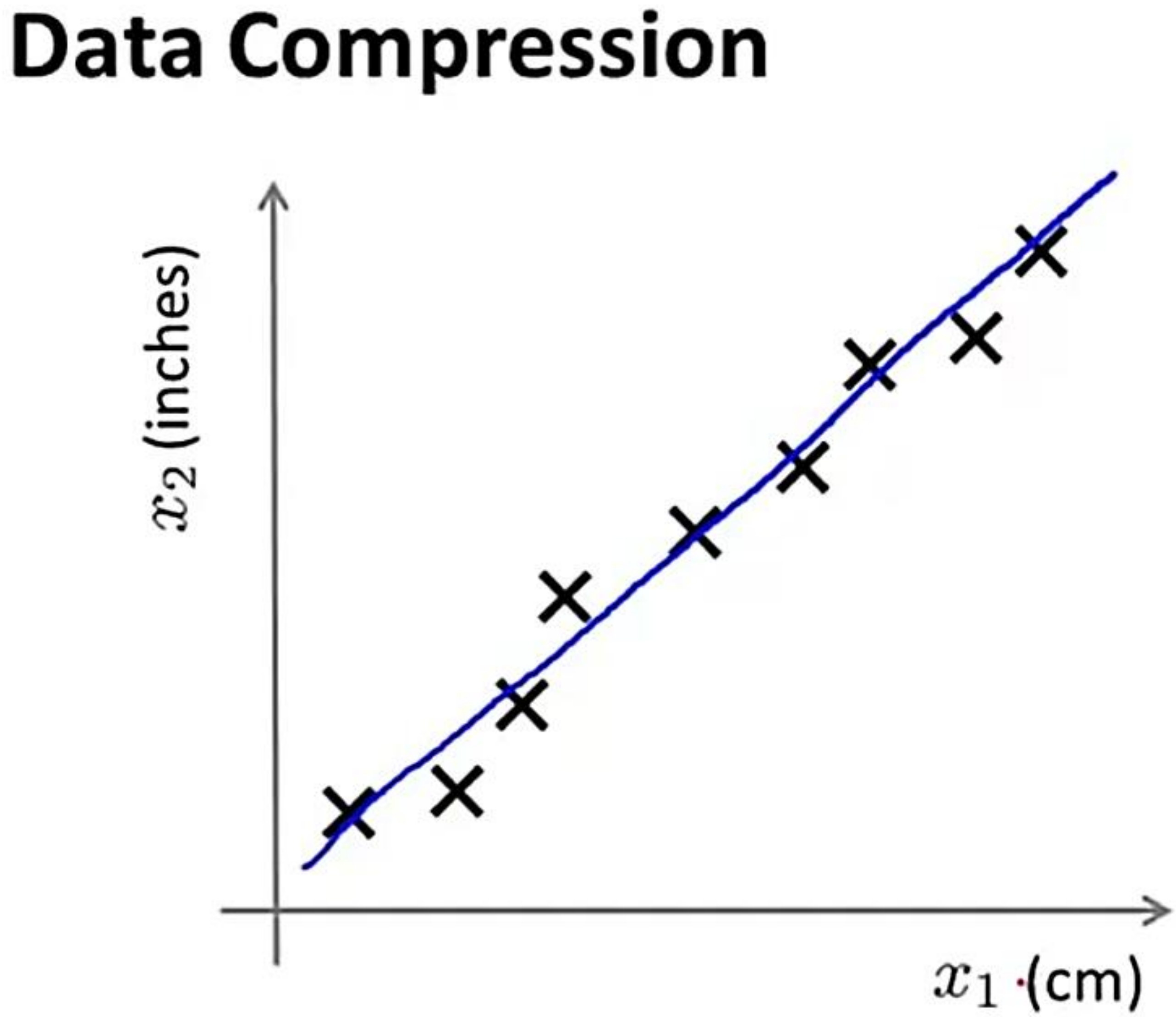
14. Dimensionality Reduction – Unsupervised Learning Problem

It will allow us **data compression** and **speeding up** the algorithm.

Example:

