# 446 Section 09

TA: Yufei Zhang

## Plans for today!

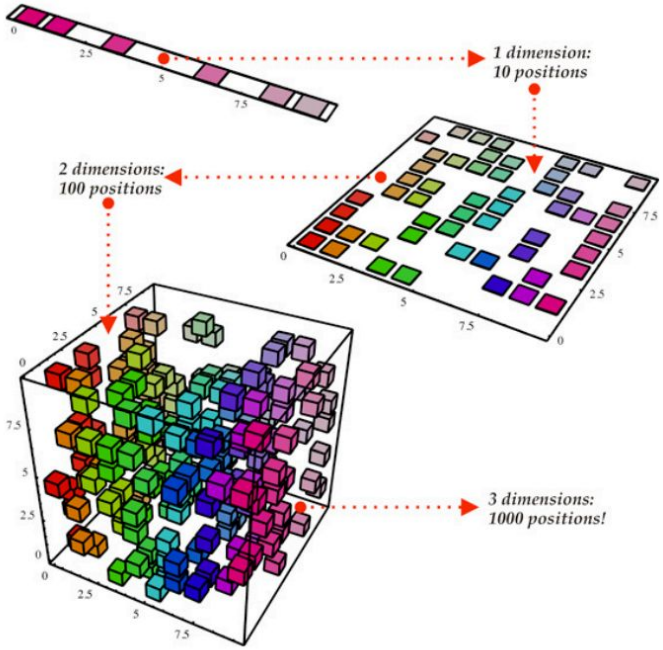
1. This
2. Reminders
3. PCA
4. SVD
5. CNNs

# Reminders

- HW4 due Mar 11th!
- **Final Exam**: 3/18, 8:30am – 10:20pm

# High Dimensional Data

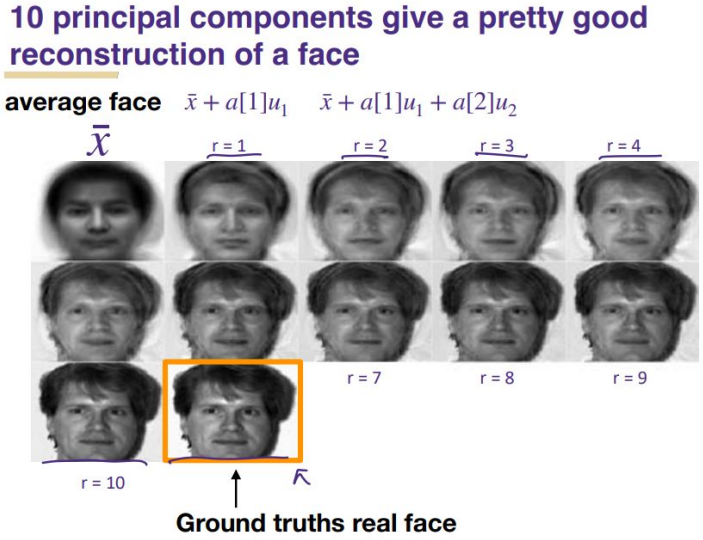
- Dealing with high-dimensional data can lead to problems (curse of dimensionality)
- Visualization is very hard for $d > 2$
- Intrinsic dimensionality may be lower



# Dimensionality Reduction

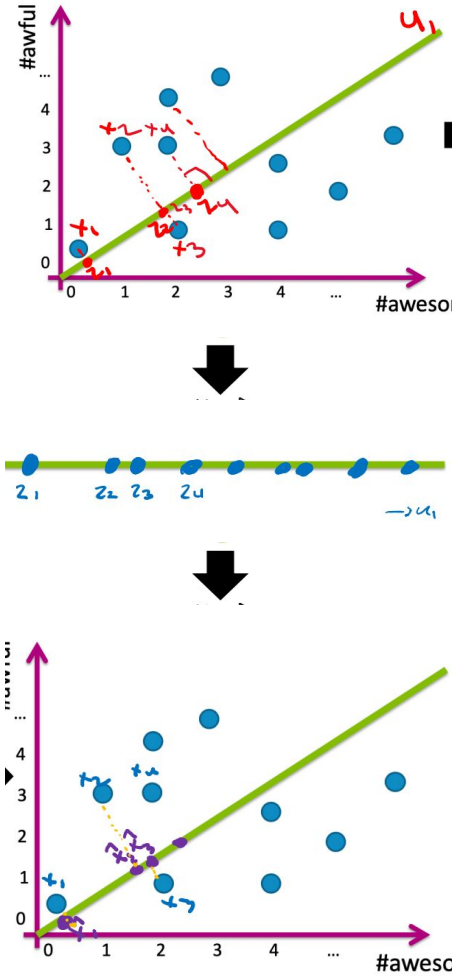
**Goal:** Find a lower-dimensional representation of your data **X** with the least loss of information

We need to find a subspace of the original data span **to project down to**. The **Principal Components** of our data can help us do this



# Principal Component Analysis

- **Idea**: Use a linear projection from $d$-dimensional data to $k$-dimensional data, where $d > k$
  - 1000-dimensional word vectors to 3-dimensional
- Choose the projection that minimizes reconstruction error
  - Intuition: The information lost if you were to ”undo” the projection can give us meaningful information
  - Choosing vectors that can capture the most **variation** (during projection)



# Recall from lecture…

## PCA: a high-fidelity linear projection

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} \left|\left|(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^\top (x_i - \bar{x})\right|\right|_2^2$$

$\mathbf{V}_q \mathbf{V}_q^T$ is a *projection matrix* that minimizes error in basis of size $q$

$$\widehat{x}_i := \bar{x} + \mathbf{V}_q \mathbf{V}_q^\top (x_i - \bar{x}) = \bar{x} + \sum_{j=1}^{q} v_j v_j^\top (x_i - \bar{x})$$

Case when $q = 1$

$$v_1 = \arg \min_{v:\|v\|_2=1} \sum_{i=1}^{N} \|(x_i - \bar{x}) - vv^\top (x_i - \bar{x})\|_2^2$$

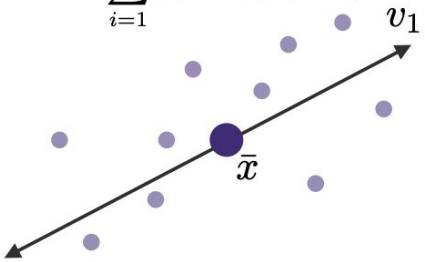
$$= \arg \min_{v:\|v\|_2=1} \sum_{i=1}^{N} \|x_i - \bar{x}\|_2^2 - 2(x_i - \bar{x})^\top vv^\top (x_i - \bar{x}) + (x_i - \bar{x})^\top vv^\top vv^\top (x_i - \bar{x})$$

$$= \arg \min_{v:\|v\|_2=1} \sum_{i=1}^{N} \|x_i - \bar{x}\|_2^2 - \sum_{i=1}^{N} (x_i - \bar{x})^\top vv^\top (x_i - \bar{x})$$

$$= \arg \max_{v:\|v\|_2=1} \sum_{i=1}^{N} (x_i - \bar{x})^\top vv^\top (x_i - \bar{x})$$

$$= \arg \max_{v:\|v\|_2=1} v^\top \Sigma v$$

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T$$

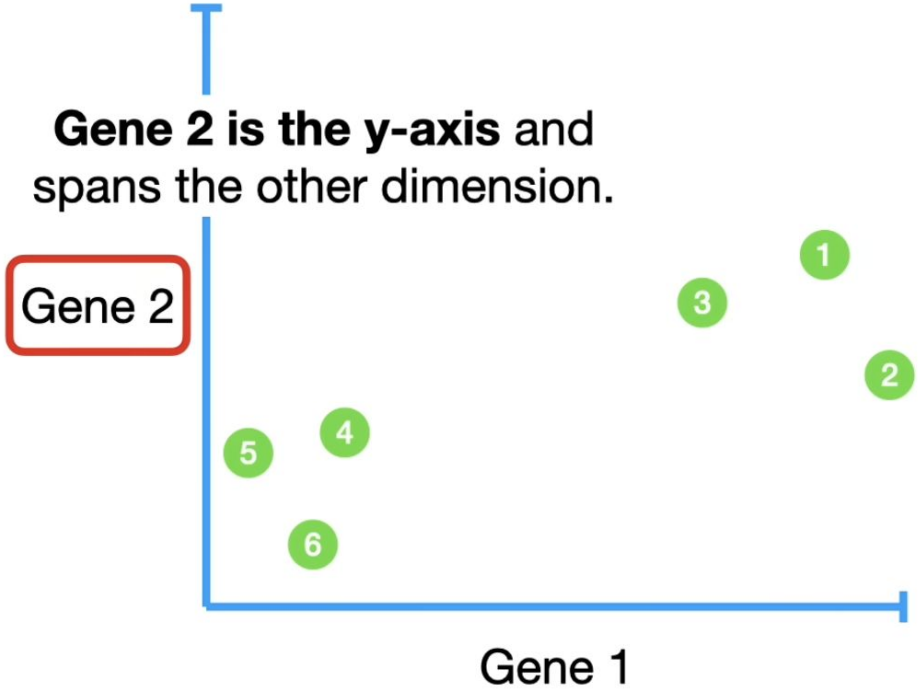



# Let’s do some PCA Intuition Building

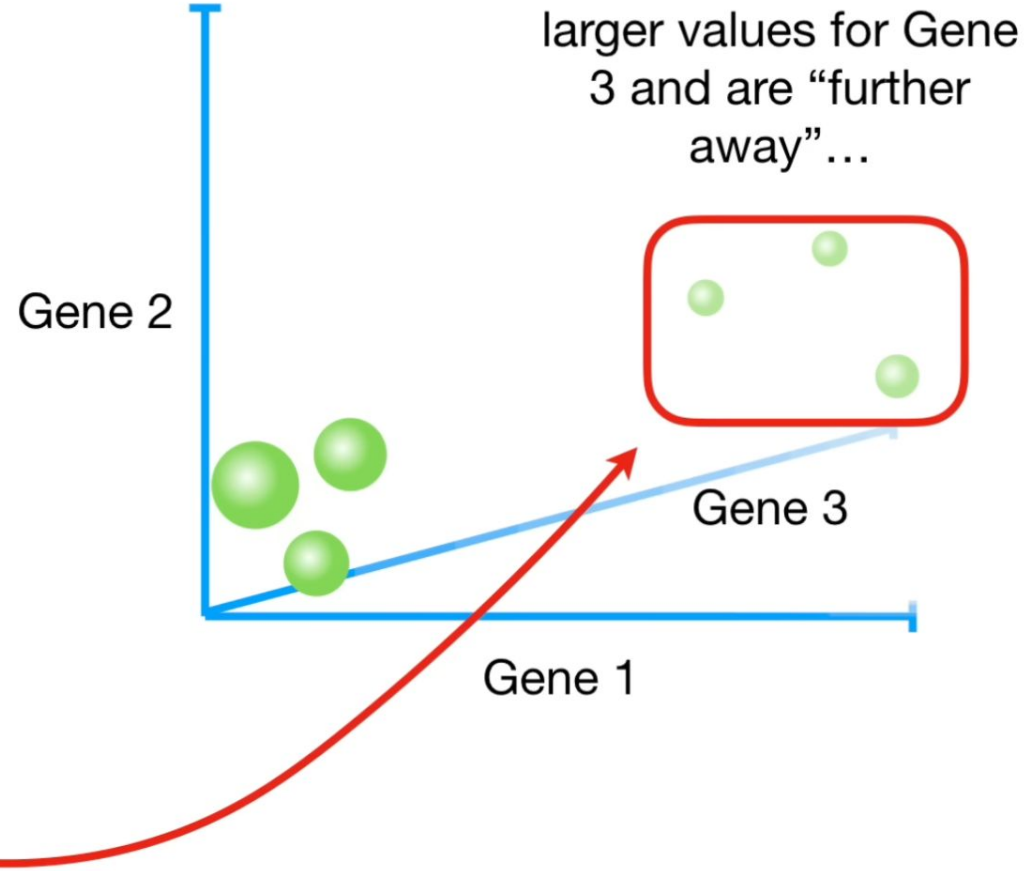
| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| Gene 1 | 10 | 11 | 8 | 3 | 1 | 2 |

Mice 1, 2 and 3 have relatively high values…

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 1 | 2 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |



| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |
| **Gene 3** | 12 | 9 | 10 | 2.5 | 1.3 | 2 |



# Understanding the decomposition

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i=1}^{r} u_i \sigma_i v_i^\top$$

Rank-r approximation

Rank-2 approximation

Rank-1 approximation

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = u_1\sigma_1 v_1^\top + u_2\sigma_2 v_2^\top + \ldots + u_r\sigma_r v_r^\top$$

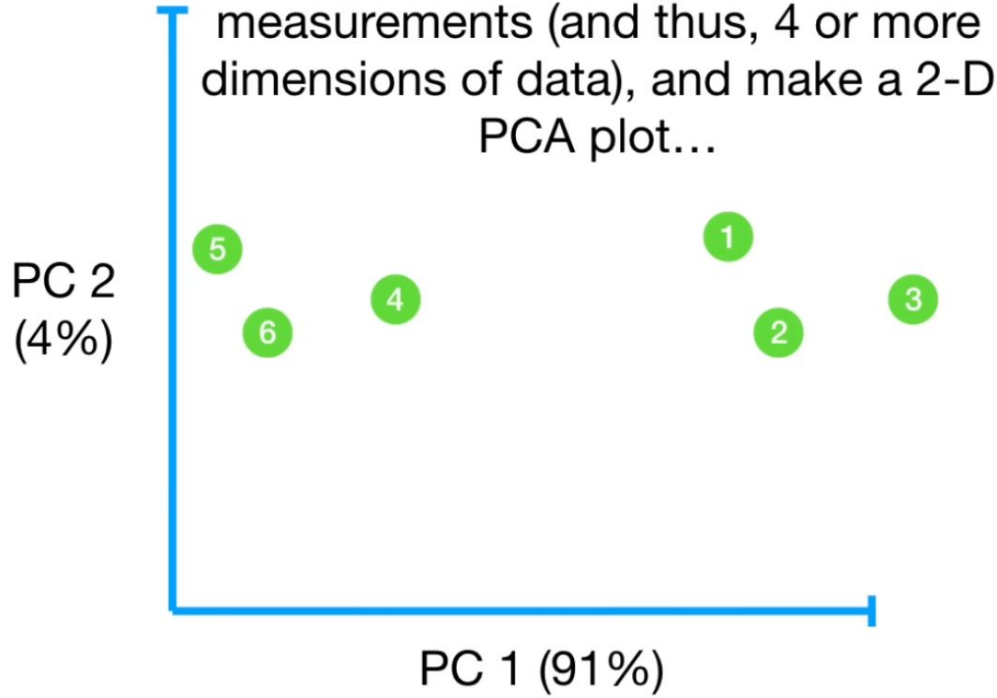
| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |
| **Gene 3** | 12 | 9 | 10 | 2.5 | 1.3 | 2 |
| **Gene 4** | 5 | 7 | 6 | 2 | 4 | 7 |

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |
| **Gene 3** | 12 | 9 | 10 | 2.5 | 1.3 | 2 |
| **Gene 4** | 5 | 7 | 6 | 2 | 4 | 7 |

If we measured 4 genes, however, we can no longer plot the data - 4 genes require 4 dimensions.

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |
| **Gene 3** | 12 | 9 | 10 | 2.5 | 1.3 | 2 |
| **Gene 4** | 5 | 7 | 6 | 2 | 4 | 7 |

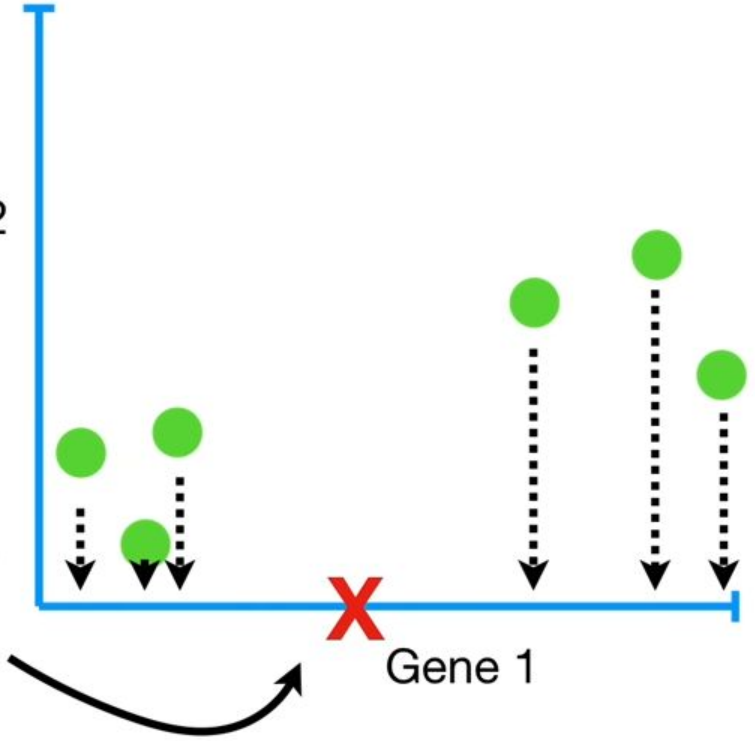
So we're going to talk about how PCA can take 4 or more gene measurements (and thus, 4 or more dimensions of data), and make a 2-D PCA plot…



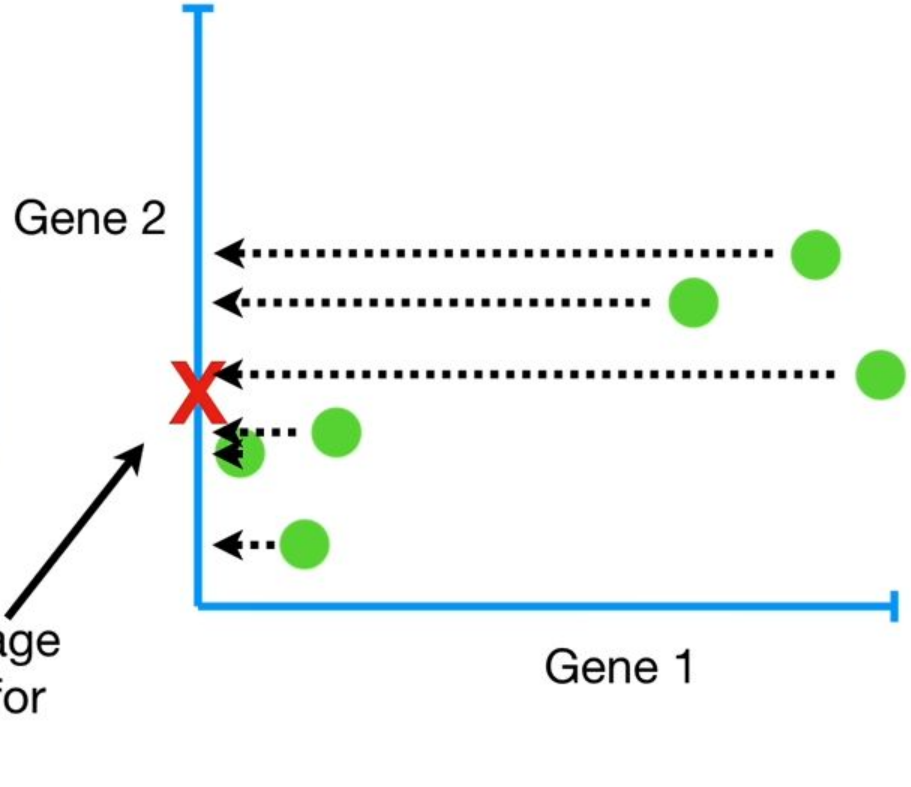
# Step 1: Center the data

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |

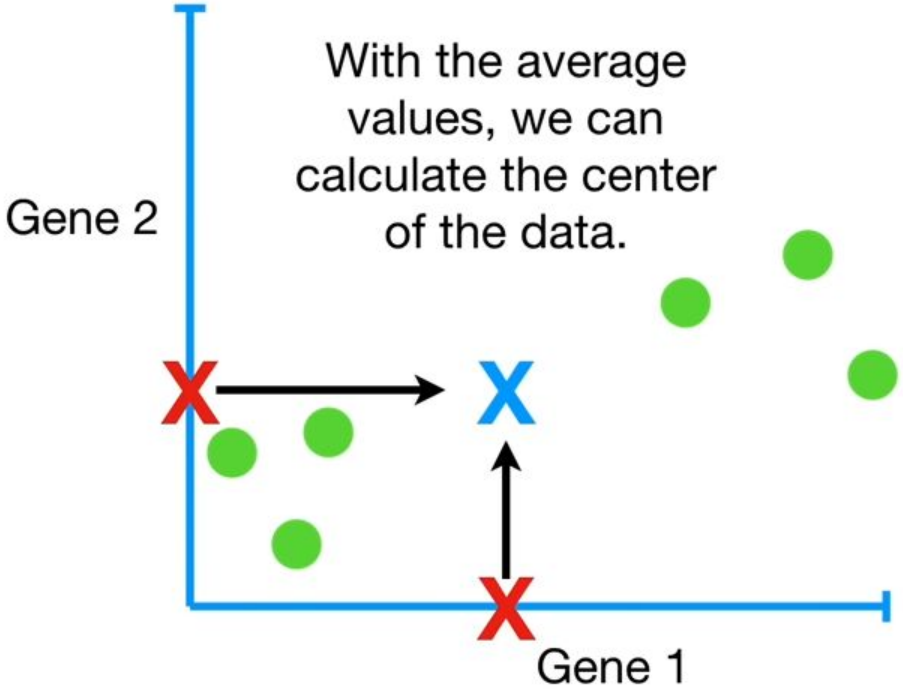
Then we'll calculate the average measurement for Gene 1…



