# CS 189/289

Today's lecture outline

1. Finish Transformers
2. Unsupervised learning, dimensionality reduction
3. PCA

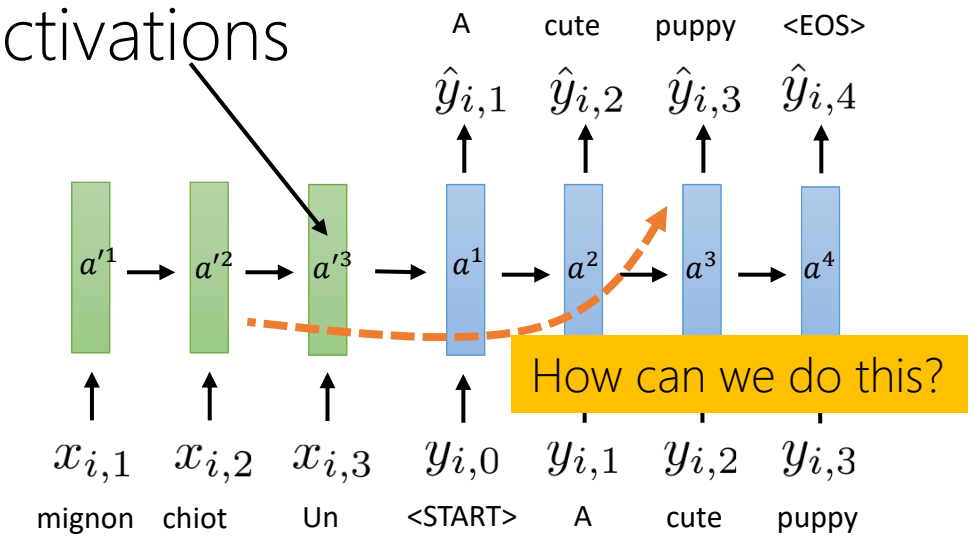
# CS 189/289

## Today's lecture outline

1. Finish Transformers
2. Unsupervised learning, dimensionality reduction
3. PCA

# Recall: RNN bottleneck problem

all information about the conditioned sequence is contained in these activations




**Idea:** what if we could somehow "peek" at the source sentence while decoding?

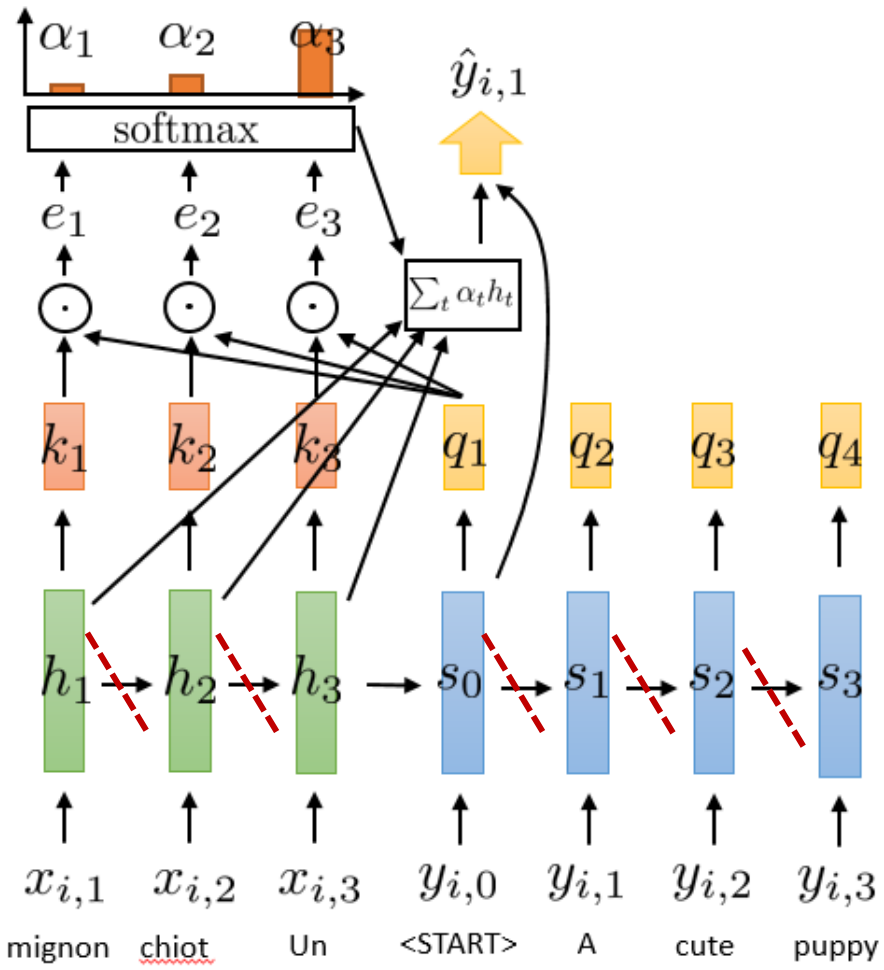
**Attention to the rescue!**

# Recall: Is Attention All We Need?

- If we have **attention**, do we even need recurrent connections?
- Can we **transform** our RNN into a **purely attention-based** model?

This has a few issues we must overcome:

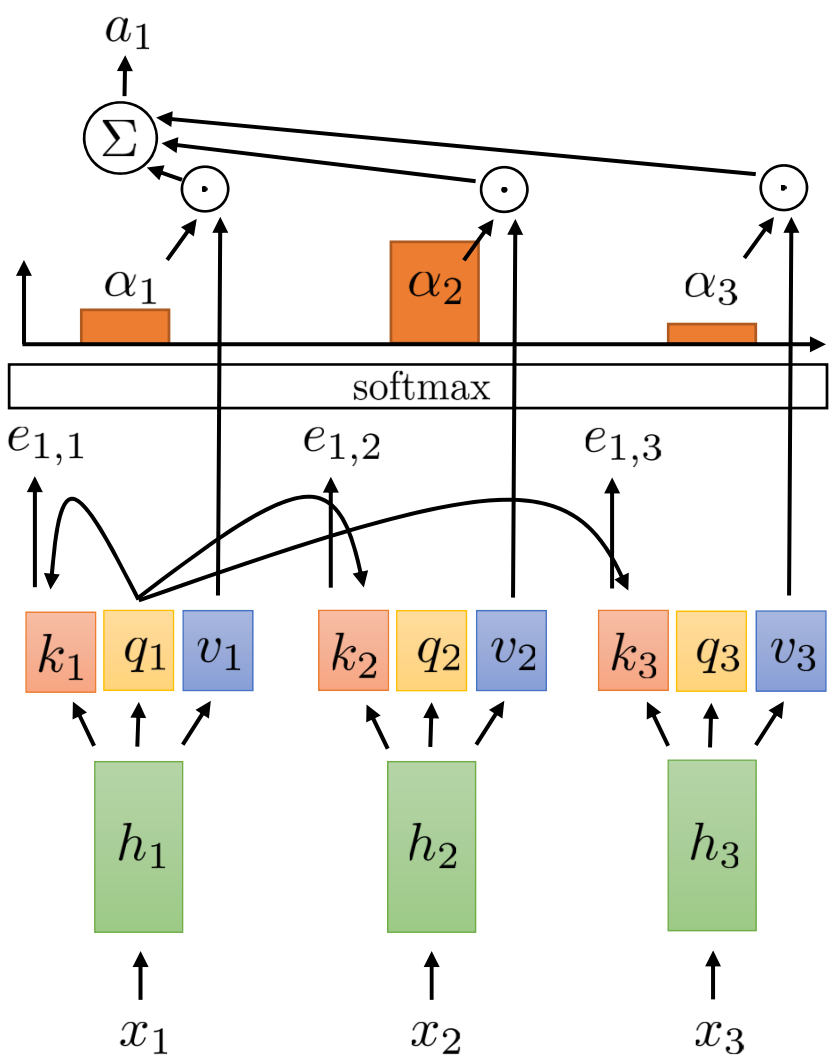
- Decoding position 3 can't access $s_1$ or $s_0$.
- Solution: self-attention.

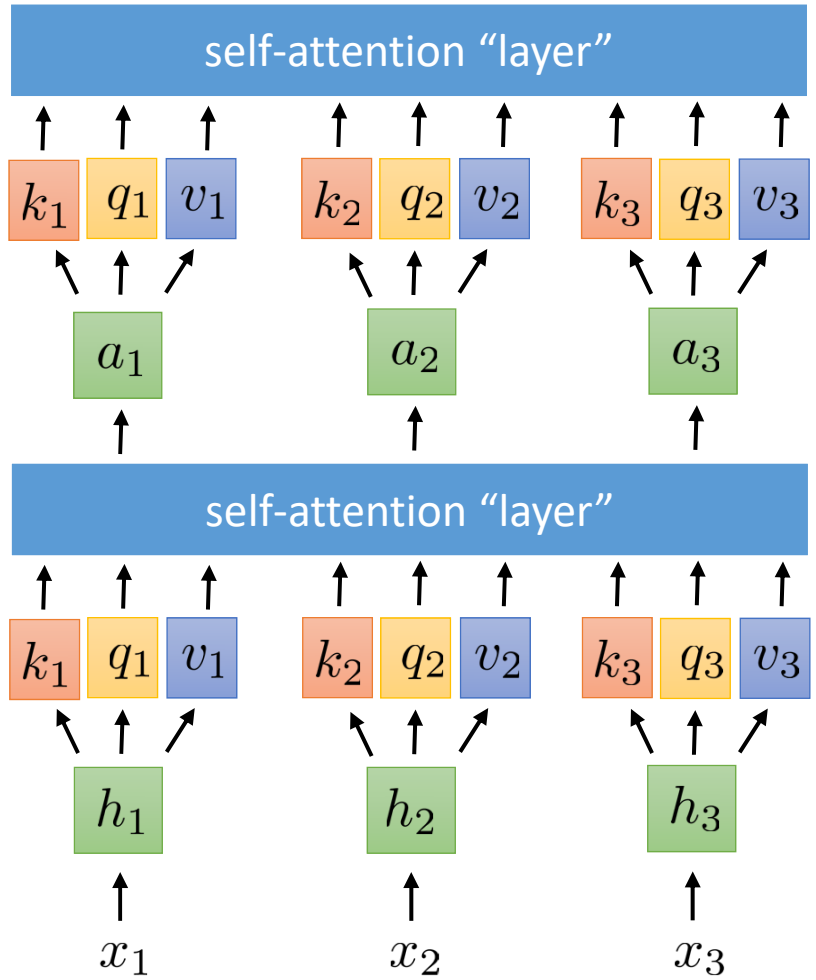


# Recall: Self-Attention

keep repeating until we've processed this enough

then hand off to next part of overall model



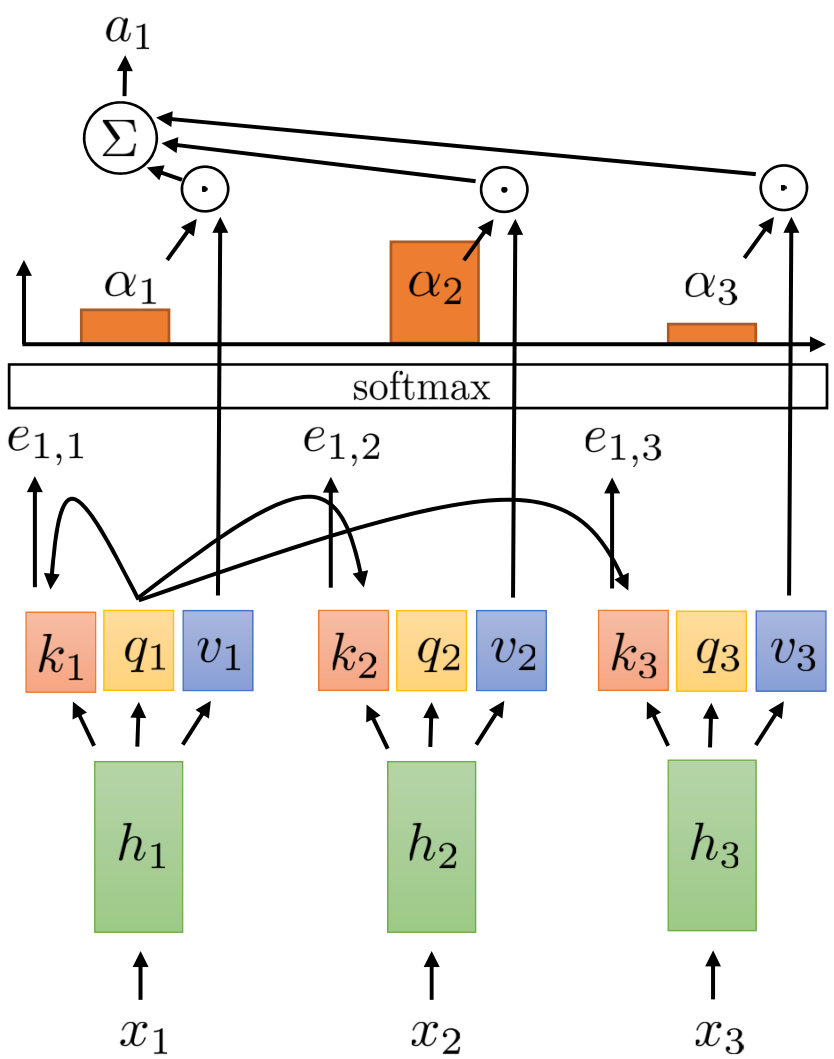



# From Self-Attention to Transformers

- Self-attention lets us remove recurrence entirely, yielding the now pervasively used Transformer model for sequences.
- But we need a few additional components to fix some problems:

| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Positional encoding: what is the order?




**what we see:**

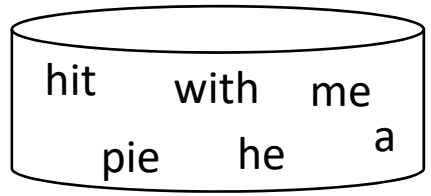
he hit me with a pie

**what naïve self-attention sees:**

a pie hit me with he

a hit with me he pie

he pie me with a hit



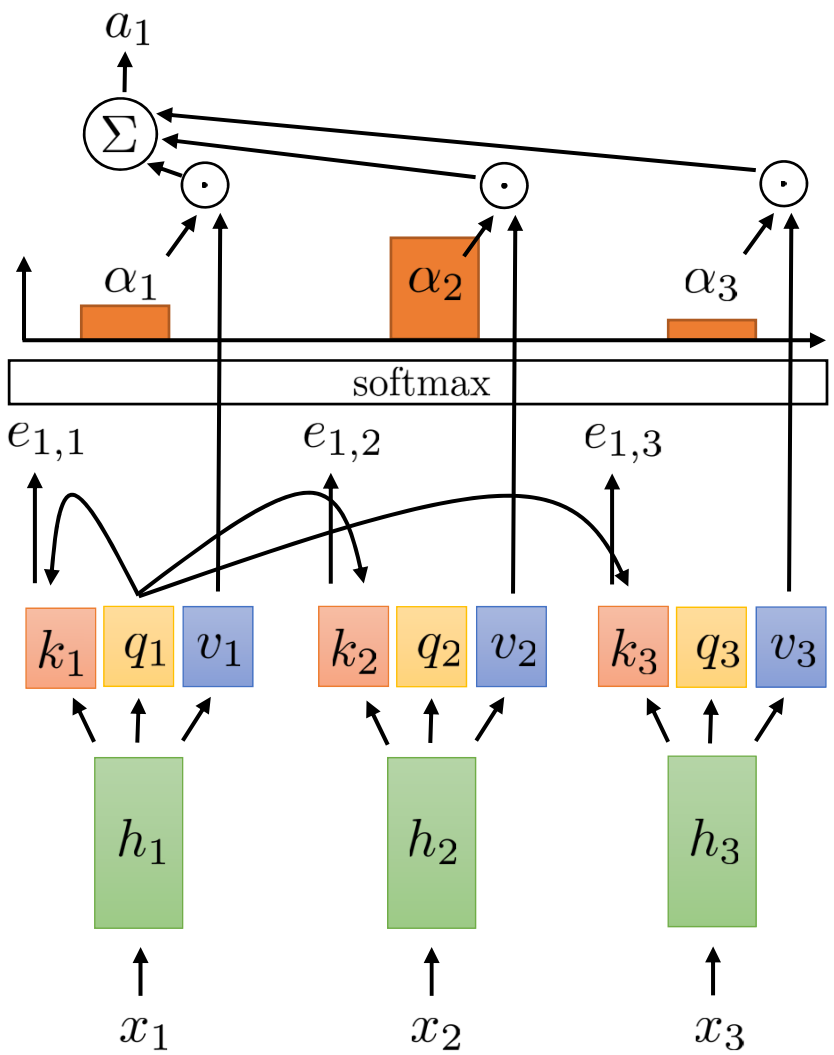

Permutation Equivariant!