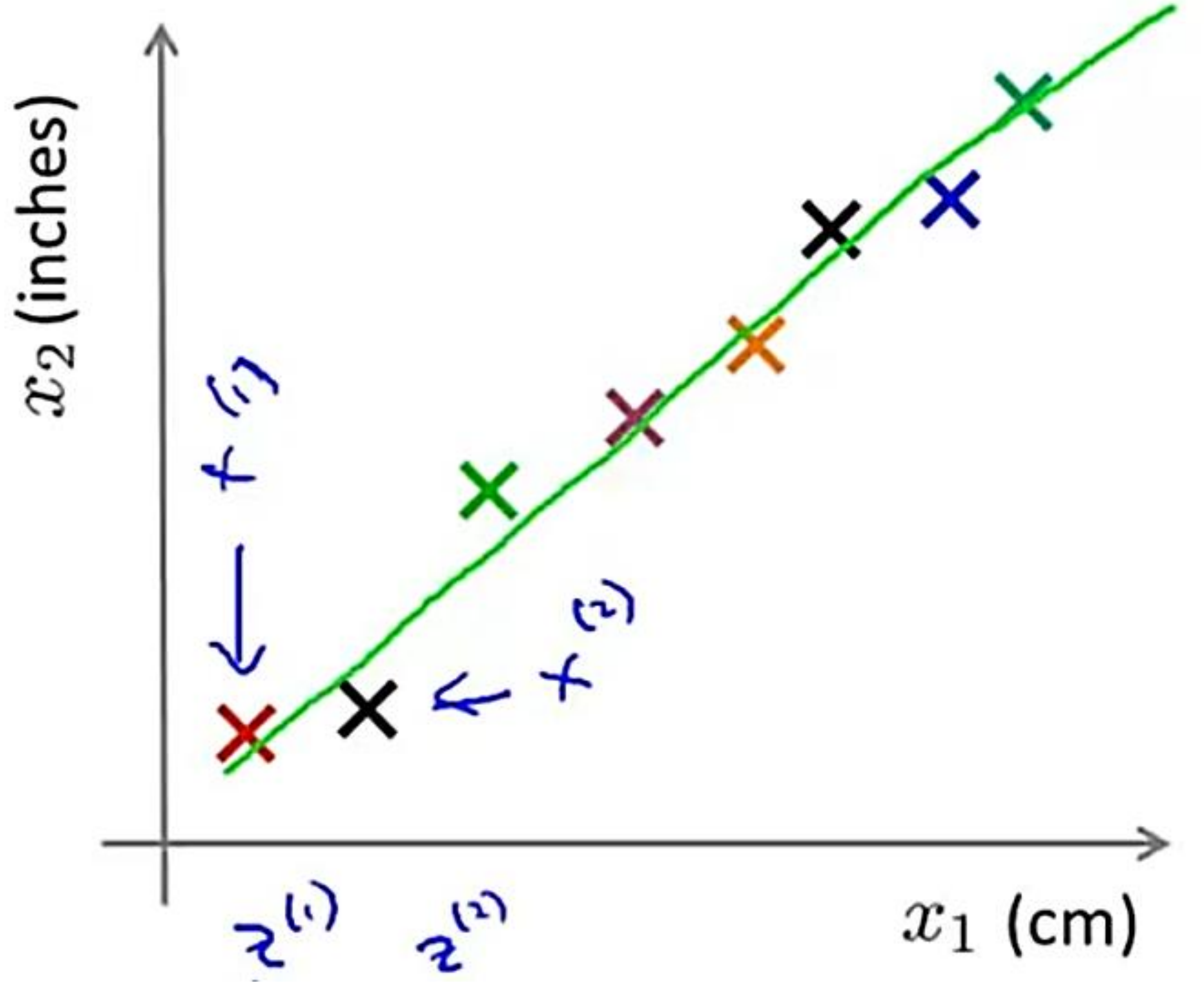


The points don’t lie on the straight line due to rounding off.

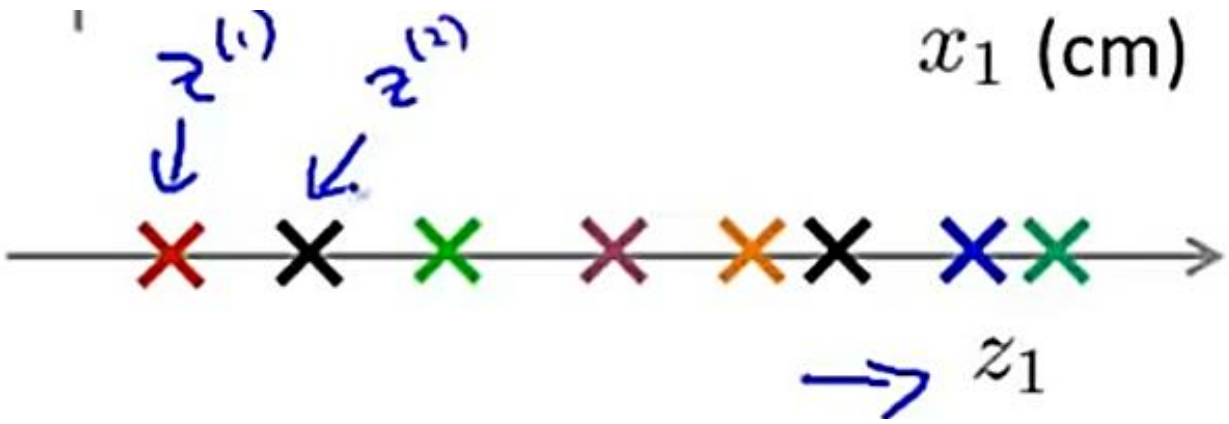
≫ Reduce Data from 2D to 1D

- **Since the two features are proportional (not exactly, just approximately):**

We try to project each data point on a line(green)



We represent this line as a **new feature → z :**

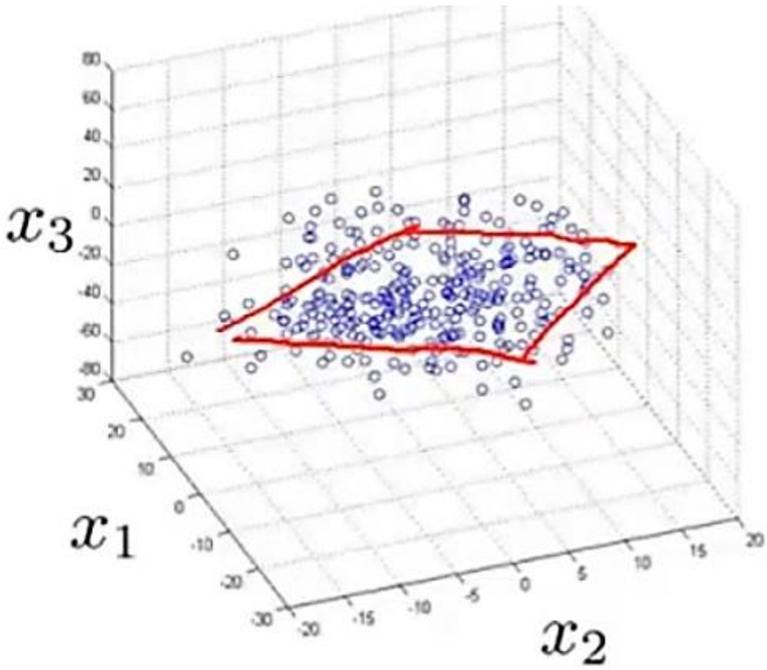


This reduces one dimension from out data:

Reduce data from
2D to 1D

$$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$$
$$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$$
$$\vdots$$
$$x^{(m)} \rightarrow z^{(m)}$$