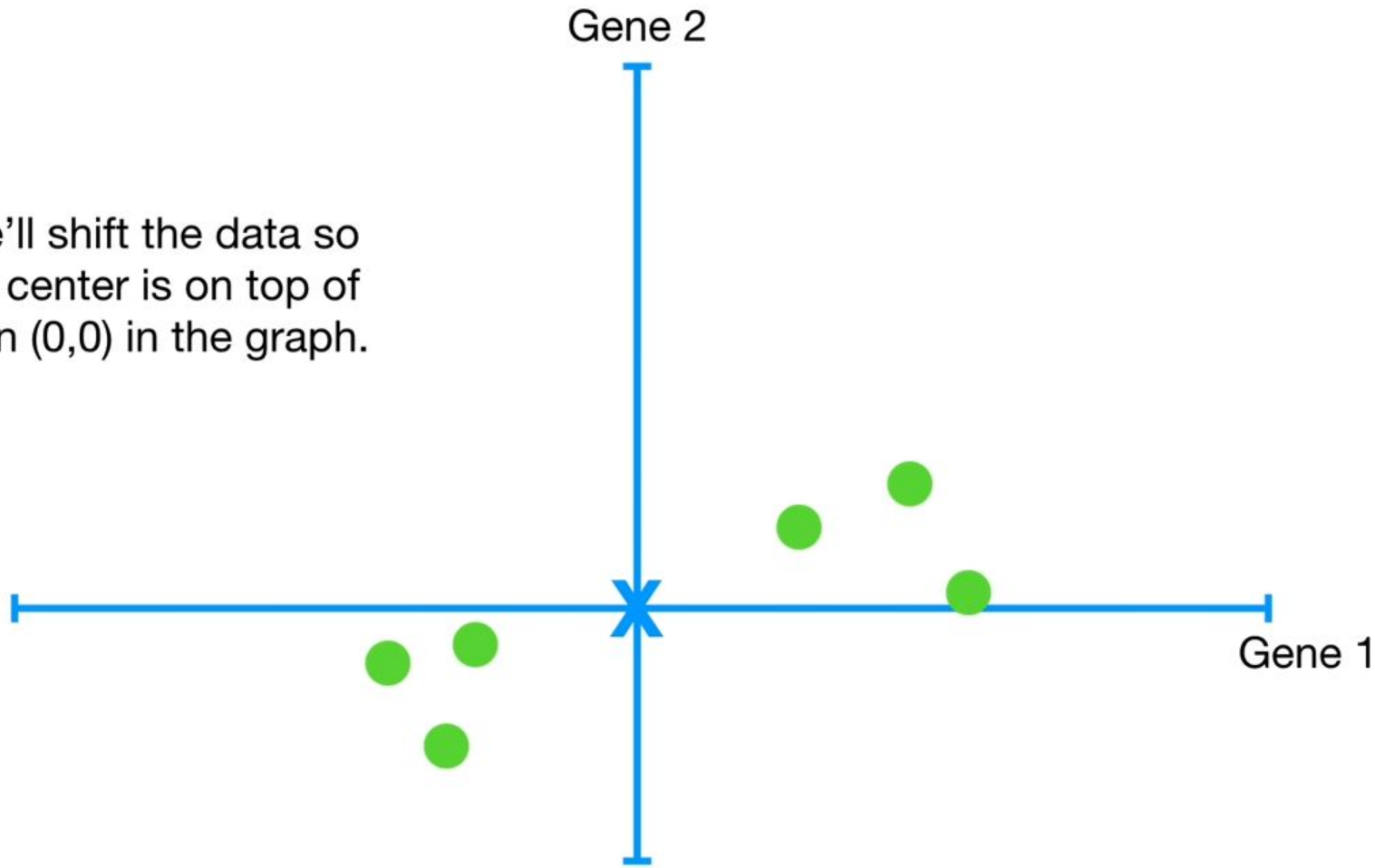
| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| Gene 1 | 10 | 11 | 8 | 3 | 2 | 1 |
| Gene 2 | 6 | 4 | 5 | 3 | 2.8 | 1 |



| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |

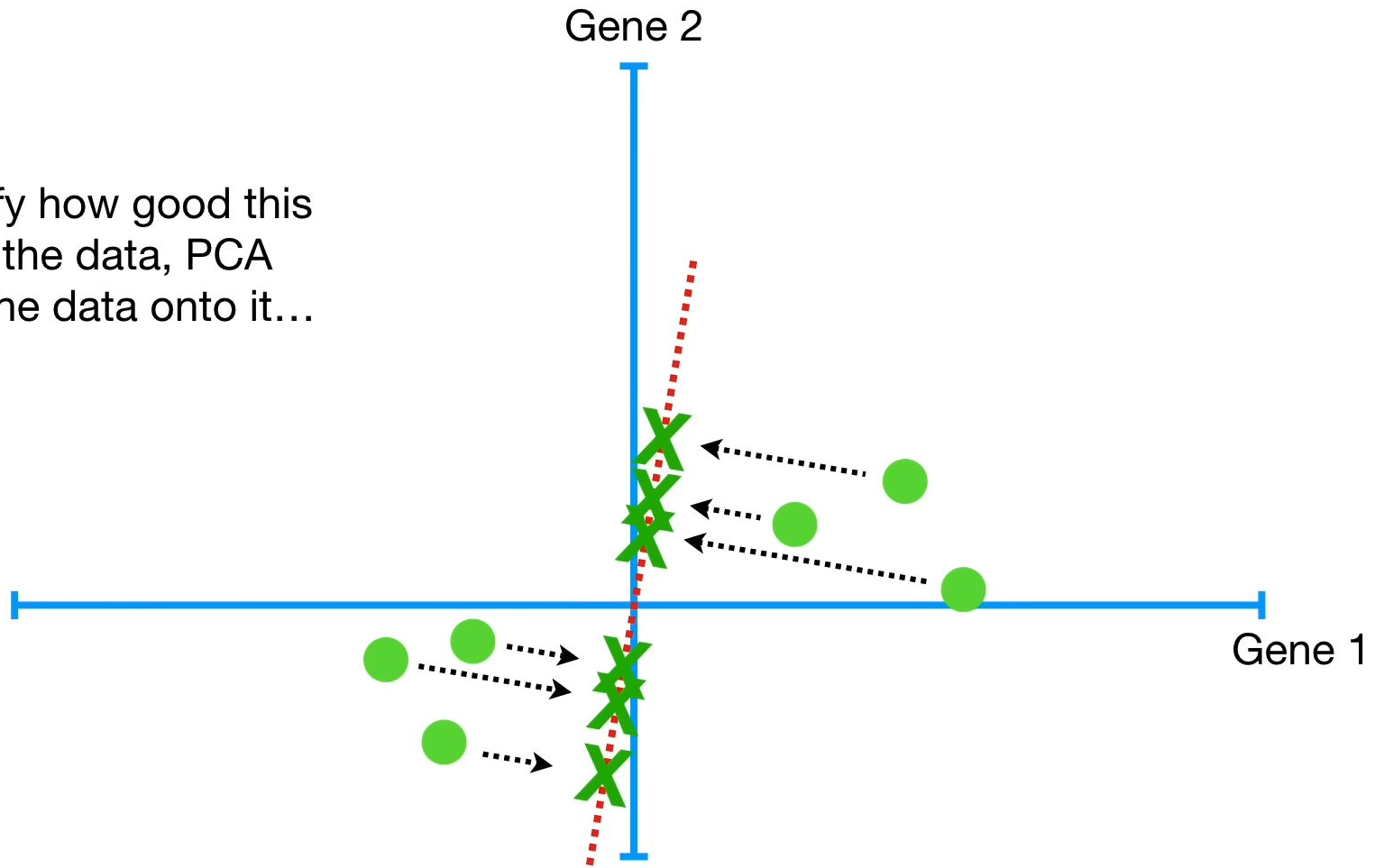


Now we'll shift the data so that the center is on top of the origin (0,0) in the graph.