**Idea:** add "positional" information, i.e. that indicates where it is in the sequence!

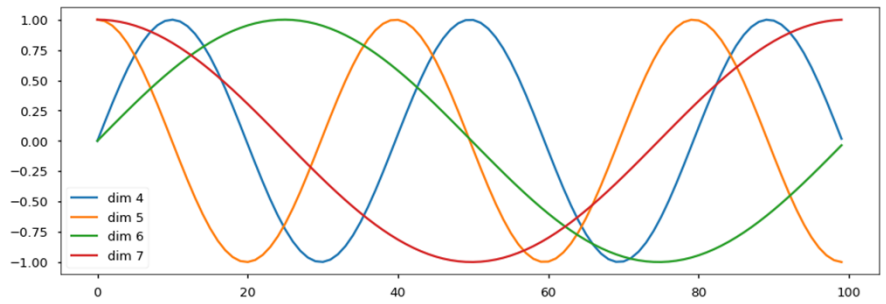
$$h_t = f(x_t, t)$$

some function

# Positional encoding: what is the order?




$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$




$d$, is the dimensionality of positional encoding

$$h_t = f(x_t, t)$$

some function

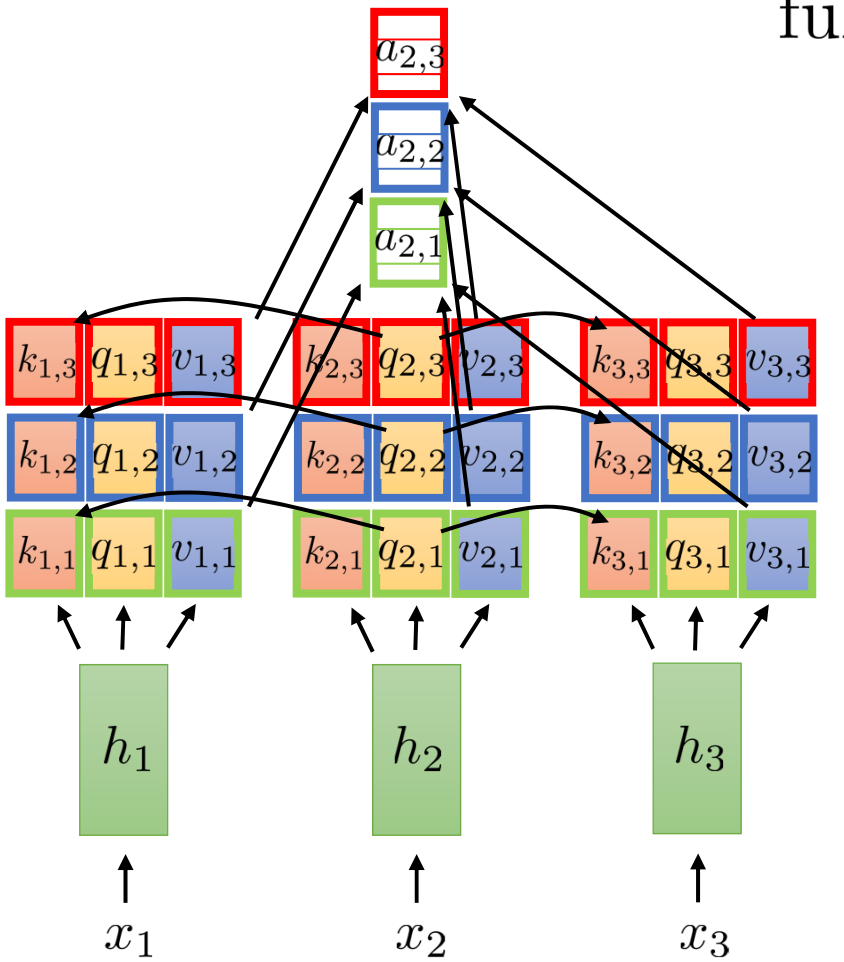
# From Self-Attention to Transformers

- The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**
- But to make this actually work, we need to develop a few additional components to address some fundamental limitations

| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Multi-head attention

Idea: have multiple keys, queries, and values for every time step!




full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

compute weights **independently** for each head

$$e_{l,t,i} = q_{l,i} \cdot k_{l,i}$$

$$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$$

$$a_{l,i} = \sum_{t} \alpha_{l,t,i} v_{t,i}$$

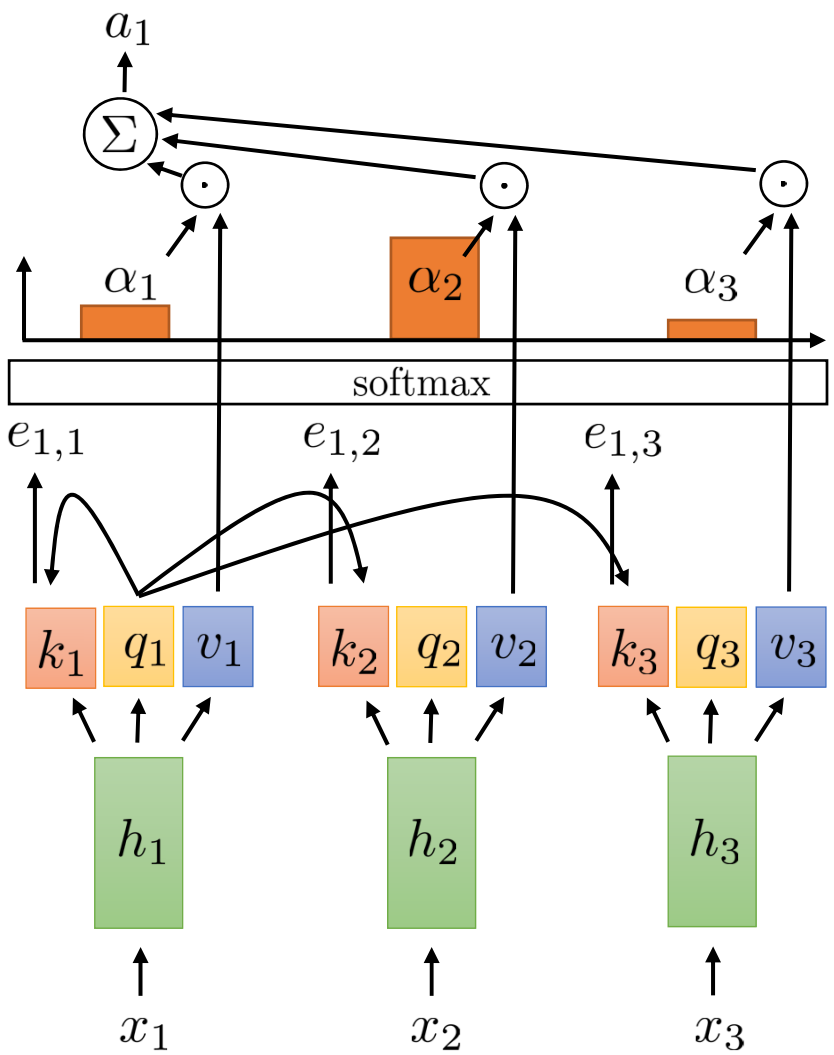
*around 8 heads seems to work pretty well for big models*

# From Self-Attention to Transformers

- The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**
- But to make this actually work, we need to develop a few additional components to address some fundamental limitations

| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Self-Attention is **Linear**




$$k_t = W_k h_t \qquad q_t = W_q h_t \qquad v_t = W_v h_t$$

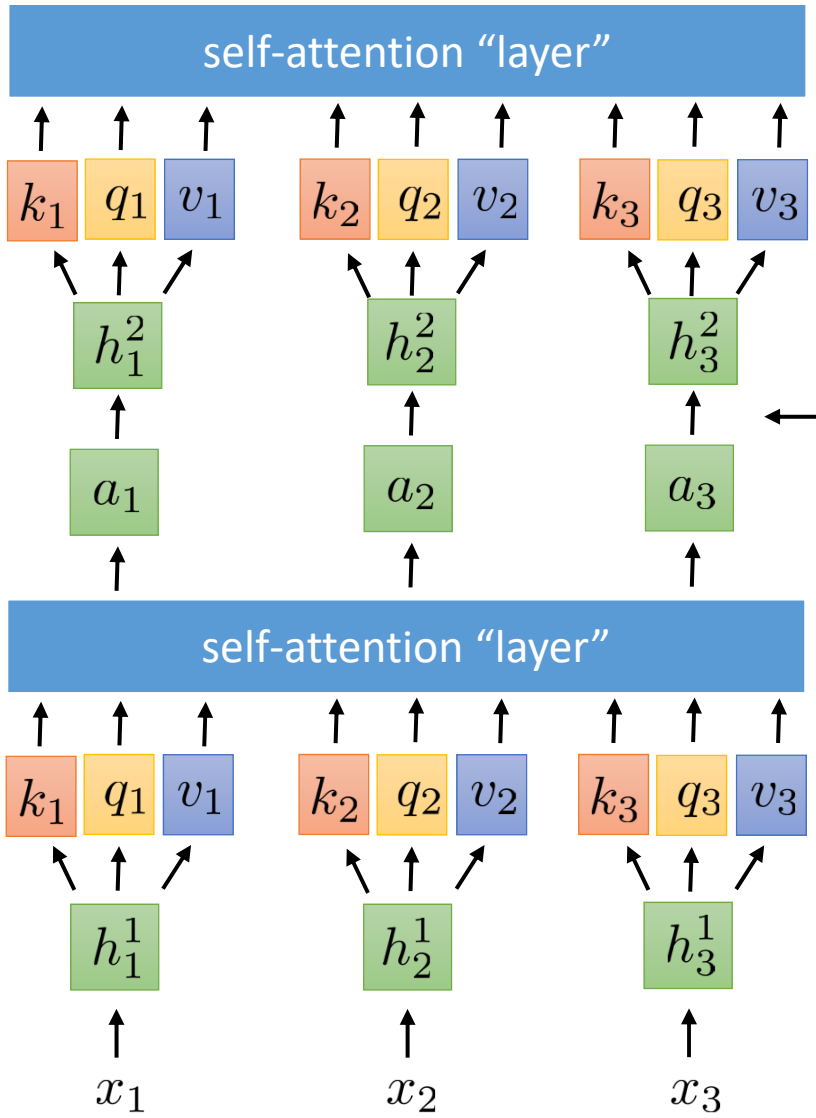
$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$$a_l = \sum_t \alpha_{l,t} v_t = \sum_t \alpha_{l,t} W_v h_t = W_v \sum_t \alpha_{l,t} h_t$$

Every self-attention "layer" is a linear transformation of the previous layer

# Alternating self-attention & non-linearity




some non-linear (learned) function

e.g., $h_t^\ell = \sigma(W^\ell a_t^\ell + b^\ell)$

just a neural net applied at every position after every self-attention layer

# From Self-Attention to Transformers

- The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**
- But to make this actually work, we need to develop a few additional components to address some fundamental limitations

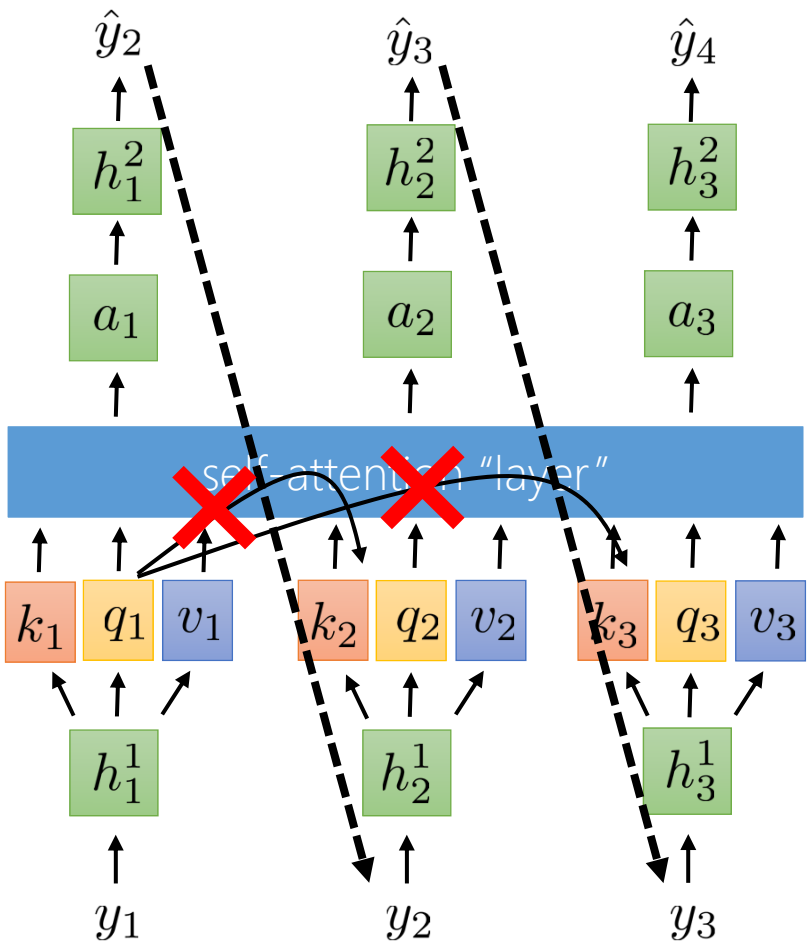
| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Self-attention can see the future!

*e.g.* self-attention "language model":

$$a_l = \sum_t \alpha_{l,t} v_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$



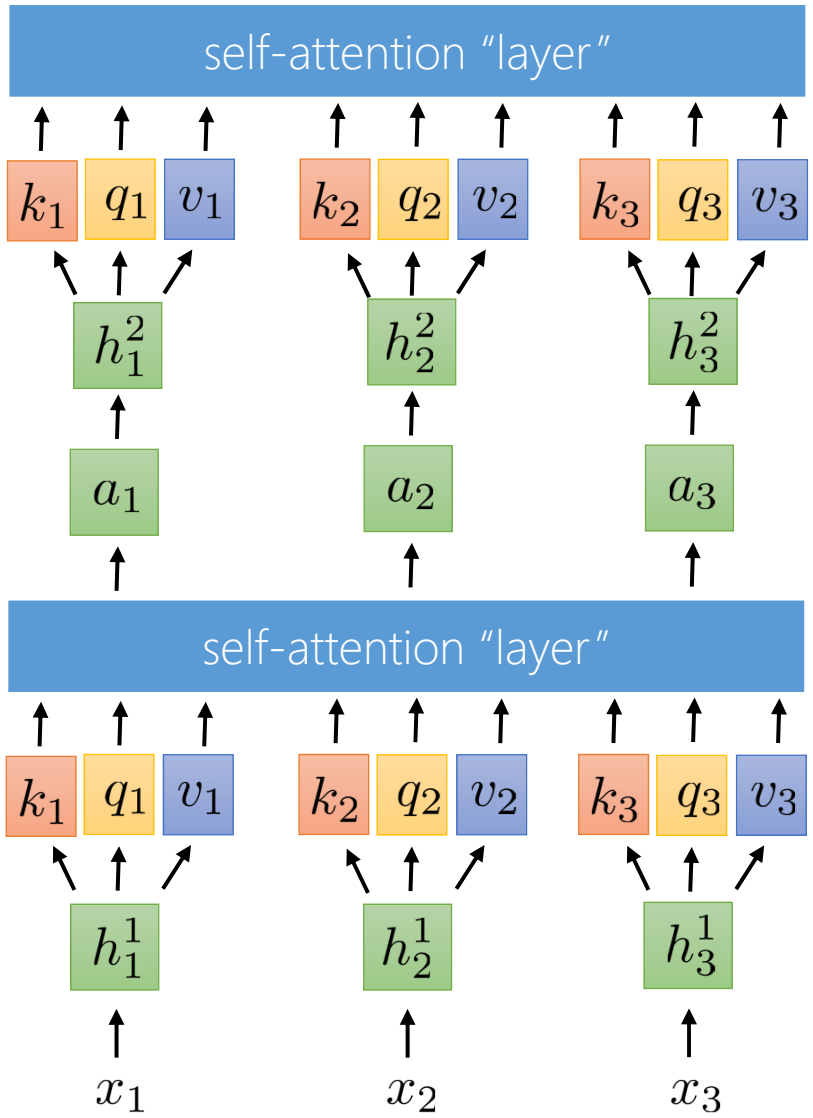

Problem:

- Step 1 can look at future values (hence inputs).
- **At test time** ("decoding"), the output at step 1 will see the input at step 2 ...
- Also cyclic: output 1 depends on input 2 which depends on output 1.
- So it can see itself, thereby "cheating".

Solution: $$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } t \leq l \\ -\infty & \text{otherwise} \end{cases}$$

Now we are read for
The Transformer!

# Sequence-to-sequence with self-attention




"*Transformer*" architecture:

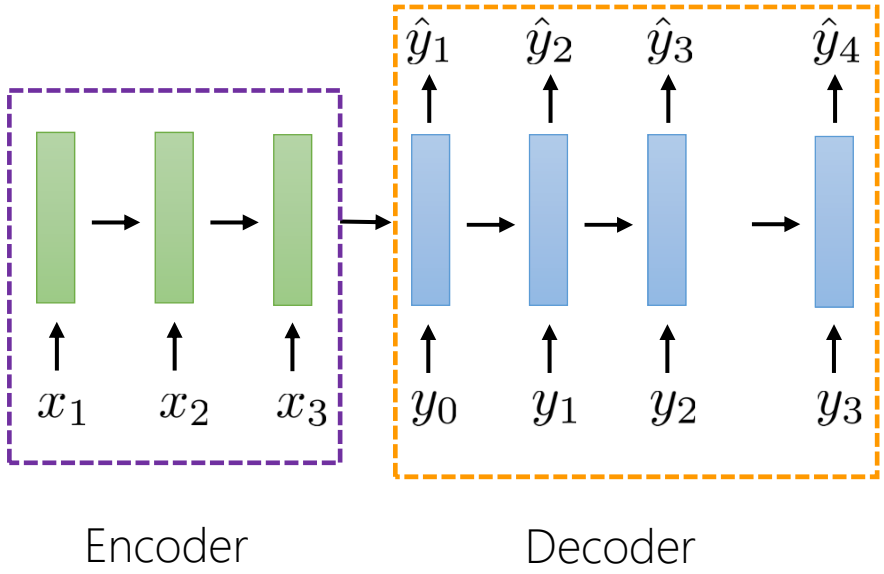
- Stacked self-attention layers with position-wise nonlinearities.
- *Transform* one sequence into another at **each** layer.
- For sequence data.