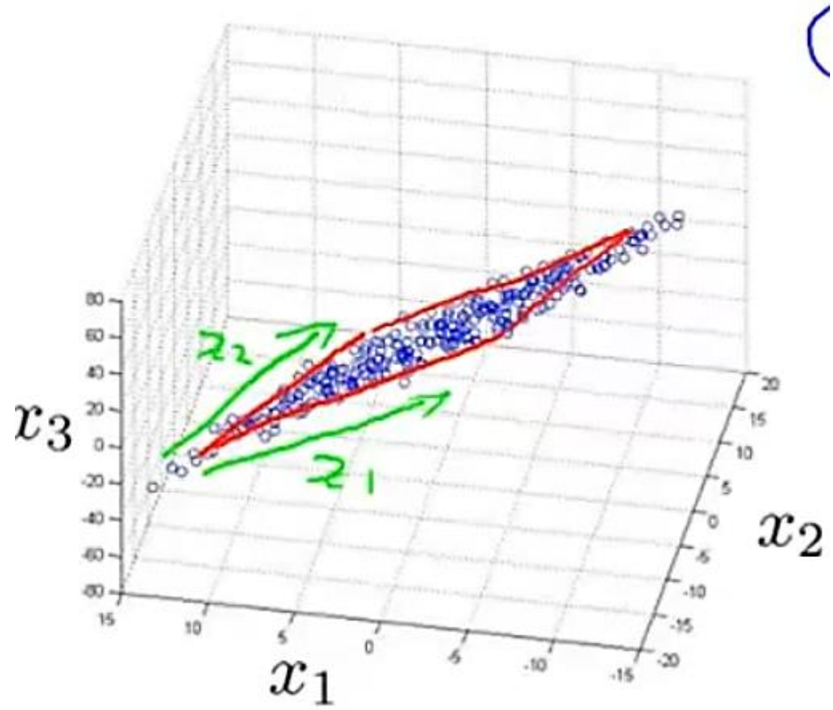
≫ Reduce Data from 3D to 2D



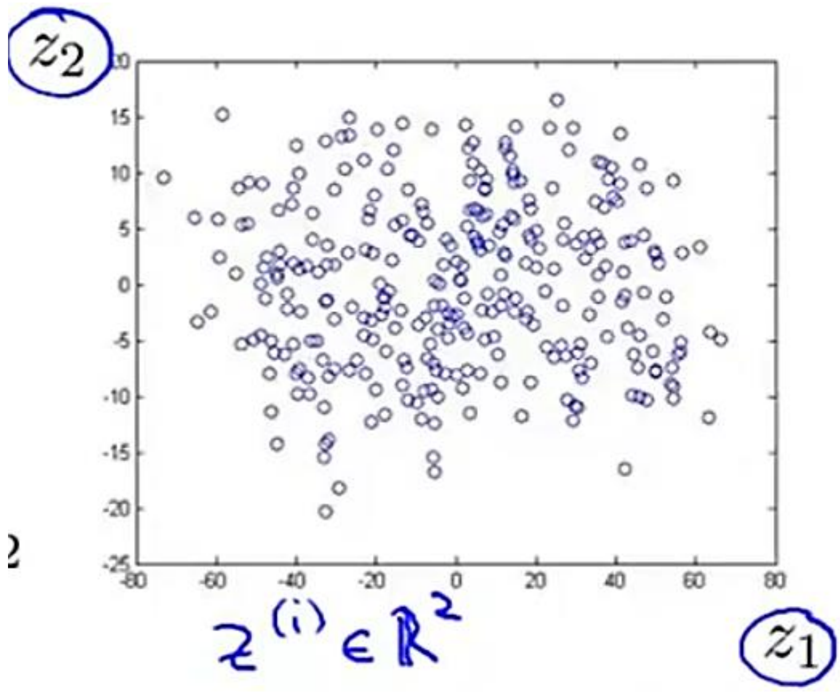
$$x^{(i)} \in \mathbb{R}^3$$

Let’s say, this data roughly lies on a plane

So, we project the data on that plane:



Hence, we have reduced the data from 3D to 2D



Where:

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

Dimensionality reduction: Data Visualization:

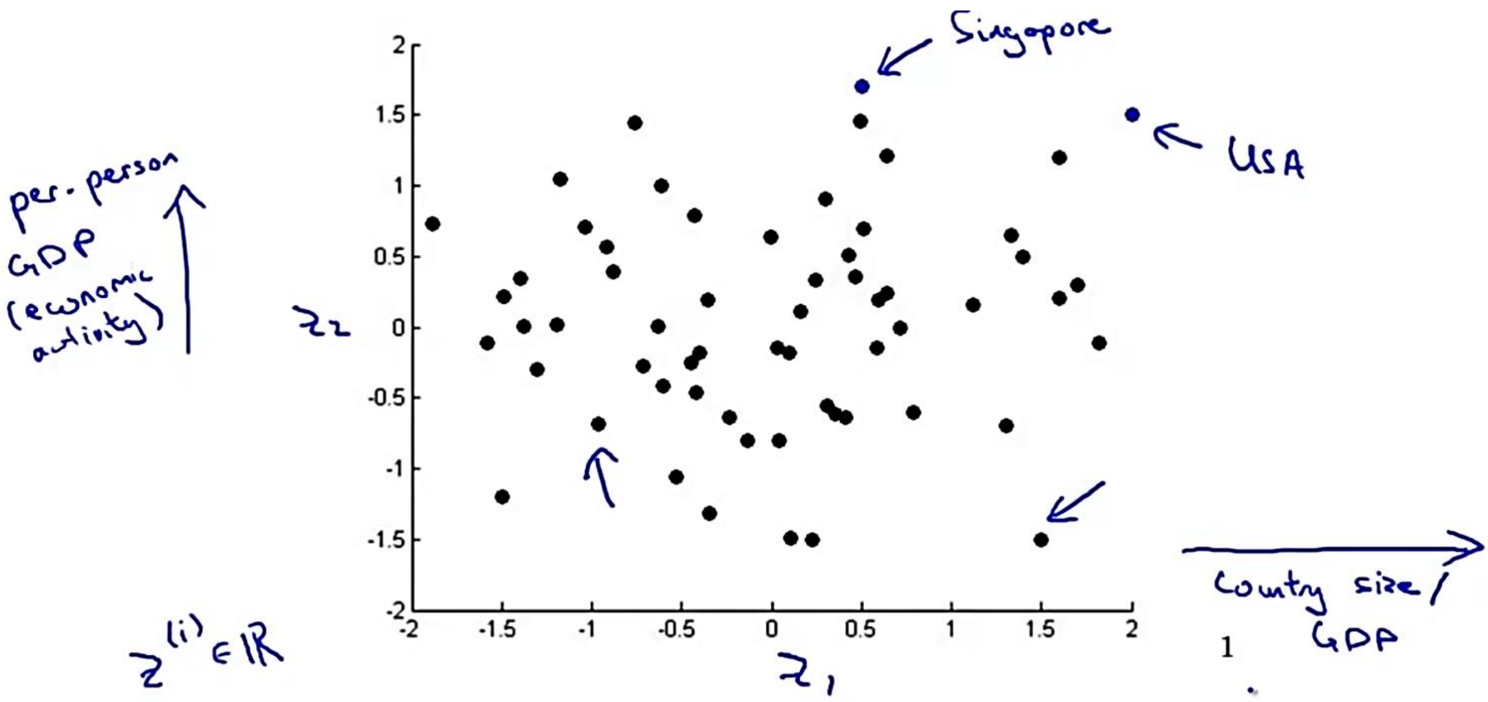
Suppose we have a 50D feature vector

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	…
Canada	1.577	39.17	0.908	80.7	32.6	67.293	…
China	5.878	7.54	0.687	73	46.9	10.22	…
India	1.632	3.41	0.547	64.7	36.8	0.735	…
Russia	1.48	19.84	0.755	65.5	39.9	0.72	…
Singapore	0.223	56.69	0.866	80	42.5	67.1	…
USA	14.527	46.86	0.91	78.3	40.8	84.3	…
…	…	…	…	…	…	…	