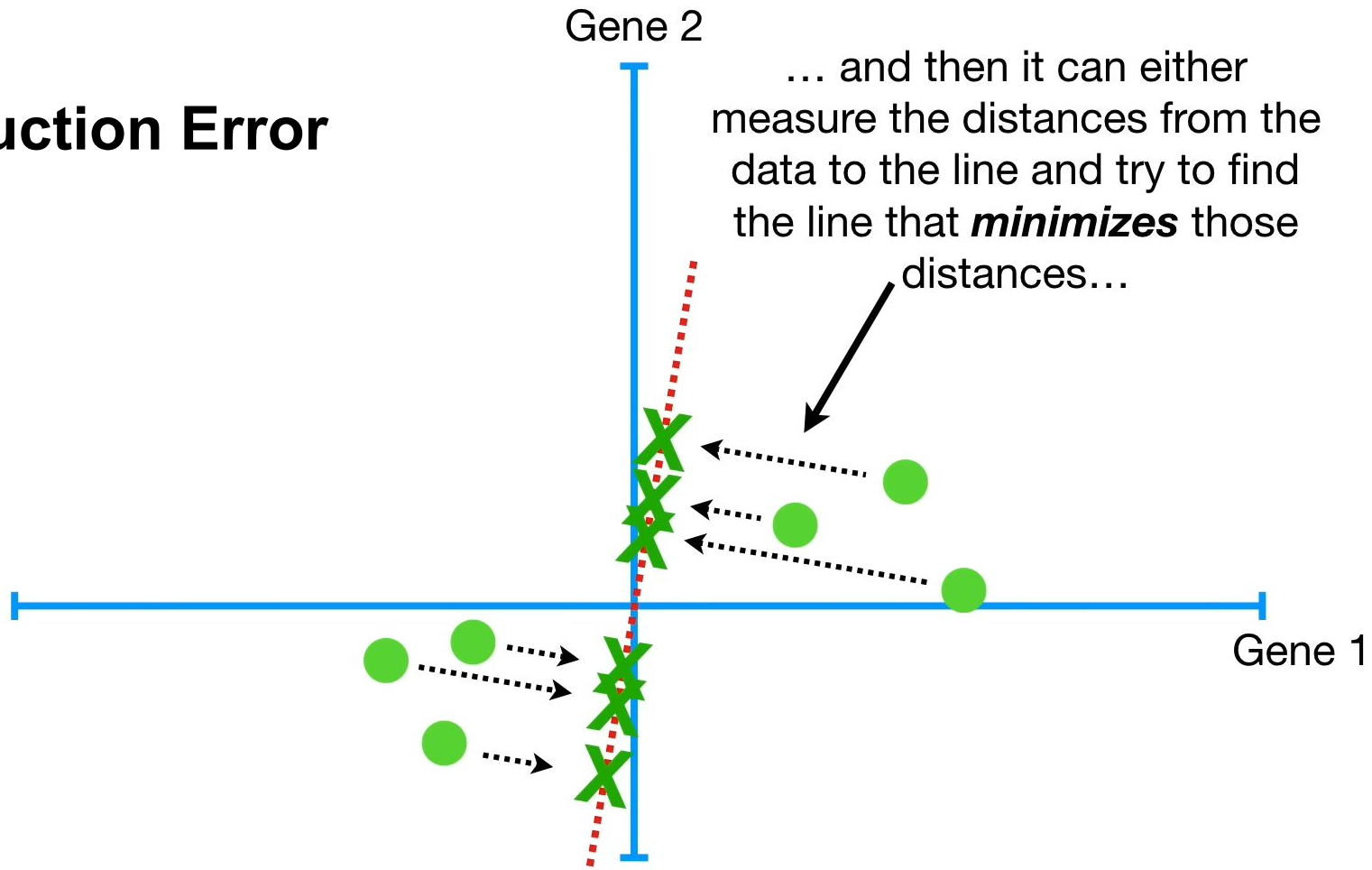


# Step 2: Find Principal Components

To quantify how good this line fits the data, PCA projects the data onto it…

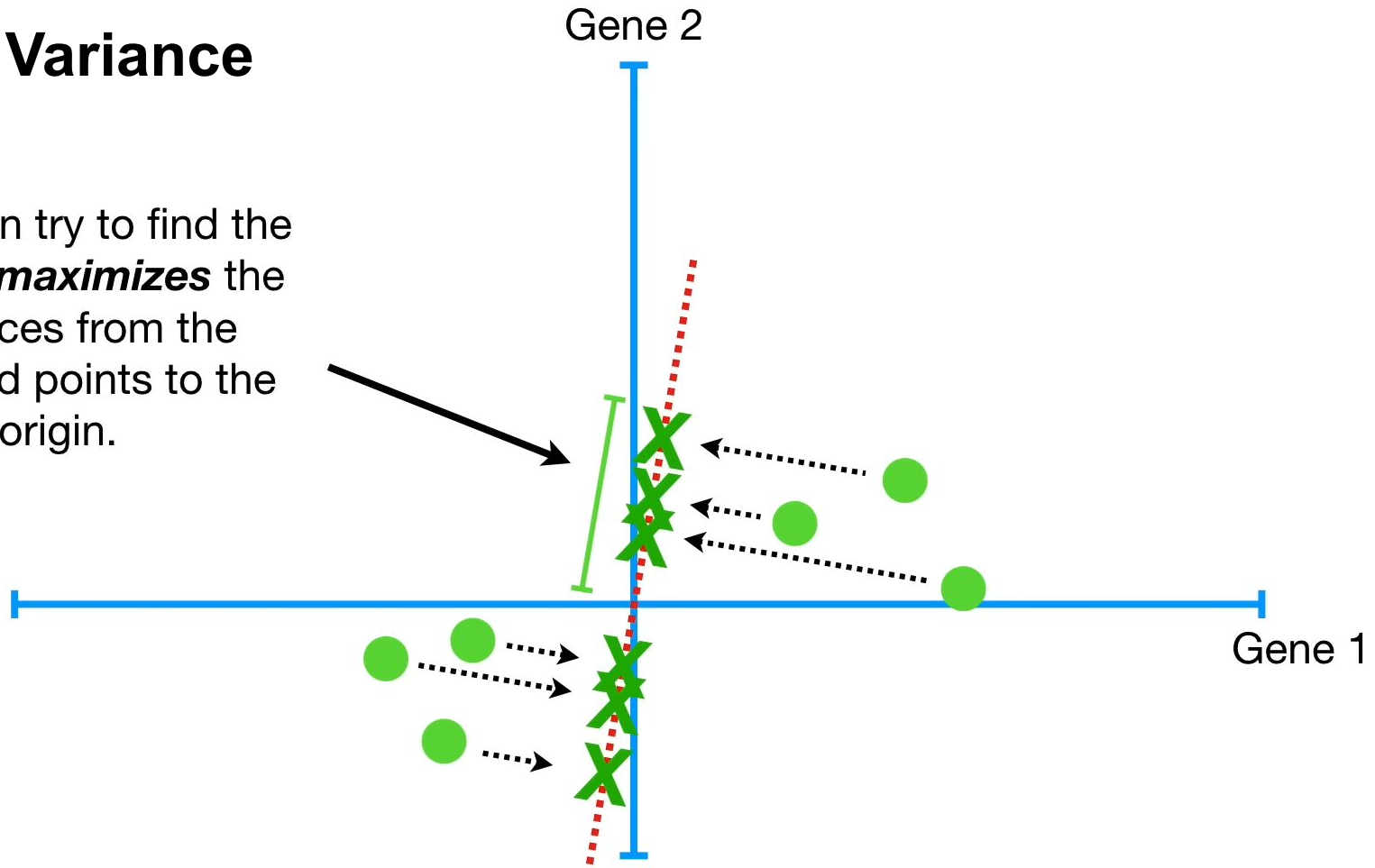


# Minimize Reconstruction Error

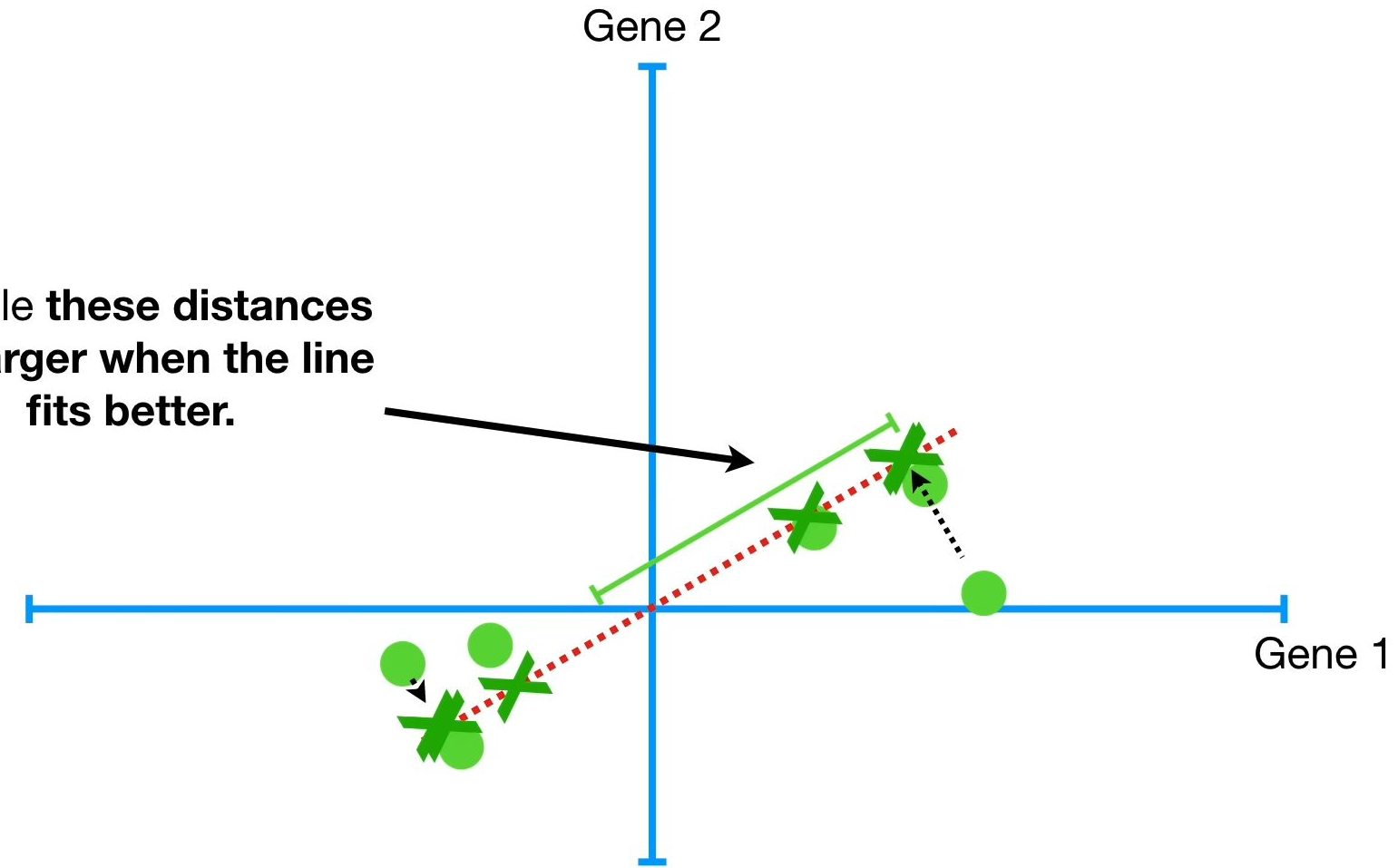


# Maximize Variance

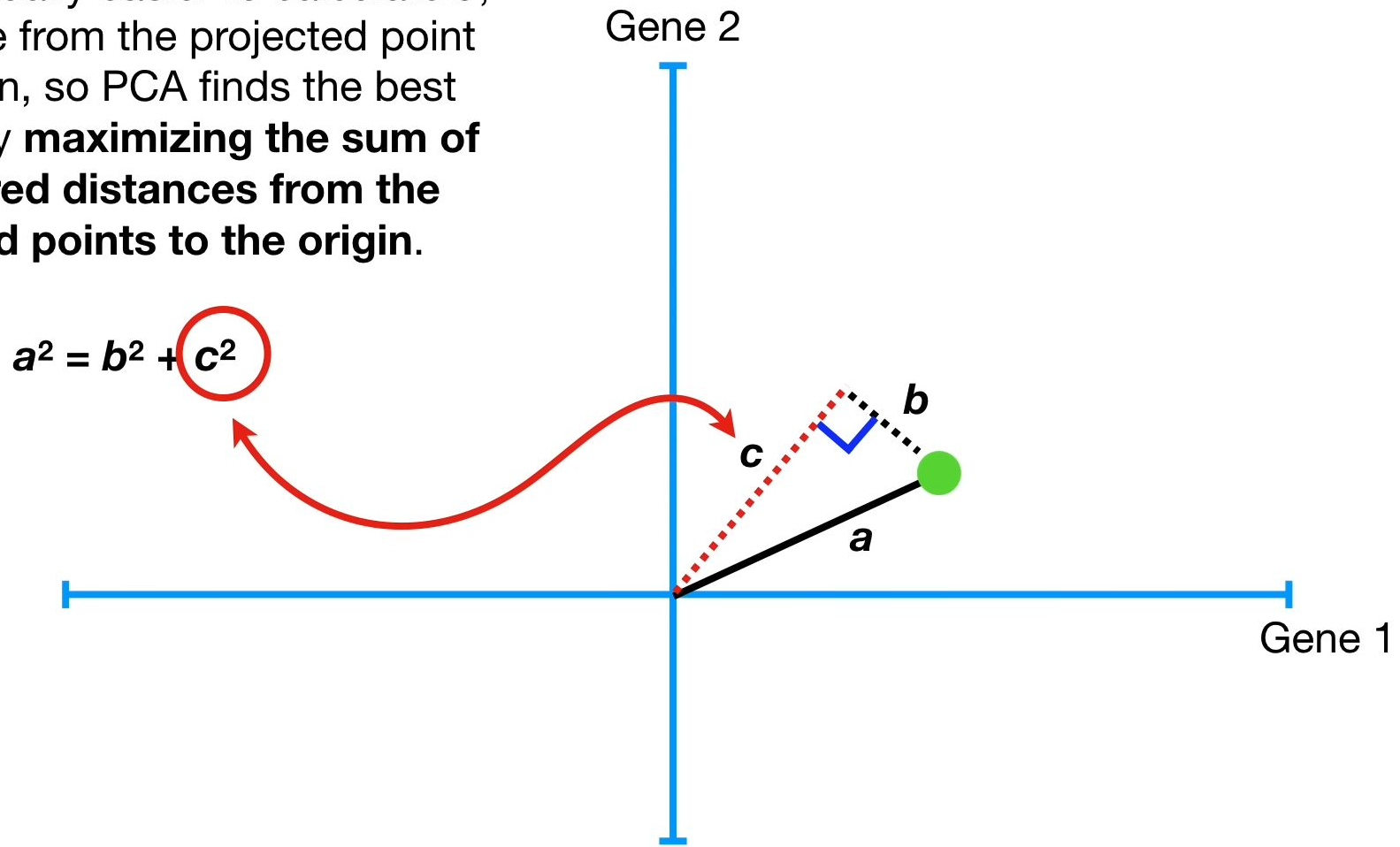
…or it can try to find the line that ***maximizes*** the distances from the projected points to the origin.



…while **these distances get larger when the line fits better.**

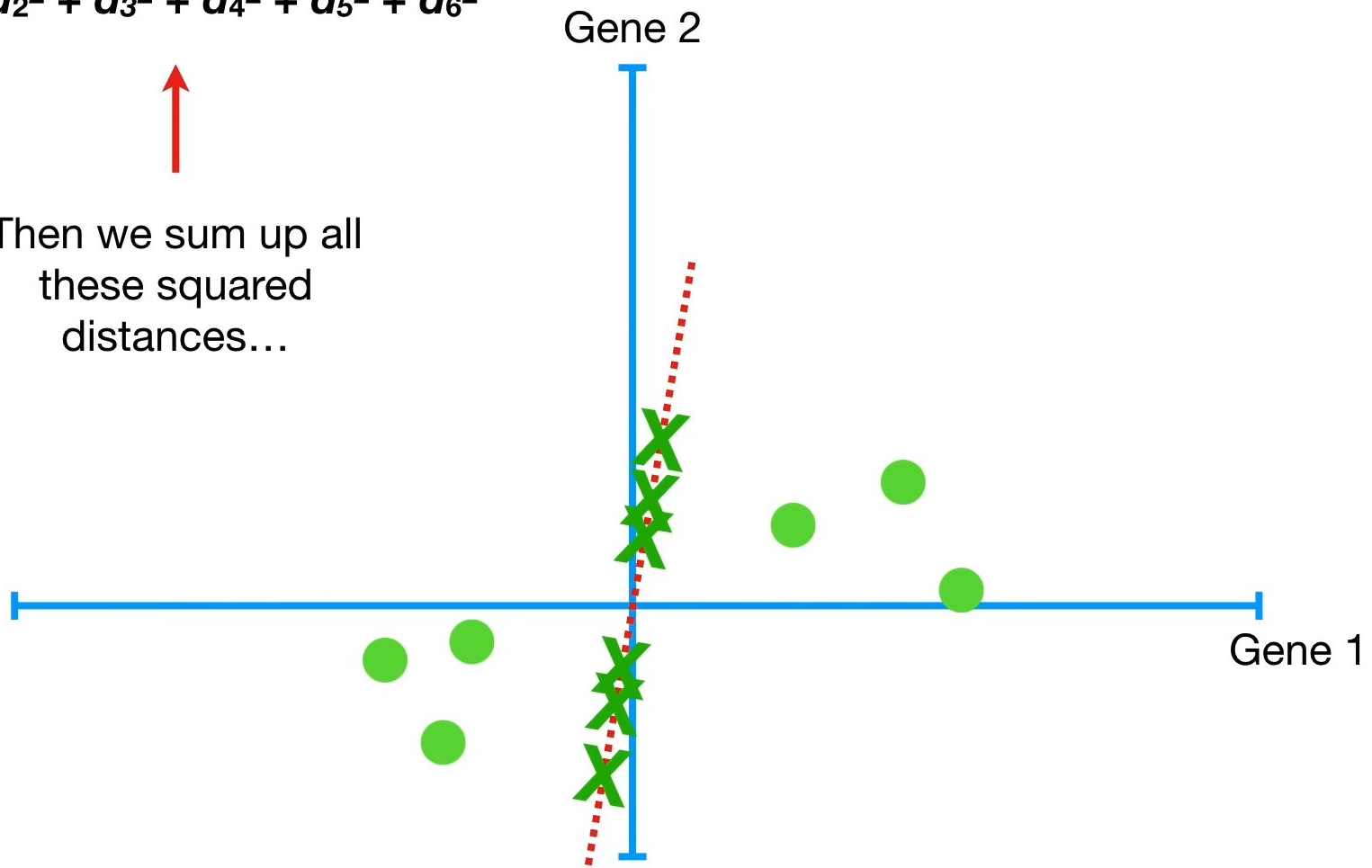


…but it's actually easier to calculate ***c***, the distance from the projected point to the origin, so PCA finds the best fitting line by **maximizing the sum of the squared distances from the projected points to the origin**.



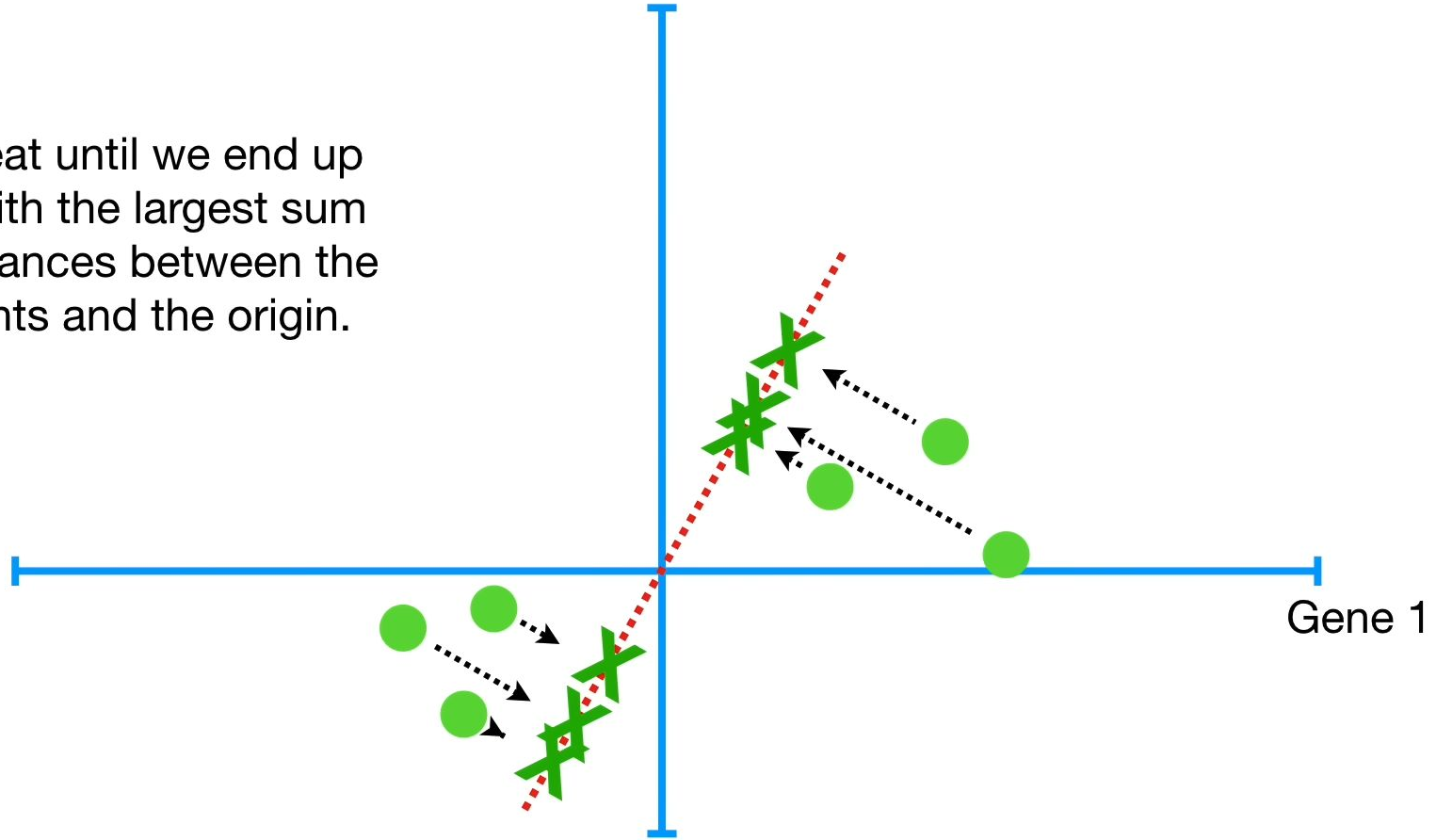
$d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2 + d_6^2$

Then we sum up all these squared distances…



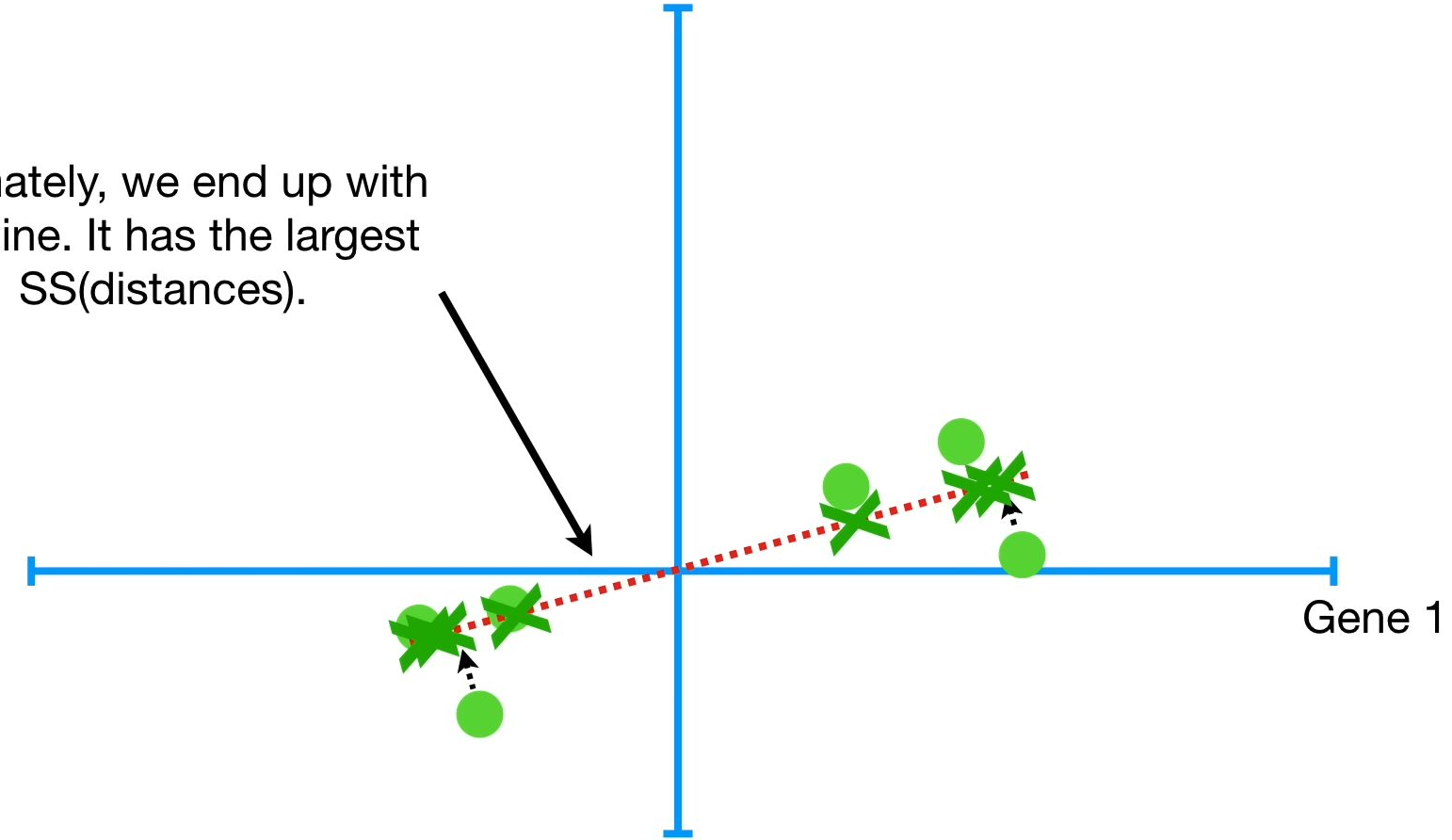
$d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2 + d_6^2$ = sum of squared distances = SS(distances)

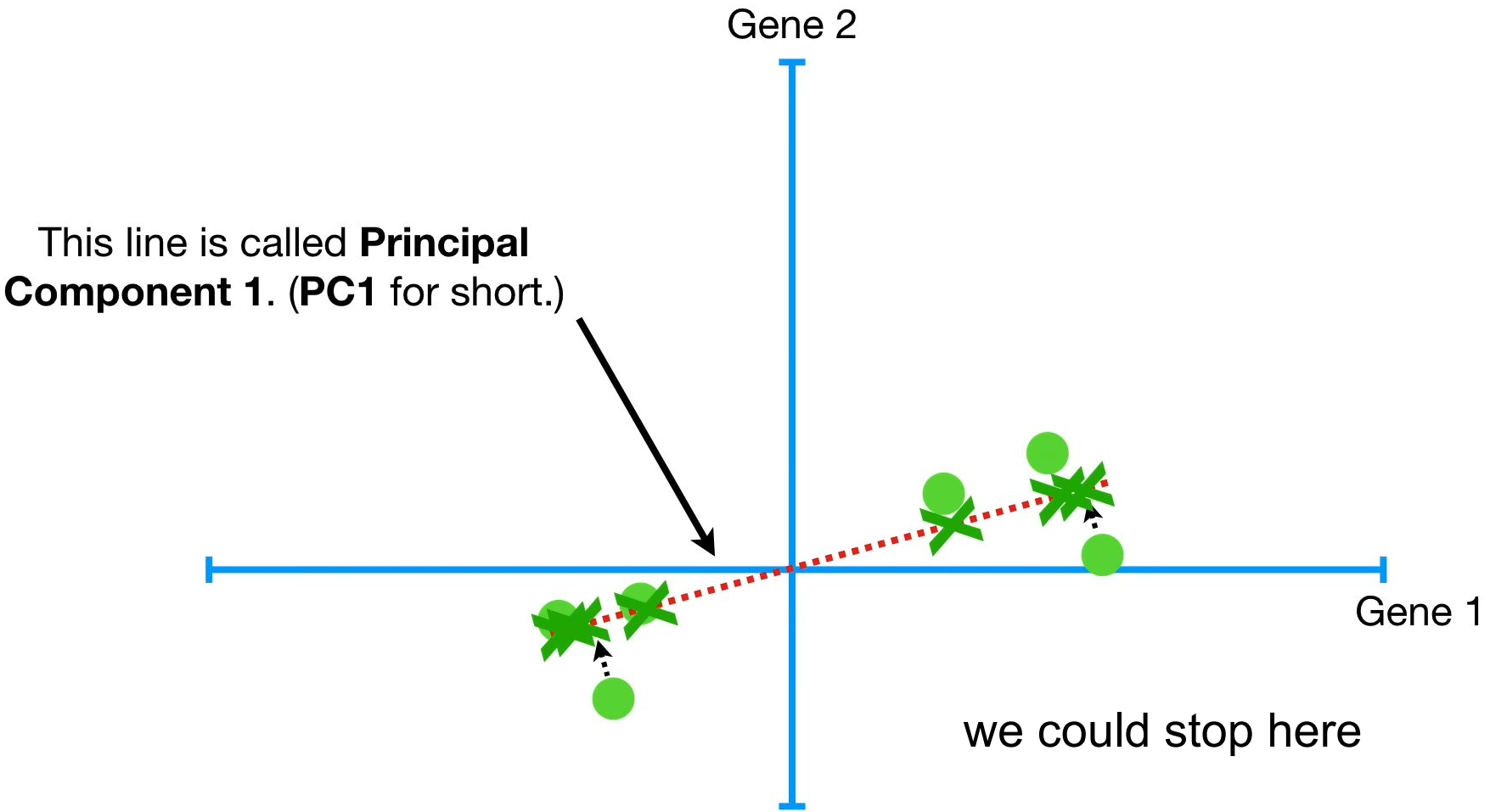
…and we repeat until we end up with the line with the largest sum of squared distances between the projected points and the origin.