[Vaswani et al. **Attention Is All You Need.** 2017]

# Encoder-Decoder Transformer

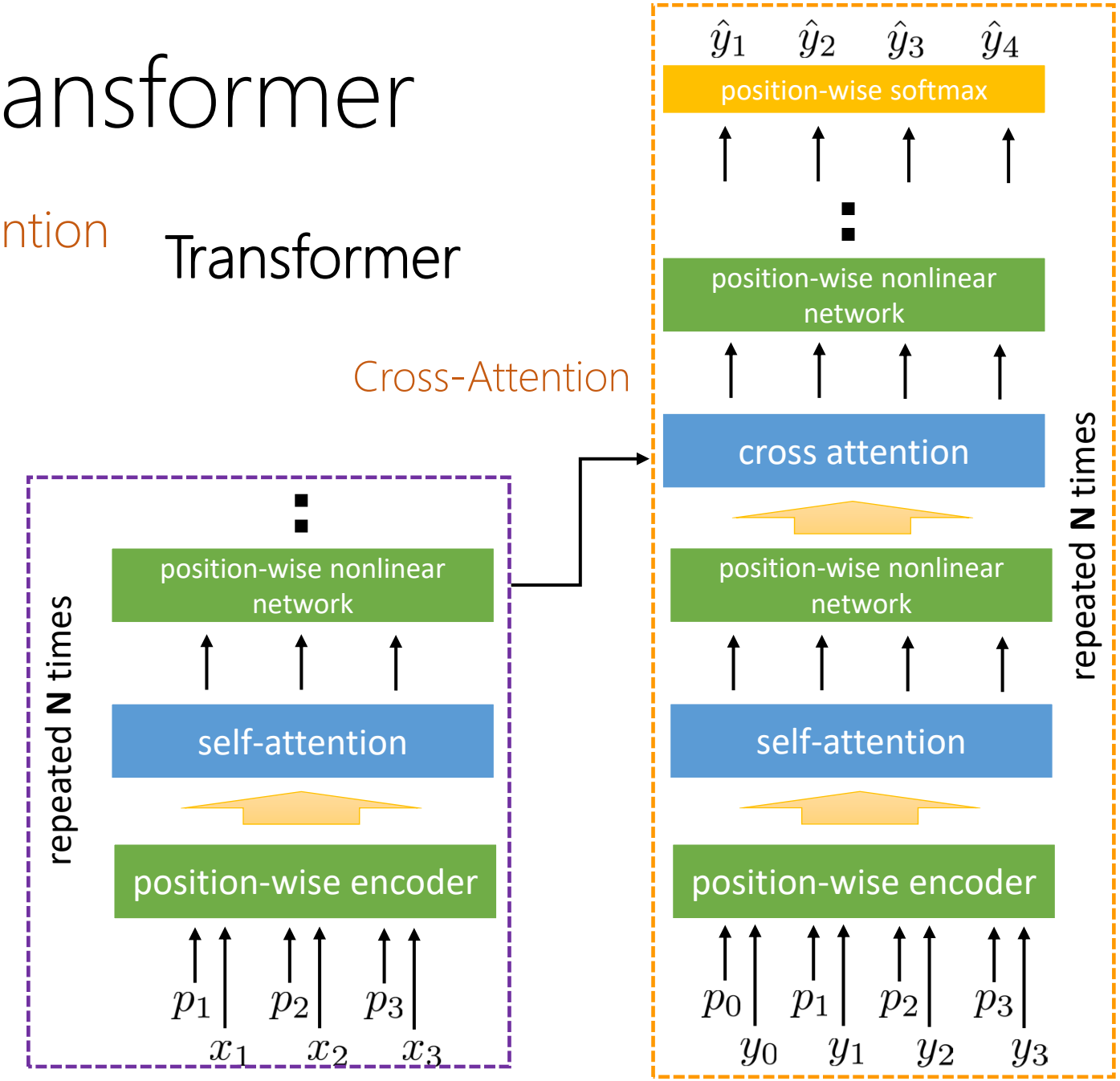
Similar to the **standard (non-self)** attention from the previous lecture

RNN



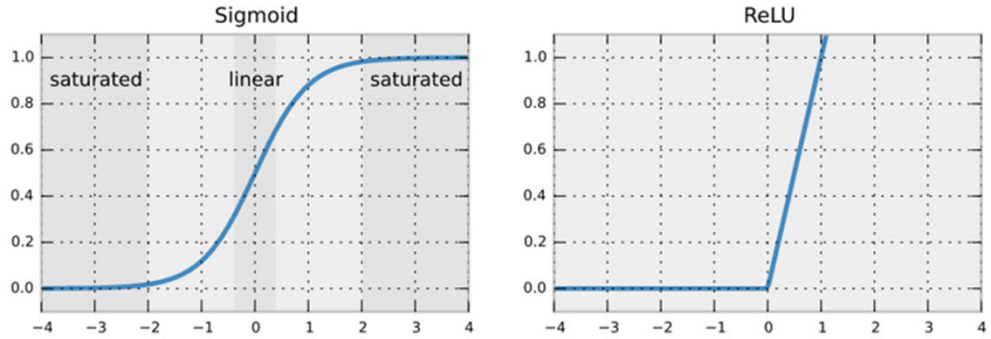

Transformer

Cross-Attention



# Recall: batch normalization

"Vanishing gradient" from saturating non-linearities




$$z^{(1)} = W^{(1)}x$$
$$h^{(1)} = \sigma(z^{(1)})$$

Activation functions saturating (problem amplified by depth)—fixed with *normalizations* (*e.g.* "batch normalization").

1. Normalize data in the mini-batch

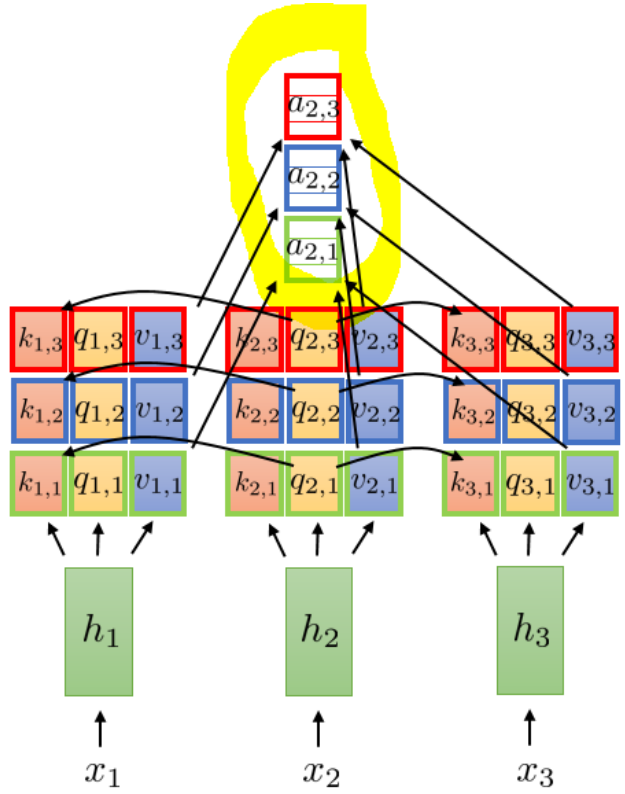
$$\widehat{z^{(1)}} = \frac{z^{(1)} - E\left[z^{(1)}\right]}{\sqrt{Var[z^{(1)}]}}$$

2. Add scale and shift parameters, $\gamma, \beta$:

$$h^{(1)} = \sigma(\gamma\widehat{z^{(1)}} + \beta)$$

# From batch to layer normalization

- Batch normalization tricky in sequence models: long sequences have small batches/poor stats.
- *Layer normalization*: multi-headed attention vectors for one position in a layer are stacked together to form vector $\boldsymbol{a}$, over which mean & std. dev. are computed for one sample.
- Layer normalization is independent of the batch size.




$$\widehat{z^{(1)}} = \frac{z^{(1)} - E\left[z^{(1)}\right]}{\sqrt{Var\left[z^{(1)}\right]}}$$

$$h^{(1)} = \sigma(\gamma \widehat{z^{(1)}} + \beta)$$

# Transformers pros and cons

Downsides:

- Attention computations are theoretically* $O(n^2)$.
- Somewhat more complex to implement (positional encodings, etc.)

Benefits:

+ Better long-range connections (compared to RNN).
+ *Much easier to parallelize.
+ In practice, can make it much deeper (more layers) than RNN.

- Benefits often **vastly** outweigh the downsides.
- Transformers work **much** better than RNNs in general.
- One of the most important sequence modeling improvements of the past decade.
- Can use just encoder, just decoder.

# CS 189/289

## Today's lecture outline

1. Finish Transformers
2. Unsupervised learning, dimensionality reduction
3. PCA

# Unsupervised learning

- So far: *supervised learning*, $\{(x_i, y_i)\}$ for $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$ or $y \in \mathbb{Z}$.
- Often model just $\{x_i\}$: *unsupervised learning*, includes:

*i.* *Dimensionality reduction*, $z \in \mathbb{R}^m = f_\theta(x \in \mathbb{R}^d)$, $m \ll d$.

*ii.* *Clustering*, for each $x_i$, assign cluster label, $z_i \in \{1,2,3 \ldots K\}$

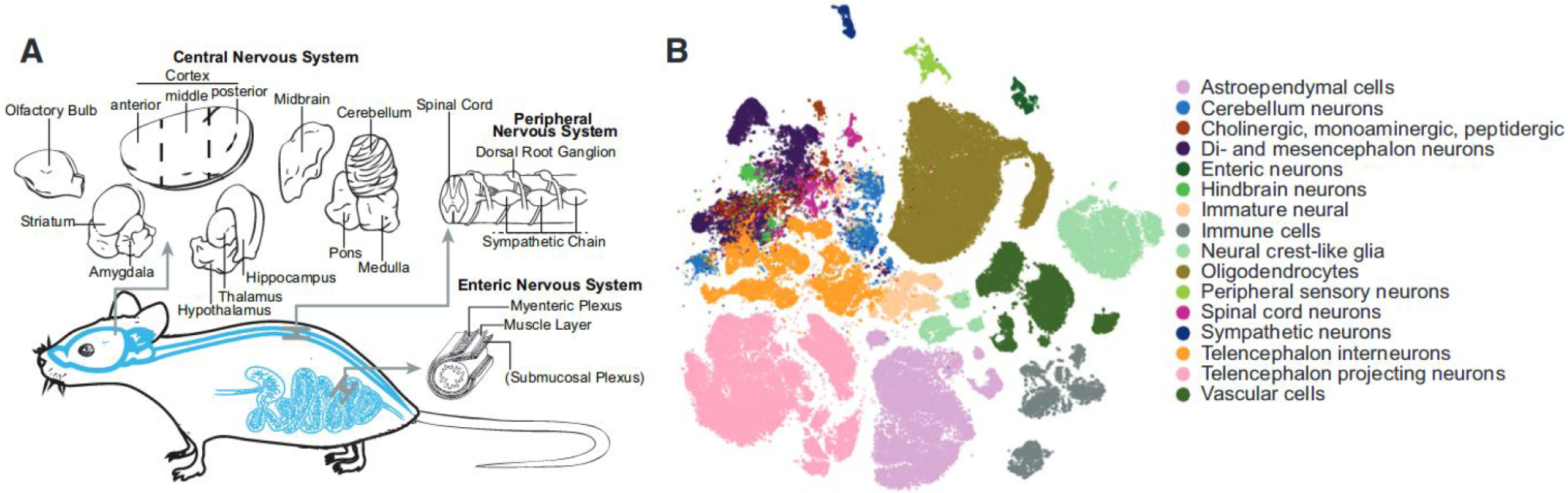
*iii.* *Representation learning*, $z \in \mathbb{R}^m$, $z = f_\theta(x)$, or $z \sim p_\theta(x)$.

*iv.* *Density estimation*, evaluate $p_\theta(x)$.

*v.* *"Generative" modeling*, $x \sim p_\theta(x)$

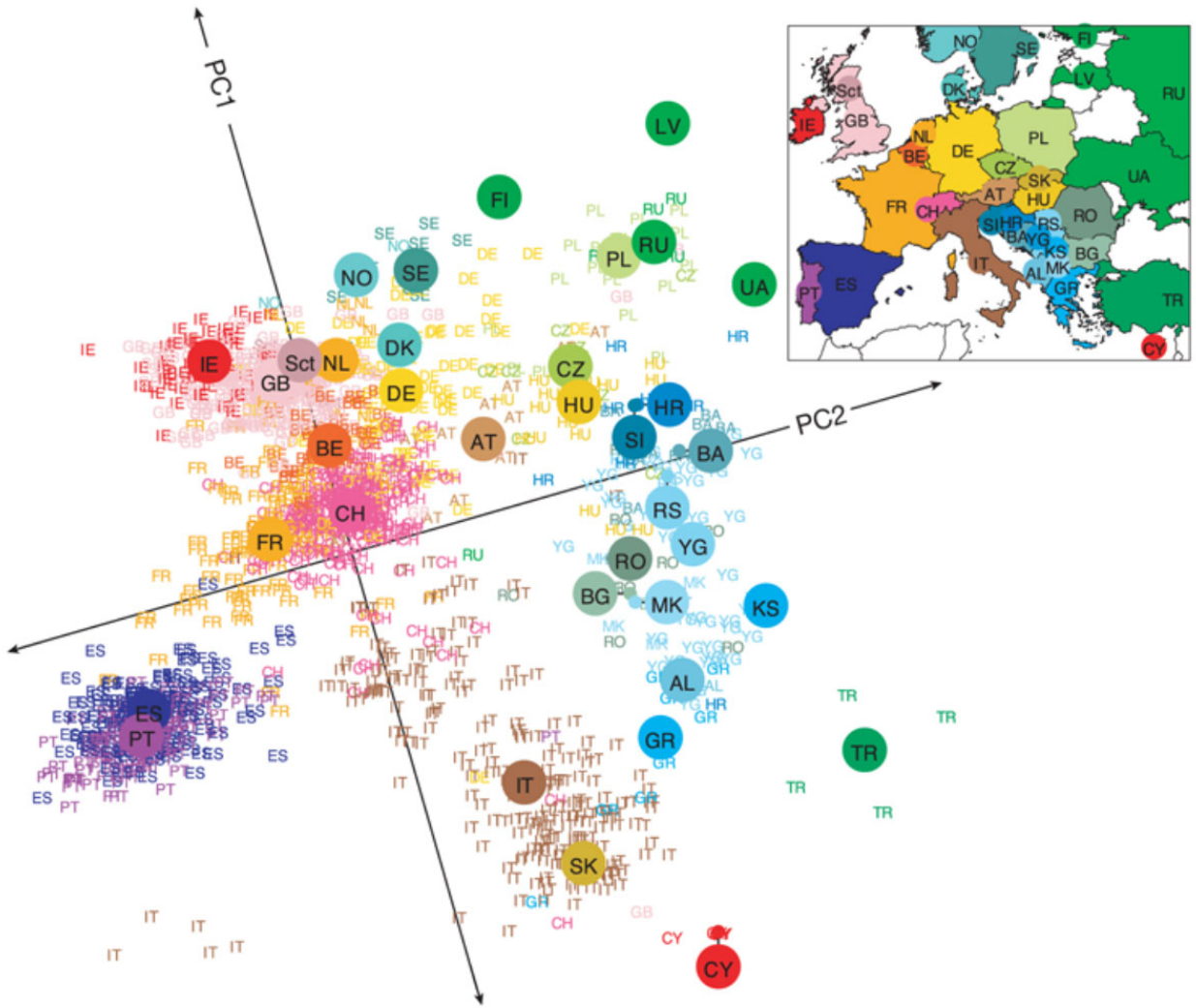
# e.g. of Dimensionality reduction

Single-cell transcriptomics (single-cell RNA sequencing): samples are cells, features are genes.



