

## Week 9: Anomaly Detection & Recommender Systems

### Anomaly Detection: Problem Motivation

- Let's use an example to explain anomaly detection. Let's assume we have a couple of features for an aircraft engine and the manufacturer wants to know if a new engine coming off the assembly line is anomalous in any way. Plotting  $x_1$  and  $x_2$  we have the chart below. If we take the new engine  $x_{test}$  how do we know if it's anomalous? If it looks like ones we have seen before (i.e. sits somewhere close to where other engines have been) we probably can say it is OK. If it's outside where previous examples have been then we can call it an anomaly. Anomaly detection can be used in supervised and unsupervised problems.

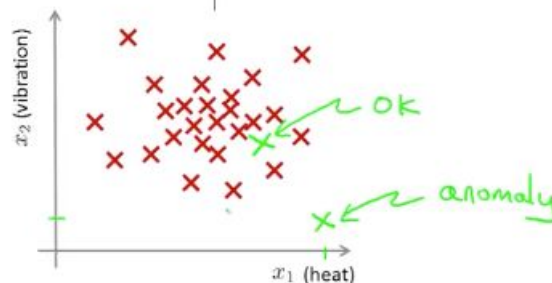
#### Anomaly detection example

Aircraft engine features:

- $x_1$  = heat generated
- $x_2$  = vibration intensity
- ...

Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

New engine:  $x_{test}$



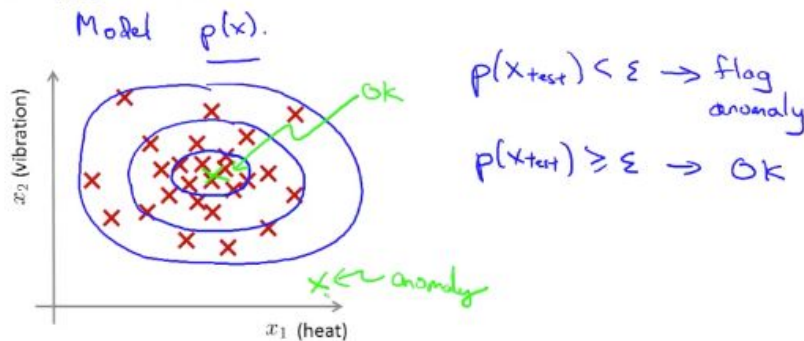
Andrew N

- More formally, given a dataset  $\{x_1, x_2, \dots, x_m\}$  we then ask the question: is  $x_{test}$  anomalous? To answer this question we create a model  $p(x)$ . If  $p(x_{test}) < \epsilon \rightarrow \text{flag anomaly}$ . If  $p(x_{test}) \geq \epsilon \rightarrow \text{OK}$ . Our model would hopefully be the circles in the middle and the examples in the middle are then ok. The example outside the circles is then an anomaly

## Density estimation

→ Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

→ Is  $x_{test}$  anomalous?



- Some applications of anomaly detection are:
  - fraud detection where my model  $p(x)$  can help to identify users who are behaving badly.
  - Manufacturing: the above aircraft engine example
  - Monitoring computers in a data center where the model  $p(x)$  can help us identify computers that are about to go down and should be checked by the sysadmin.

## Anomaly detection example

→ Fraud detection:

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

→ Model  $p(x)$  from data.

→ Identify unusual users by checking which have  $\underline{p(x) < \epsilon}$

→ Manufacturing

→ Monitoring computers in a data center.

→  $x^{(i)}$  = features of machine  $i$

$x_1$  = memory use,  $x_2$  = number of disk accesses/sec,

$x_3$  = CPU load,  $x_4$  = CPU load/network traffic.

...

$p(x) < \epsilon$

$x_1$   
 $x_2$   
 $x_3$   
 $x_4$        $p(x)$

## Anomaly Detection: Gaussian Distribution

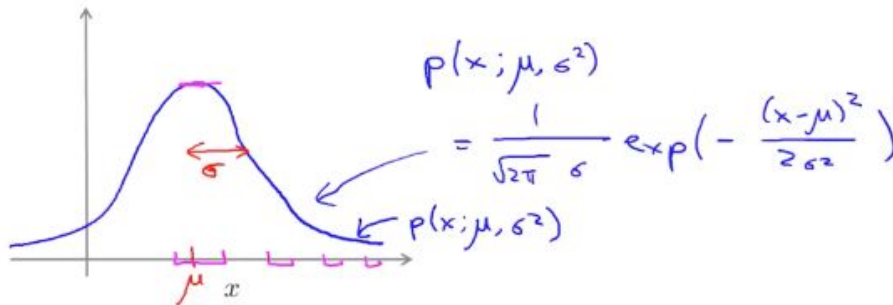
- The Gaussian distribution is also known as the normal distribution and is parametrized by the mean  $\mu$  and variance  $\sigma^2$ . We can write this as  $x \sim N(\mu, \sigma^2)$ . The chart below shows the well-known bell shaped form of the gaussian distribution. It tells us that the probability of  $x$  being one of the values in the center is high and it decreases as  $x$  moves to the sides. What we are charting is our model  $p(x; \mu, \sigma^2)$ . The formula for the distribution is shown below. Finally, sigma is the width of the curve as shown below.

## Gaussian (Normal) distribution

Say  $x \in \mathbb{R}$ . If  $x$  is a distributed Gaussian with mean  $\mu$ , variance  $\sigma^2$ .

$$x \sim \mathcal{N}(\mu, \sigma^2) \quad \sigma \text{ standard deviation}$$

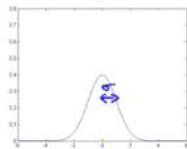
↑ "distributed as"



- Some examples of Gaussian distributions are shown below. In general, when  $\sigma$  is small the gaussian curve tends to be thinner and when  $\sigma$  is big the gaussian curve tends to be fatter as shown below. A good property to know is that the area under the gaussian curve must integrate to 1.

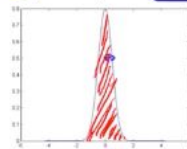
### Gaussian distribution example

→  $\mu = 0, \sigma = 1$

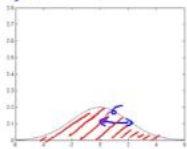


→  $\mu = 0, \sigma = 0.5$

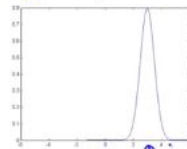
$$\sigma^2 = 0.25$$



→  $\mu = 0, \sigma = 2$



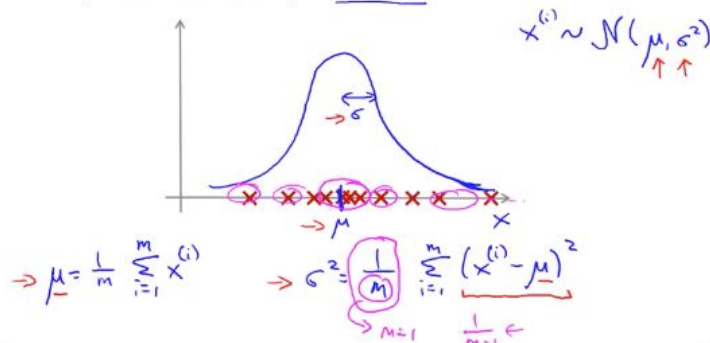
→  $\mu = 3, \sigma = 0.5$



- How do estimate the parameters  $\mu$  and  $\sigma$ ? Let's consider the example below. Given a dataset  $\{x_1, x_2, \dots, x_m\}$  we plot the  $x$ 's as shown.

### Parameter estimation

→ Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$   $x^{(i)} \in \mathbb{R}$



We then need to estimate the parameters with the formulas below

- $\mu = (1/m) \sum_{i=1}^m x^{(i)}$
- $\sigma^2 = (1/m) \sum_{i=1}^m (x^{(i)} - \mu)^2$

### Anomaly Detection: Algorithm

- We are going to use the gaussian distribution to create an anomaly detection algorithm.