$d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2 + d_6^2$ = sum of squared distances = SS(distances)

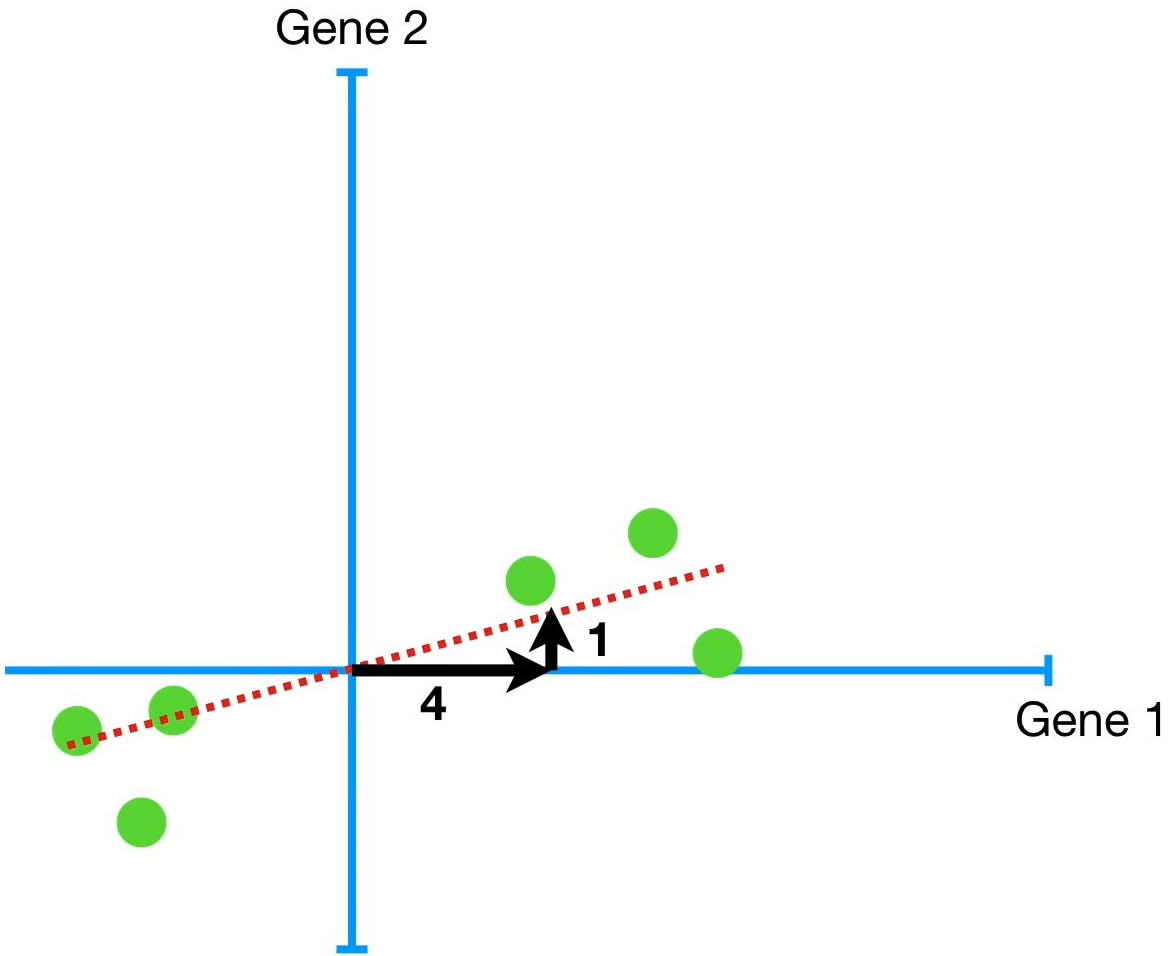
Ultimately, we end up with this line. It has the largest SS(distances).



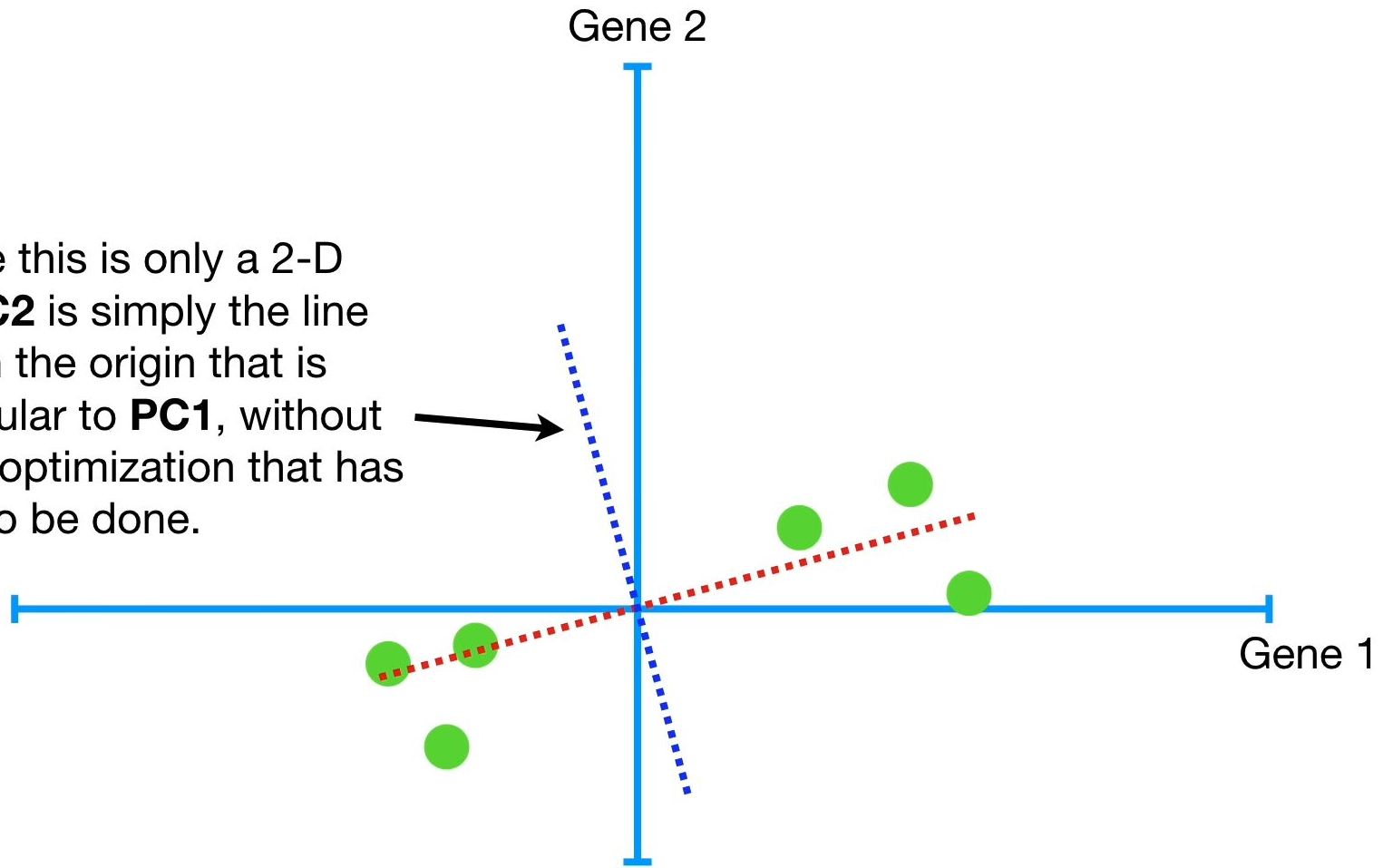

Gene 2
This line is called **Principal Component 1**. (**PC1** for short.)
Gene 1
we could stop here

**To make PC1**
Mix **4** parts Gene 1
with **1** part Gene 2

The ratio of Gene 1 to Gene 2 tells you that Gene 1 is more important when it comes to describing how the data are spread out..

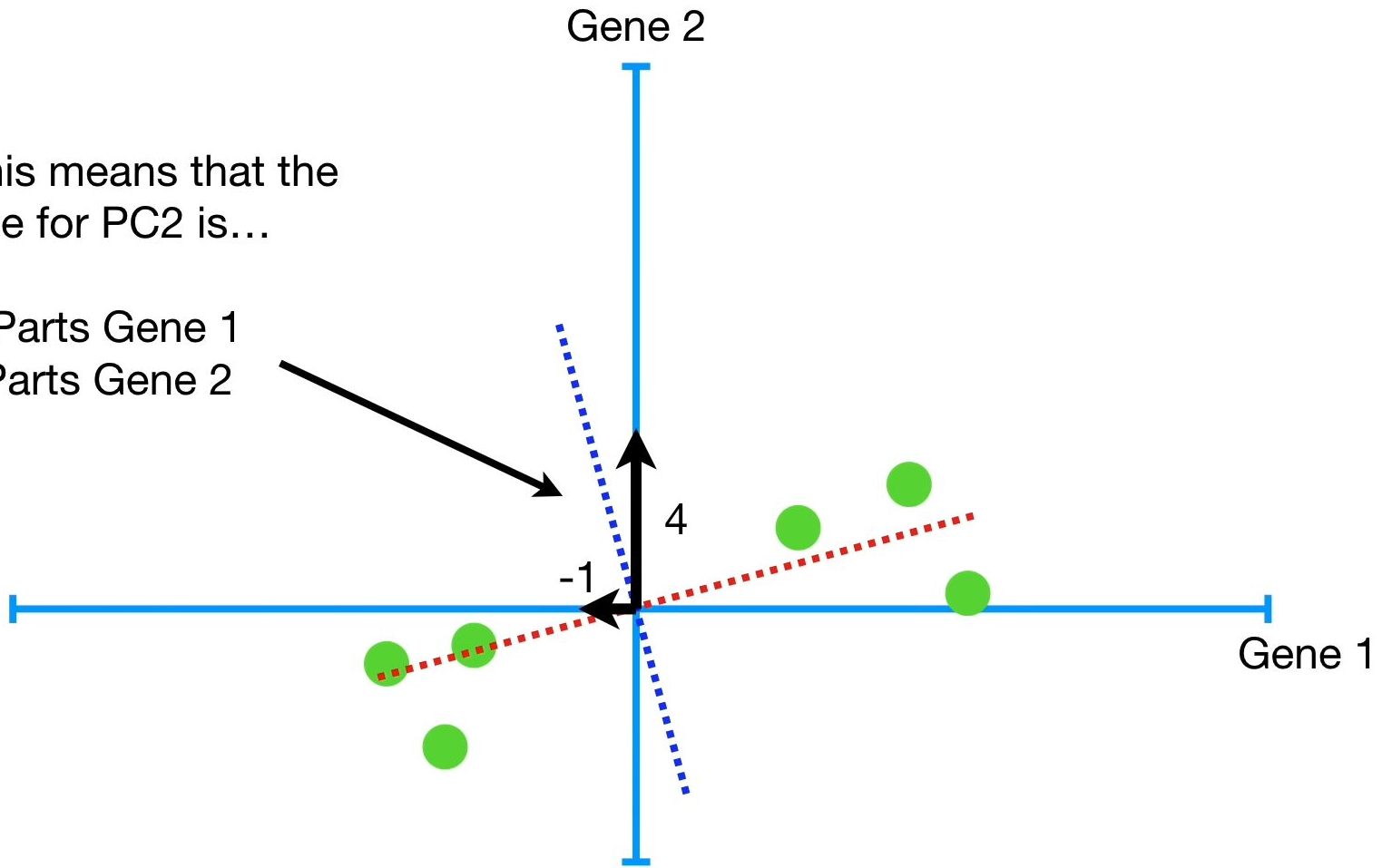


Because this is only a 2-D graph, **PC2** is simply the line through the origin that is perpendicular to **PC1**, without any further optimization that has to be done.



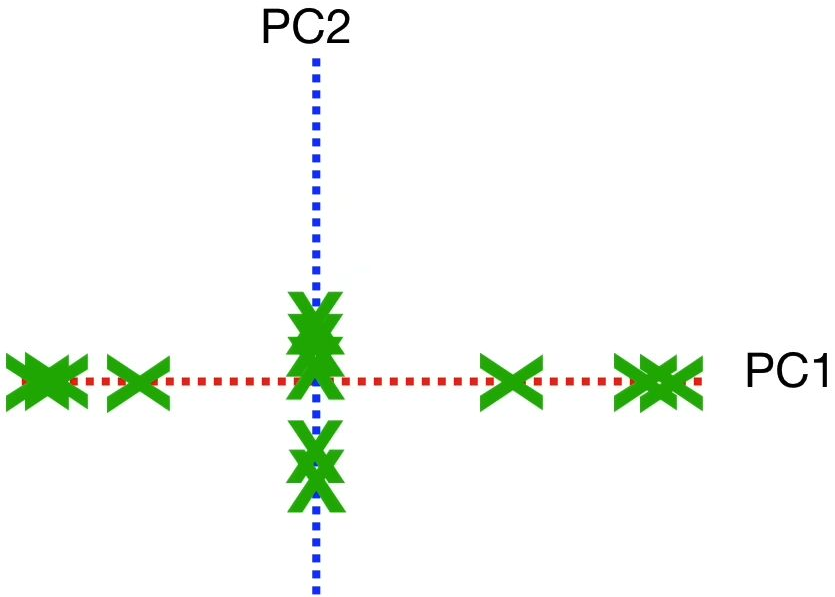
…and this means that the recipe for PC2 is…

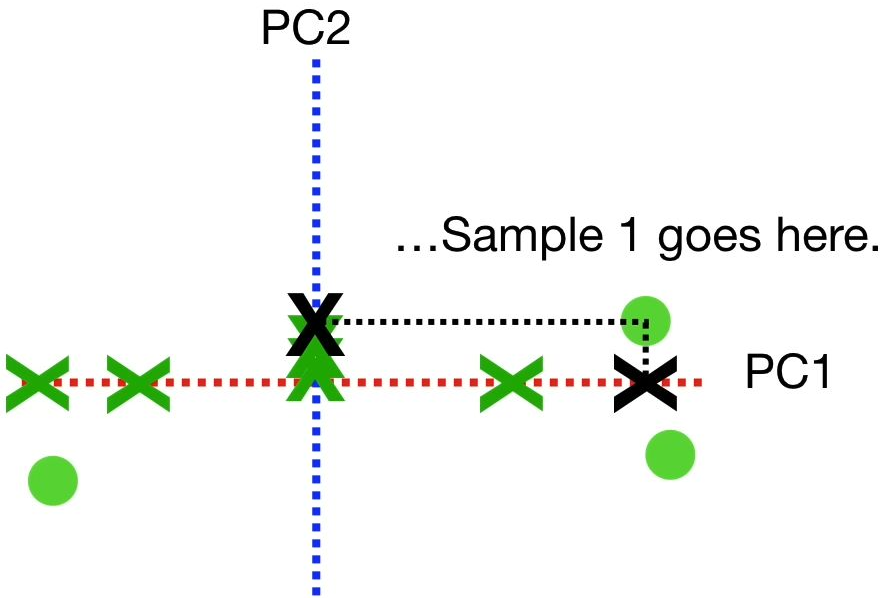
**-1** Parts Gene 1
**4** Parts Gene 2



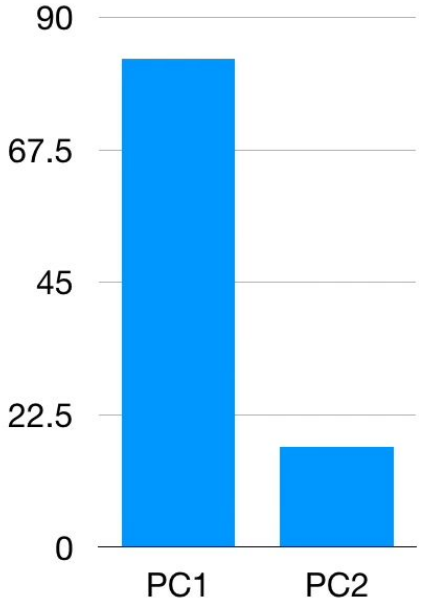
# Step 3: Change of Basis

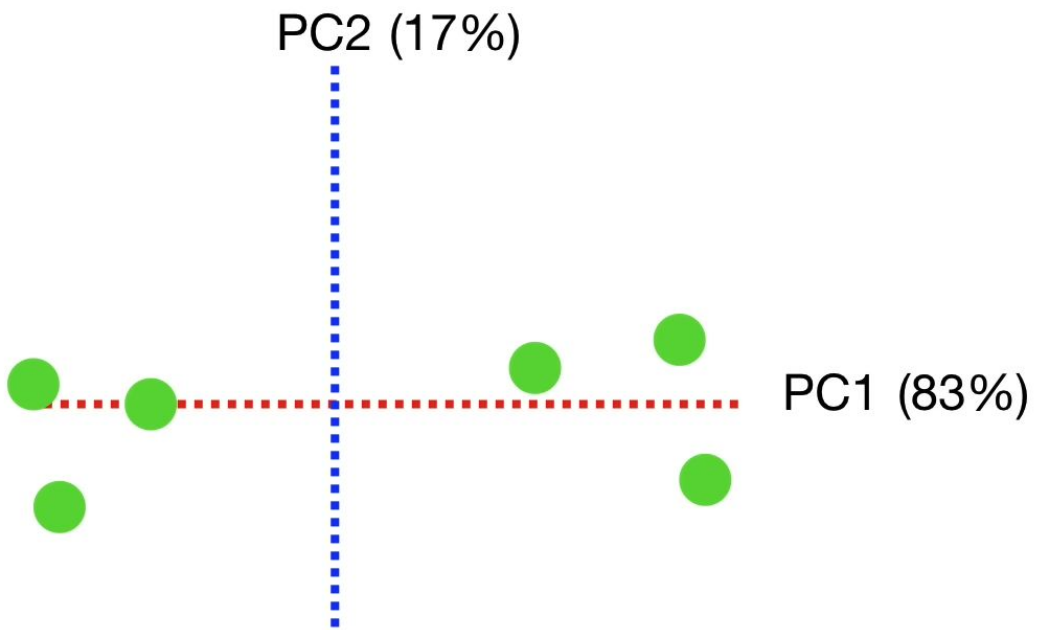
…then we use the projected points to find where the samples go in the PCA plot.