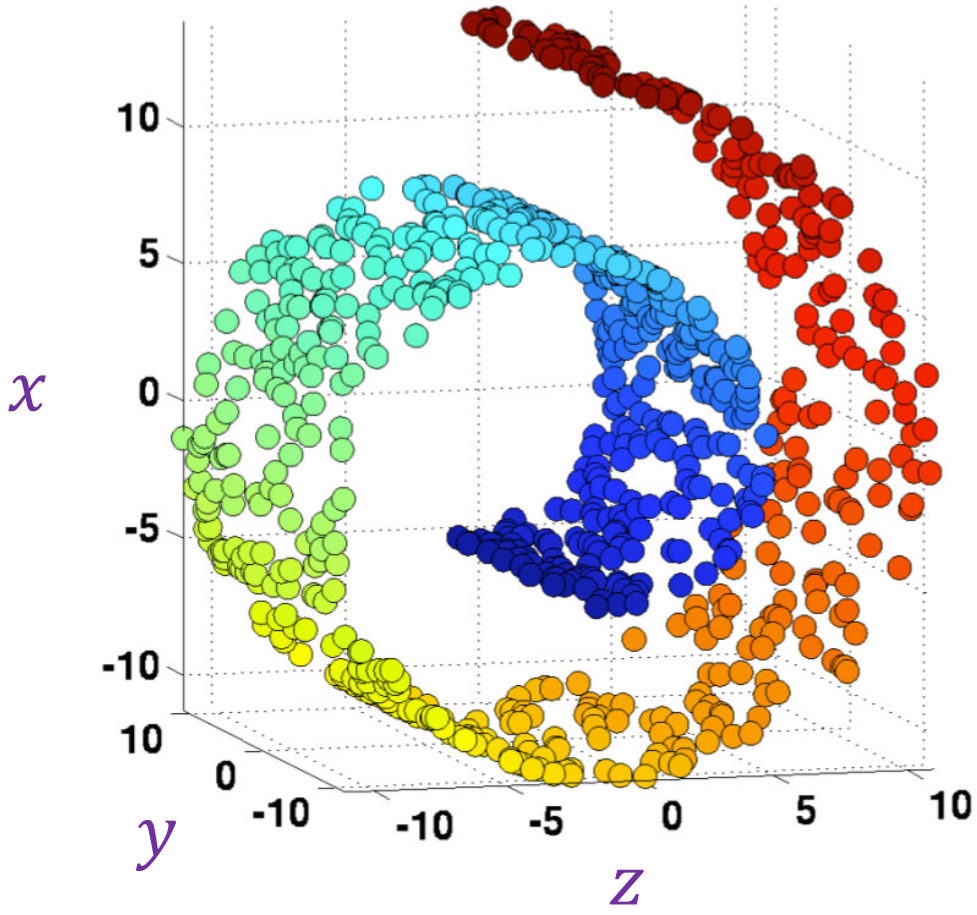

Zeisel et al. (2018)
$n \approx 500{,}000$

# e.g. of Dimensionality reduction



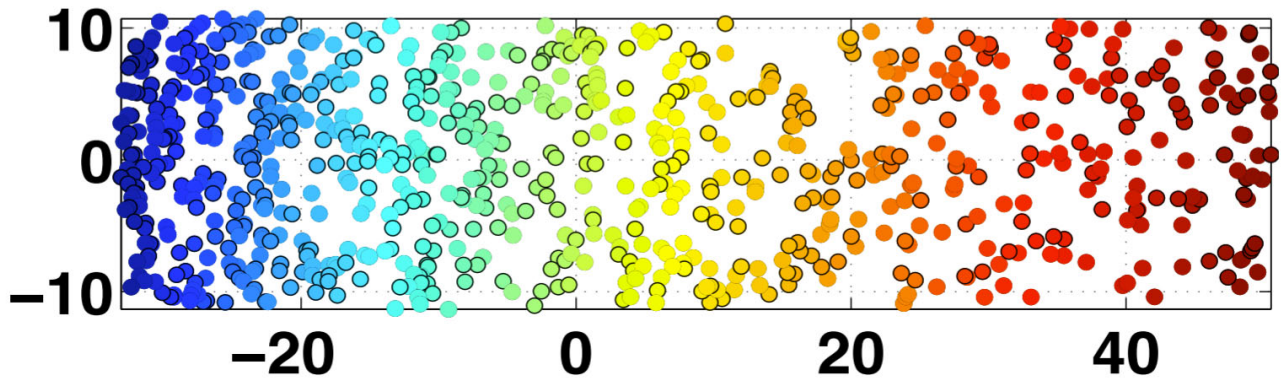

Novembre *et al Nature* 2008

# What dimensionality do these points live in?



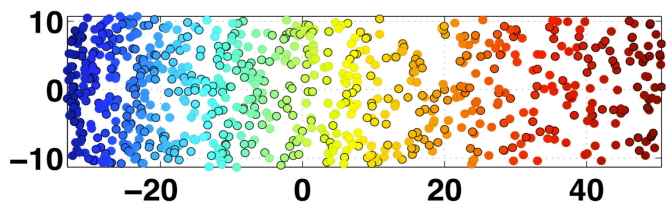

$x \in \mathbb{R}^3 = [x, y, z]$.

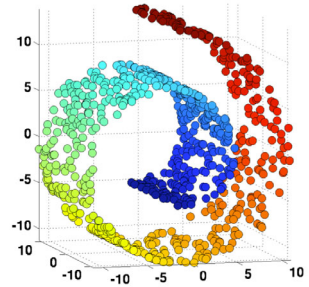
But could uniquely describe each point with just 2 *coordinates*.




*Weinberger et al ICML 2004*

# The "manifold hypothesis"




- "*High dimensional data tend to lie in the vicinity of a low dimensional manifold.*" e.g. [Fefferman 2013]
- Manifold: roughly speaking, a space that locally feels like a Euclidean space.
- For us: a manifold is a lower dimensional part of the observation space in which the data tend to lie.
- "*embedding* the data in a lower dimensional manifold", or "an *embedding* of the data".

# What dimensionality do these points live in?

- 5000 faces, $\boldsymbol{x_i} \in \mathbb{R}^{\mathbf{32\times32=1024}}$
- How low a dimension do you think we can go and still "keep" the image?
- Turns out we can go down to $\sim\mathbf{100}$ from the "ambient" $\mathbf{1024}$ dimensions!
- Trick: carefully create 100 special "basis" images.
- Principal Components Analysis (PCA) will yield the PC basis vectors.

*https://mikedusenberry.com/on-eigenfaces*

# What dimensionality do these points live in?

- 5000 f
- How l we can g
- Turns dimen
- Trick: asis" image
- Princip A) will tell us

PCA "basis" images, $x \in \mathbb{R}^{1024}$

https://mikedusenberry.com/on-eigenfaces

# What dimensionality do these points live in?

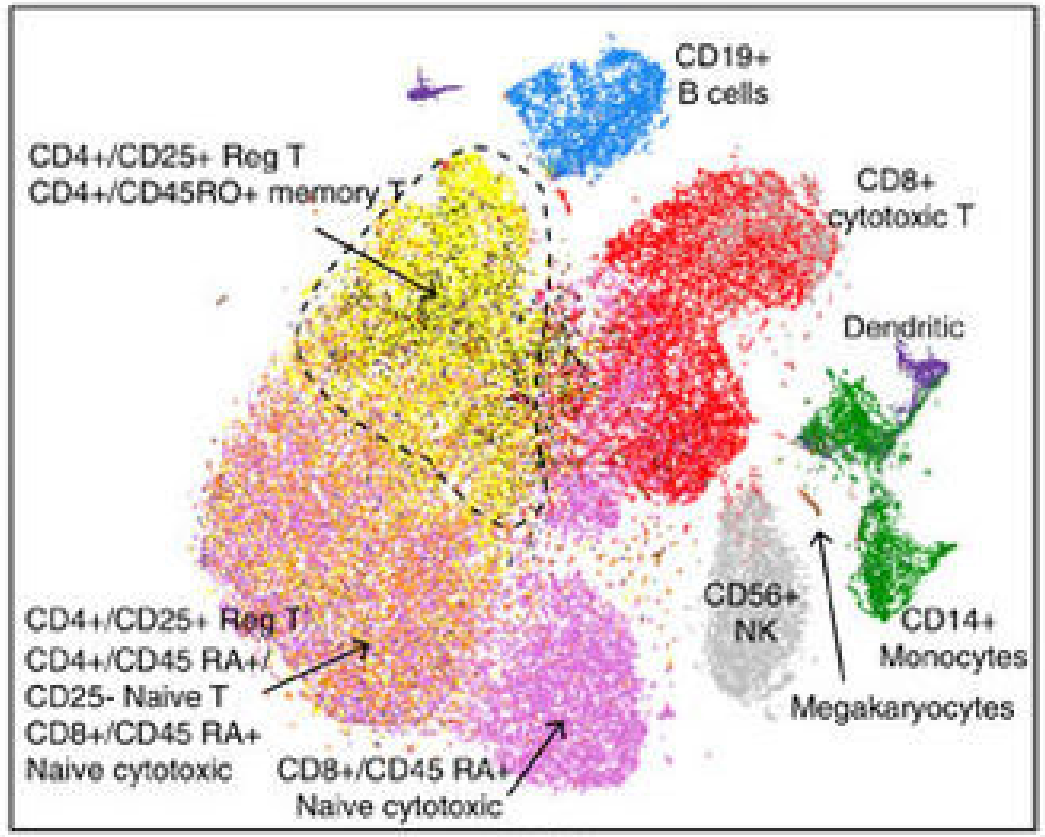
original faces, $x \in \mathbb{R}^{1024}$

approximate faces, $x' \in \mathbb{R}^{100}$

https://mikedusenberry.com/on-eigenfaces

# Why might we want to reduce dimensionality?

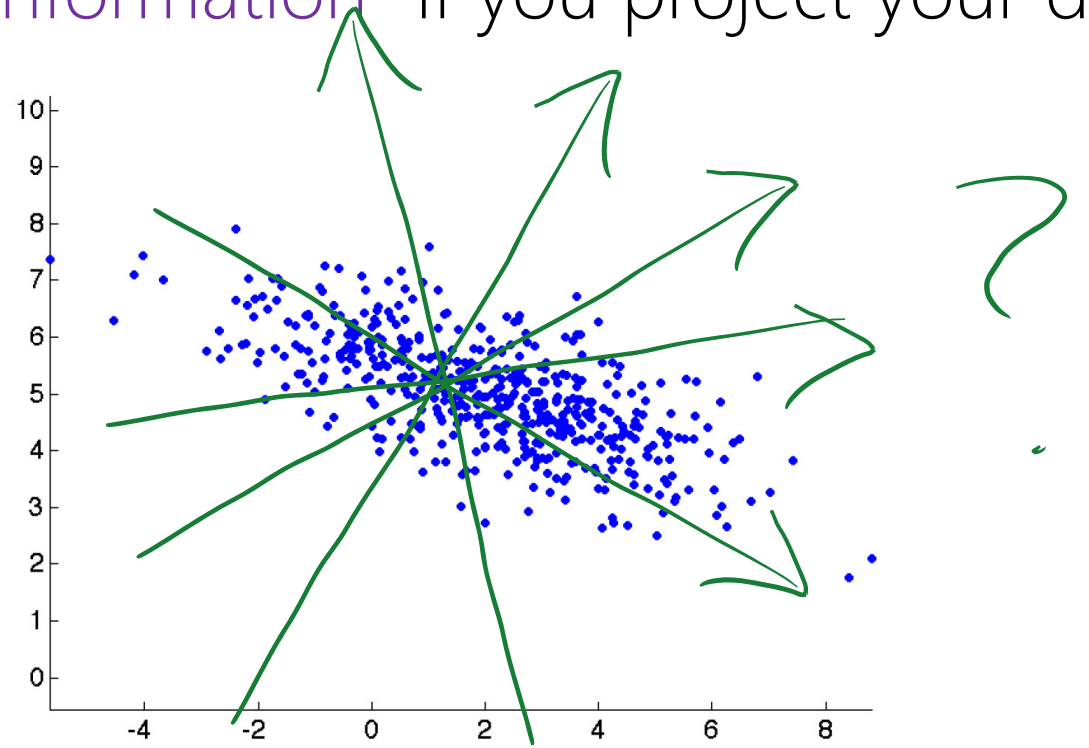
- Visualization, e.g. 2D plots.
- To denoise the data, or remove systematic artifacts (big one in biology).
- To compress the data (e.g. audio, images).
- To speed up supervised learning, or other analyses.

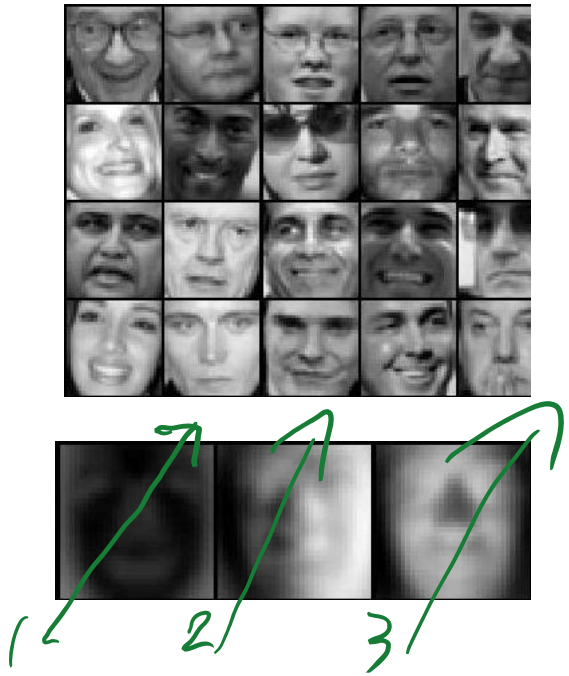



https://www.nature.com/articles/ncomms14049/figures/3

# Principal Components Analysis (PCA)

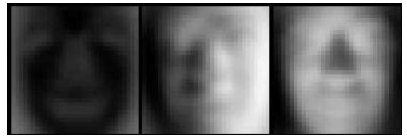
- Those special faces, "eigenfaces", are the Principal Component basis vectors that PCA yields:
- Look for the direction in the original space that "retains most of the "information" if you project your data down on to it.



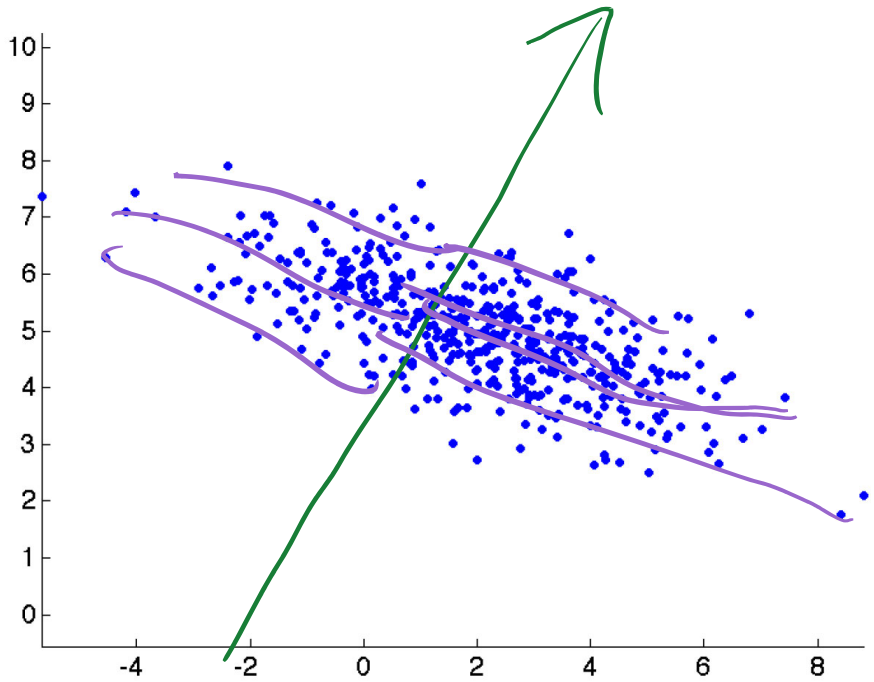
# Principal Components Analysis (PCA)

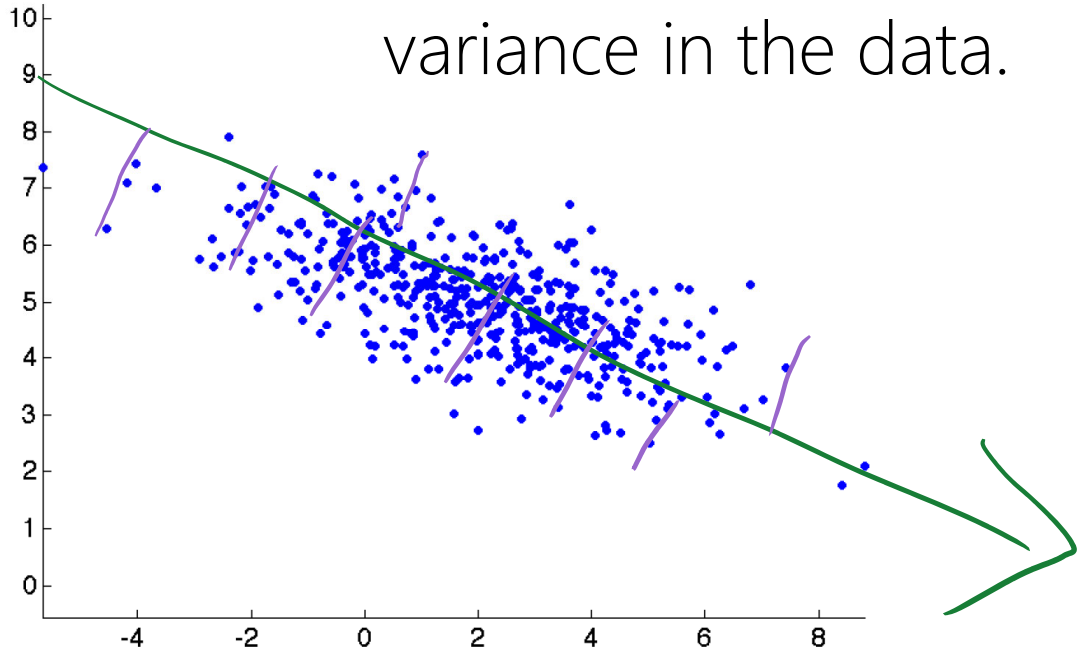
- "eigenfaces" are the PCA basis



- Look for the direction in the original space that "retains most of the "information" if you project your data down on to it.