Given a dataset  $\{x_1, x_2, \dots, x_m\}$ , our model

$p(x) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \dots * p(x_n; \mu_n, \sigma_n^2)$ . A more compact way of saying this is  $p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$

### Density estimation

→ Training set:  $\{x^{(1)}, \dots, x^{(m)}\}$

Each example is  $x \in \mathbb{R}^n$

$$\begin{aligned}
 p(x) &= p(x_1; \mu_1, \sigma_1^2) p(x_2; \mu_2, \sigma_2^2) p(x_3; \mu_3, \sigma_3^2) \dots p(x_n; \mu_n, \sigma_n^2) \leftarrow \\
 &= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)
 \end{aligned}$$

Handwritten notes on the right:

$$\begin{aligned}
 x_1 &\sim \mathcal{N}(\mu_1, \sigma_1^2) \\
 x_2 &\sim \mathcal{N}(\mu_2, \sigma_2^2) \\
 x_3 &\sim \mathcal{N}(\mu_3, \sigma_3^2)
 \end{aligned}$$

Handwritten notes at the bottom right:

$$\begin{aligned}
 \sum_{i=1}^n i &= 1+2+3+\dots+n \\
 \prod_{i=1}^n i &= 1 \times 2 \times 3 \times \dots \times n
 \end{aligned}$$

- Here is then our anomaly detection algorithm: choose features  $x_j$  that might be indicative of problematic examples. Fit the parameters using the formulas we know. Given a new example  $x$ , compute  $p(x)$  as shown below. If  $p(x) < \epsilon \rightarrow \text{flag as anomaly}$

### Anomaly detection algorithm

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

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

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

Handwritten notes for parameter fitting:

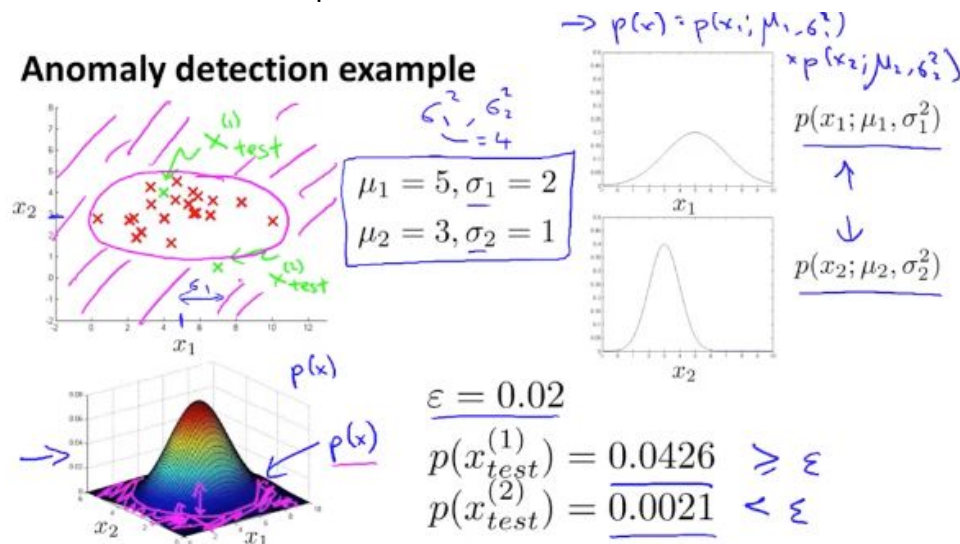
$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

→ 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  $p(x) < \epsilon$

- Here's an example of anomaly detection. Suppose we have the examples shown below. And we compute the corresponding  $\mu$ 's and  $\sigma$ 's. Since we only have two examples we can see their gaussian distributions on the right. Plotting that on 3d gives us the plot in the bottom left corner. We choose  $\varepsilon = 0.02$ . (later we'll find out how to choose epsilon). Then we get two new examples:  $x_{test}^{(1)}$  and  $x_{test}^{(2)}$ . Using our formulas we calculate  $p(x_{test}^{(1)})$  and  $p(x_{test}^{(2)})$  and we find out that  $p(x_{test}^{(1)}) \geq \varepsilon$  and  $p(x_{test}^{(2)}) < \varepsilon$  so the first test example is not an anomaly and the second one is. This is reflected in our 3d chart by showing that anything inside our 'cone' is not an anomaly and anything outside most likely is. The 2d version is shown on the top left corner.



Andrew Ng

## Developing and Evaluating an Anomaly Detection System

- We start by stressing the importance of having a way to test our learning algorithm.

### The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

- Assume we have some labeled data, of anomalous and non-anomalous examples. ( $y = 0$  if normal,  $y = 1$  if anomalous).
- Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal examples/not anomalous)
- Cross validation set:  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
- Test set:  $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$



For that we assume that we have some labeled data (where 1=anomalous and 0=normal). We then have a training set  $\{x_1, x_2, \dots, x_m\}$  with just normal examples, not anomalous ones. The cross validation set and the test set will contain anomalous examples.

- Let's see an example of how we construct our different datasets. Given 10,000 good engines and 20 flawed ones, we create
  - A training set of 6,000 good engines (where  $y=0$ )
  - A cross validation set of 2,000 good engines ( $y=0$ ) and 10 anomalous ones ( $y=1$ )
  - A test set of 2,000 good engines ( $y=0$ ) and 10 anomalous ones ( $y=1$ )

### Aircraft engines motivating example

- 10000 good (normal) engines
- 20 flawed engines (anomalous) 2-50  $y=1$
- Training set: 6000 good engines ( $y=0$ )  $p(x) = p(x_1, \mu_1, \sigma_1^2) \dots p(x_n, \mu_n, \sigma_n^2)$
- CV: 2000 good engines ( $y=0$ ), 10 anomalous ( $y=1$ )
- Test: 2000 good engines ( $y=0$ ), 10 anomalous ( $y=1$ )

Alternative:

- Training set: 6000 good engines
- CV: 4000 good engines ( $y=0$ ), 10 anomalous ( $y=1$ )
- Test: 4000 good engines ( $y=0$ ), 10 anomalous ( $y=1$ )

Some people repeat the same engines in the cross validation and test sets (as shown in the alternative example above) but that's not good machine learning practice and we will not do that

- The algorithm evaluation steps are then
  - Fit the model  $p(x)$  to the training set  $\{x_1, x_2, \dots, x_m\}$  using the Gaussian formulas we know (mean and variance)
  - Predict for the cross validation and test sets if  $y=1$  (anomalous) or  $y=0$  (normal)

### Algorithm evaluation

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

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

Possible evaluation metrics:

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

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

- Once we fit the model  $p(x)$  to the training set we can predict  $y$  for a cross validation/test set example  $x$  using  $p(x)$  and  $\epsilon$ . Since we have a highly skewed dataset accuracy is not a good performance evaluator for this algorithm. We are better off using precision/recall,  $F_1$  score or true positive/false positive/true negative/false negative.

### Algorithm evaluation

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

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

Possible evaluation metrics:

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

Can also use cross validation set to choose parameter  $\epsilon$

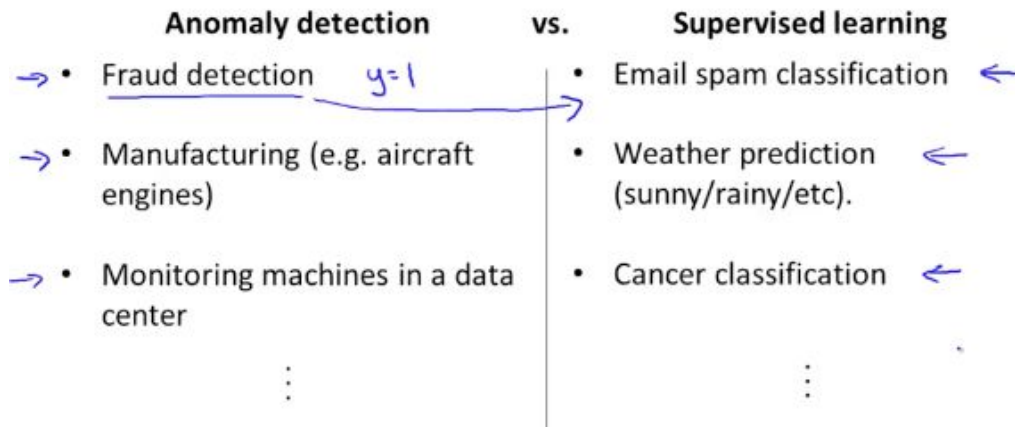
One way to choose epsilon is to try different values of epsilon on the cross validation set and select the one that maximizes the  $F_1$  score or otherwise does well with the cross validation set. We then do the final evaluation of the algorithm in the test set.