PC2
…Sample 1 goes here.
PC1

# Explained Variance

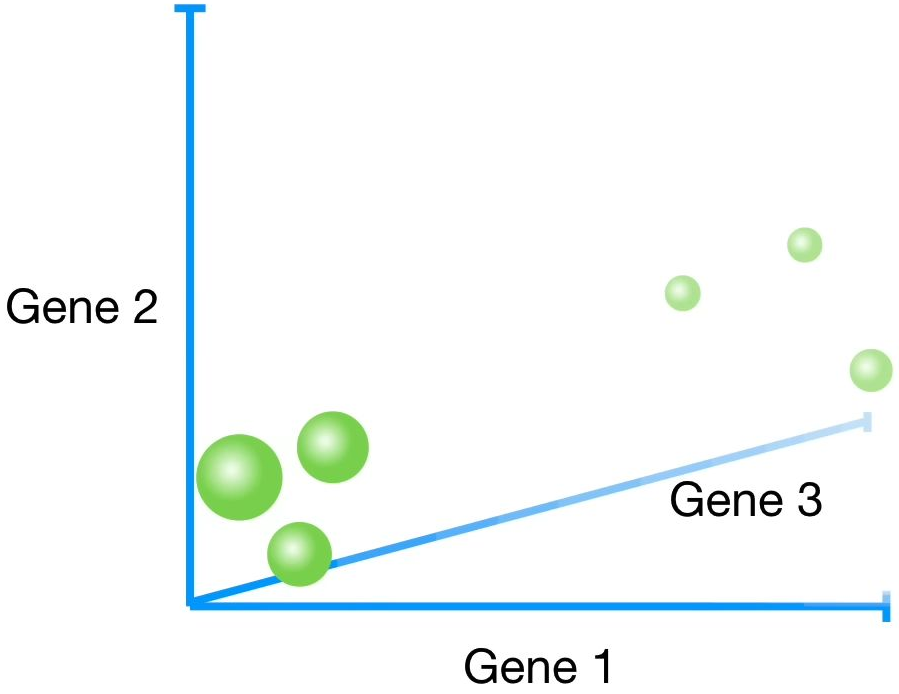


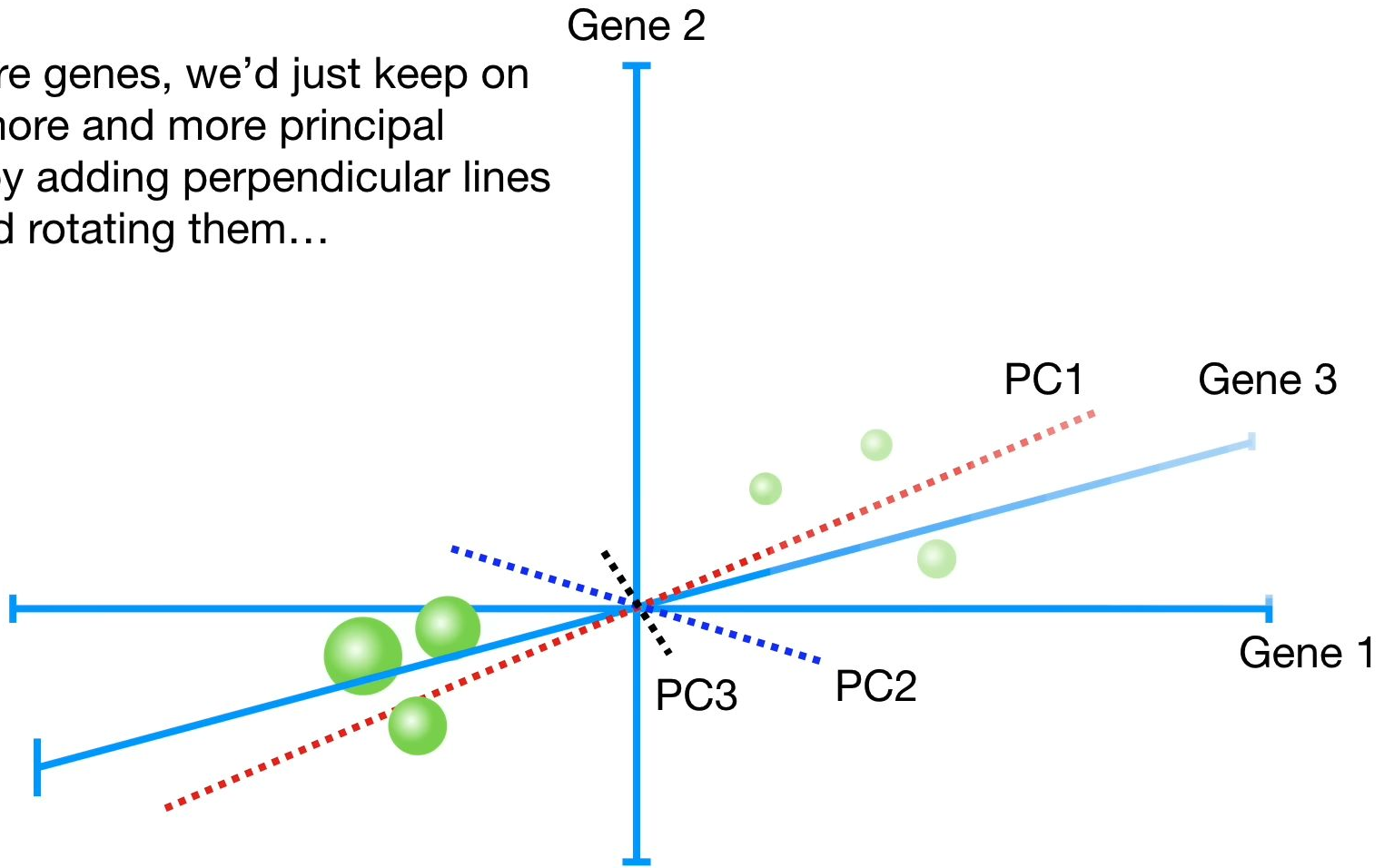
# Example w 3 Genes
# → 3 Dimensions

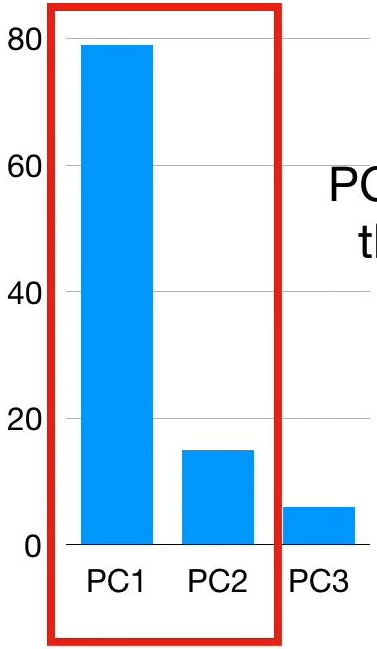
PCA with 3 variables (in this case, that means 3 genes) is pretty much the same as 2 variables…

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |
| **Gene 3** | 12 | 9 | 10 | 2.5 | 1.3 | 2 |

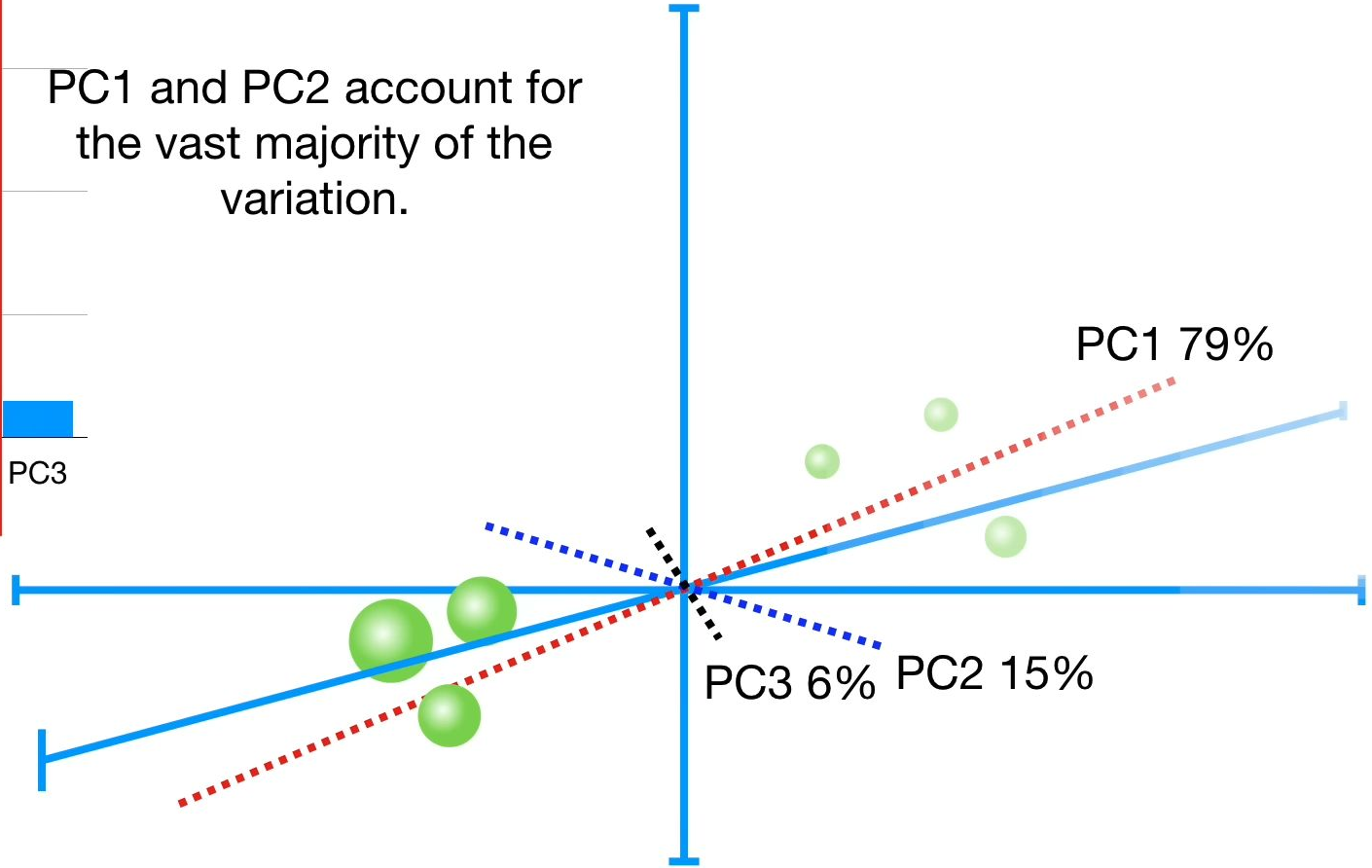


If we had more genes, we'd just keep on finding more and more principal components by adding perpendicular lines and rotating them…

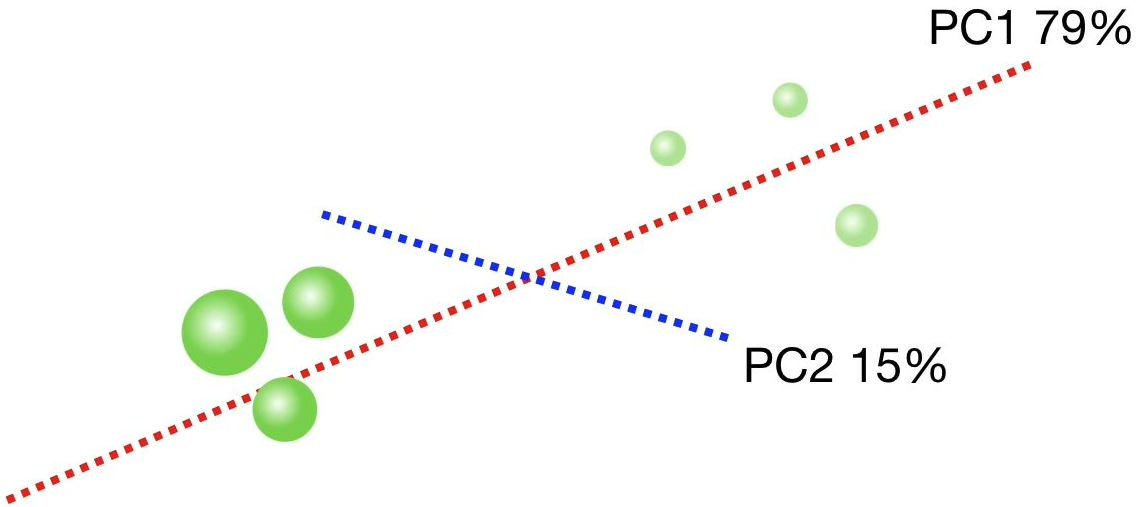


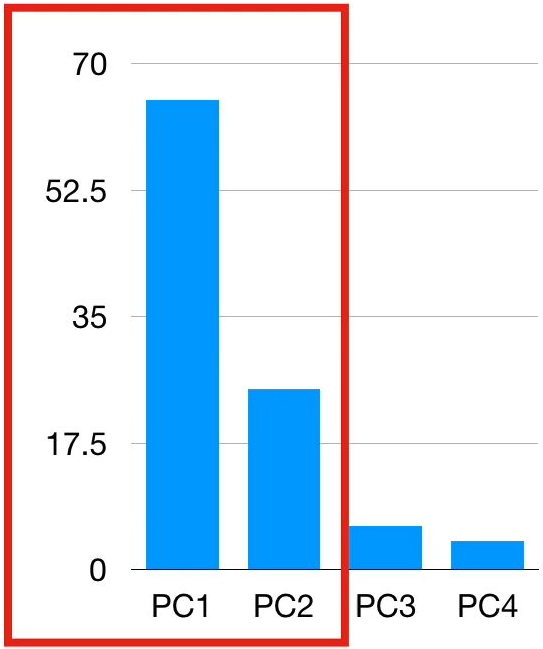



PC1 and PC2 account for the vast majority of the variation.

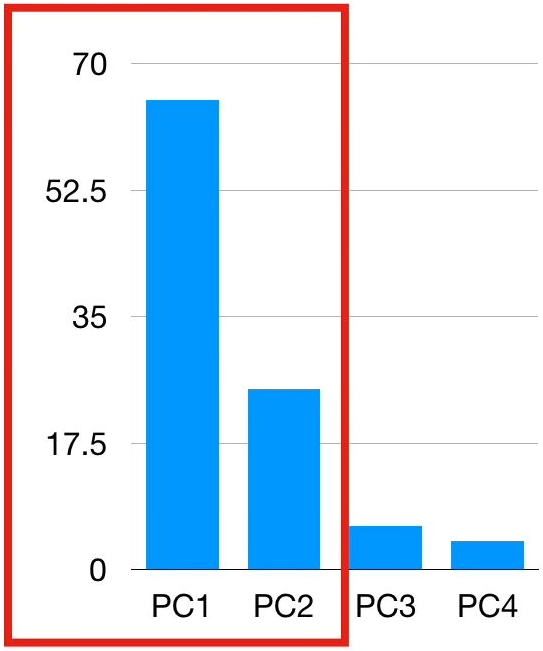


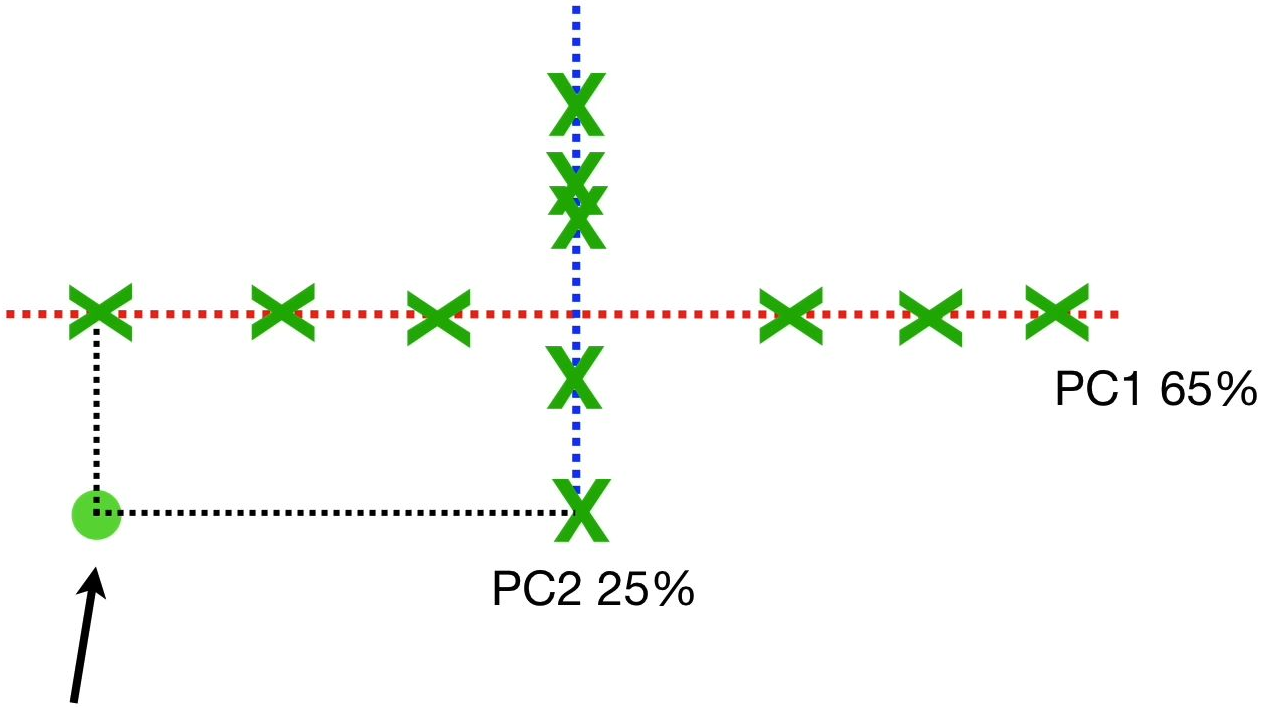
To convert the 3-D graph into a 2-D PCA graph, we just strip away everything but the data and PC1 and PC2…






…in this case, PC1 and PC2 account for 90% of the variation, so we can just use those to draw a 2-dimensional PCA graph.



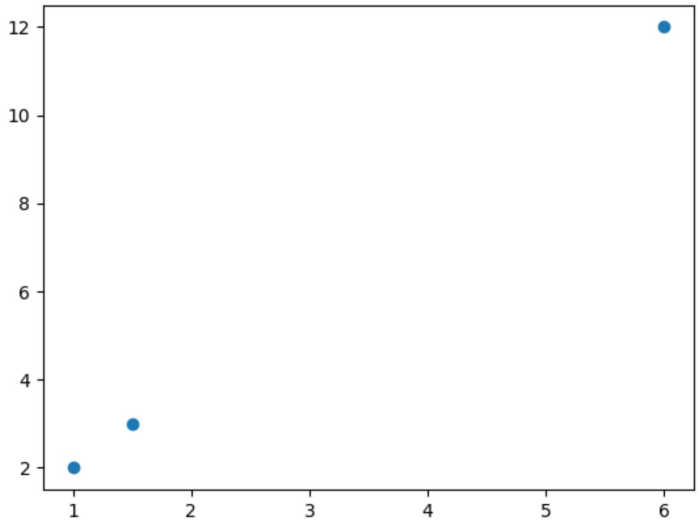




So Sample 2 goes here.

# 1. Principal Component

Consider the following dataset, which is represented as three points in $\mathbb{R}^2$. Note that in this problem we will **not** demean the dataset.

$$\begin{bmatrix} 1 & 2 \\ 1.5 & 3 \\ 6 & 12 \end{bmatrix}$$




(a) What is the first principal component vector, $v_1$?

## (a) What is the first principal component vector, $v_1$?

Each point has second coordinate twice the first, so every point is on the line $y = 2x$, or equivalently is a multiple of the vector $[1, 2]$.

That direction, normalized, is the first principal component, so $v_1 = [1/\sqrt{5}, 2/\sqrt{5}] \approx [0.45, 0.89]$.

## (b) What is the second principal component, $v_2$?

**Solution:**

Since every data is in the span of the first principal component, any unit norm vector perpendicular to $v_1$ is an acceptable choice. One such vector is $[-2/\sqrt{5}, 1/\sqrt{5}] \approx [-0.89, 0.45]$.

(c) If we use only the first principal component to compress the dataset, what will the representation of each point be?

**Solution:**

The first point is $\sqrt{5}v_1$, the second one is $1.5 \cdot \sqrt{5}v_1$, and the third one is $6 \cdot \sqrt{5}v_1$.

(d) Will this representation be lossy, or perfectly preserve the dataset?

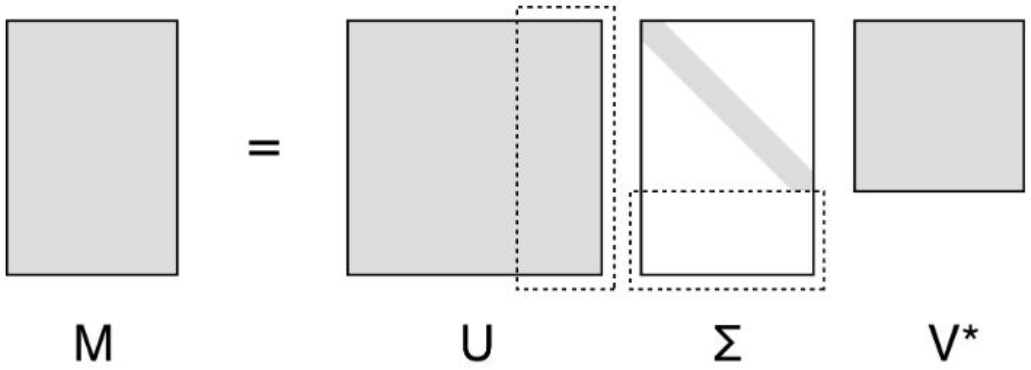
**Solution:**

In this particular dataset, we perfectly preserve this dataset (the points are all multiples of $v_1$).