$x^{(i)} \in \mathbb{R}^{50}$

We have to find out two **new features that can summarise these 50 features**… then we can easily plot the data in 2D

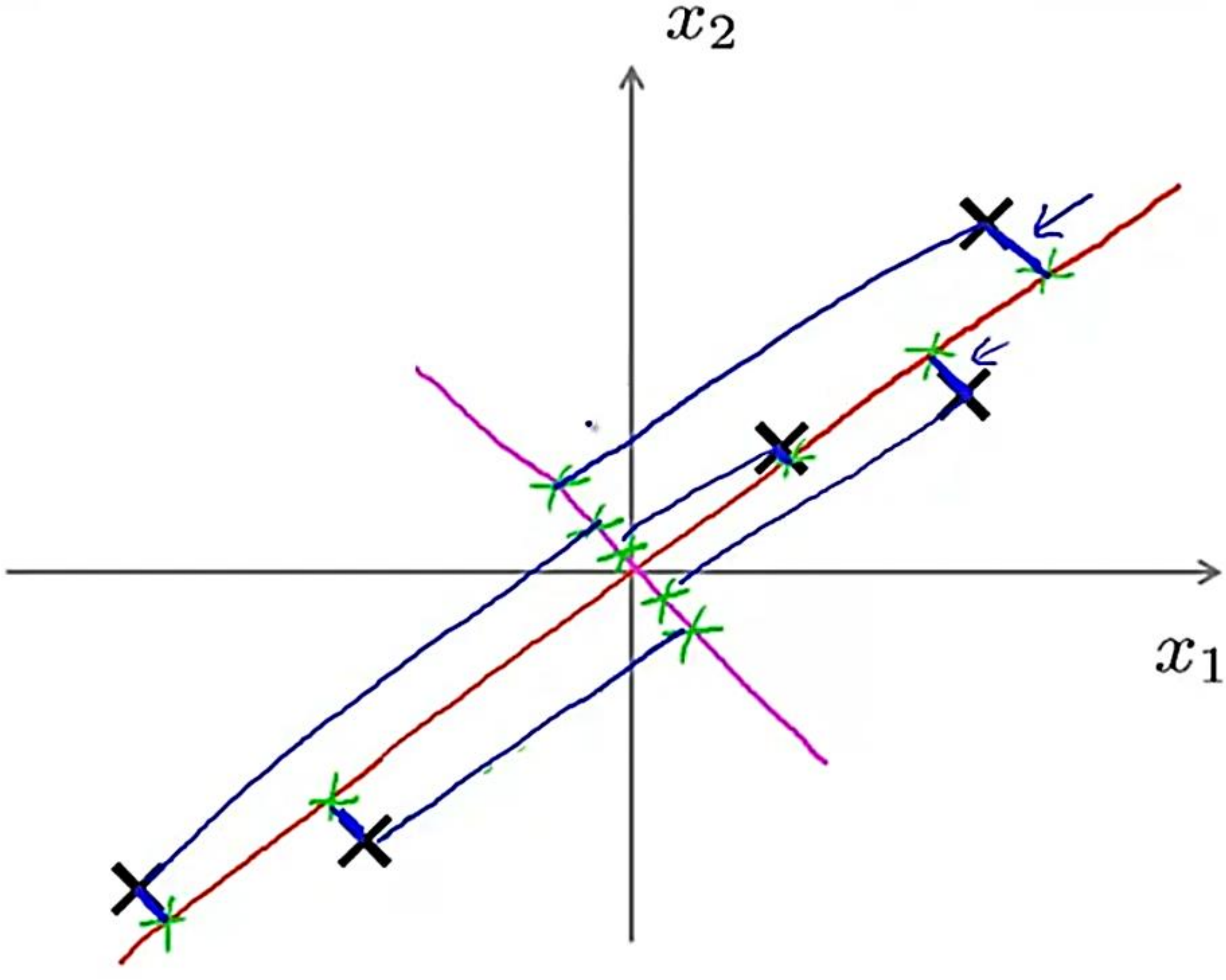
We can use the two most important features of deviation



≫ PRINCIPLE COMPONENET ANALYSIS:

--algo for dimensionality reduction

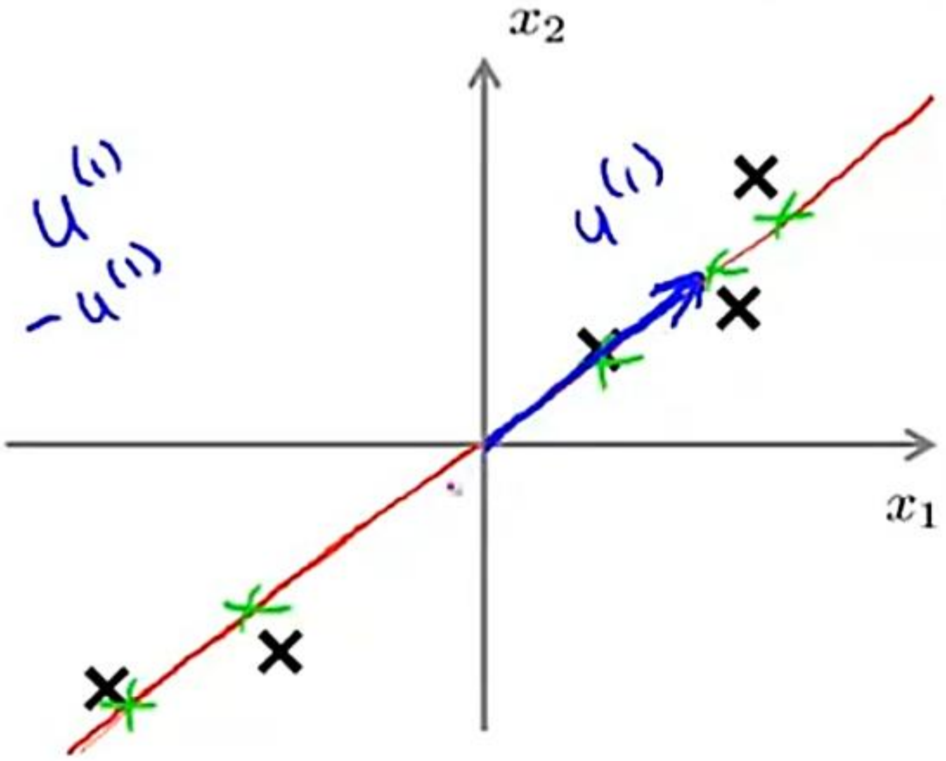
Problem formulation:



The PCA algo will chose red line to reduce the dimensions of problem from 2D to 1D because the projection errors (blue lines) in magenta line are much higher.