### Anomaly Detection vs Supervised Learning

- How do we know when to use anomaly detection and when to use a supervised learning algorithm? There are two guidelines to follow
  - If we have a small number of positive examples (0-20) and a large number of negative examples we should use anomaly detection. If we have a large number of both positive and negative examples use supervised learning.
  - If our positive examples have many different types of anomalies any algorithm will find it hard to learn what anomalies look like, therefore we use anomaly detection. If we have enough positive examples we use supervised learning.

Anomaly detection	vs.	Supervised learning
→ Very small number of positive examples ( $y = 1$ ). (0-20 is common).		Large number of positive and negative examples. ←
→ Large number of negative ( $y = 0$ ) examples. $p(x)$ ←		
→ Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like;		Enough positive examples for algorithm to get a sense of what positive examples are like, future ←
→ future anomalies may look nothing like any of the anomalous examples we've seen so far.		positive examples likely to be similar to ones in training set. ←

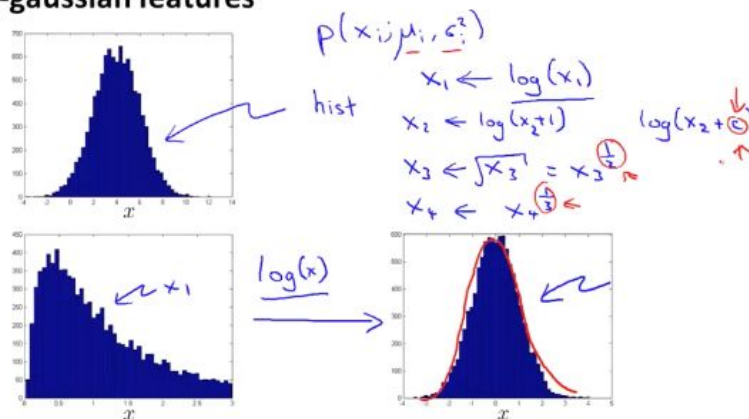
- Some of the applications for each type of algorithm are shown below. For anomaly detection some examples are: Fraud Detection (if you are a major website with a lot of examples of anomalous behavior fraud detection can shift to use supervised learning), Manufacturing and monitoring machines in a data center. Applications for supervised learning are email spam classification, weather prediction and cancer classification.



### Choosing what features to use

- One of the things to do once having the data is to plot it. If it looks gaussian like on the top left we are fine and can proceed. If it doesn't look gaussian (like the bottom left example below) there are some things we can do:
  - We can apply the log function to  $x$  and  $\log(x)$  will then look gaussian.
  - We can add a constant  $c$
  - We can do the square root of a feature or we can do another power

### Non-gaussian features



We use the octave command hist to play with the data to see if applying one of our various choices (log, square root, etc.) will make the distribution more gaussian.

- Next is how do we come up with new features for our algorithm? One way is to use an error analysis process similar to the one we use for supervised learning. A very common problem is that  $p(x)$  is comparable for normal and anomalous examples so we cannot



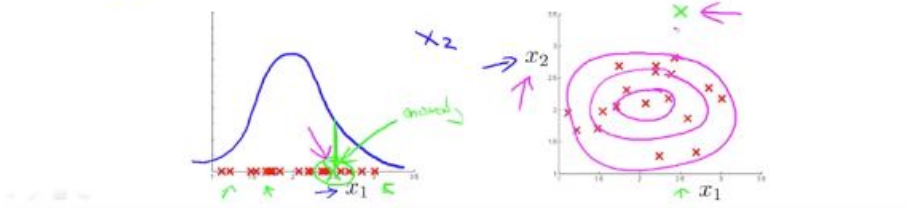
tell them apart. In the example below (left chart) we have an anomalous example buried in between of normal examples. The recommendation is to take a look at that anomalous example and see if it can help us identify a new feature, say  $x_2$ , that will give us more clarity. Assume we got new feature  $x_2$  and re-plotting the data gives us the second chart. In that chart we can expect my anomaly detection algorithm to give high probability to the center regions and low probability to the outer regions. Our new plot then helps us to distinguish anomalies from good examples.

### → Error analysis for anomaly detection

Want  $p(x)$  large for normal examples  $x$ .  
 $p(x)$  small for anomalous examples  $x$ .

Most common problem:

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



- Here's an example of how the professor will try to come up with new features. Let's take the example of monitoring computers in the data center. We have the features below and let's say I think the normal features show a linear relationship between CPU and network traffic. I can then have a suspicion of a 'bad'/anomalous server if we have high CPU but low(er) network traffic. To detect this type of anomaly I create a new feature,  $x_5 = \text{CPU load} / \text{network traffic}$ . I can also get creative and create  $x_6 = (\text{CPU load})^2 / \text{network traffic}$  which would be another feature that help us to catch the high CPU/no network traffic anomaly.

### → Monitoring computers in a data center

→ 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 ←

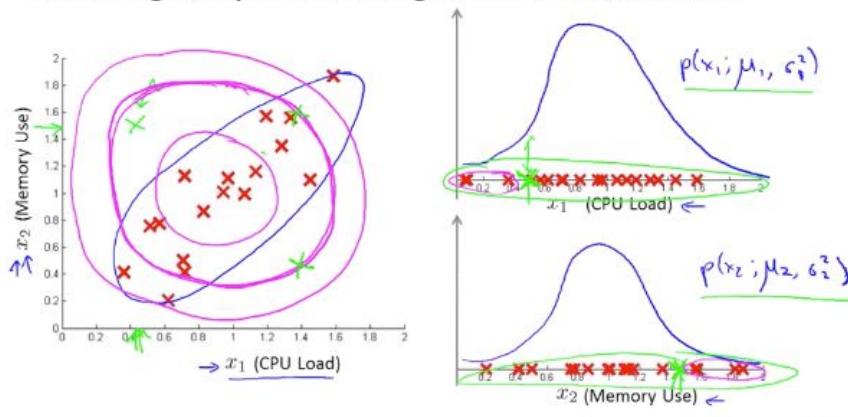
$$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$$

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

## Multivariate Gaussian Distribution (Optional)

- We are going to talk about a possible extension to our anomaly detection algorithm called multivariate gaussian distribution. First we will talk about the motivation for such an extension: It can catch some anomalies than the previous version could not catch. Let's illustrate that with our data center machines example. We have two features:  $x_1 = \text{CPU load}$  and  $x_2 = \text{Memory use}$ . We plot the gaussian for each one and we have the gaussian charts on the right. Now let's say we plot both of our features and we end up with the right plot. In that plot we see most of the examples are inside the blue shape. Now let's assume there is an example outside: the green example on the top left with  $x_1 \approx 0.4$  and  $x_2 \approx 1.5$ . If we plot this example in each gaussian chart we see they kind of are bunched up with other examples so regular anomaly detection will not flag them as anomalies. What our regular anomaly detection algorithm is not doing is not realizing that as we draw more circles the probability decreases

### Motivating example: Monitoring machines in a data center



- To fix this we are going to develop a modified version of anomaly detection: multivariate gaussian distribution.

### Multivariate Gaussian (Normal) distribution

→  $x \in \mathbb{R}^n$ . Don't model  $p(x_1), p(x_2), \dots$ , etc. separately.

Model  $p(x)$  all in one go.

Parameters:  $\mu \in \mathbb{R}^n$ ,  $\Sigma \in \mathbb{R}^{n \times n}$  (covariance matrix)

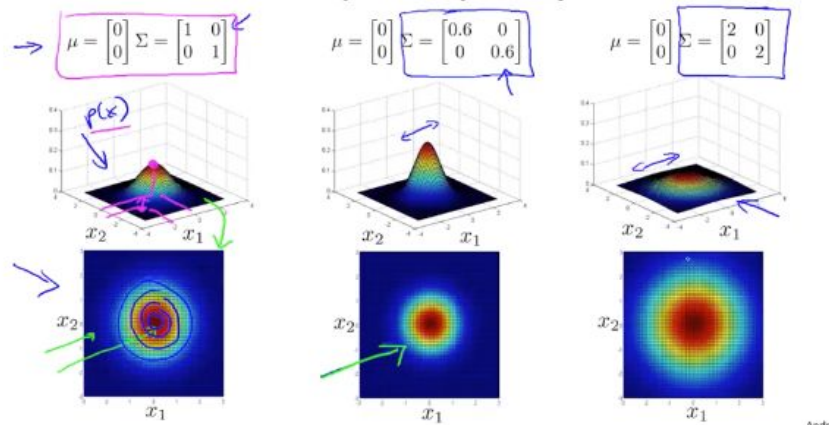
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

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

In this version we don't model the  $\mu$ 's separately, we model all of them in one go. We have two parameters:  $\mu$  and  $\Sigma$  (covariance matrix). The formula is shown above.