# Singular Value Decomposition (SVD)

# Singular Value Decomposition

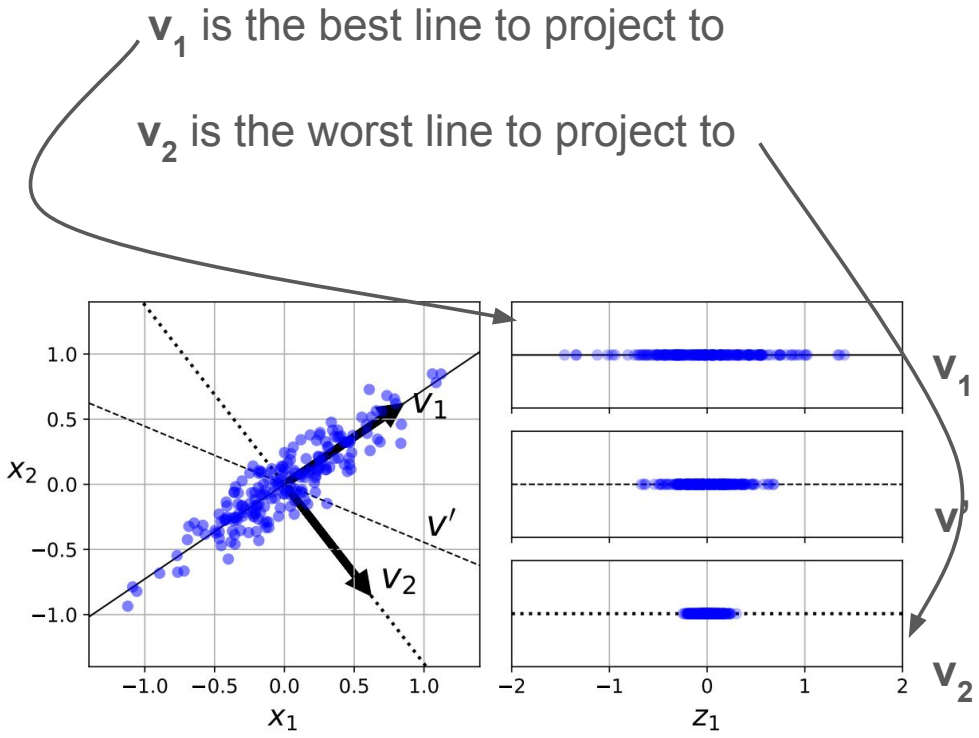
**Fact:** Any matrix **M** can be written as the multiplication three separate matrices



# Relating PCA & SVD

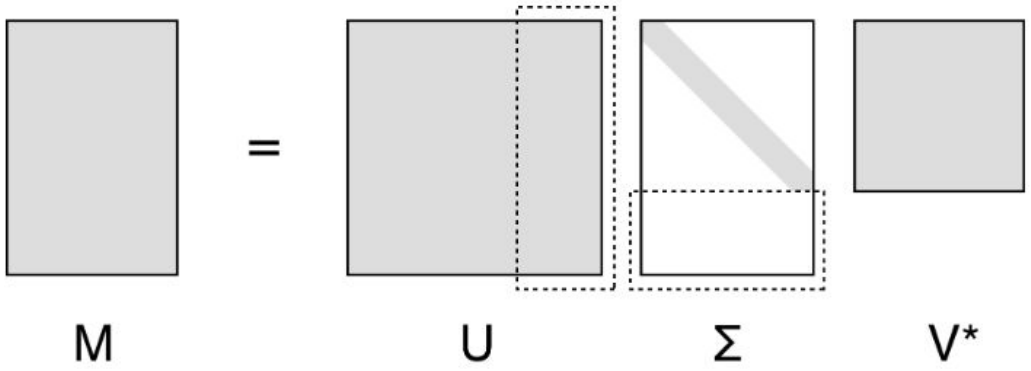
- PCA finds the **directions of greatest variance**
- Same as the **eigenvectors** of the **covariance matrix**

**So how do we find the eigenvectors of $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ ?**



# Singular Value Decomposition

**Fact:** Any matrix **M** can be written as the multiplication three separate matrices




SVD will help us find **V!**

**The matrix with PCs.**

# Rank-r approximations

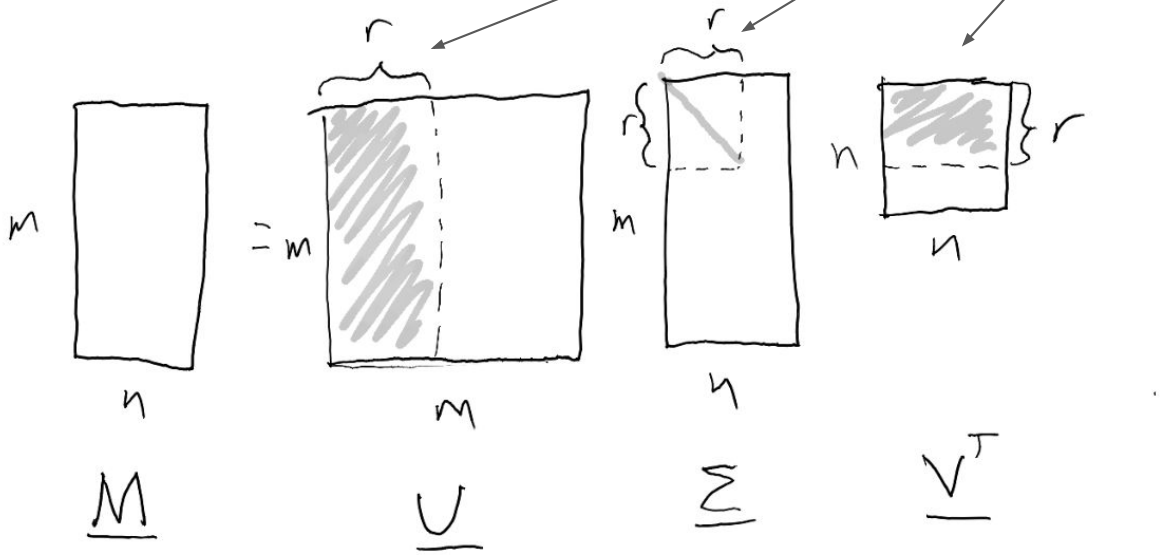
Say that **matrix X** is **“bloated”**

- many **linearly dependent vectors** increase its size
- the “information” of **M** lives the subspace spanned by its linearly independent vectors

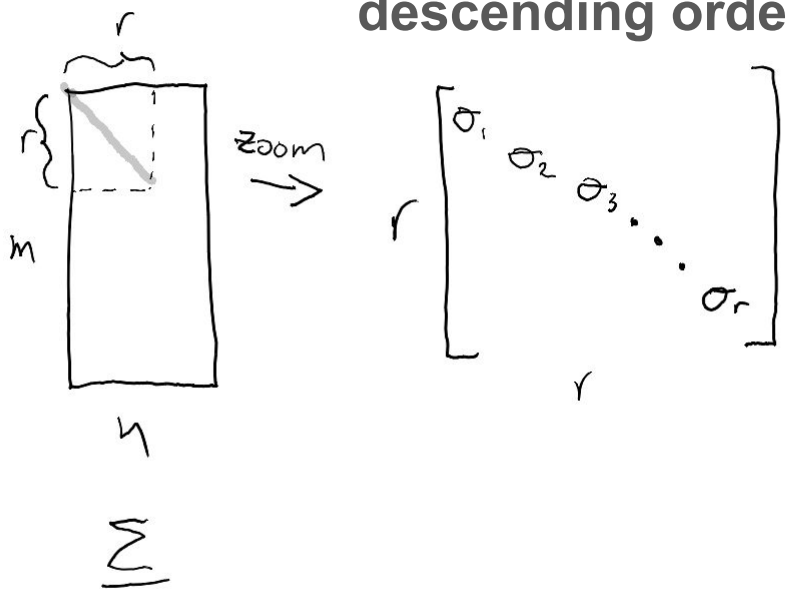
| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| **Gene 1** | 10 | 11 | 8 | 3 | 2 | 1 |
| **Gene 2** | 6 | 4 | 5 | 3 | 2.8 | 1 |
| **Gene 3** | 12 | 9 | 10 | 2.5 | 1.3 | 2 |
| **Gene 4** | 5 | 7 | 6 | 2 | 4 | 7 |

# Rank-r approximations

- The optimal rank-$r$ approximation is $L = U_{1:r} S_{1:r,1:r} V_{1:r}^T$



# Understanding the decomposition




$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i=1}^{r} u_i \sigma_i v_i^\top$$

# Understanding the decomposition

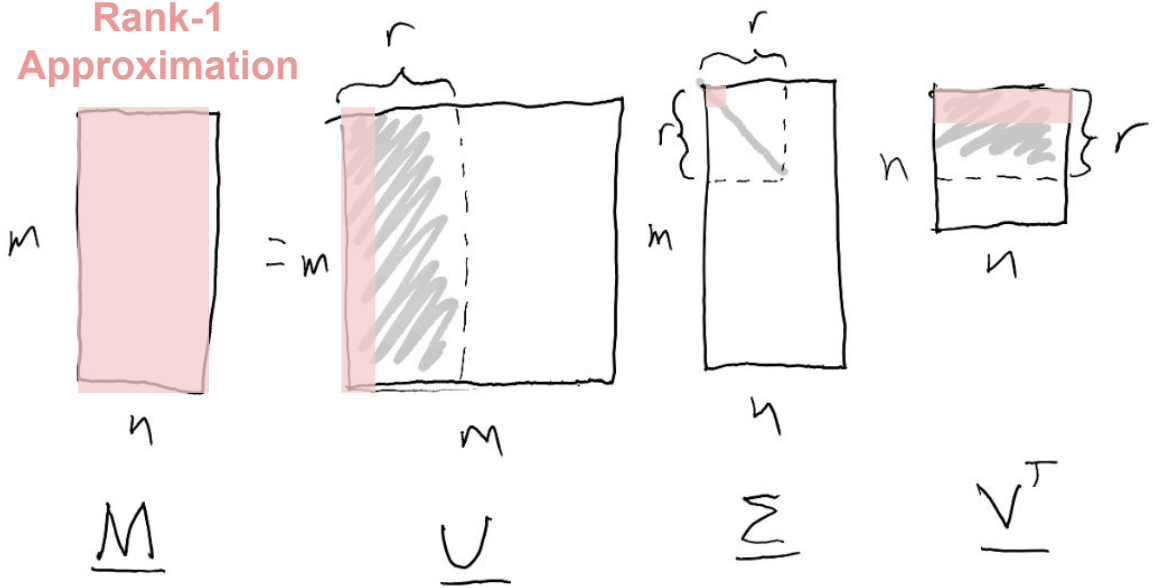
$$\mathbf{M} \in \mathbb{R}^{m \times n}$$

$$u_i \in \mathbb{R}^m;\ \sigma_i \in \mathbb{R};\ v_i \in \mathbb{R}^n \therefore$$

$$u_i \sigma_i v_i^\top \in \mathbb{R}^{m \times n}$$

**Analogy:**
Coats of paint

Rank-1 Approximation




$$\mathbf{U\Sigma V}^\top = u_1\sigma_1 v_1^\top + u_2\sigma_2 v_2^\top + \ldots + u_r\sigma_r v_r^\top$$

# Understanding the decomposition

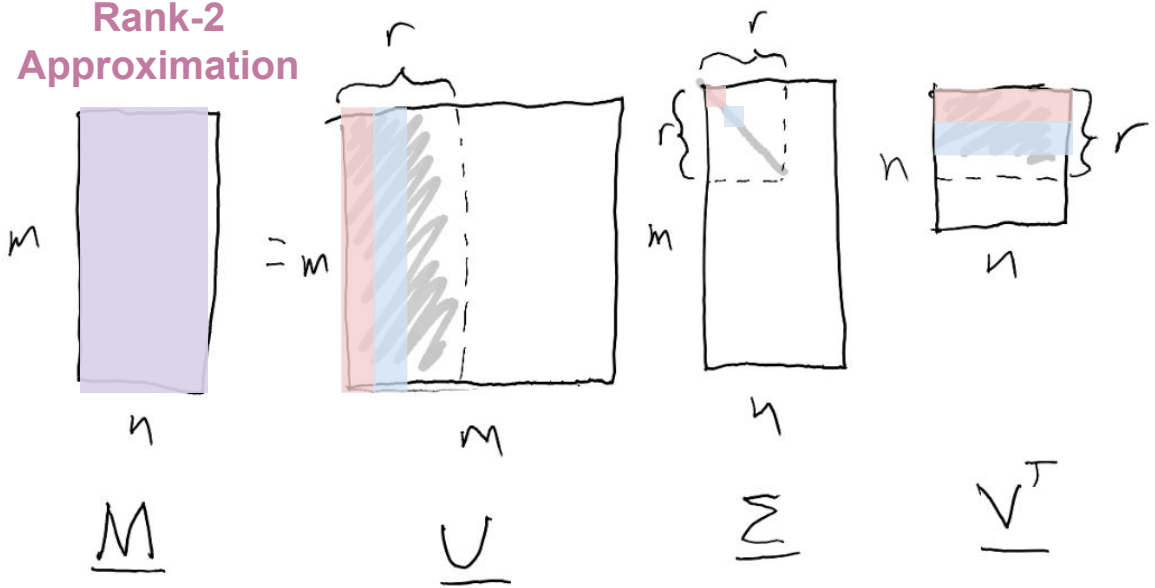
$$\mathbf{M} \in \mathbb{R}^{m \times n}$$

$$u_i \in \mathbb{R}^m;\ \sigma_i \in \mathbb{R};\ v_i \in \mathbb{R}^n \therefore$$

$$u_i \sigma_i v_i^\top \in \mathbb{R}^{m \times n}$$

**Analogy:**
Coats of paint

Rank-2 Approximation




$$\mathbf{U\Sigma V}^\top = u_1\sigma_1 v_1^\top + u_2\sigma_2 v_2^\top + \ldots + u_r\sigma_r v_r^\top$$

# Understanding the decomposition

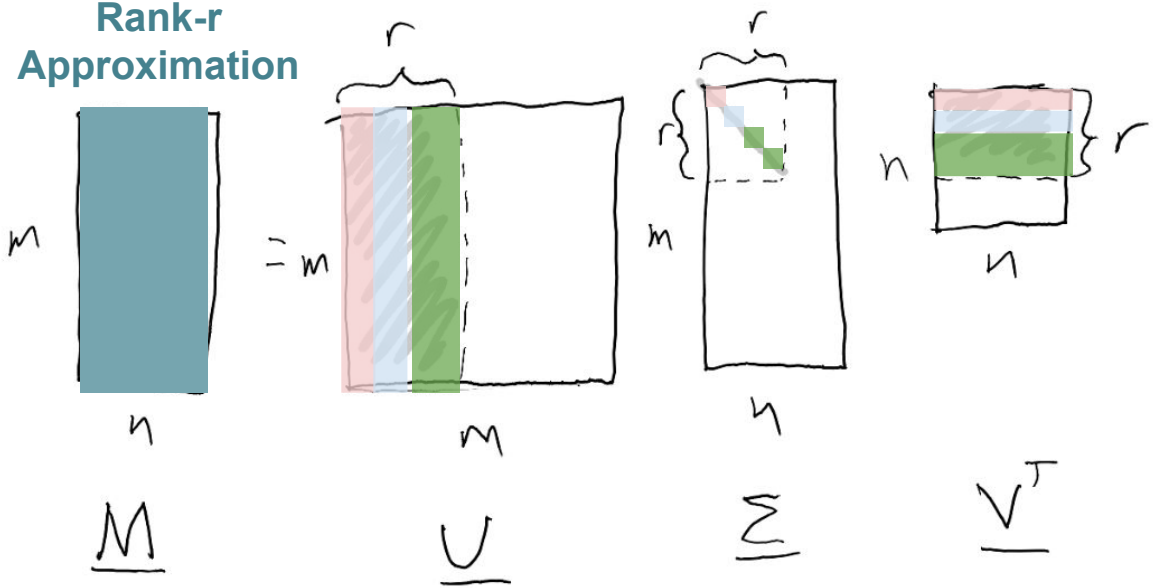
$$\mathbf{M} \in \mathbb{R}^{m \times n}$$

$$u_i \in \mathbb{R}^m;\ \sigma_i \in \mathbb{R};\ v_i \in \mathbb{R}^n \therefore$$

$$u_i \sigma_i v_i^\top \in \mathbb{R}^{m \times n}$$

**Analogy:**
Coats of paint

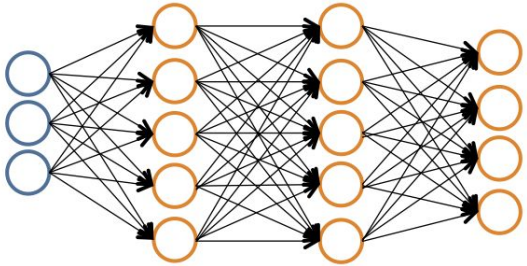
Rank-r Approximation




$$\mathbf{U\Sigma V}^\top = u_1\sigma_1 v_1^\top + u_2\sigma_2 v_2^\top + \ldots + u_r\sigma_r v_r^\top$$

# Convolutional Neural Networks (CNNs)

# Neural Networks

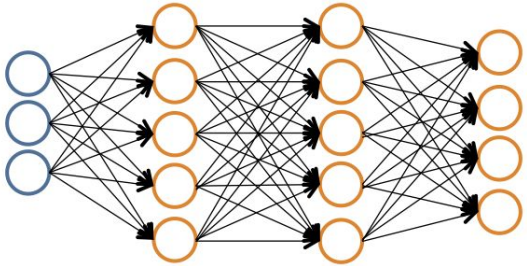


So far, we've looked at neural networks that look like this:

for the XOR problem and on the MNIST dataset (HW3).

- Both of these are pretty simple problems and our models were fairly small.

# Neural Networks



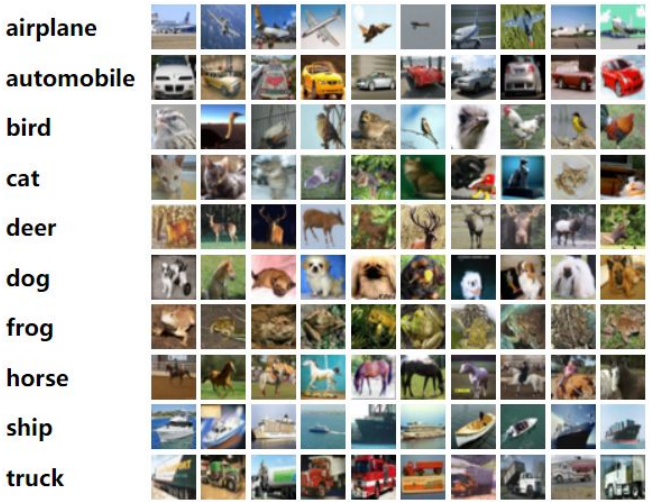
So far, we've looked at neural networks that look like this:

for the XOR problem and on the MNIST dataset (HW3).

- Both of these are pretty simple problems and our models were fairly small.

Now, consider more complex image data, like:

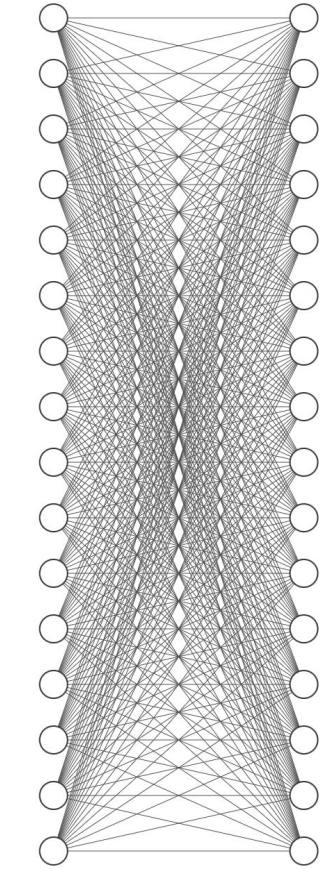
- Why might the same fully connected neural network architecture not work very well for these images?