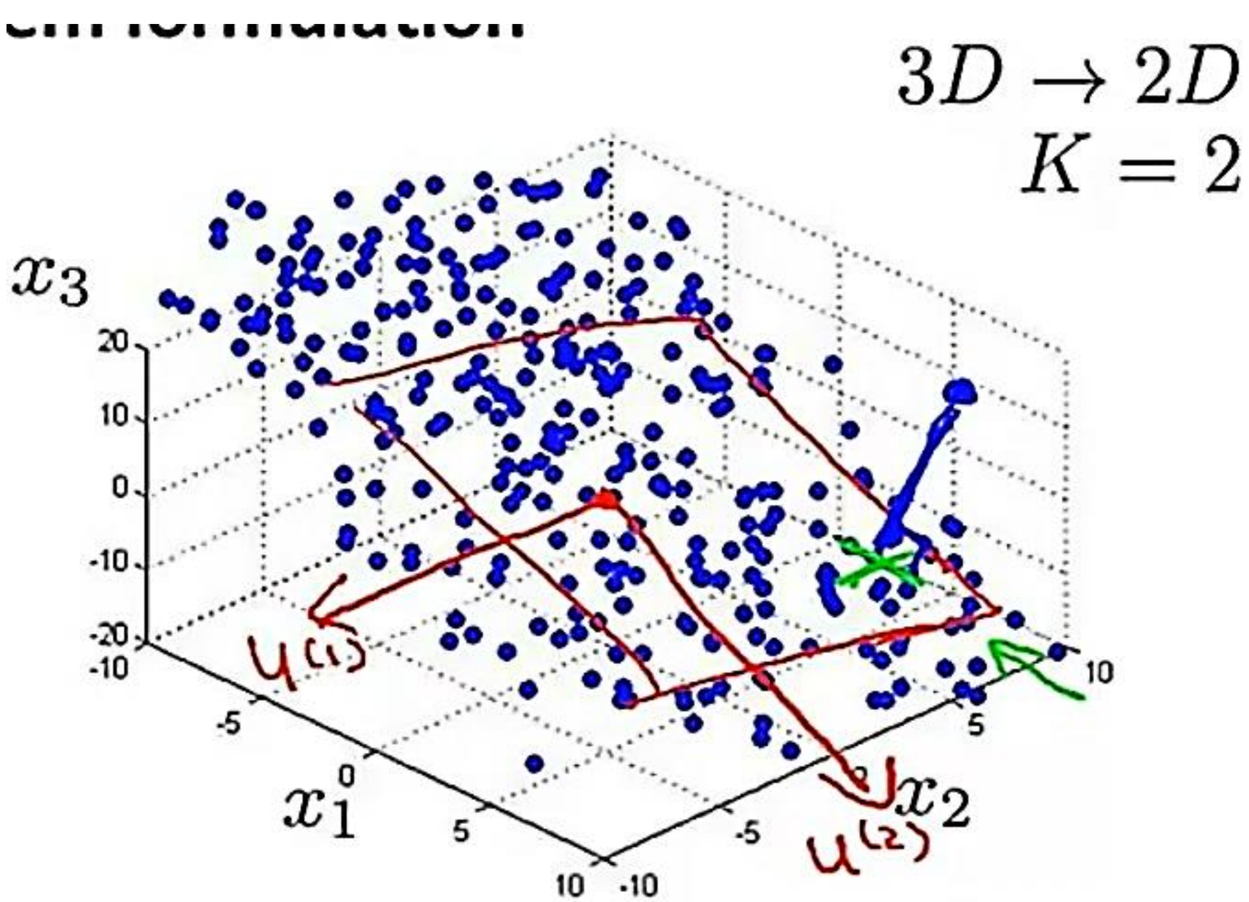
- **Projection errors = reduction errors**

Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.



More generally:

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



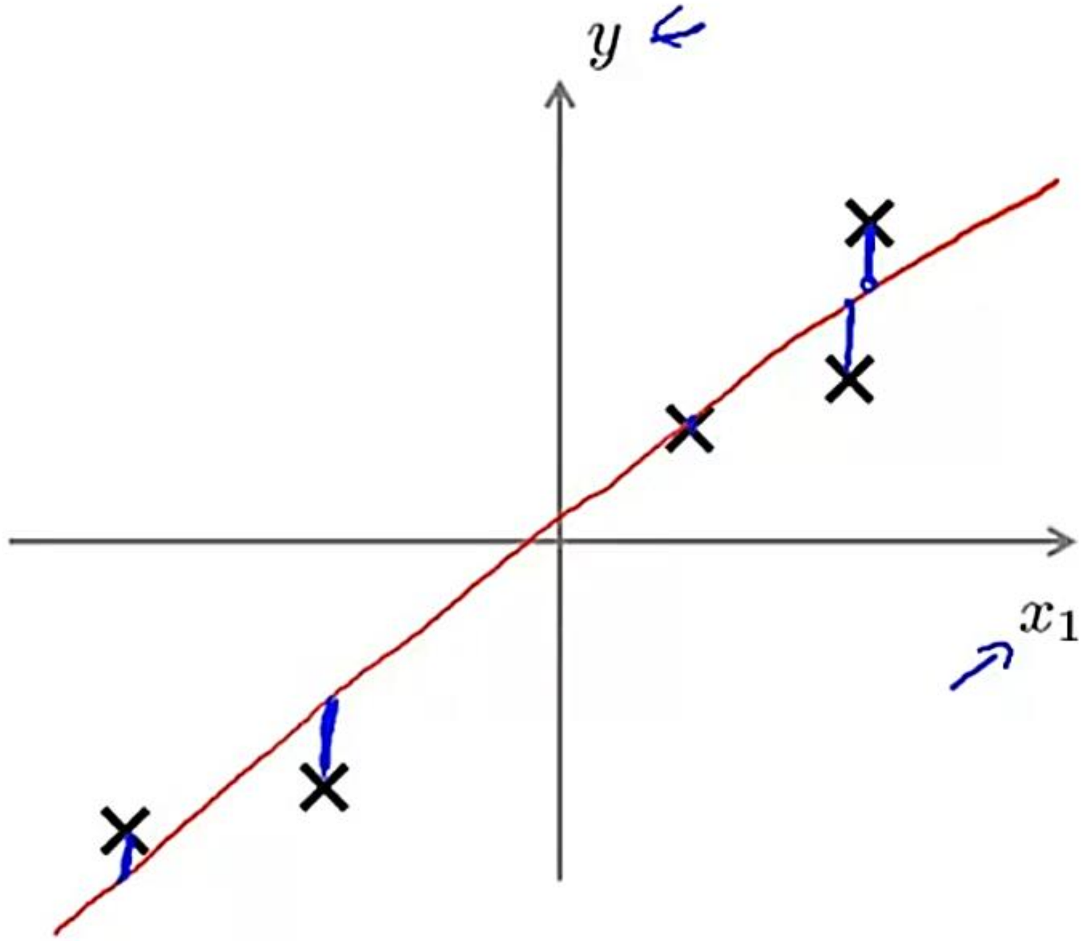
≫ It may seem that PCA is like Linear Regression, but its not.