Direction with lowest *reconstruction loss*, is the direction with maximal variance in the data.

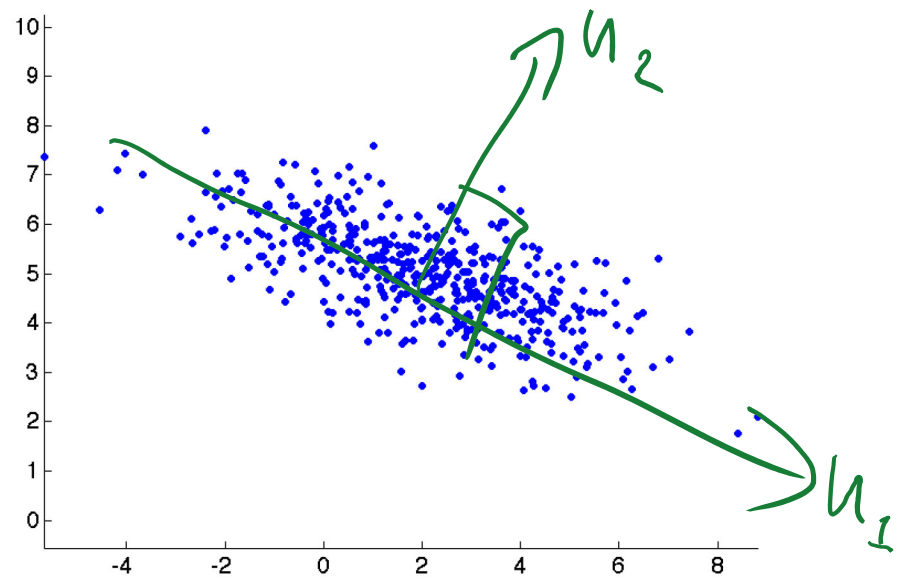


# Principal Components Analysis (PCA)

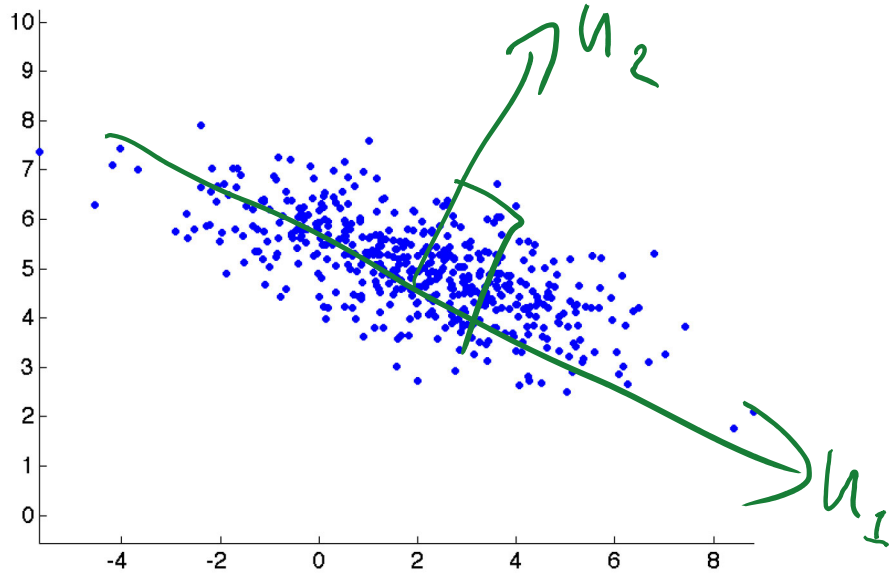
Recursively apply this idea to find 2nd best direction, then 3rd best:

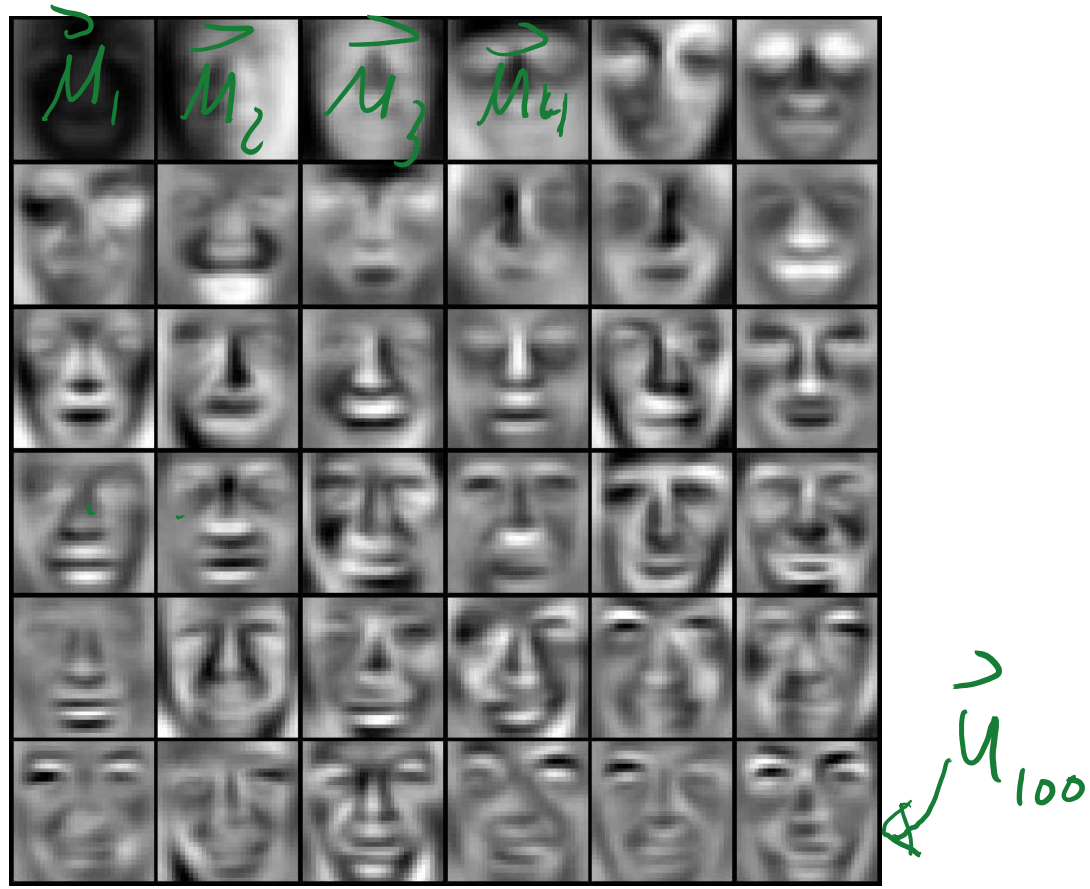
- 2nd direction should be orthogonal to the first… and…
- … be direction of most variance subject to that constraint.
- What's the maximum number of such directions we can find?



# Principal Components Analysis (PCA)

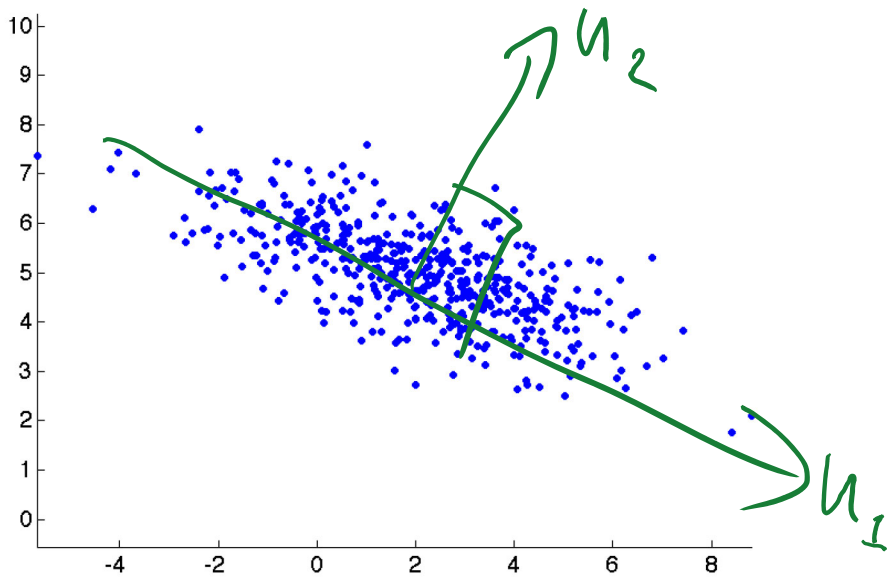
- Each of the 100 eigenfaces was one of these special directions in the original 1024-dimensional image space.



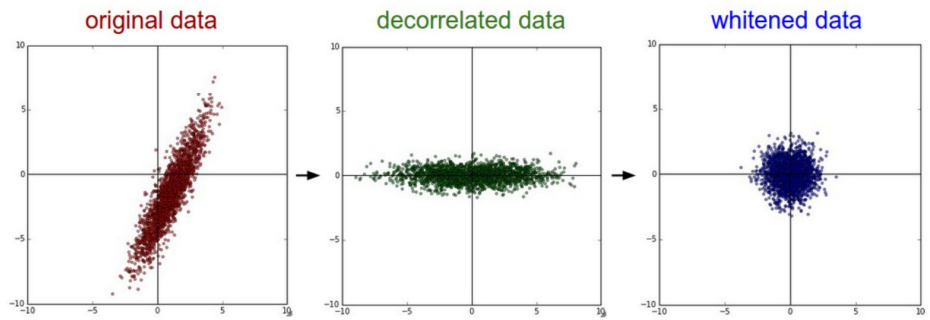
# Principal Components Analysis (PCA)

Is this starting to remind you of anything?




## Diagonalizing a MVG ("sphering")

- To sphere a MVG is to make all its contour lines be spheres (also called "whitening").
- Thus we need to make the ellipses look like spheres.
- To do this, we need to understand how to diagonalize a matrix.




Can be derived with MLE for params $\mu, W$, assuming $p(x \in \mathbb{R}^d) = N(\mu + xW; I\sigma^2)$, for $\sigma^2 \to 0$ and $W \in \mathbb{R}^{d \times k}$.

# Principal Components Analysis (PCA)

Recall: to diagonalize a MVG distribution, we made use of a special factorization of its covariance matrix, $\boldsymbol{A} = \boldsymbol{QDQ^T}$, an "eigen" or "spectral"-decomposition:

## Linear Algebra: Diagonalizing a matrix

Spectral theorem: When A is symmetric $A = A^T$

$A = \varphi D \varphi^T$ with real eigenvalues in D

and orthonormal vectors in $S = \varphi$

$\varphi$ is an orthonormal matrix $[q_1\ q_2 \cdots q_n]$

$\Downarrow$

$\varphi^{-1} = \varphi^T$

(rotations & reflections)

each is of length 1

any two are orthogonal

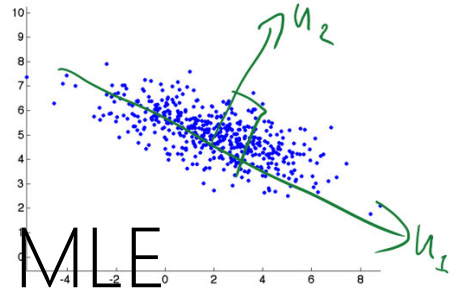
true for col and row

$A = \varphi D \varphi^{-1}$

Can use this to do PCA.

(Even if data are not Gaussian).

# PCA overview



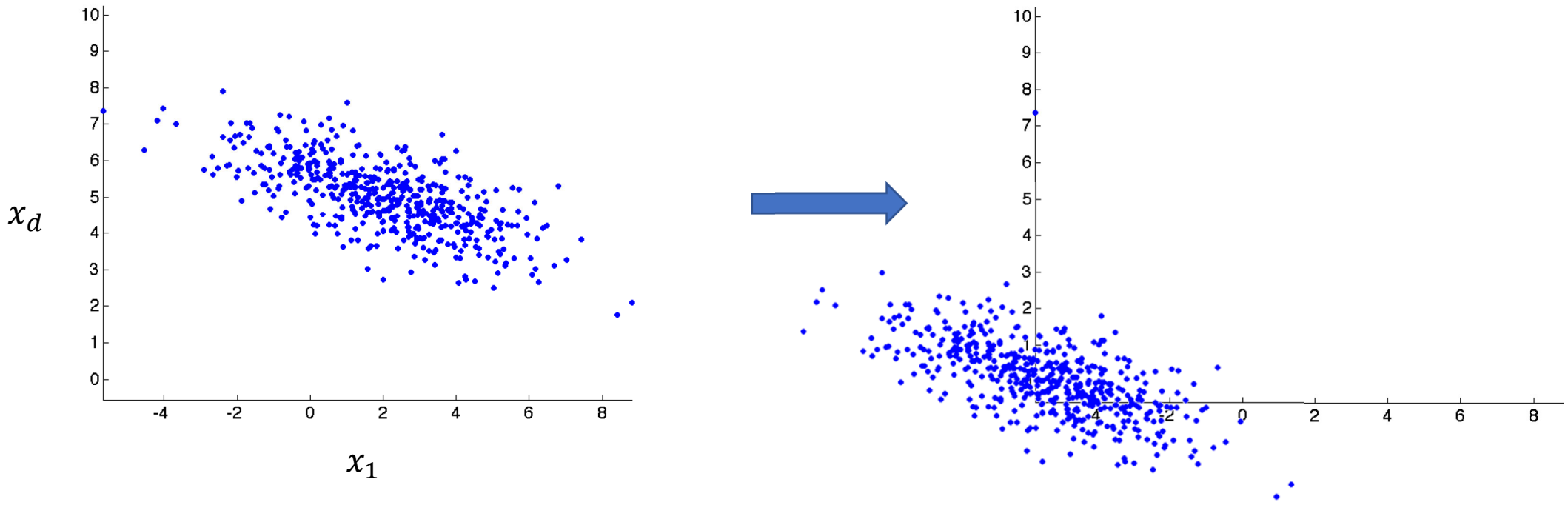

Intuitively: pretend our data are Gaussian; compute the MLE "covariance matrix"; and pick off the directions with the top $\boldsymbol{k}$ eigenvalues (all $\boldsymbol{\lambda_i \geq 0}$ because covariance is PSD).

- Given data matrix, $\boldsymbol{X \in \mathbb{R}^{n \times d}}$.
- Construct, $\boldsymbol{X^T X \in \mathbb{R}^{d \times d}}$ (after mean-centering each feature).
- Apply spectral theorem, $\boldsymbol{X^T X = QDQ^T}$ to pick off $\boldsymbol{k}$ directions.
- Now approximate this covariance matrix with the "best" low rank approximation to it (rank $\boldsymbol{k}$).
- Best: lowest "reconstruction loss", and highest variance directions.

# PCA step-by-step

Given $\boldsymbol{n}$ data points of dimension $\boldsymbol{d}$, $X \in \mathbb{R}^{n \times d}$, to perform PCA, we:

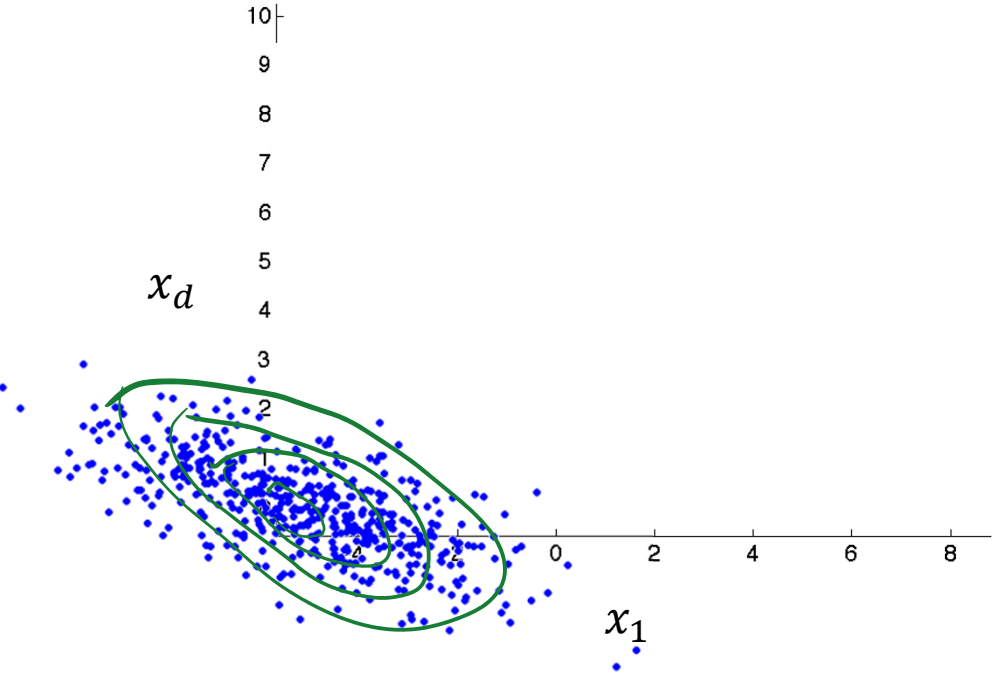
1. Center the data (make each feature zero mean), $\bar{X} = X - [\overline{x_1}, \ldots, \overline{x_d}]$. (We will continue on only with the matrix $\bar{X}$).