# Neural Networks: Images

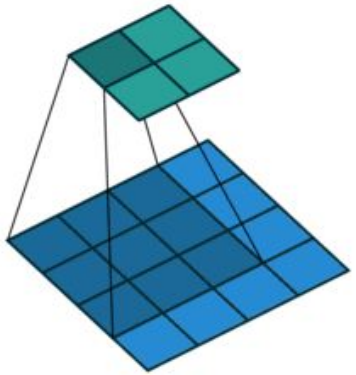
- A dense neural network has *a lot* of parameters to learn.
  - Moderate sized images might be 3 x 200 x 200 pixels each - 1200 inputs!

- With lots of images, we'd like to have fewer weights
- Typically, pixels that are closer to each other are more related while regions that are farther apart are less related

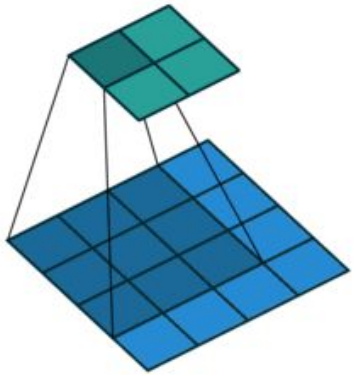


# Convolutions! (crash course)

Taking advantage of the structured nature of image data, we can use weighted sums in small areas of images to process the data.



# Convolutions! (crash course)



Taking advantage of the structured nature of image data, we can use weighted sums in small areas of images to process the data.

- We use a **kernel** filter that slides over the image and compute the sum of the element-wise product between the kernel and the image

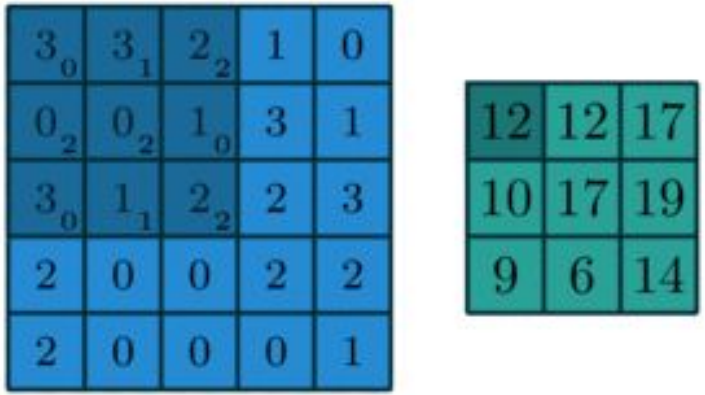
Image

| | | | | |
|---|---|---|---|---|
| 3 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 3 | 1 |
| 3 | 1 | 2 | 2 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 2 | 0 | 0 | 0 | 1 |

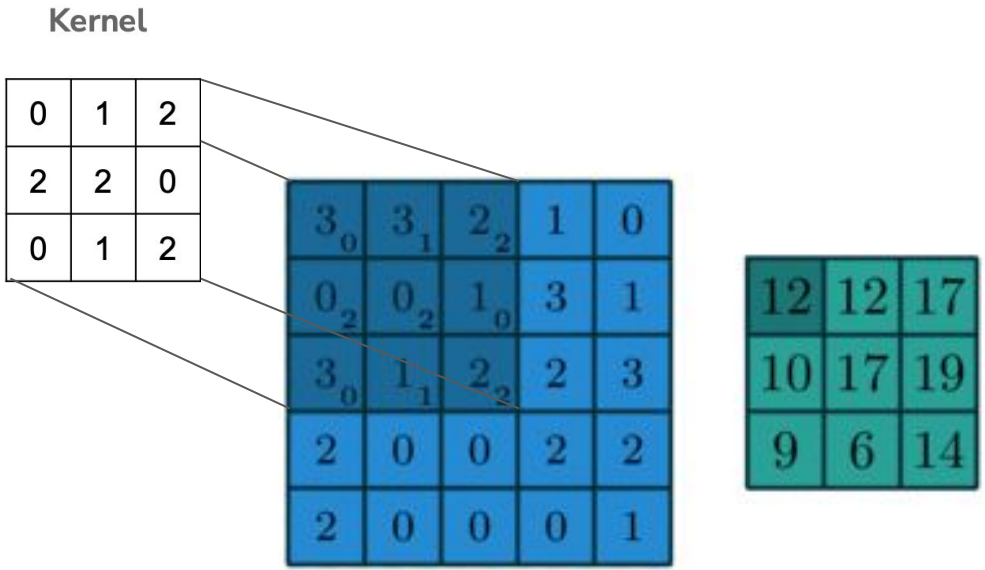
Kernel

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 2 | 2 | 0 |
| 0 | 1 | 2 |

# Convolutions - cont'd.



# Convolutions - cont'd.

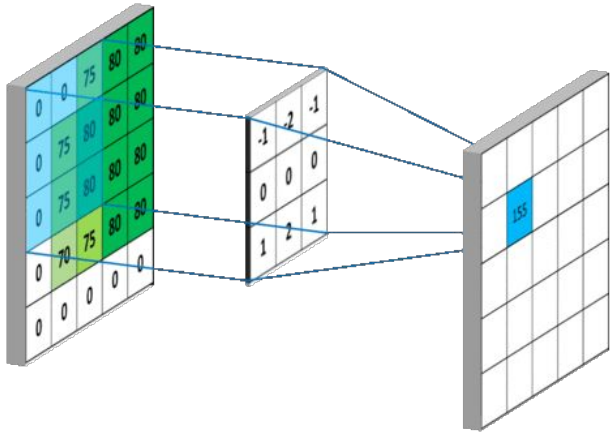



$$(3*0) + (3*1) + (2*2)$$
$$+ (0*2) + (0*2) + (1*0)$$
$$+ (3*0) + (1*1) + (2*2) = 12$$

# Convolutions - cont’d.

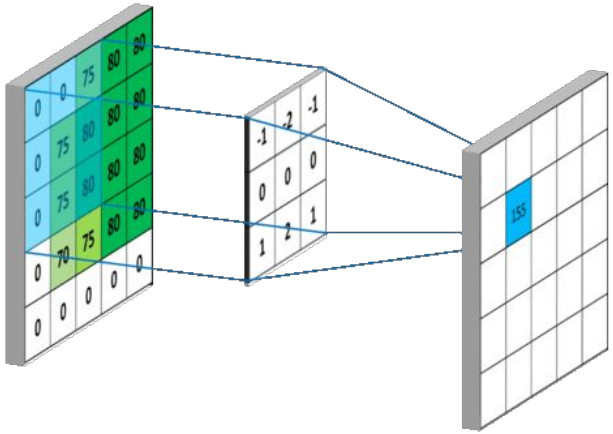
- What 3x3 kernel would result in the same image? (ignoring the edges)

- Can you think of any cool operations convolutions could do with an image?

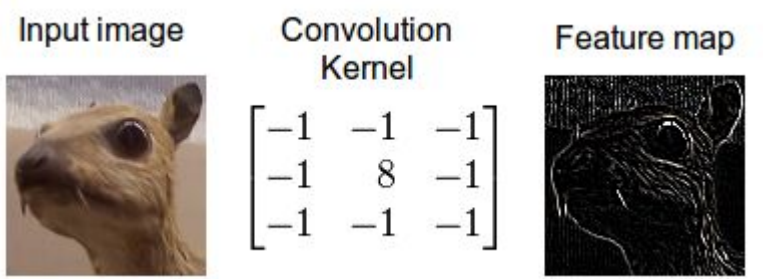


# Convolutions - cont'd.

- What 3x3 kernel would result in the same image? (ignoring the edges)

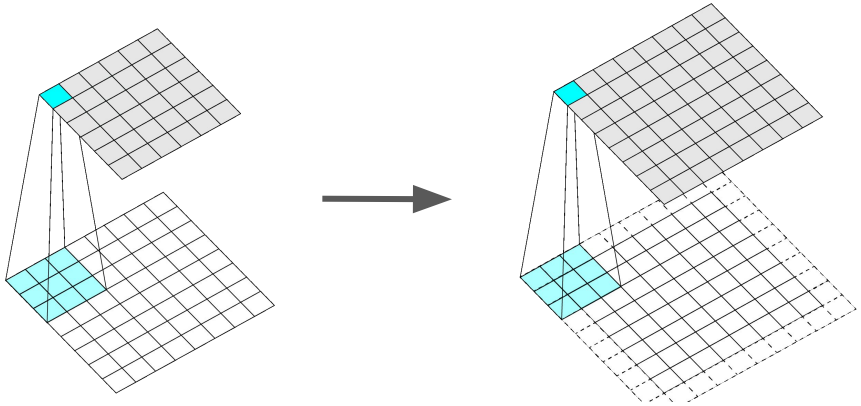


- Can you think of any cool operations convolutions could do with an image?



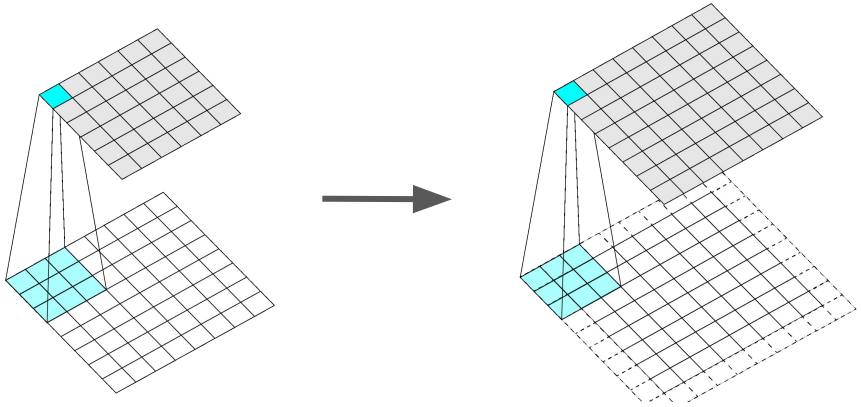
# Padding & Stride

- Convolutions can have problems on the edges. So, we can add “padding” to the edges
  - Can vary the value and amount of padding to include



# Padding & Stride

- Convolutions can have problems on the edges. So, we can add "padding" to the edges
  - Can vary the value and amount of padding to include



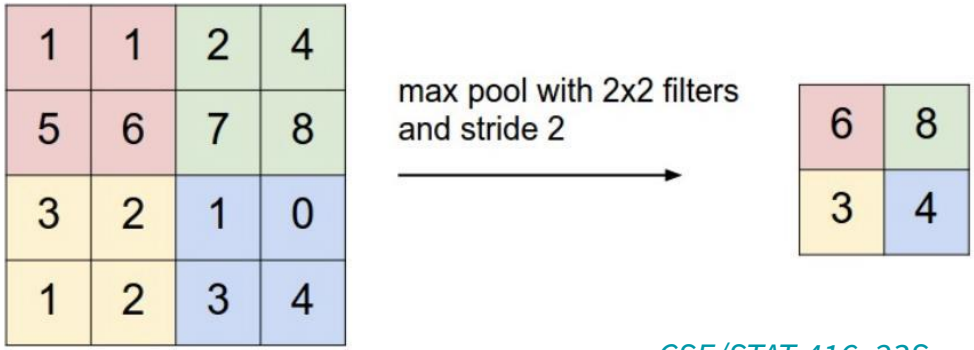
CSE 455, 23Sp

- Can also set a "stride", which determines how far to "jump" values. We've looked at convolutions with stride 1 so far; how would a larger stride affect your convolution output?

# Pooling

Similar to convolutions, **pooling** is an operation that is commonly used to **downsample** images.

A pooling layer will iterate over an image like a convolution, pooling pixels in each region. You can average pixels, take the minimum value, take the median, etc. But typically, we will use **max pooling**, where we take the maximum value at each region.

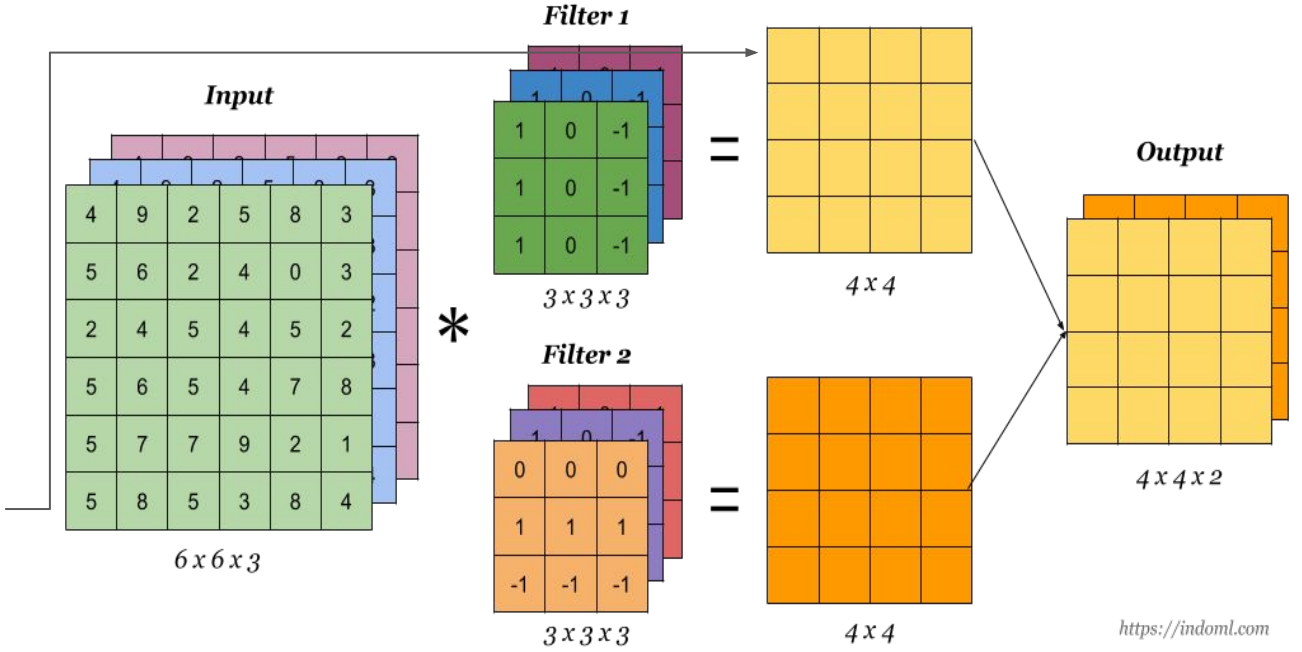


# What each filter looks like

Number of filters influence the output channel through stacking

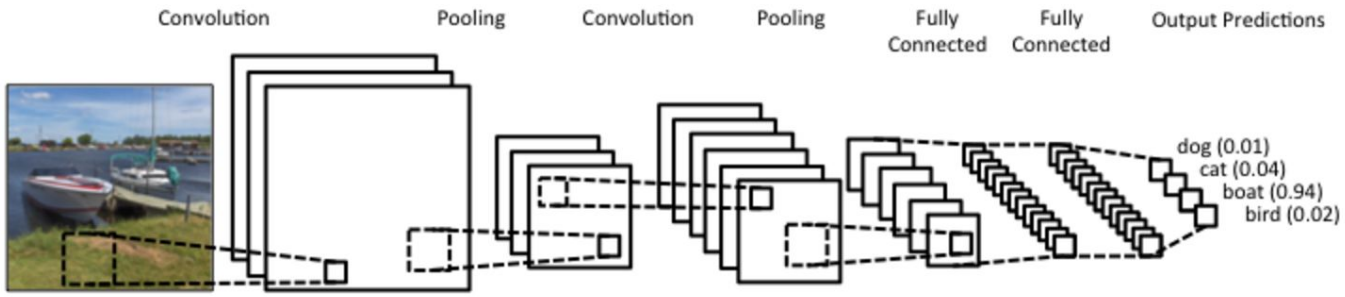
Each filter learns some feature of the input.

For this example, input is a 6x6 image with RGB channels. The filter will have the size of third dimension equal to the input channel (3). Output is just the elementwise dot-product.



# Convolutional Neural Networks

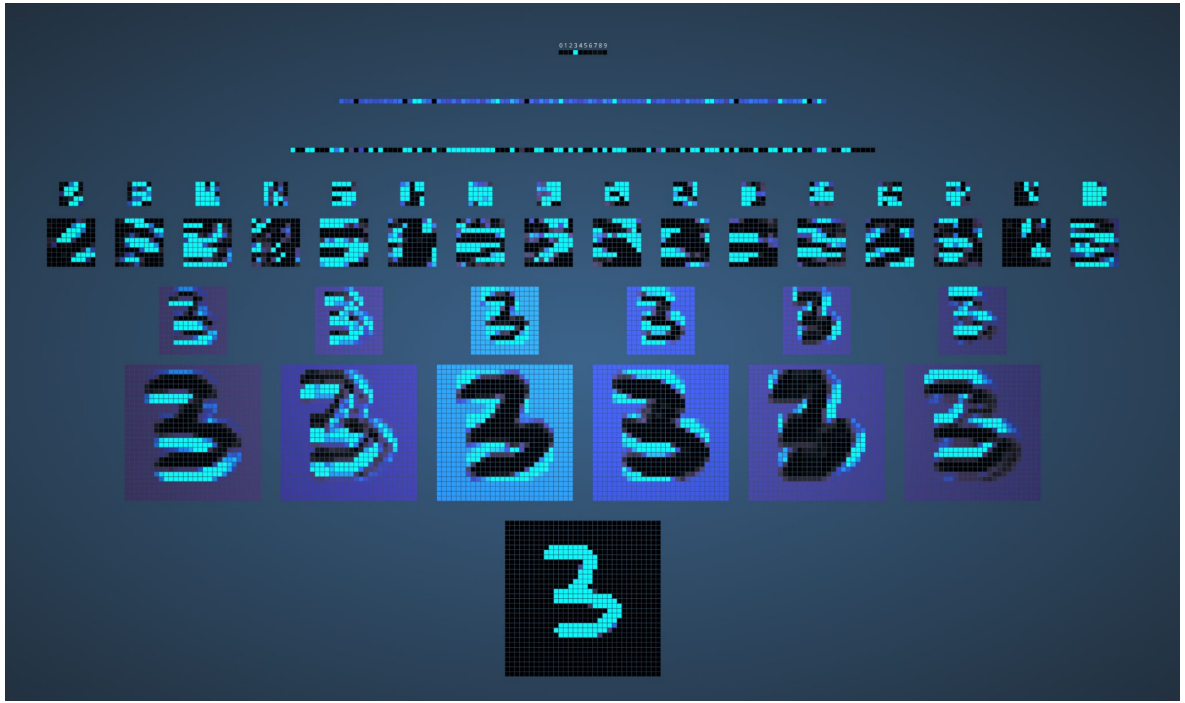
- Lots of different architectures possible for CNNs! In general, they often take the form:



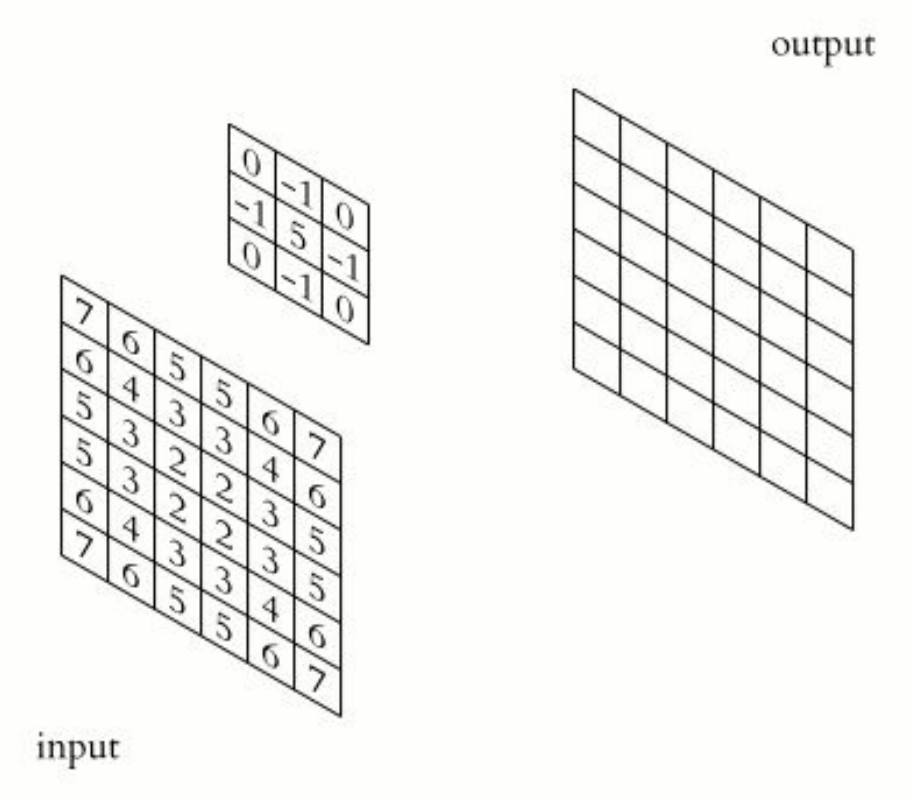

  - The basic building blocks are convolutional layers, pooling layers, and fully connected layers. Fully connected layers are often used as a last layer to map your image features to predictions.