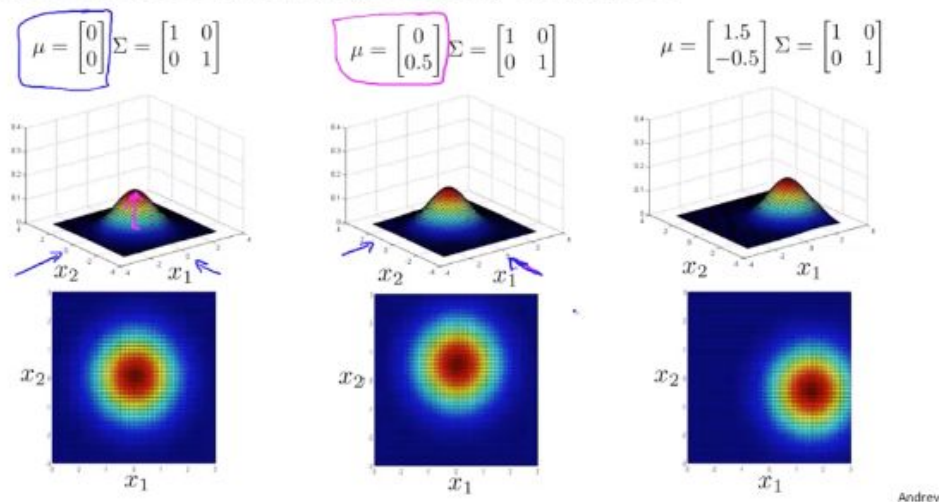
- Now let's look at some examples of how multivariate gaussian distributions look. These examples have  $\mu$  on (0,0) (red color) so that's the place where the maximum probability is. This probability decreases as you move away from the center. If we vary  $\Sigma$  we see the following: if it's smaller the 'cone' is narrower but taller. If it's smaller the 'cone' is wider but smaller.

### Multivariate Gaussian (Normal) examples



- If we vary  $\mu$  then the center of the distribution moves from (0,0) to the new values as shown in a couple of examples below. The center of the distribution is the point where the height of the 'cone' is the highest. In summary, varying  $\mu$  shifts around the center of the distribution.

### Multivariate Gaussian (Normal) examples



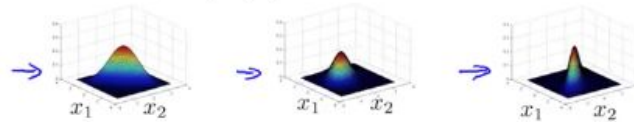
- To recap, a multivariate gaussian distribution has parameters  $\mu$  and  $\Sigma$  and the formula is shown below. How do we estimate these parameters? We use the formulas below and apply them using the training set  $\{x_1, x_2, \dots, x_m\}$

### Multivariate Gaussian (Normal) distribution

Parameters  $\mu, \Sigma$

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

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



Parameter fitting:

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

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

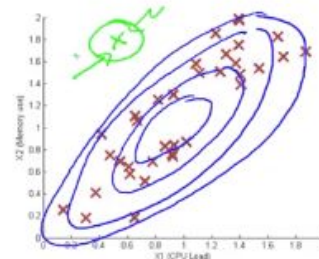
$$\rightarrow \boxed{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad \rightarrow \boxed{\Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

- How do we do anomaly detection using a multivariate gaussian distribution? We start by fitting  $p(x)$  using the known formulas for  $\mu$  and  $\Sigma$ . Then, when given a new example  $x$ , we compute  $p(x)$  using the formula below. If  $p(x) < \epsilon$  then it's an anomaly. We end up with a gaussian probability that places high probability to the circle in the middle of the examples in the chart below and the probability decreases as you move away from the first circle. This algorithm will correctly identify the new  $x$  (the green  $x$  below) as an anomaly

### Anomaly detection with the multivariate Gaussian

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

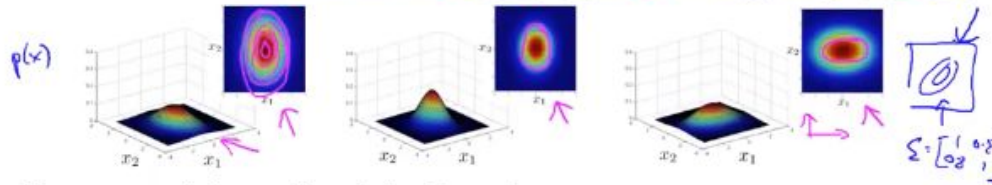
$$\left[ p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right) \right.$$

Flag an anomaly if  $p(x) < \epsilon$

- What is the relationship between the original model and multivariate gaussian distribution? Turns out the original model is a multivariate gaussian distribution with one constraint. The constraint is that the contours of the gaussian distribution are axis aligned. It also means that the covariance matrix has values  $\{\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2\}$  on its diagonal and 0's everywhere else.

## Relationship to original model

Original model:  $p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \dots \times p(x_n; \mu_n, \sigma_n^2)$



Corresponds to multivariate Gaussian

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

where

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \dots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \dots & \sigma_n^2 \end{bmatrix}$$

- When would you use each model? The original model is used more often but it requires you to manually create features to capture anomalies that are combinations of values while multivariate gaussian automatically captures these correlations. Original model is computationally cheaper and scales better to a large  $n$  while multivariate gaussian is computationally more expensive. Finally, we can use the original model even if  $m$  (the training set size) is small. Multivariate gaussian must have  $m > n$ , if not  $\Sigma$  is not-invertible.  $\Sigma$  is also non-invertible if we have redundant features (i.e.  $x_1 = x_2$  or  $x_3 = x_4 + x_5$ ). In practice, it is hard we find this case but it is good to be aware of it.

### → Original model

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

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  $n$ )  $n = 10,000, m = 100,000$

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

$$\Sigma \in \mathbb{R}^{n \times n} \quad \Sigma^{-1}$$

Computationally more expensive

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

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

Andrew

## Recommender Systems: Problem Formulation

- We will talk about Recommender Systems for two reasons:



- Because it is an important application of machine learning that most big websites (netflix, amazon, etc) want to improve in performance and it has a direct impact on their bottom line.
  - Because there are some algorithms that can learn new features for us instead of us trying to design them manually. Recommendation systems happen to be one of them.
- Let's use an example of predicting movie ratings to illustrate the problem formulation. Given 5 movies and 4 users and some ratings we have the following:
  - $n_u$  = number of users
  - $n_m$  = number of movies
  - $r(i, j) = 1$  if user  $j$  has rated movie  $i$ , otherwise it is  $= 0$
  - $y^{(i,j)}$  = rating given to movie  $i$  by user  $j$  (only defined if  $r(i, j) = 1$ )


**Example: Predicting movie ratings**

→ User rates movies using one to five stars  
 zero

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$n_u = 4$        $n_m = 5$

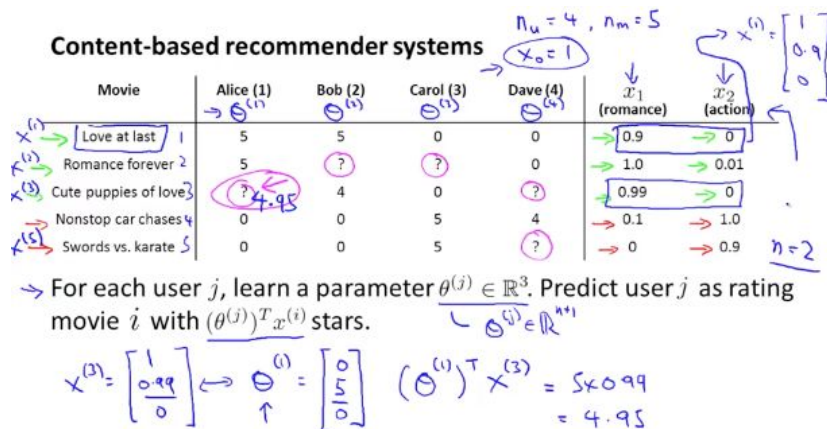
→  $n_u$  = no. users  
 →  $n_m$  = no. movies  
 →  $r(i, j) = 1$  if user  $j$  has rated movie  $i$   
 →  $y^{(i,j)}$  = rating given by user  $j$  to movie  $i$  (defined only if  $r(i, j) = 1$ )  
 →  $0, \dots, 5$



The recommender system problem is to look at the data, look for the ratings that are missing and try to predict what would be the values for the question marks above. Our job here is to come up with a learning algorithm that can automatically fill in the missing ratings and look at the movies the user has not watched yet and recommend to him/her new movies to watch.