# PCA step-by-step

Given $\boldsymbol{n}$ data points of dimension $\boldsymbol{d}$, $X \in \mathbb{R}^{n \times d}$, to perform PCA, we:

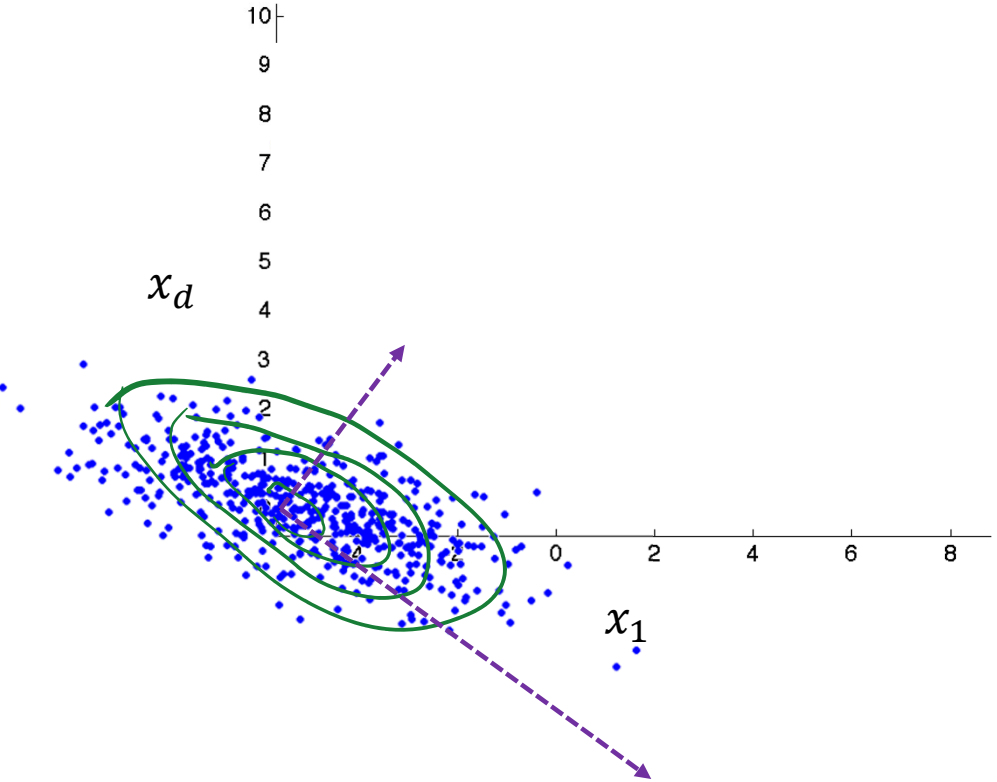
2. Compute the covariance matrix, $\Sigma = \bar{X}^T \bar{X}$.



# PCA step-by-step

Given $\boldsymbol{n}$ data points of dimension $\boldsymbol{d}$, $X \in \mathbb{R}^{n \times d}$, to perform PCA, we:

2. Compute the covariance matrix, $\Sigma = \bar{X}^T \bar{X}$.
3. Compute $\bar{X}^T \bar{X} = QDQ^T$ to get eigenvectors (aka *principal directions*).

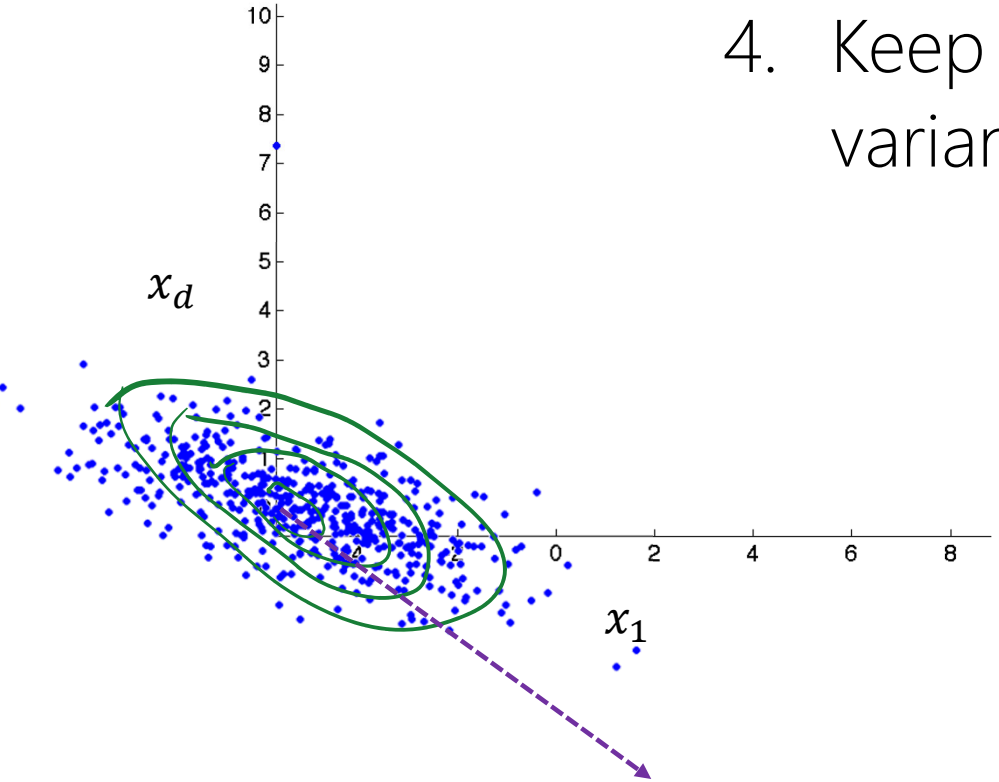



- *This decomposition is typically implemented with a call to "eig" function (linalg package in python), but can also be obtained from "svd".*
- *Are the same for PSD matrices, but svd may be more stable.*

# PCA step-by-step

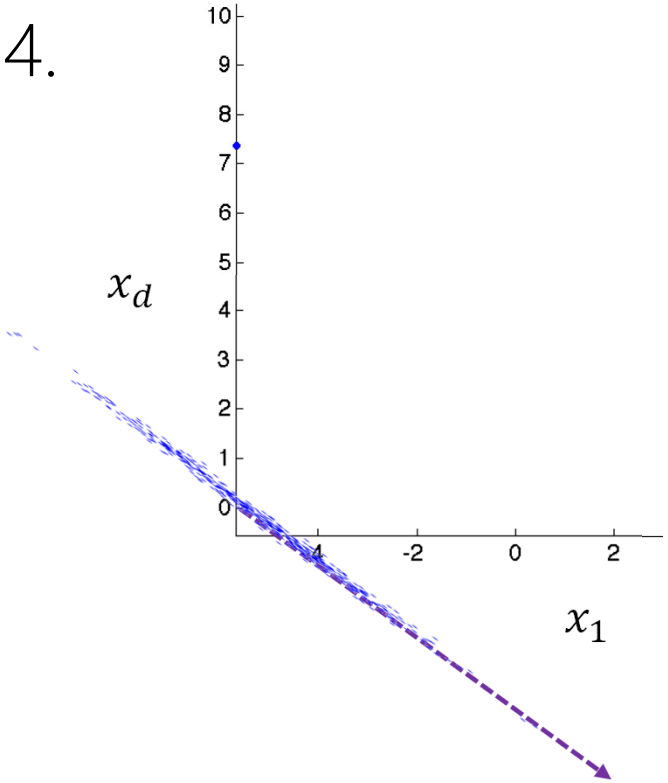
Given $\boldsymbol{n}$ data points of dimension $\boldsymbol{d}$, $X \in \mathbb{R}^{n \times d}$, to perform PCA, we:

2. Compute the covariance matrix, $\Sigma = \bar{X}^T \bar{X}$.
3. Compute $\bar{X}^T \bar{X} = QDQ^T$ to get eigenvectors (aka *principal component axes*).
4. Keep the $\boldsymbol{k}$ eigenvectors, $Q_k \equiv Q_{:,1:k}$, with the most variance (highest eigenvalues in $D$).



# PCA step-by-step

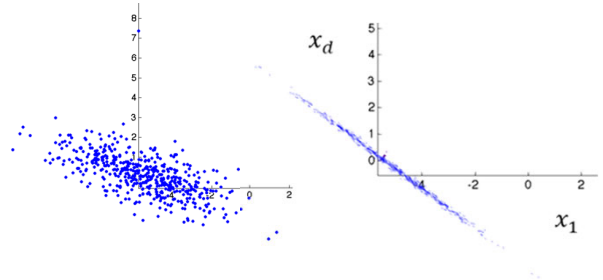
Given $\boldsymbol{n}$ data points of dimension $\boldsymbol{d}$, $X \in \mathbb{R}^{n \times d}$, to perform PCA, we:

2. Compute the covariance matrix, $\Sigma = \bar{X}^T \bar{X}$.
3. Compute $\bar{X}^T \bar{X} = QDQ^T$ to get eigenvectors (aka *principal component axes*).
4. 

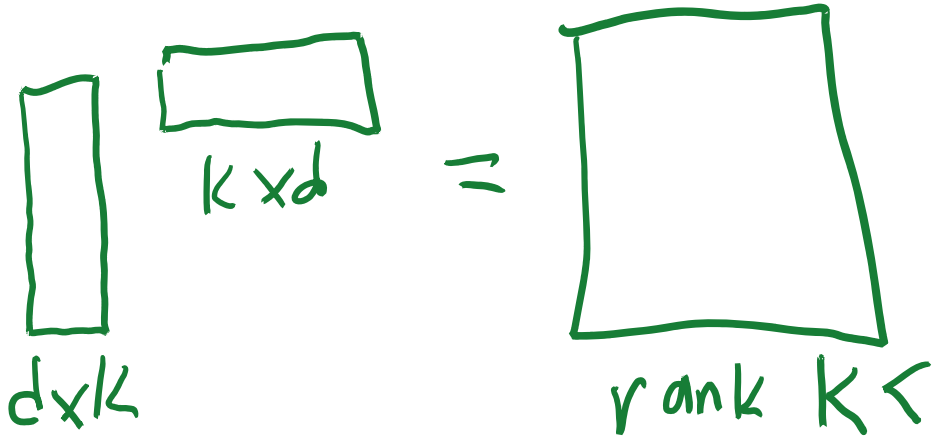

   4. Keep the $\boldsymbol{k}$ eigenvectors, $Q_k \equiv Q_{:,1:k}$, with the most variance (highest eigenvalues in $D$).
   5. Project your points (original, or new ones) down to this subspace, $\bar{X}_k = \bar{X} Q_k \in \mathbb{R}^{n \times k}$, these are your *principal components scores*.

   Final, dimensionality-reduced data is.

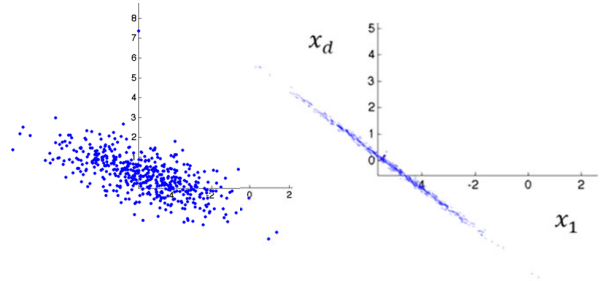
# PCA step-by-step



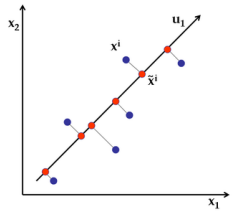

- Final, dimensionality-reduced data is $\bar{X}_k = \bar{X}Q_k \in \mathbb{R}^{n\times k}$.
- What if we wanted our data in the original dimension ($d$), but with only the information retained from our PCA-k analysis? (e.g. reconstructed faces)
- We need to "reconstruct" the original points. We can do this by expanding back from PCA basis to original basis, but with only the first $k$ PCA basis vectors, $\bar{X}_{recon-k} = \underbrace{\bar{X}_k}_{n\times k}\underbrace{Q_k^T}_{k\times d} = \bar{X}\underbrace{Q_k Q_k^T}_{d\times d} \in \mathbb{R}^{n\times d}$



# PCA step-by-step




- Final, dimensionality-reduced data is $\bar{X}_k = \bar{X}Q_k \in \mathbb{R}^{n\times k}$.
- What if we wanted our data in the original dimension ($d$), but with only the information retained from our PCA-k analysis? (e.g. reconstructed faces)
- We need to "reconstruct" the original points. We can do this by expanding back from PCA basis to original basis, but with only the first $k$ PCA basis vectors, $\bar{X}_{recon-k} = \bar{X}_k Q_k^T = \bar{X}Q_k Q_k^T \in \mathbb{R}^{n\times d}$
- Now we talk about the "reconstruction loss", as the difference between original and reconstructed data, $\left\|\bar{X}_{recon-k} - \bar{X}\right\|_{\mathrm{F}}$.
- The larger the reconstruction loss, the more information we have lost.




and Frobenius norm as

$$\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2} = \sqrt{\mathrm{tr}(X^\top X)} = \sqrt{\sum s_i^2},$$

where $s_i$ are singular values of $X$, i.e. diagonal

# Original, and reconstructed faces

original faces, $x \in \mathbb{R}^{1024}$

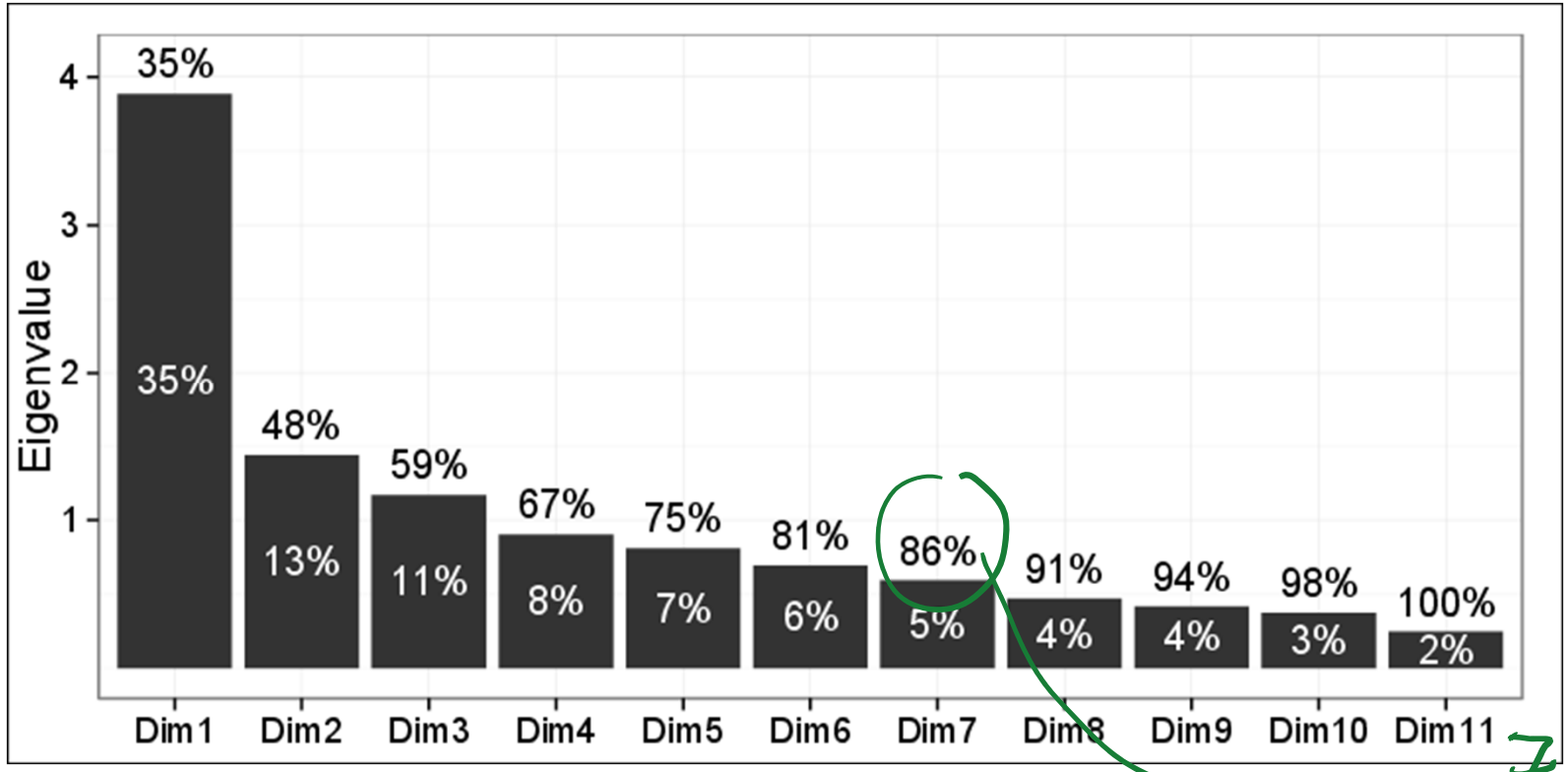
reconstructed faces

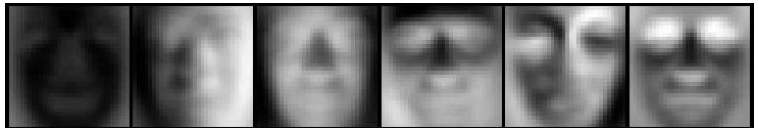
$x' \in \mathbb{R}^{100} \rightarrow x_k \in \mathbb{R}^{1024}$

https://mikedusenberry.com/on-eigenfaces

# PCA: % variance explained to help pick hyper-parameter, $k$

$X^T X = Q D Q^T$



If you normalize the eigenvalues in $D$ by their sum, then they correspond to % variance explained.