In linear regression:

- ➔ The graph is b/w features vs output

$x \rightarrow \underline{y}$

- ➔ The error is the vertical distance b/w actual o/p and predicted o/p(line)

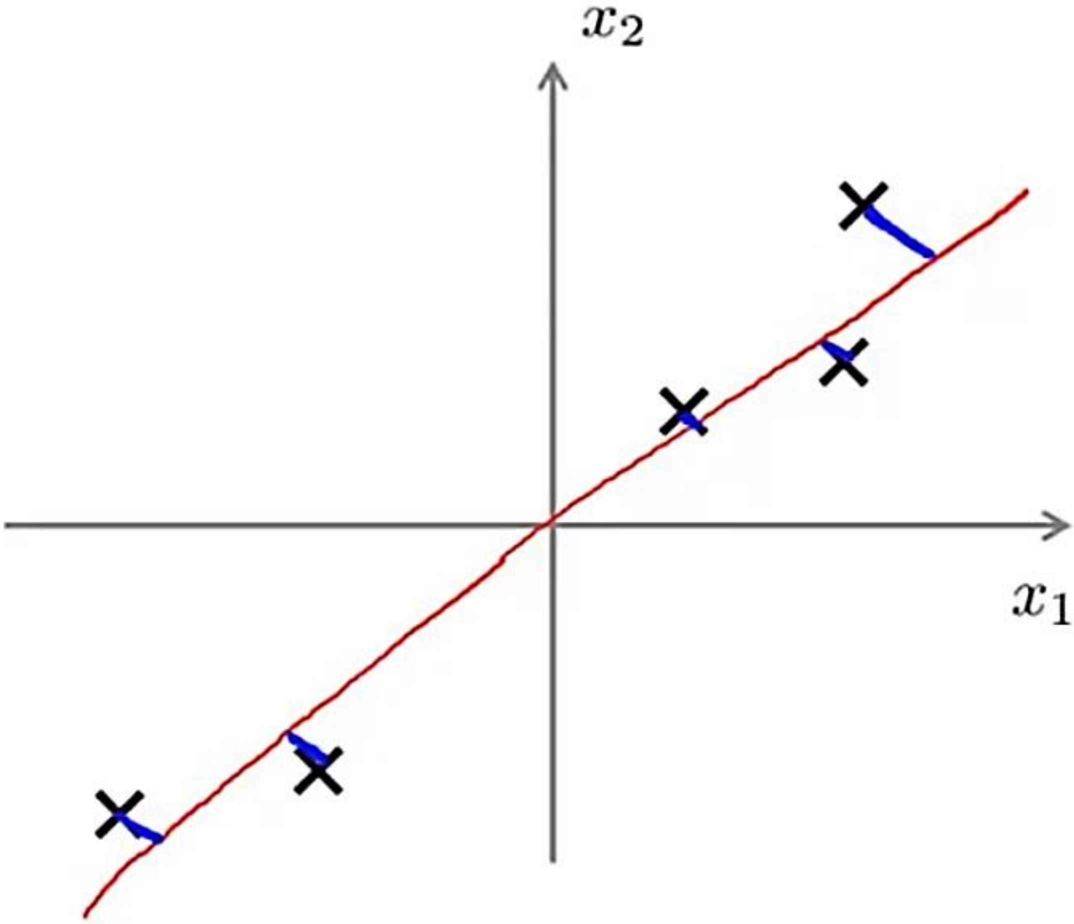


In PCA:

- ➔ The graph is b/w features only

$x_1, x_2, \dots, x_n$

➔ The error is projection of data on the PCA line.



≫ PRINCIPAL COMPOENT ANALYSIS – ALGOORITHM:

- First we do the data pre-processing:

