in learning with changing user preferences.

### Other online learning example:

Product search (learning to search)

User searches for "Android phone 1080p camera" ←

Have 100 phones in store. Will return 10 results.

→  $x =$  features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc.

→  $y = 1$  if user clicks on link.  $y = 0$  otherwise.

→ Learn  $p(y = 1|x; \theta)$ . ← predicted CTR

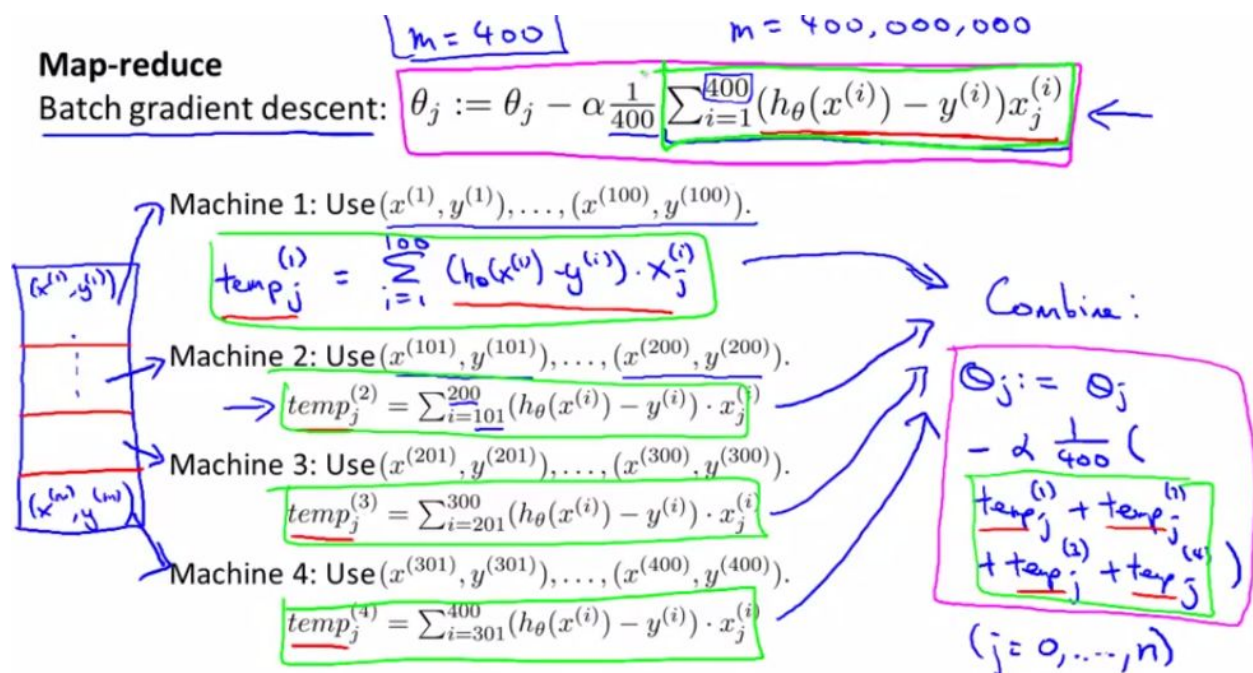
→ Use to show user the 10 phones they're most likely to click on.

Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; ...

## XVI (IV) -Map Reduce (Data Parallelism)

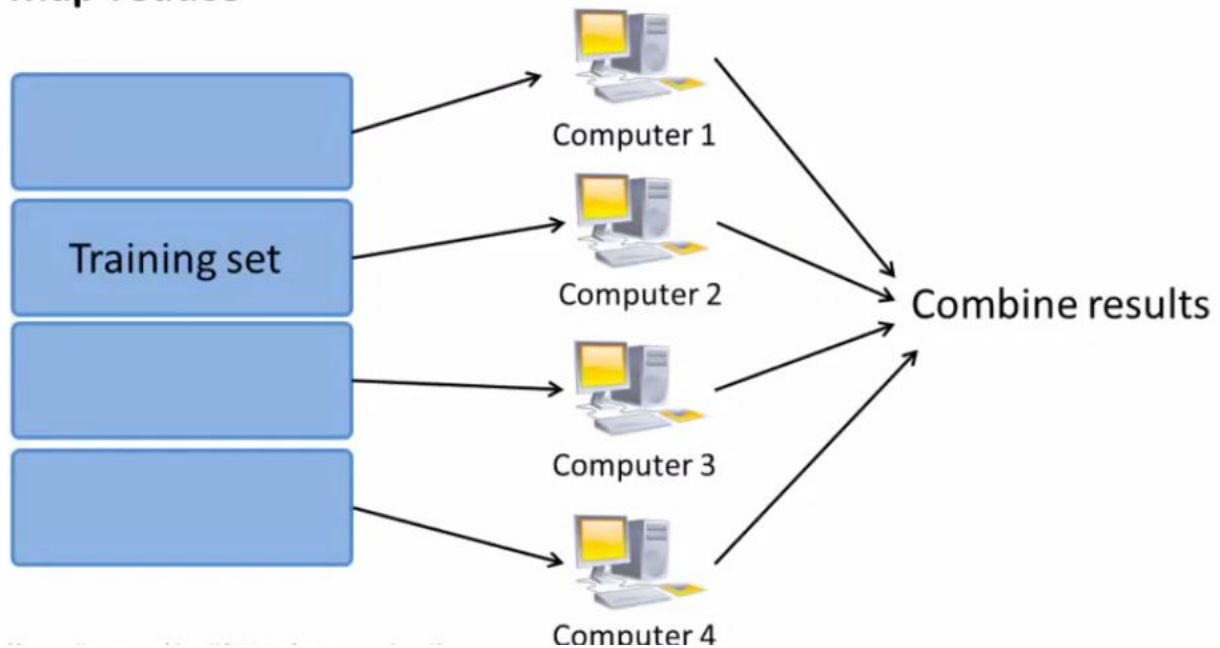
If our learning algorithm is such that the cost function and the gradient descent can be seen as a summation (like batch gradient descent) over the training set, the computation can be distributed over several machines or cores.