$$V_i \equiv \frac{D_{i,i}}{\sum_j D_{j,j}}$$

$$\sum_{j=1}^{7} V_j$$

https://stats.stackexchange.com/questions/133451/is-there-any-required-amount-of-variance-captured-by-pca-in-order-to-do-later-an

# PCA: reconstruction loss and % variance

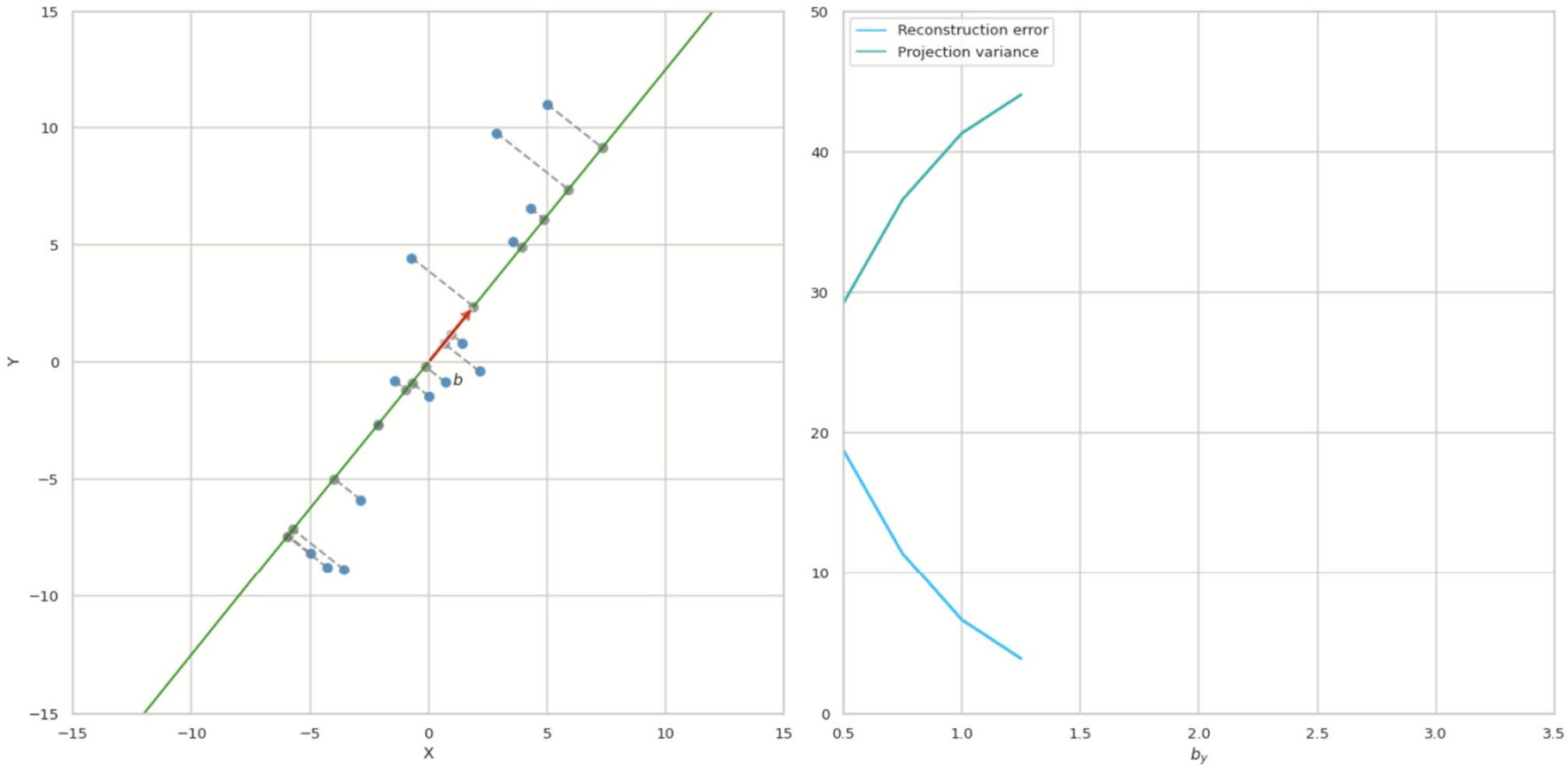


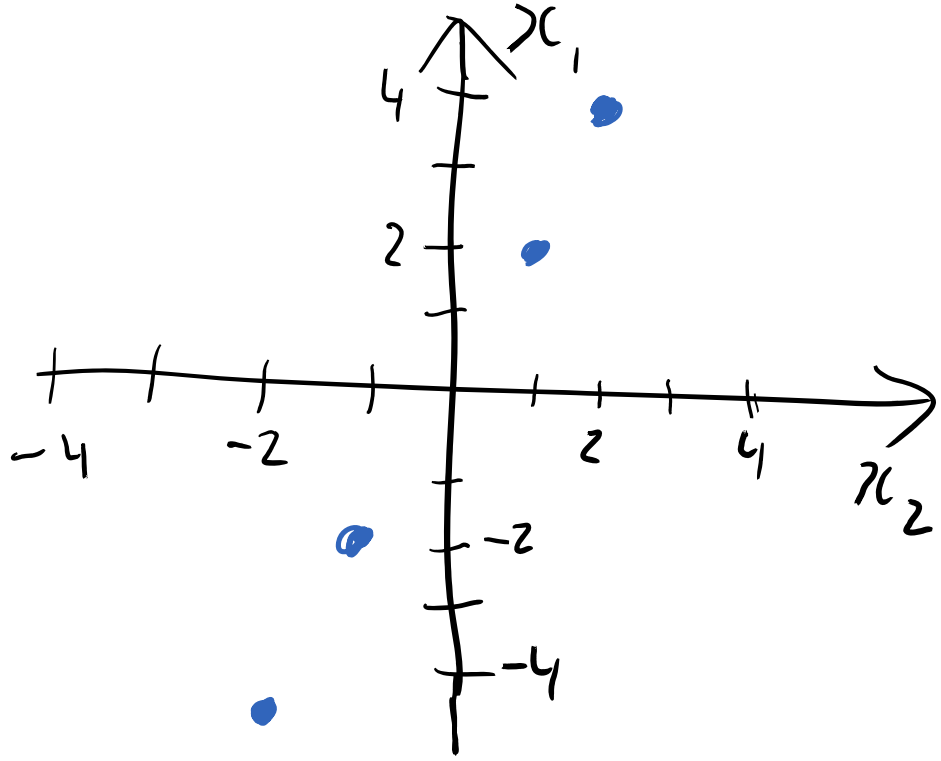

Fig.5. We rotate basis vector b and project data on corresponding subspace, then calculate reconstruction error L and projection variance Var(λ).

*https://towardsdatascience.com/dimensionality-reduction-with-pca-from-basic-ideas-to-full-derivation-37921e13cae7*

# Example of PCA

$$\underset{n \times d}{X} = \begin{array}{c} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} \\ \begin{array}{cc} x_1 & x_2 \end{array} \end{array}$$

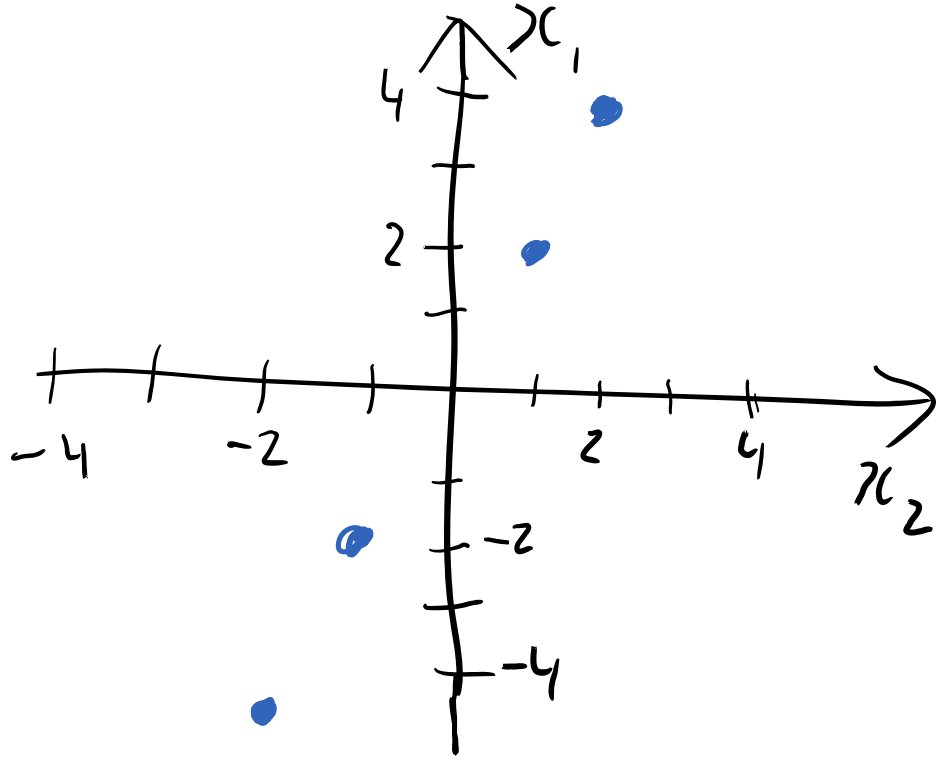
$d = 2$
$n = 4$



# Example of PCA

$$\underset{n \times d}{X} = \begin{array}{c} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} \\ \begin{array}{cc} x_1 & x_2 \end{array} \end{array}$$

$d = 2$
$n = 4$

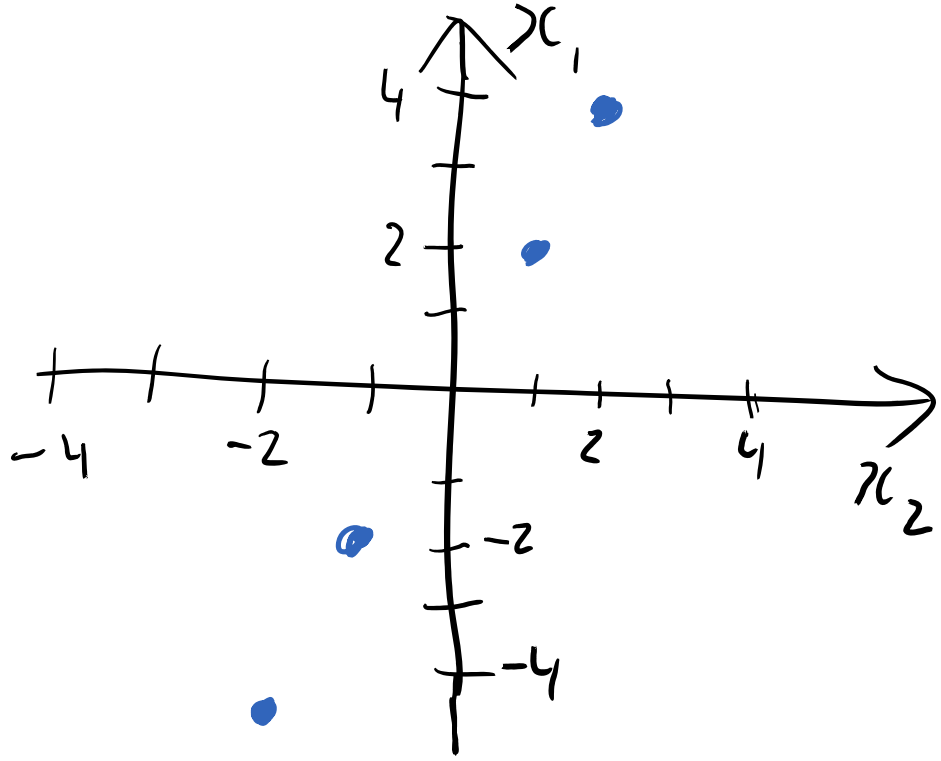



mean of each dimension $[0 \quad 0] \Rightarrow \bar{X} = X$

# Example of PCA

$$\underset{n \times d}{X} = \begin{array}{c} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} \\ \begin{array}{cc} x_1 & x_2 \end{array} \end{array}$$

$d = 2$
$n = 4$

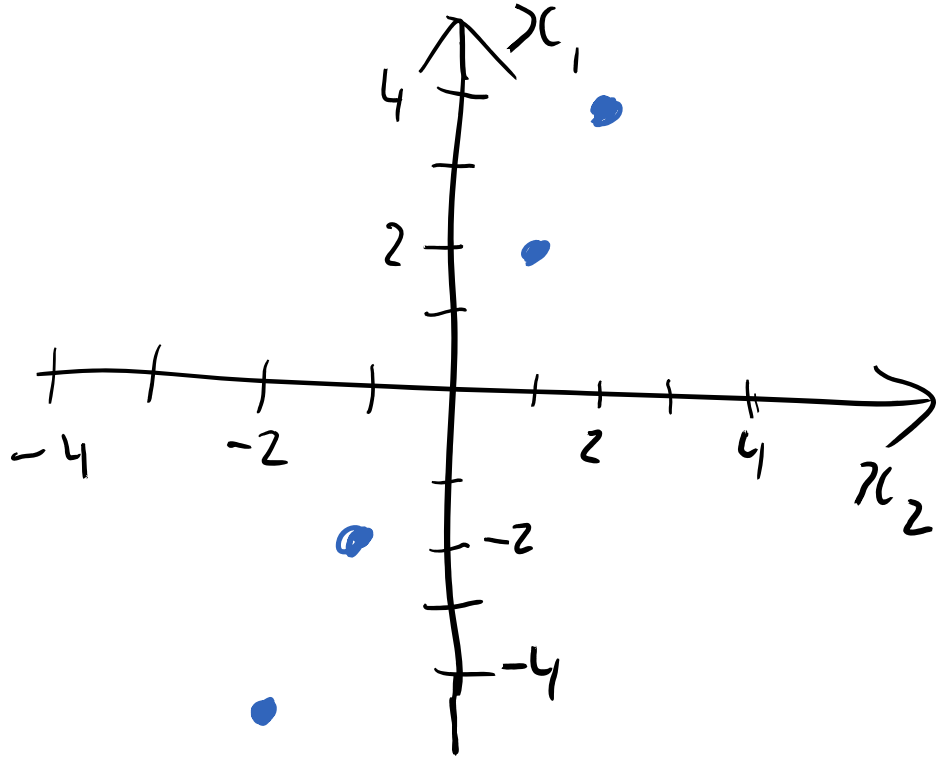



$$\underset{d \times d}{X^T X} = \begin{bmatrix} 1 & 2 & -1 & -2 \\ 2 & 4 & -2 & -4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 20 & 40 \end{bmatrix}$$

# Example of PCA

$$\underset{n \times d}{X} = \begin{array}{c} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} \\ \begin{array}{cc} x_1 & x_2 \end{array} \end{array}$$

$$\underset{d \times d}{X^T X} = \begin{bmatrix} 10 & 20 \\ 20 & 40 \end{bmatrix}$$



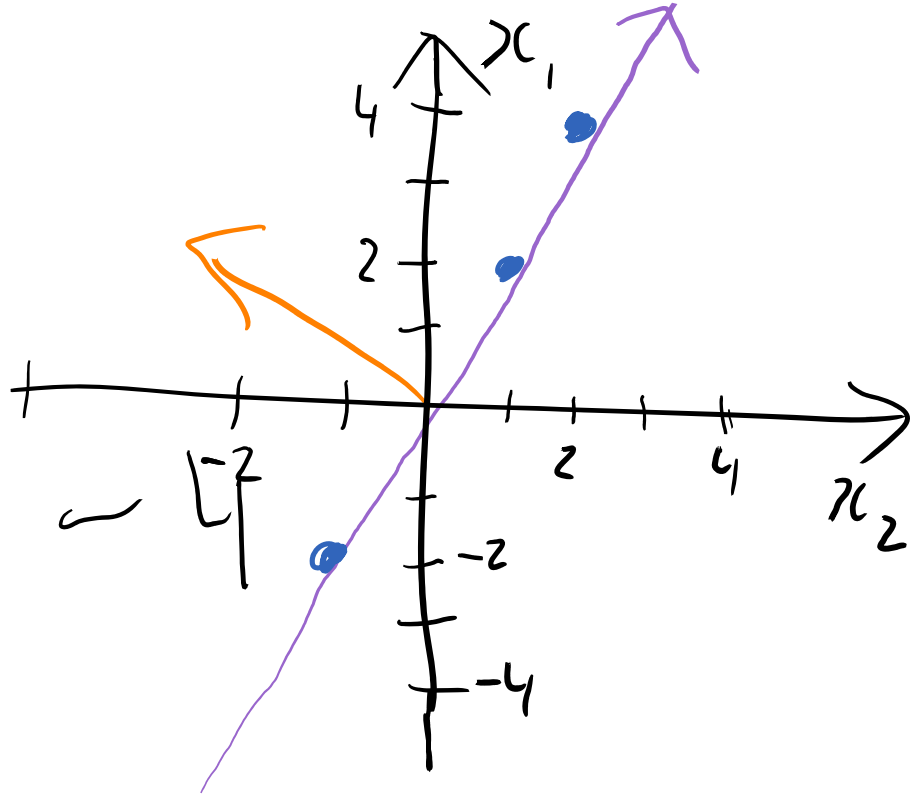
# Example of PCA

$$\underset{n \times d}{X} = \begin{array}{c} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} \\ \begin{array}{cc} x_1 & x_2 \end{array} \end{array}$$

linalg.eig($X^T X$)