### Recommender Systems: Content-based Recommendations

- How do we then predict the missing values? Consider the dataset below.



In this set we have 5 movies, 4 users and a few missing ratings. We also have two features,  $x_1$  and  $x_2$ , that measure the degree of romance and action respectively. We can then have a vector for each movie. For example, for 'love at last' our vector would be  $x^{(1)} = [1 \ 0.9 \ 0]$ . We can do the same for all the other movies. How then do we predict a missing value? For each user  $j$ , we learn a  $\theta$  parameter and predict user  $j$  of rating movie  $i$  as  $(\theta^{(j)})^T * x^{(i)}$ . Let's take the example of trying to predict Alice's rating ( $\theta^{(1)}$ ) for the movie cute puppies of love ( $x^{(3)}$ ). We will look later at how do we get the thetas, for now let's assume  $\theta^{(1)} = [0 \ 5 \ 0]^T$ . Our product is then  $([0 \ 5 \ 0]^T * ([0 \ 0.99 \ 0]))$  and our result is 4.95 which, considering Alice's previous results (she likes romance movies and not action movies so it is likely she will give a high rating to another romantic movie), seems like a good rating.

- More formally, our problem formulation is as follows: given  $r(i,j)$ ,  $y^{(i,j)}$ ,  $\theta^{(j)}$ ,  $x^{(i)}$ , we know that for user  $j$ , movie  $i$  the predicted rating is  $(\theta^{(j)})^T * (x^{(i)})$ . Finally,  $m^{(j)} = \text{number of movies user } j \text{ has rated}$ . We have everything except  $\theta^{(j)}$  so to learn it we apply the following formula:  $\sum_{i:r(i,j)=1} (\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} )^2$ . So this is like the linear regression formula and we can add the regularization term and also  $1/(2*m)$ . To simplify the math we get rid of the constant  $m^{(j)}$  and it will not change the value of theta.

### Problem formulation

- $r(i, j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)
- $y^{(i,j)}$  = rating by user  $j$  on movie  $i$  (if defined)

→  $\theta^{(j)}$  = parameter vector for user  $j$

→  $x^{(i)}$  = feature vector for movie  $i$

→ For user  $j$ , movie  $i$ , predicted rating:  $\underline{(\theta^{(j)})^T x^{(i)}}$

$$\theta^{(j)} \in \mathbb{R}^{n+1}$$

→  $m^{(j)}$  = no. of movies rated by user  $j$

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

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

- My optimization objective is then to learn  $\theta^{(j)}$  for user  $j$  using the formula we just learned which is also shown below. To learn the thetas for all users we use the 2nd formula below. The difference between the 2 formulas is the extra summation that iterates over  $n_u$  which is number of users. Our formula looks similar to logistic regression as it has both the square error term and the regularization term.

### Optimization objective:

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

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

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

- Assembling everything we then have a variation of linear regression called content-based recommendation (called like that because we assume we have features for the different movies). The optimization algorithm is shown below and the gradient descent update is also shown below.

## Optimization algorithm:

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

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

## Gradient descent update:

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

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

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

## Recommender Systems: Collaborative Filtering

- We now will talk about a different approach to recommender systems: Collaborative Filtering. Our previous recommender algorithm assumed we always had features for all the different movies; this one doesn't. In this case we don't have any info for  $x_1$  and  $x_2$  but we assume we have a way of asking our users what they like and therefore can learn the  $\theta_S$  for each user. For example, Alice (user 1) likes romance so  $\theta^{(1)} = [0 \ 0 \ 5]$ , Bob is similar, Carol and Dave are the opposite: they like action and not romance so their thetas are:  $[0 \ 0 \ 5]$ .

## Problem motivation

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	$x_1$ (romance)	$x_2$ (action)
$x^{(1)}$ Love at first sight	5	5	0	0	1.0	0.0
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

$x^{(1)} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix}$   
 $\theta^{(1)} x^{(1)} \approx 5$   
 $\theta^{(2)} x^{(1)} \approx 5$   
 $\theta^{(3)} x^{(1)} \approx 0$   
 $\theta^{(4)} x^{(1)} \approx 0$

With this information we can then predict  $x_1$  and  $x_2$  for the first movie (assume we don't know the movie's title and therefore do not know it is a romance). All we know Alice and Bob (the romance people) liked the movie and Carol and Dave (the action people) hated

it so we can then predict  $x^{(1)} = \text{romantic movie}$  and therefore  $x_1 = 1$  and  $x_2 = 0$ . Our vector is then  $x^{(1)} = [1 \ 1 \ 0]$ . What we are asking here is what feature vector  $x^{(1)}$  should be so that  $(\theta^{(1)})^T * x^{(1)} \approx 5$  and  $(\theta^{(2)})^T * x^{(1)} \approx 5$  and  $(\theta^{(3)})^T * x^{(1)} \approx 0$  and  $(\theta^{(4)})^T * x^{(1)} \approx 0$  and  $x^{(1)} = [1 \ 1 \ 0]$  satisfies this equation.

- Now let's formalize the process of learning the features  $x^{(i)}$ : Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$  we try to learn  $x^{(i)}$  using the formulas below.

### Optimization algorithm

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  $x^{(1)}, \dots, x^{(n_m)}$ :

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

- Finally, putting everything together we have the algorithm from the previous video where given  $\{x^{(1)}, x^{(2)}, \dots, x^{(n_u)}\}$  and movie ratings we can estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$  and we have the algorithm we just saw: given  $\theta^{(1)}, \dots, \theta^{(n_u)}$  we can estimate  $\{x^{(1)}, x^{(2)}, \dots, x^{(n_u)}\}$ . Now we can go back and forth between the two methods: we can initially guess some of thetas and then we use the algorithm we just learned and we get some features. Now that we have some features we go back to the first method and get a better estimate for our thetas. So we can iterate our way and go back and forth between the 2 algorithms and we will converge at reasonable estimates for both our features and our thetas.

### Collaborative filtering

Given  $x^{(1)}, \dots, x^{(n_m)}$  (and movie ratings),  
can estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$

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

Guess  $\Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \dots$

$\sigma^{(i,j)}$   
 $y^{(i,j)}$



This is a basic collaborative filtering algorithm but we are going to use something a bit more complicated. It is called collaborative because the users giving us ratings are helping us to get better estimates for the missing values.

### Recommender Systems: Collaborative Filtering Algorithm

- We had two algorithms before and we said how we could go back and forth between them. Now we are going to have an optimization objective that minimizes both the features and the thetas. This algorithm is shown below, in this version we minimize  $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$  with the equation below. In this version we don't have  $x_0$

**Collaborative filtering optimization objective**

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

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

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

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

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

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

→  $\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$

Andrew Ng

- Our algorithm then works as follows:
  - Initialize features and thetas to small random values. This serves as symmetry breaking and ensures the algorithm learns features  $x^{(1)}, \dots, x^{(n_u)}$  that are different from each other.
  - Minimize cost function  $J$  using gradient descent (or another advanced optimization algorithm)
  - For a user with parameters  $\theta$  and a movie with learned features  $x$  we predict the rating by doing  $(\theta^{(j)})^T * (x^{(i)})$

### Collaborative filtering algorithm

- 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  $j = 1, \dots, n_u, i = 1, \dots, n_m$ :

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

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

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

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

Andrew Ng

### Low Rank Matrix Factorization: Vectorization

- Now we talk about other applications for our collaborative filtering algorithm like given one product, can we recommend other products. We are going to see another way of writing out the predictions of our algorithm

### Collaborative filtering

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

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

What we do initially is we take the ratings and the movies we have and we create a matrix of ratings  $Y$  as shown above.

- Given our rating matrix  $Y$  we can substitute each of its values for the product of the corresponding thetas and features. For example, for  $Y(0, 0)$  this value would be  $(\theta^{(1)})^T x^{(1)}$  and so on for each value in the matrix.