Note : Map Reduce works for linear regression, logistic regression as well as neural networks.



MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

## Map-reduce



Because of network latency and other overhead associated with map-reduce, if we run map-reduce using  $N$  computers, we might get less than an  $N$ -fold speedup compared to using 1 computer.

---

## XVII (I) -Problem Description and Pipeline

Photo OCR Pipeline involves :

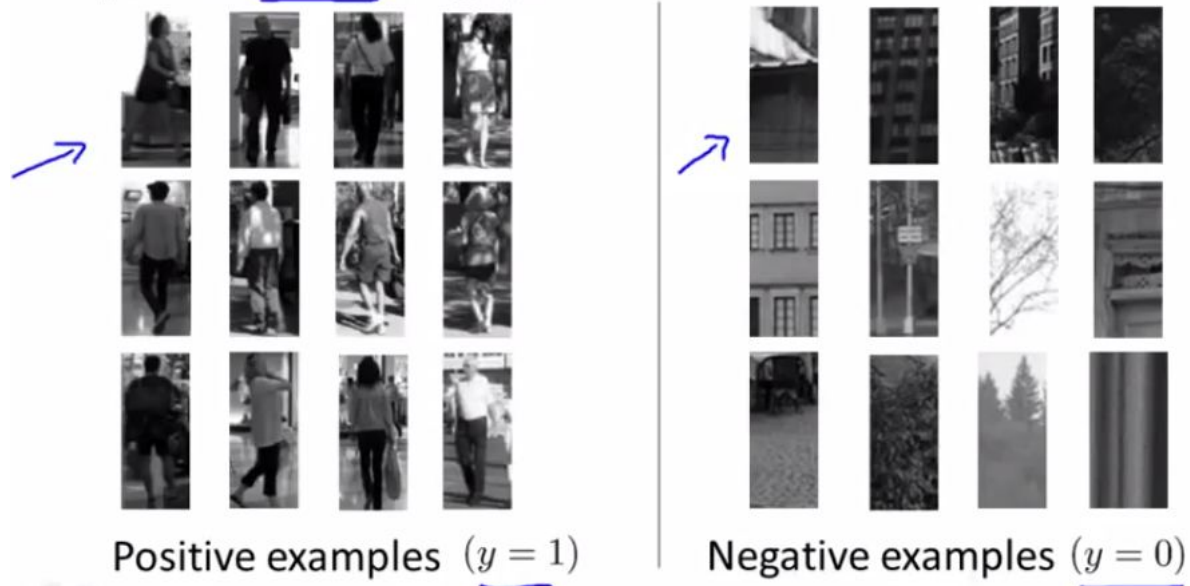
- Text Detection : finding areas in a digital image containing texts
- Character Segmentation : reading individual characters in text
- Character recognition : Recognizing what is written as a whole and take some decisions based on it.

## XVII (II) -Sliding Window

We choose an aspect-ratio (like 82 X 36) . Then we get a large training set and train our algorithm to classify negative and positive examples

### Supervised learning for pedestrian detection

$x$  = pixels in 82x36 image patches



Then we run the sliding window (of the same aspect ratio) in other image to identify objects (like pedestrian) in a new image. The amount by which we shift the sliding window is called the **step-size/stride-parameter**.

For text-detection, once we have identified text areas, we create a rectangle around those regions. The regions whose aspect ratio looks right (generally boxes are text should be more wider than tall) are kept, while other regions are rejected.



For character segmentation, we again use a supervised learning algorithm to identify if there is a split between two distinct characters at the middle. We again, use sliding window (running left to right) to find characters in the text region.



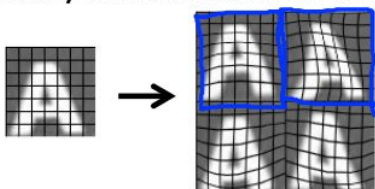
In the last step of character-recognition, we use a multi-class classification algorithm to identify the characters in the text region.

### XVII (III) -Artificial Data Synthesis

Artificial data synthesis should be done in the context of the problem.

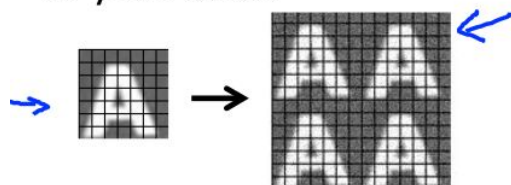
#### Synthesizing data by introducing distortions

- Distortion introduced should be representation of the type of noise/distortions in the test set.



➔ Audio:  
Background noise,  
bad cellphone connection

- Usually does not help to add purely random/meaningless noise to your data.



➔  $x_i = \text{intensity (brightness) of pixel } i$   
➔  $x_i \leftarrow x_i + \text{random noise}$

It is important to make sure that we have a low bias classifier before attempting to generate more training examples (by adding more features or adding more layers if we are using neural network)

### XVII (III) -What to Work on In a Pipeline(Ceiling Analysis)

We manually feed correct labels for one feature and observe overall improvement in the prediction (over some chosen metrics). This helps us in determining which part of the pipeline to focus on

Component	Accuracy
Overall system	72% ← ↓ 17%
→ Text detection	89% ← ↓ 1%
Character segmentation	<u>90%</u> ← ↓ 10%
Character recognition	100% ←