# Filters visualized

You can clearly see the feature mapping here as a result of the convolutional operation with the kernel filters.



CNN visualization demo by Adam Harley



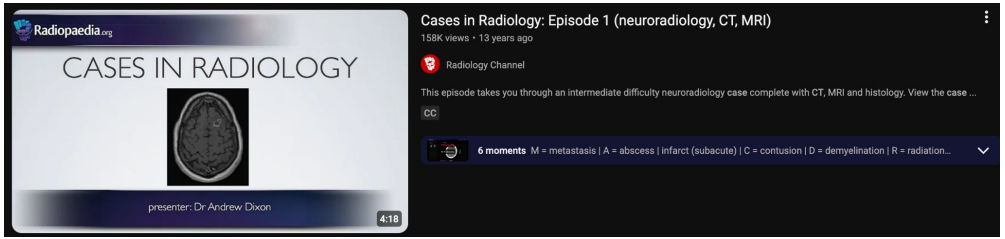

This video shows a convolutional pass being done. The kernel they use here is a “sharpening kernel”

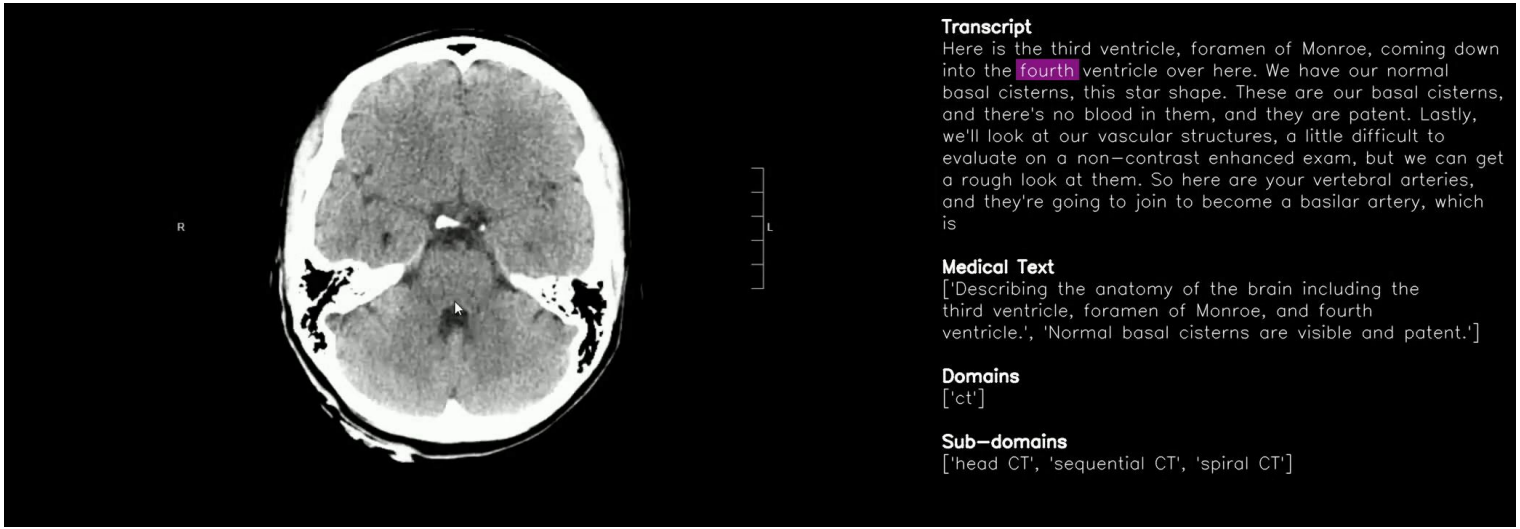
**Shape of a convolutional layer / maxpooling output:** For a $n \times n$ input, $f \times f$ filter, padding $p$ and stride $s$, the output size is $o \times o$ where:

$$o = \frac{n - f + 2p}{s} + 1$$

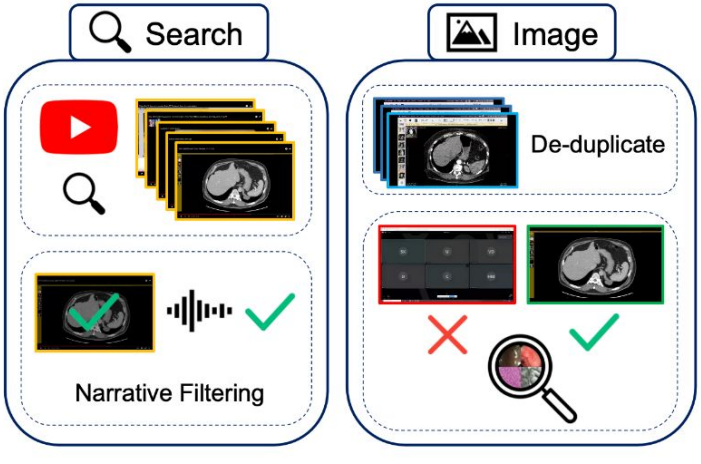
An equation worth memorizing…

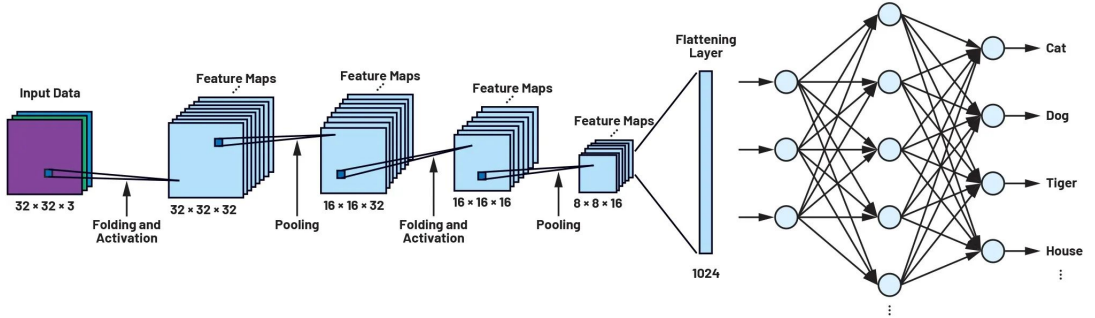
# Applications of CNNs

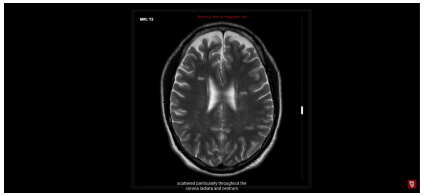


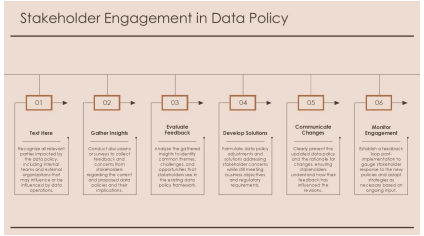



# Applications of CNNs





# Questions/Chat Time!

# Variance Maximization

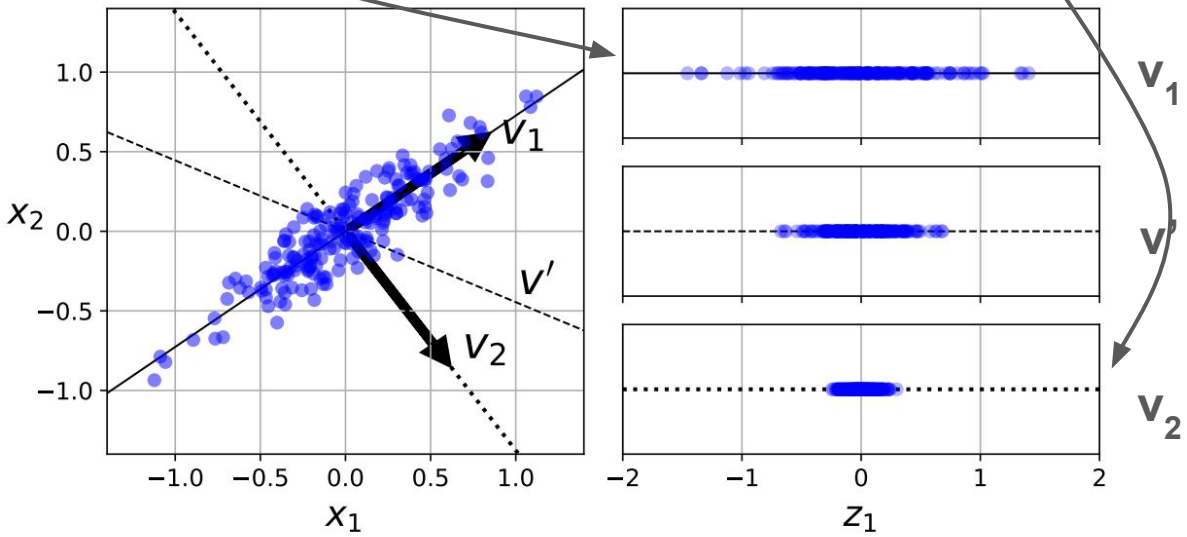
One way to think about “information” in data is its spread, or variance.

Projection data down to one single point = loss of all information.

**So what is the opposite?**

$\mathbf{v}_1$ is the best line to project to

$\mathbf{v}_2$ is the worst line to project to



Our goal is to find a projection matrix $\mathbf{V}_q \in \mathbb{R}^{d \times q}$ that minimizes the reconstruction error of our demeaned data. Mathematically, we want to find:

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^\top (x_i - \bar{x})||_2^2$$

Turns out, the $\mathbf{V}_q$ that minimizes this equation is **the first $q$ eigenvectors of $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$**. There is a proof in the lecture notes, but here is the inuition:

- Eigenvalues/eigenvectors tell you the "direction and magnitude of greatest stretch" when a matrix $\mathbf{M}$ is applied as a transformation.

- If $\mathbf{X}$ is our data, make $\tilde{\mathbf{X}}$ the demeaned data. This makes $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ proportional to the **sample covariance matrix** of $\mathbf{X}$. Essentially, a matrix which holds information about the variance of $\mathbf{X}$.

- The eigenvectors of $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ will tell us the directions of greatest variance. Intuitively, finding the eigenvectors of this should give us **principal components** to project onto!

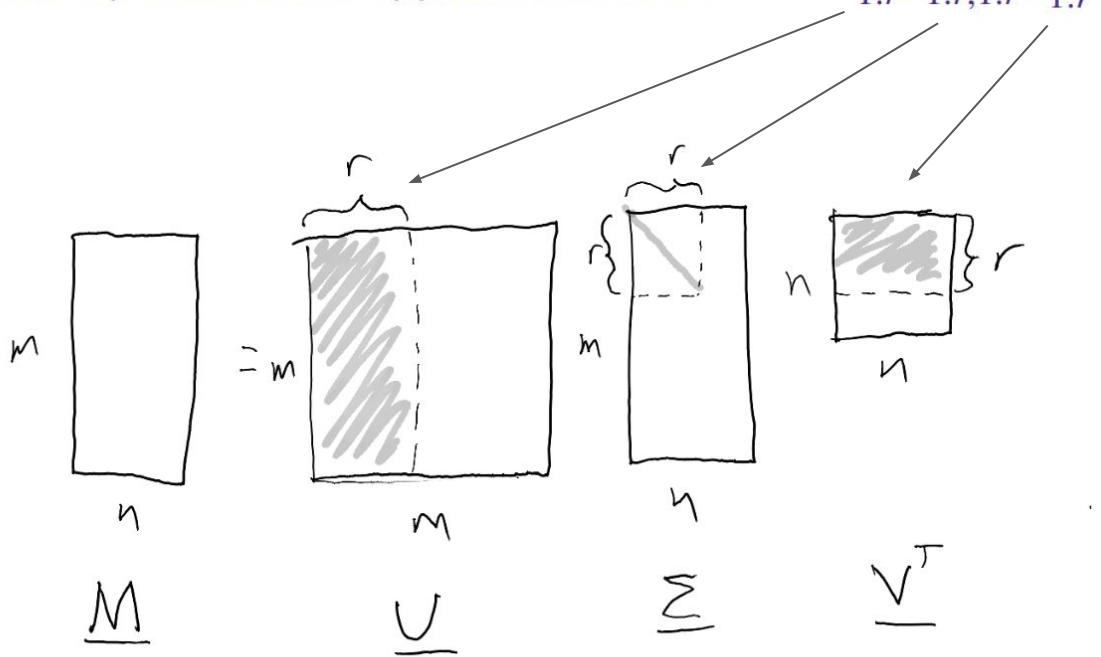
# CNN Slides

# Rank-r approximations

Shaded areas are what is "enough" to reconstruct **M**. Blank areas are 0

For compression using SVD, we pick **r** to be small to save us lots of space!

- The optimal rank-$r$ approximation is $L = U_{1:r}S_{1:r,1:r}V^T_{1:r}$



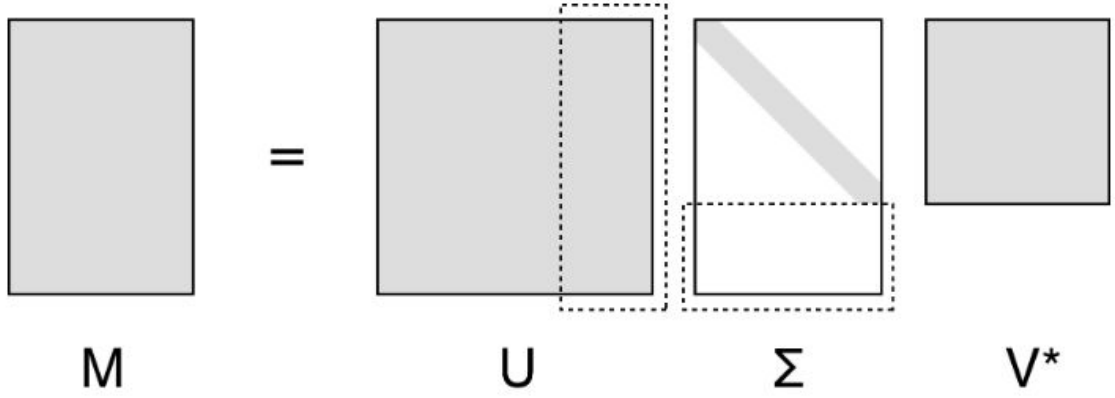
# Singular Value Decomposition

**Fact:** Any matrix **M** can be decomposed into three separate matrices.

- I will not be discussing how to *solve* SVD, rather what its solution *means.*

(BTW: This is equivalent to thinking of **M** as a rotation, scaling, and another rotation).

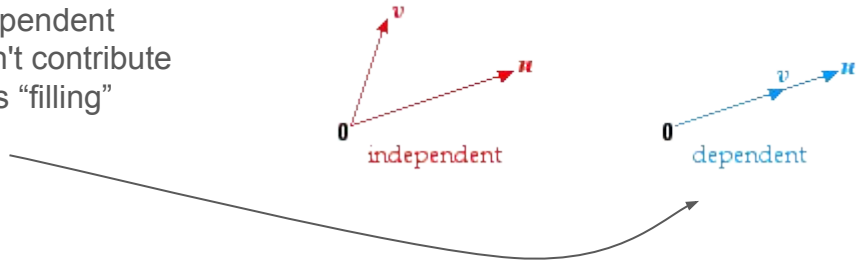
Say **M** is rank **r**




Alternative explanation:
Rank is how much space the matrix “fills”.

Linearly dependent vectors don't contribute more to this “filling”

How I like to think of rank: How much space is **actually** taken up by the vectors in the matrix?



# Relating PCA & SVD

From lecture:

**Theorem [SVD]**: Let $A \in \mathbb{R}^{m \times n}$ with rank $r \leq \min\{m, n\}$. Then $A = USV^T$ where $S \in \mathbb{R}^{r \times r}$ is diagonal with non-negative entries, $U^TU = I$, and $V^TV = I$.

$\mathbf{V}$ are the first $r$ eigenvectors of $\mathbf{A}^T\mathbf{A}$ with eigenvalues $\text{diag}(\mathbf{S}^2)$

$\mathbf{U}$ are the first $r$ eigenvectors of $\mathbf{A}\mathbf{A}^T$ with eigenvalues $\text{diag}(\mathbf{S}^2)$

So how do we find the eigenvectors of $\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}}$ ?

$\text{SVD}(\tilde{\mathbf{X}}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ $\therefore$ *(definition of SVD)*

$\mathbf{V}$ contains the first $\mathbf{r}$ eigenvectors of $\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}}$, $\therefore$ *(fact from lecture notes)*

the principal components of $\mathbf{X}$ exist in $\mathbf{V}$ from the SVD of $\tilde{\mathbf{X}}$ *(PCs of* $\mathbf{X}$ *= eigenvectors of* $\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}}$*)*

Remember this?: $\sigma_1 > \sigma_2 > \ldots > \sigma_r$

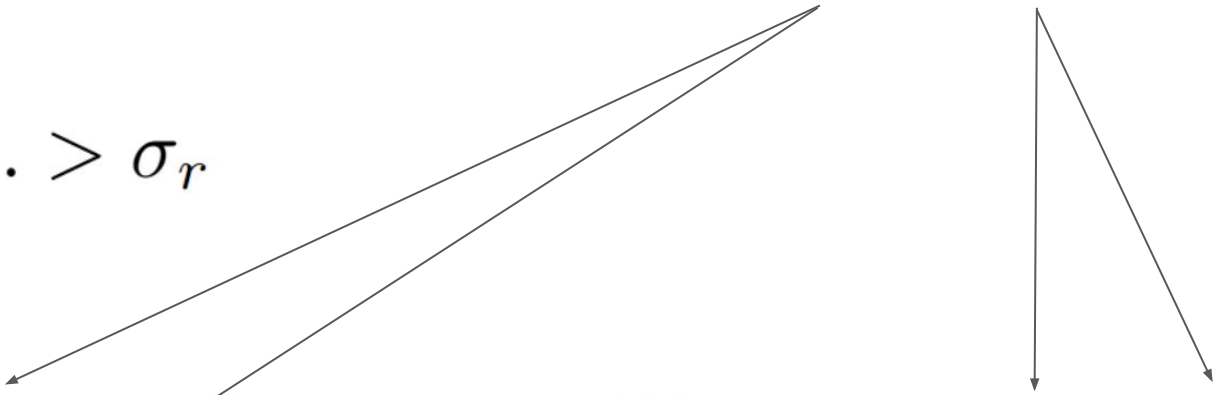
Selecting the first *q* vectors from **V** gives us the directions with the GREATEST variance… **principal components!**

# Understanding the decomposition

**Q:** Which singular vectors affect the reconstruction of **M** the most and least. Why?

$$\sigma_1 > \sigma_2 > \ldots > \sigma_r$$

**A:**



$$\mathbf{U\Sigma V}^\top = u_1\sigma_1 v_1^\top + u_2\sigma_2 v_2^\top + \ldots + u_r\sigma_r v_r^\top$$