**Collaborative filtering**

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

Predicted ratings:  $(i, j)$

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

$X = \begin{bmatrix} -(x^{(1)})^T \\ -(x^{(2)})^T \\ \vdots \\ -(x^{(n_m)})^T \end{bmatrix}$

$\Theta = \begin{bmatrix} -(\theta^{(1)})^T \\ -(\theta^{(2)})^T \\ \vdots \\ -(\theta^{(n_u)})^T \end{bmatrix}$

→ Low rank matrix factorization

Andrew

There is a vectorized version of this product which takes a matrix  $X$  of all the features and a matrix  $\Theta$  of all the transposes of all the thetas and our product is then just  $X * \Theta^T$ . Our algorithm is also known as low rank matrix factorization.

- Here's something else we can do with our collaborative filtering algorithm: given a movie  $i$  can I find  $j$  movies related to movie  $i$ ? How do we do that? Each movie has a vector  $x$  so this gives us a very convenient way to measure if the movies are similar. We can compute the difference between the vectors for two movies and if that difference is small then we know the movies are somewhat related. Formalizing it we can then say that  $small \|x^{(i)} - x^{(j)}\|$  gives us an indication of relation. So we can do things like find 5 most similar movies to movie  $i$  by finding 5 movies  $j$  with the smallest  $\|x^{(i)} - x^{(j)}\|$

### Finding related movies

For each product  $i$ , we learn a feature vector  $x^{(i)} \in \mathbb{R}^n$ .

→  $x_1 = \text{romance}, x_2 = \text{action}, x_3 = \text{comedy}, x_4 = \dots$

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

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

5 most similar movies to movie  $i$ :

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

### Low Rank Matrix Factorization: Mean Normalization

- Mean Normalization can make our collaborative filtering algorithm work better. Here's an example where it will be needed. We have our previous users but we have added a new one, Eve, who has not rated any movies. So what will we end up having for Eve in terms of predicted ratings? Applying our equation we find the first term is going to be just  $\theta$  since  $y$  is always zero (since Eve has not rated any movie) and we are then left

with the last term in the equation. This regularization term encourages us to go to zero and we will end up with a theta vector of zeros. Using that theta vector times any x will give us zeros for all of Eve's ratings. This is not very useful since we will not be able to recommend any movies to her.

**Users who have not rated any movies**

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

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

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

$n=2$   
 $\theta^{(5)} \in \mathbb{R}^2$   
 $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$   
 $(\theta^{(5)})^T x^{(i)} = 0$

$\frac{\lambda}{2} [(\theta_1^{(5)})^2 + (\theta_2^{(5)})^2] \leftarrow$

Andrew 1

- Mean Normalization can fix this problem. We start by moving all our ratings (including Eve's as question marks) to our Y matrix. We then compute a vector  $\mu$  which has the mean rating for every movie and we subtract it from matrix Y. The resulting matrix Y is the input into my collaborative filtering algorithm. Our prediction equation changes a bit though, because we need to add back the mean so the new equation is  $(\theta^{(j)})^T * (x^{(i)}) + \mu_i$ . For Eve then our first term would be zero because the theta vector would still be zeros so we end up with just the mean vector and that becomes the ratings given to Eve.

#### Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \rightarrow \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

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

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

$$\downarrow$$

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

User 5 (Eve):

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

$$\underbrace{(\theta^{(5)})^T (x^{(i)})}_0 + \mu_i$$

## Week 10: Large Scale Machine Learning

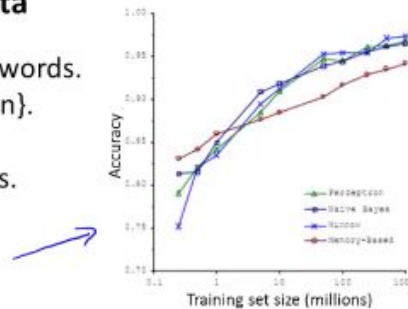
## Gradient Descent with Large Datasets: Learning with large datasets

- We want to learn from large datasets. We saw earlier how an algorithm that classified confusable words would perform better if it had more data to use. This and other instances led to the saying below, indicating the importance of data.

### Machine learning and data

Classify between confusable words.  
E.g., {to, two, too}, {then, than}.

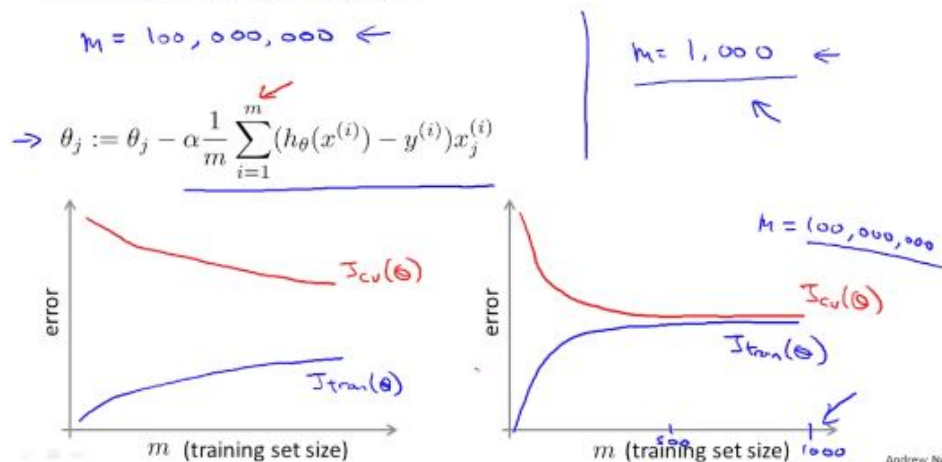
For breakfast I ate two eggs.



→ “It’s not who has the best algorithm that wins.  
It’s who has the most data.”

- However, using large datasets bring their own challenges. Assume we have a training set of 100 M records (i.e. some part of the US census) and you want to perform gradient descent. Every gradient descent step will need to sum over a 100 million records, which will be computationally expensive. One way to check if we need to do this is use a smaller  $m$  and then plot the learning curves to check if more data would make a difference.

### Learning with large datasets



In the left chart above we have a high variance algorithm which tells us getting more data would help the algorithm. On the right side we have the contrary: an algorithm with high bias which wouldn’t benefit from adding more data. In that case the right thing to do is to add more features. Finally, as we saw, gradient descent is no computationally



effective. Next we are going to see a couple of new methods that are better in this dimension

### Gradient Descent with Large Datasets: Stochastic Gradient Descent

- We are going to talk about Stochastic Gradient Descent, a modification to gradient descent that allows us to scale learning algorithms to large datasets. Let's suppose we are using linear regression with gradient descent. These equations are shown below. Graphing the cost function  $J(\theta)$  would give us the bow shaped function below.

#### Linear regression with gradient descent

$$\rightarrow h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

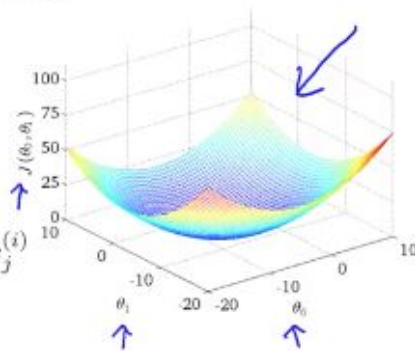
$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



- Just to refresh our memory, gradient descent starts from where the parameters are initialized and works its way to the global minimum as shown below. The problem with this version of gradient descent, known as batch gradient descent, is that for each step you have to use your whole training set. If that training set is say, 300 million records each step is going to be slow and inefficient.

#### Linear regression with gradient descent

$$h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

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

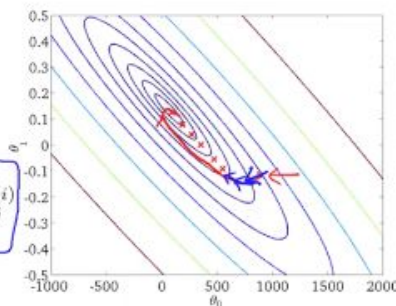
Repeat {

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

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

}

$M = 300,000,000$   
Batch gradient descent



By the way, we call this method batch gradient descent because each step requires us to use all the records in the training set.