Data preprocessing

Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

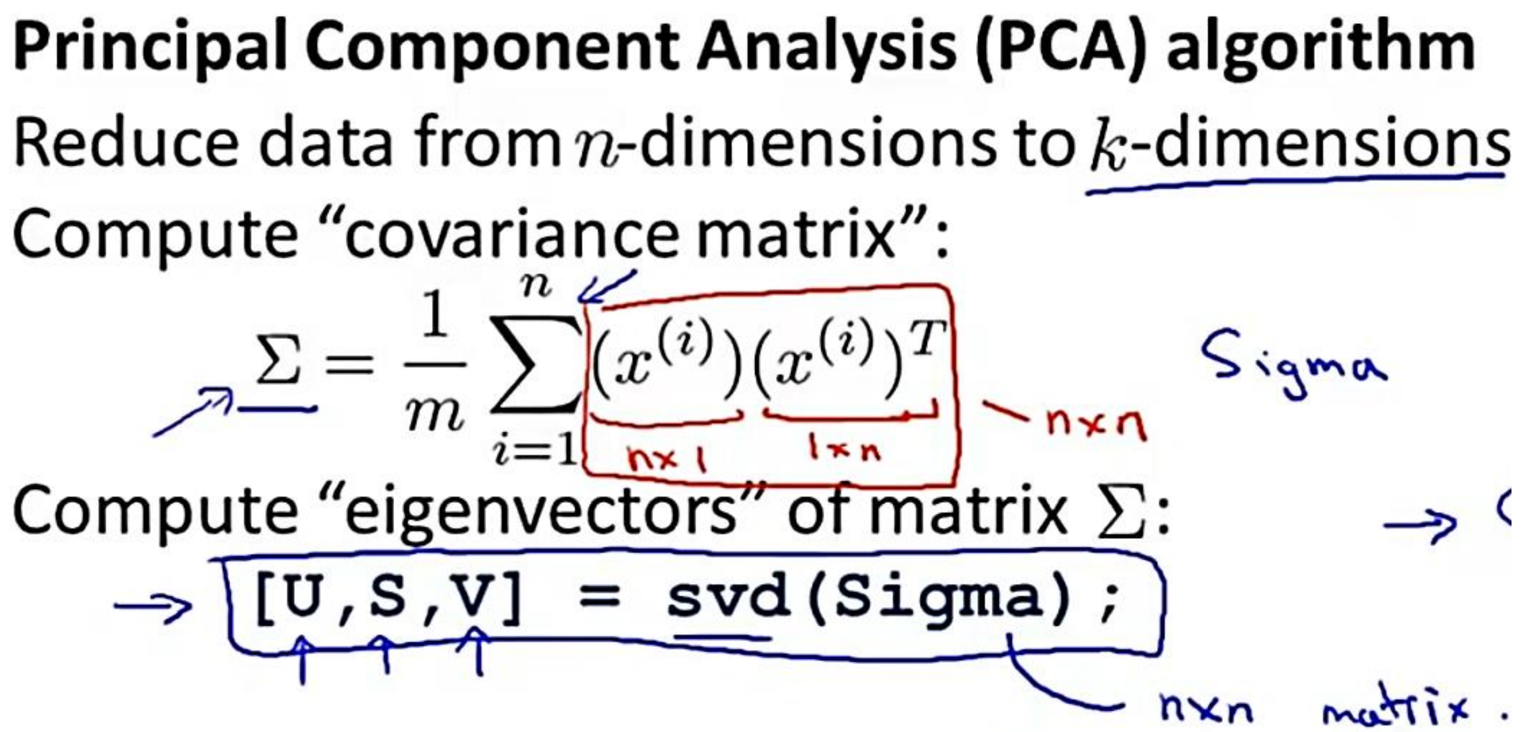
If different features on different scales (e.g., $x_1 =$ size of house, $x_2 =$ number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

μ_j → it's the mean of all values of that feature

S_j → it's the range of that feature values → standard deviation

➢ **Algorithm:**



➢ **SVD** = Singular Value Decomposition

We can also use eig() instead of svd()

➢ **Sigma** = Covariance matrix obtained

⇨ **U** = the matrix of all the vectors we need, we just take the first "k" vectors from it, and those are our Dimensionality reduction vectors

From [U,S,V] = svd(Sigma), we get:

$$\rightarrow U = \begin{bmatrix} \underbrace{\begin{matrix} | & | \\ u^{(1)} & u^{(2)} \\ | & | \end{matrix}}_{k} & \dots & \begin{matrix} | \\ u^{(n)} \\ | \end{matrix} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

Now, to find the "z" feature vector:

$$x \in \mathbb{R}^{\cancel{n}\, k} \longrightarrow z \in \mathbb{R}^k$$

$$z = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}}_{\substack{n\times k \\ U_{reduce}}}^T x = \underbrace{\underbrace{\begin{bmatrix} — (u^{(1)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix}}_{k\times n} \underbrace{x}_{n\times 1}}_{k\times 1}$$

$z \in \mathbb{R}^k$

Here, **X** → can be training data vector, cross validation, test data..

➔ It can also be a particular example: $\mathbf{X}^{(i)}$

Principal Component Analysis (PCA) algorithm summary

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

$$\texttt{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T \qquad X = \begin{bmatrix} — x^{(1)T} — \\ \vdots \\ — x^{(m)T} — \end{bmatrix}$$

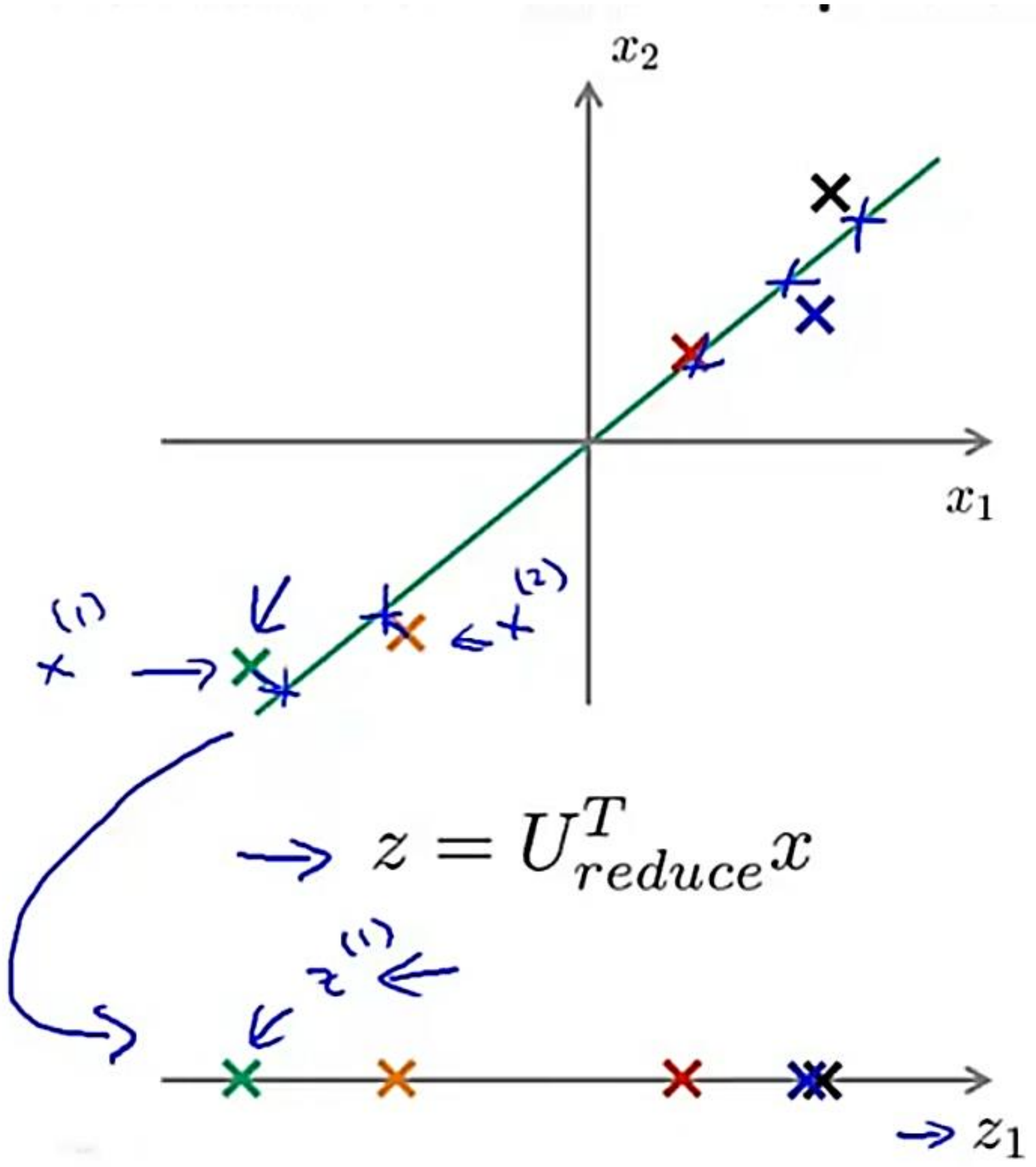
$$\boxed{\texttt{Sigma} = (1/m) * X' * X;}$$

```
→ [U,S,V] = svd(Sigma);
→ Ureduce = U(:,1:k);
→ z = Ureduce'*x;
```