- Now we are going to see some methods that do not require us to use the whole training but one training example at a time. On the left we have the cost function and gradient

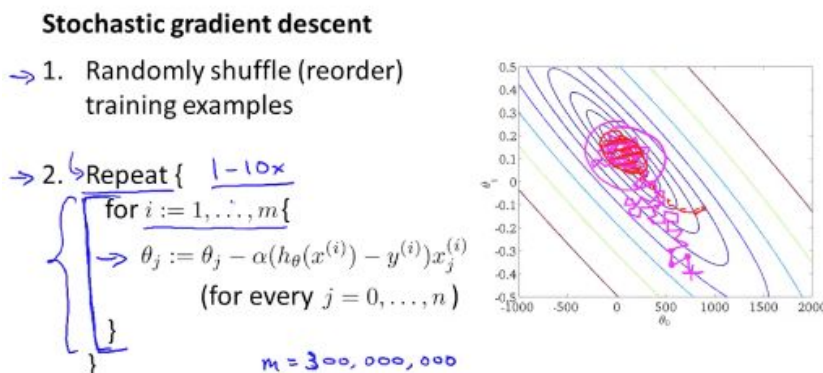
descent equation for the batch case. On the right we have the same for stochastic gradient descent. Notice how the cost function is written as the cost of the parameter  $\theta$  with respect to training example  $x^{(i)}, y^{(i)}$ . Stochastic gradient descent then does the following steps:

- Randomly shuffle the dataset.
- Repeat for  $i = 1, \dots, m$ :  $\theta_j = \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)})^2$

Batch gradient descent	Stochastic gradient descent
$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$	$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$
Repeat { $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ <div style="text-align: center;"> <math>\frac{\partial}{\partial \theta_j} J_{train}(\theta)</math>            (for every <math>j = 0, \dots, n</math>)         </div> }	$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\theta, (x^{(i)}, y^{(i)}))$
	1. Randomly shuffle dataset. ← 2. Repeat { for $i = 1, \dots, m$ { $\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ <div style="text-align: center;"> <math>\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))</math> </div> }           }
	$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots$

Basically what we are doing is we take one training example and make a small modification to the parameter  $\theta_j$ , then take the 2nd training example and do another small modification to  $\theta_j$  until it goes through the whole dataset. Instead of looping through the whole dataset in each gradient descent iteration, stochastic gradient descent takes each training example and makes small modification to the parameters.

- Now let's look at how stochastic gradient descent operates: Whereas batch gradient descent would take a somewhat direct route to the global minimum and converge there stochastic gradient descent would, in general, go towards the global minimum but not all the time.



Additionally, stochastic gradient descent does not converge to the global minimum, it kind of wanders around the global minimum. This is not a problem. Finally, it is common to repeat the loop 1-10 times, that is, taking 1-10 passes through your dataset

## Gradient Descent with Large Datasets: Mini Batch Gradient Descent

- We are going to see another variation of gradient descent: Mini Batch Gradient Descent. In Batch Gradient Descent we use all  $m$  examples in each iteration, in stochastic gradient descent we use 1 example each iteration. In mini batch gradient descent we use  $b$  examples each iteration where  $b = 2 - 100$ . Let's look at an example, assume  $b = 10$  so we have 10 training examples:  $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$ . Our gradient descent update is then:  $\theta_j = \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) * x_j^{(k)}$  and we repeat it.

### Mini-batch gradient descent

→ Batch gradient descent: Use all  $m$  examples in each iteration

→ Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use  $b$  examples in each iteration

$b = \text{mini-batch size. } b = 10. \quad 2 - 100$

Get  $b = 10$  examples  $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

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

$i := i + 10$

- Writing out our mini batch gradient descent update a bit more formally for an example of  $b=10$  and  $m=1000$  we have the equation below. Mini batch gradient descent is faster than batch gradient descent because we only need  $b$  examples on each iteration instead of the whole data set.

### Mini-batch gradient descent

Say  $b = 10, m = 1000$ . →  $b$  examples  
Repeat { → 1 example  
Vectorization  
→ 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  $j = 0, \dots, n$ )  
}  
}  
 $m = 300,000,000$   $b = 10$

It is likely to outperform stochastic if we have a good vectorized implementation which allows us to parallelize our computation; there is not much to parallelize with the 1 example used in stochastic gradient descent.

## Gradient Descent with Large Datasets: Stochastic Gradient Descent Convergence

- In this session we'll talk about how to make sure Stochastic Gradient Descent is converging and for picking the learning rate  $\alpha$ . In batch gradient descent we would plot  $J_{train}(\theta)$  as a function of the number of iterations to make sure gradient descent was converging which was fine with small training sets. However, for stochastic gradient descent we don't want to pause it periodically since the whole point is to start making progress faster (using just 1 training example) than with batch gradient descent.

### Checking for convergence

→ Batch gradient descent:

→ Plot  $J_{train}(\theta)$  as a function of the number of iterations of gradient descent.

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad M = 300,000,000$$

→ Stochastic gradient descent:

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad \rightarrow (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots$$

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

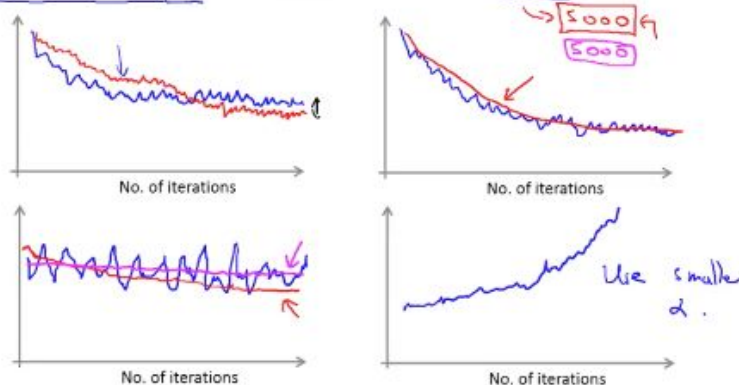
→ Every 1000 iterations (say), plot  $cost(\theta, (x^{(i)}, y^{(i)}))$  averaged over the last 1000 examples processed by algorithm.

So we do this instead: during learning we compute the cost before updating  $\theta$  and every say, 1000 iterations we plot the  $cost(\theta, x^{(i)}, y^{(i)})$  to see how well the algorithm is doing and the plot will let us know if the algorithm is converging.

- Below are some examples of how plotting the cost function would look like.

### Checking for convergence

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



Supposed we are showing the cost averaged over 1,000 examples. You might get a plot like the first one where it seems the algorithm is converging. You can use a bigger average size, say 5,000, to make sure that is the case. With a bigger size you will get a smoother curve but you have to wait longer since now you have 5,000 examples and not

1,000. The 3rd charts show a plot that initially leads us to think it is not decreasing (blue line) but it seems it is (red line) once we use a bigger sample size. The 4th plot shows us an example of the algorithm not converging and the recommendation here is to use a smaller  $\alpha$

- Finally, we are going to examine the learning rate a bit more. We remember stochastic gradient descent doesn't really converge to a minimum but rather sticks around the minimum and takes a somewhat of a meandering route to it. We usually keep the learning rate  $\alpha$  constant but there is a way to slowly decrease it over time if want  $\theta$  to converge. We set  $\alpha = \text{constant1}/(\text{iterationNumber} + \text{constant2})$ . The advantage of doing this is that stochastic gradient descent will take smaller steps and the meandering is smaller and you are, therefore, closer to the global minimum. The disadvantage is you have to play with the 2 new parameters (*constant1 and constant2*) to find a good value for each. People do this in practice but, in general, we are satisfied with just keeping  $\alpha$  constant and having stochastic gradient descent 'hang around' the global minimum.

### Stochastic gradient descent

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

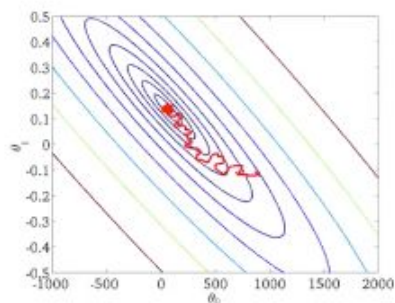
$$J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^m \text{cost}(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {
 

for  $i := 1, \dots, m$  {
 

$\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$ 

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



Learning rate  $\alpha$  is typically held constant. Can slowly decrease  $\alpha$  over time if we want  $\theta$  to converge. (E.g.  $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$ )  $\alpha \rightarrow 0$

Andrew Ng

### Advanced Topics: Online Learning

- We are going to talk about a new machine learning setting: online learning. It allows us to model problems where we have a continuous flood of data coming in and we want an algorithm to learn from that data.
- Assume we have a shipping service where people come and pay us to ship their packages from point A to point B. My features  $x$  captures user, origin/destination and price.  $y = 1$  if the user shipped the package with us,  $y = 0$  otherwise. What we want to do is optimize prices and we do that by finding out  $p(y = 1 | x; \theta)$ . My online learning algorithm would like as shown below. What it does is it has an infinite loop (because I'm constantly getting new training examples, i.e. new sessions where the users ships or not ships with us) where I take one example and update  $\theta_j$  using  $(x, y)$  with my known equation:  $\theta_j =$



$\theta_j - \alpha (h_\theta(x) - y) * x_j$ . One last thing to mention is the algorithm adapts to changing user preferences because we train the algorithm using the information from those new user preferences.