$x \in \mathbb{R}^n$ $\quad \cancel{x_0 = 1}$

≫ RECONSTRUCTION OF DATA FROM COMPRESSED REPRESENTATION:

We have:

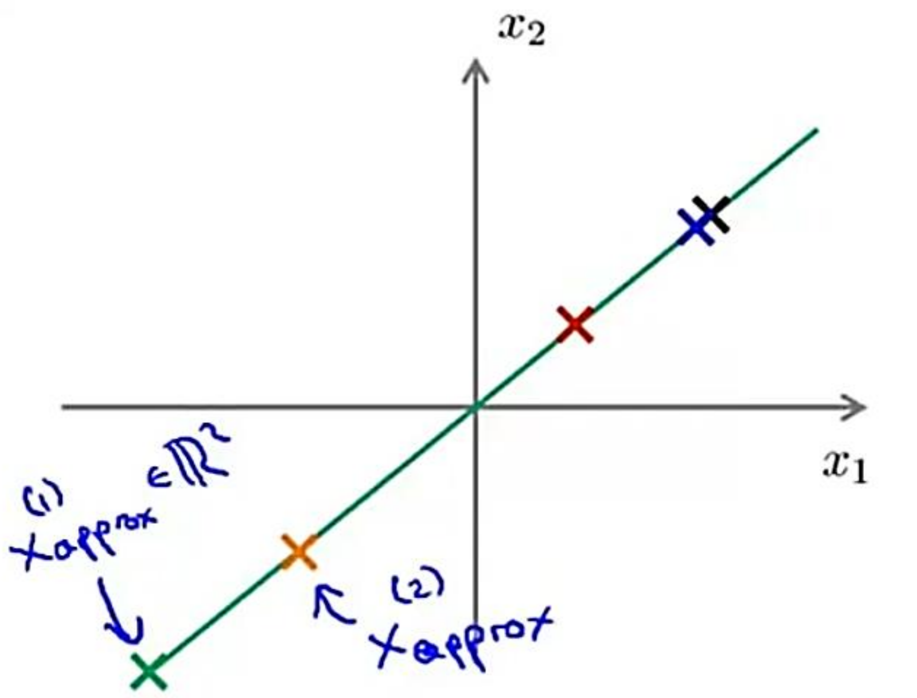


To approximate the original data:

$$\approx x$$

$$X_{approx} = \underbrace{\underbrace{U_{reduce}}_{n \times k} \cdot \underbrace{z^{(1)}}_{k \times 1}}_{n \times 1}$$

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



$$z \in \mathbb{R} \longrightarrow x \in \mathbb{R}^2$$

$$\approx x$$

$$x_{approx}^{(1)} = \underbrace{\underbrace{U_{reduce}}_{n\times k} \cdot \underbrace{z^{(1)}}_{k\times 1}}_{n\times 1}$$

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

≫ HOW TO CHOOSE THE VALUE OF "k":

Choosing k (number of principal components)

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

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \le \underline{0.01} \qquad \underline{(1\%)}$$

"<u>99%</u> of variance is retained"

Algorithm:

Choosing k (number of principal components)

Algorithm:

Try PCA with $\cancel{k=1}$ $\cancel{k=2}$ $\cancel{k=3}$ $k=4$ ...

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$

Check if

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

$k = 17$

We try diff values of k from 1 until we get the ratio ≤0.01 . But this is not an efficient way: So, **svd()** saves us:

```
→ [U,S,V] = svd(Sigma)
```

$$S = \begin{bmatrix} s_{11} & & & 0 \\ & s_{22} & & \\ & & s_{33} & \\ & & & \ddots \\ 0 & & & & s_{nn} \end{bmatrix}$$

S→ n x n diagonal matrix

The ratio → (Average Squared projection error) / (Total variance in the data):

For given k $\qquad\qquad$ k=3

$$\rightarrow 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

OR

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

```
[U,S,V] = svd(Sigma)
```

Pick smallest value of k for which

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

(99% of variance retained)

≫ Speedup Advice:

Supervised learning speedup

$$\boxed{(x^{(1)}, y^{(1)})}, (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$$

$x^{(i)} \in \mathbb{R}^{10,000}$

Extract inputs:

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

$$\downarrow PCA$$

U_{reduce}

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

New training set:

$$\boxed{(z^{(1)}, y^{(1)})}, (z^{(2)}, y^{(2)}), \ldots, (z^{(m)}, y^{(m)})$$

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.

≫ Overfitting Advice:

Bad use of PCA: To prevent overfitting

Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$. — 10000

1000

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

$$\min_\theta \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}$$

≫ Misuse of PCA:

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
- Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \ldots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

But instead:

- How about doing the whole thing without using PCA?
- Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.