### Online learning

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  $(x, y)$ .  ~~$(x, y)$~~   
 $\rightarrow \theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j \quad (j = 0, \dots, n)$   
 }  
 $\rightarrow$  Can adapt to changing user preference.

- Another example of online learning is product search where I am trying to predict the probability of a user clicking a link. In this case we are showing 10 results per page so we get 10  $(x, y)$  training examples. The  $x$  vector could contain features of the phone, how many of the query words match the phone name, how many of the query words match the description and so on. We can use the learned probability to show the user the 10 results she/he is most likely to click on.

### Other online learning example:

Product search (learning to search)

User searches for "Android phone 1080p camera"  $\leftarrow$

Have 100 phones in store. Will return 10 results.

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

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

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

$\rightarrow$  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; ...

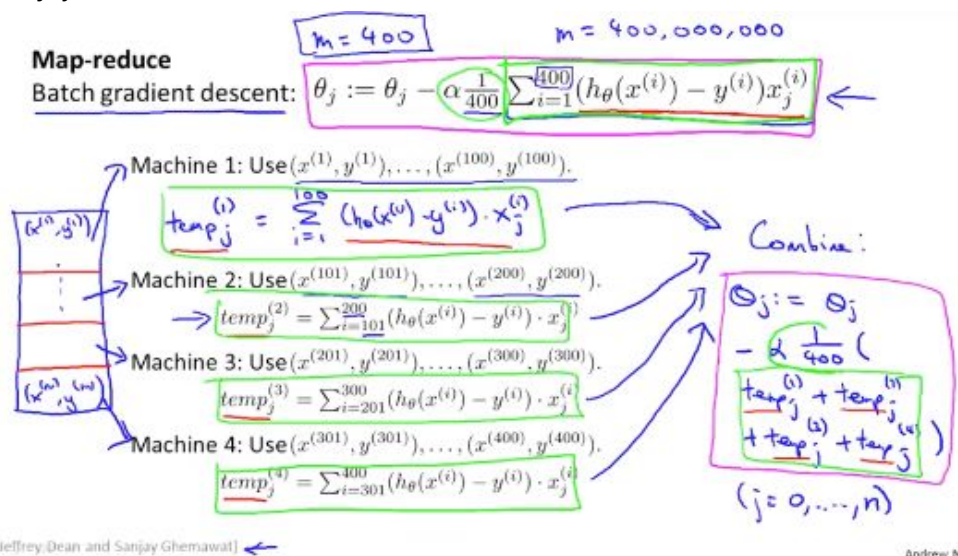
Other online learning examples are: choosing what special offers to show, customized selection of news articles and so on.

### Advanced Topics: MapReduce and Parallelism

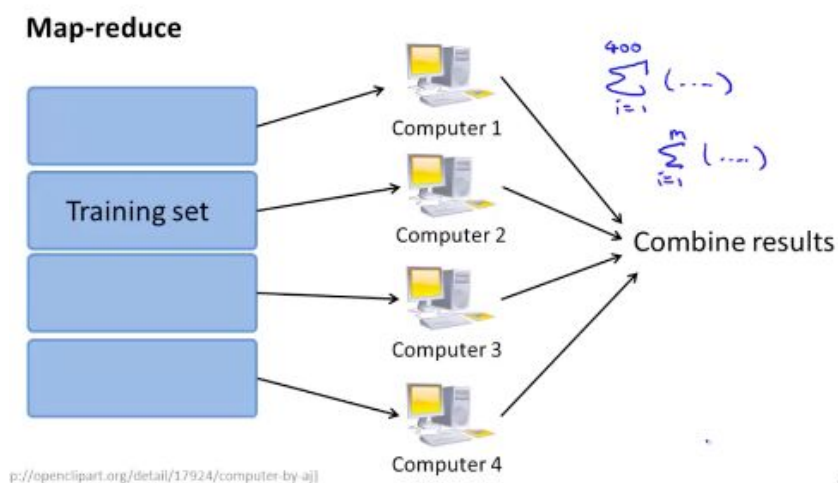
- Sometimes machine learning algorithms are too big to run on one server, we are going to see how mapReduce allows us to run them over many servers. Suppose I have a batch

gradient algorithm to run with  $m = 400$ . What map/reduce (the foundation of Hadoop) does is it breaks the the training set into 4 pieces and assigns each piece to one server (assume we have 4 servers here, could have been 8 servers and 8 pieces and so on).

Machine 1 will compute  $temp_j^{(1)} = \sum_{i=1}^{100} (h_{\theta}(x^{(i)}) - y^{(i)}) * x_j^{(i)}$ . Machine 2 will do the same computation but with examples 101-200 and so on. Once all servers are done we send the info to a central server that combines all the temp variables in the following equation:  $\theta_j = \theta_j - \alpha(1/400) (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$  and this equation is the same as if we had done it in one server. MapReduce was created by Jeff Dean and Sanjay Gimawat.



- Here's the general MapReduce picture



We take a training set and divided evenly, as much as possible (4 pieces in the example above), send each piece to a computer where each computer does some of the work and then send everything to another server where the results are combined. This helps

us to speed up the execution of our algorithm. It's not 4x because of network latencies and other factors but you do get a speed bump by using mapreduce.

- If we are thinking of using mapreduce for our speeding up a learning algorithm the key question to ask is if the algorithm can be expressed as a summation over the training set. Whenever the bulk of the work of the learning algorithm is the summation of functions over a training set MapReduce is a good candidate to speed its execution.
- Here's another example where we can use MapReduce. Let's say we are using logistic regression. We can observe how the cost function  $J_{train}(\theta)$  and its partial derivative contains summations. We can use mapreduce to break up the training set into pieces and use multiple computers to compute parts of each summation.

### Map-reduce and summation over the training set

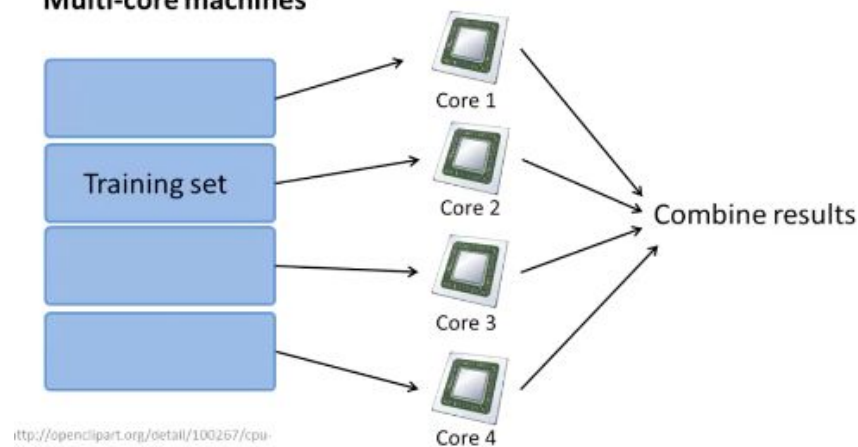
Many learning algorithms can be expressed as computing sums of functions over the training set.

E.g. for advanced optimization, with logistic regression, need:

$$\begin{aligned} \rightarrow J_{train}(\theta) &= -\frac{1}{m} \sum_{i=1}^m \underbrace{y^{(i)} \log h_{\theta}(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))}_{\text{term}^{(i)}} \\ \rightarrow \frac{\partial}{\partial \theta_j} J_{train}(\theta) &= \frac{1}{m} \sum_{i=1}^m \underbrace{(h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}_{\text{term}_j^{(i)}} \end{aligned}$$

- Finally, let's take a look how we can use mapreduce even in one server provided the computer has multiple cores. We just sent each piece of the training set to each core.

### Multi-core machines



The advantage of doing all the computation in one server is we do not have network latencies. Some linear algebra libraries take advantage of parallelizing the work among multiple cores so you don't need to implement mapreduce.