$\Rightarrow$ eigenvalues $\boxed{50 \quad 0}$

$$\underset{d \times d}{X^T X} = \begin{bmatrix} 10 & 20 \\ 20 & 40 \end{bmatrix}$$



# Example of PCA

$$\underset{n \times d}{X} = \begin{array}{c} \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ -1 & -2 \\ -2 & -4 \end{bmatrix} \\ \begin{array}{cc} x_1 & x_2 \end{array} \end{array}$$

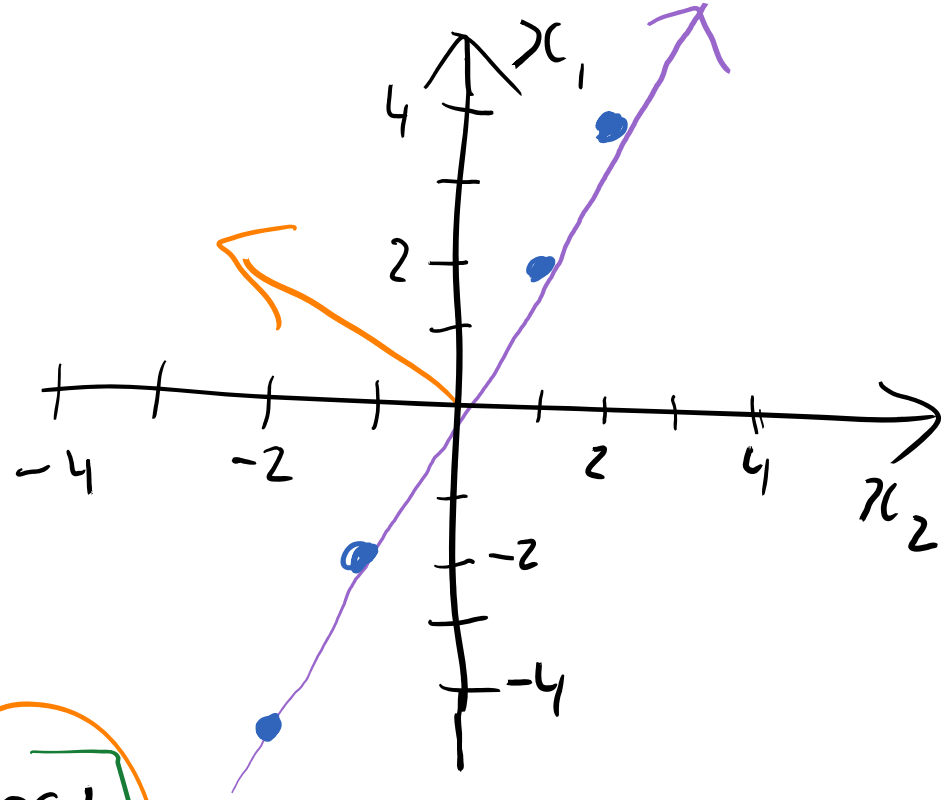
linalg.eig($X^T X$)

$\Rightarrow$ eigenvalues $\boxed{50 \quad 0}$

& eigenvectors

$$\underset{d \times d}{X^T X} = \begin{bmatrix} 10 & 20 \\ 20 & 40 \end{bmatrix}$$

$$\begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$



# Example of PCA (take 2, noisy data)

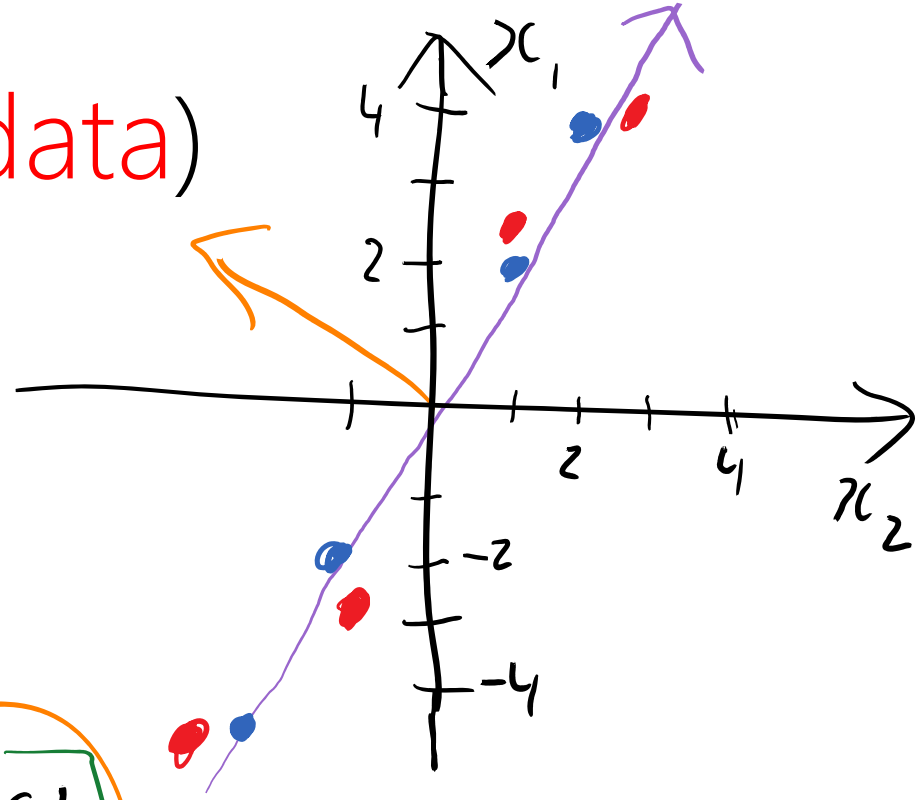
$$\underset{n \times d}{X} = \begin{array}{cc} \begin{bmatrix} 1.07 & 2.16 \\ 2.05 & 4.1 \\ -0.93 & -2.03 \\ -1.97 & -4.03 \end{bmatrix} \\ x_1 \quad\quad x_2 \end{array}$$

linalg.eig$(X^T X)$

$\Rightarrow$ eigenvalues $[50 \quad 0]$

& eigenvectors

$$\begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$



# Example of PCA (take 2, noisy data)

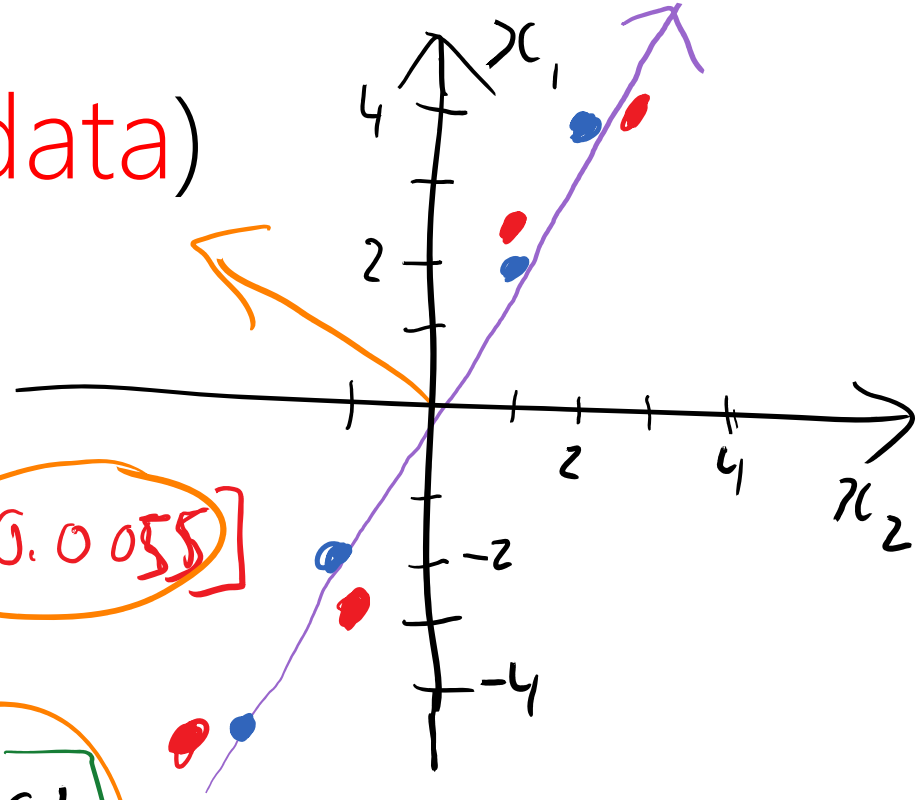
$$\underset{n \times d}{X} = \begin{array}{cc} \begin{bmatrix} 1.07 & 2.16 \\ 2.05 & 4.1 \\ -0.93 & -2.03 \\ -1.97 & -4.03 \end{bmatrix} \\ x_1 \quad x_2 \end{array}$$

linalg.eig$(X^T X)$

$\Rightarrow$ eigenvalues $[50 \quad 0]$

& eigenvectors $[52.36 \quad 0.0055]$

$$\begin{bmatrix} 0.439 & -0.898 \\ 0.898 & 0.439 \end{bmatrix} \qquad \begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$

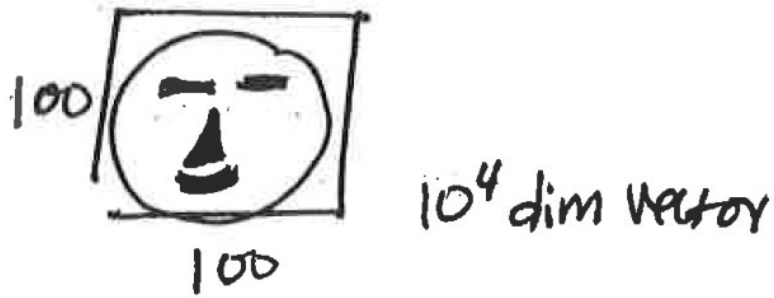


# Linear algebra for PCA

$\bar{X} \in \mathbb{R}^{n \times d}$

- Need to compute the principal axes, $Q$, of $\bar{X}^T \bar{X} = QDQ^T \in \mathbb{R}^{d \times d}$.
- This is an *eigendecomposition* of the matrix $\bar{X}^T \bar{X}$.

- Computing its eigendecomposition has time complexity $O(d^3)$.
- What if $d \gg n$? *e.g.*, images of $\mathbf{d} = \mathbf{100} \times \mathbf{100} = \mathbf{10^4}$ pixels, and $n = \mathbf{1{,}000}$ images.
- Could we do something cheaper?
- Yes. Need to understand the SVD.



# Linear algebra for PCA

$\bar{X} \in \mathbb{R}^{n \times d}$

Recall the spectral theorem (principal axis theorem) from MVG lecture, which gives a *spectral (eigen) decomposition*:.

When $A$ is symmetric $A = A^T$

$A = \varphi D \varphi^T$ with real eigenvalues in $D$

and orthonormal vectors in $S = \varphi$

$\varphi$ is an orthonormal matrix $[q_1 \; q_2 \; \dots \; q_n]$

$\Downarrow$

$\varphi^{-1} = \varphi^T$

(rotations & reflections)

each is of length 1

any two are orthogonal

true for col and row

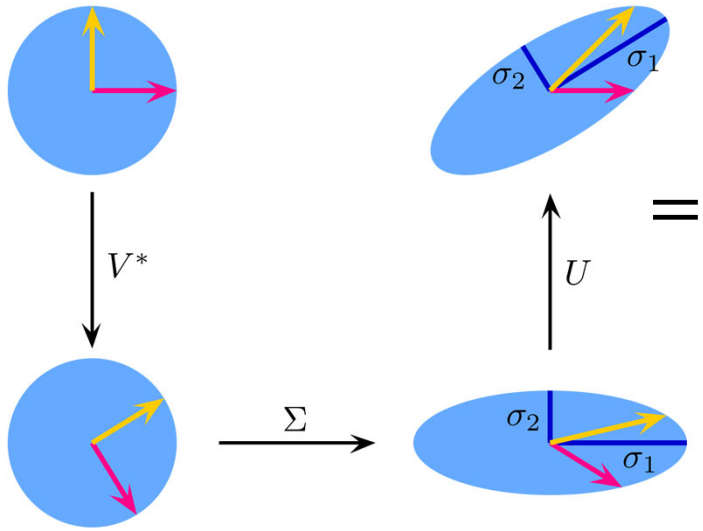
$A = \varphi D \varphi^{-1}$

- The covariance matrix for PCA, $\bar{X}^T \bar{X}$, is symmetric (and PSD).
- It turns out, there is a generalization of the spectral decomposition, for non-symmetric and non-square matrices, the SVD that will be helpful.

# Singular Value Decomposition (SVD)

**Can think of M as linear transformation broken down into three steps,** by looking at its effect on the unit disc and the two canonical unit vectors $e_1$ and $e_2$:

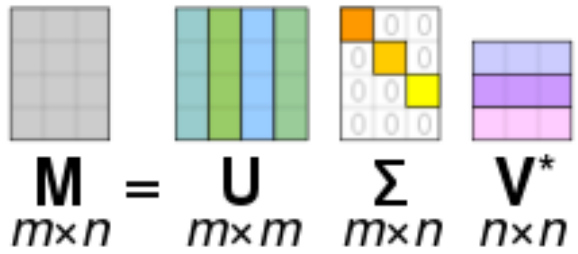
1. **Left:** $\boldsymbol{V^T}$ rotates the disc and unit vectors.
2. **Bottom: Σ** stretches scales axes by $\sigma_i = \Sigma_{i,i}$ (singular values).
3. **Right: U** performs another rotation.




$$M = U \cdot \Sigma \cdot V^*$$

$$= M \begin{bmatrix} e_2 \\ e_1 \end{bmatrix} = U \Sigma V^* \begin{bmatrix} e_2 \\ e_1 \end{bmatrix}$$

Can be applied to any matrix $\boldsymbol{M}$.




$$Mv_1 = \Sigma_{1,1} u_1$$
$$Mv_2 = \Sigma_{2,2} u_2$$
$$\dots$$
$$Mv_r = \Sigma_{r,r} u_r$$

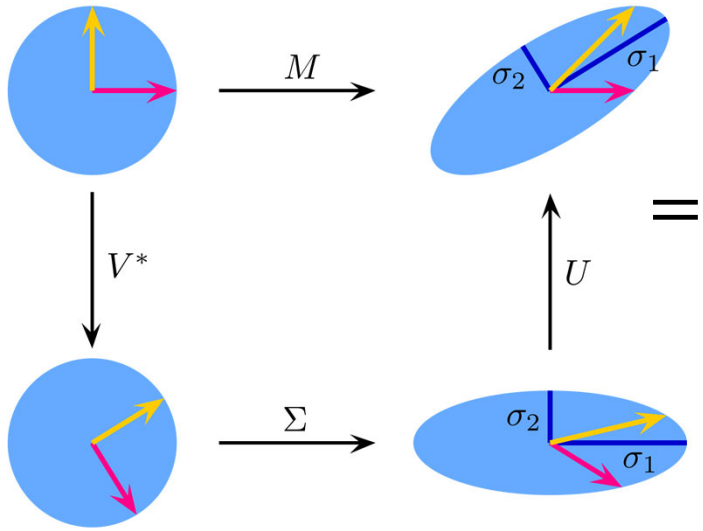
$$r = rank(M) \leq \min(m, n)$$

https://en.wikipedia.org/wiki/Singular_value_decomposition

# Singular Value Decomposition (SVD)

**Can think of M as linear transformation broken down into three steps,** by looking at its effect on the unit disc and the two canonical unit vectors $e_1$ and $e_2$:

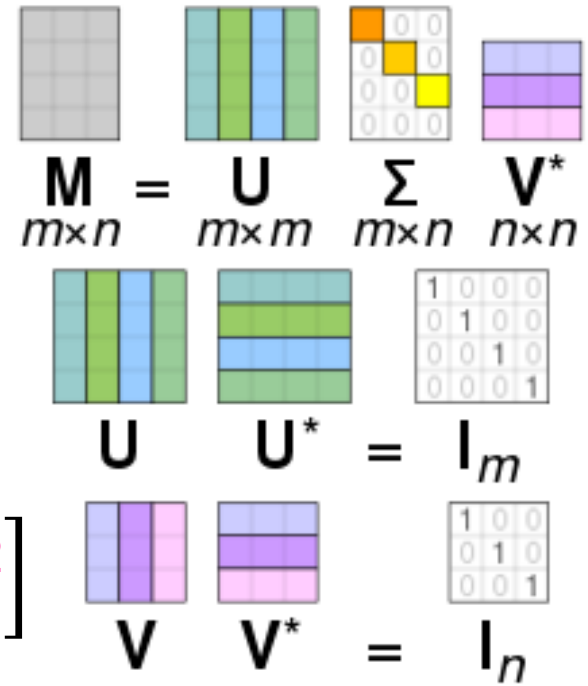
1. **Left:** $\boldsymbol{V^T}$ rotates the disc and unit vectors.
2. **Bottom: Σ** stretches scales axes by $\sigma_i = \Sigma_{i,i}$ (singular values).
3. **Right: U** performs another rotation.




$$M = U \cdot \Sigma \cdot V^*$$

$$= M \begin{bmatrix} e_2 \\ e_1 \end{bmatrix} = U \Sigma V^* \begin{bmatrix} e_2 \\ e_1 \end{bmatrix}$$

Can be applied to any matrix $\boldsymbol{M}$.



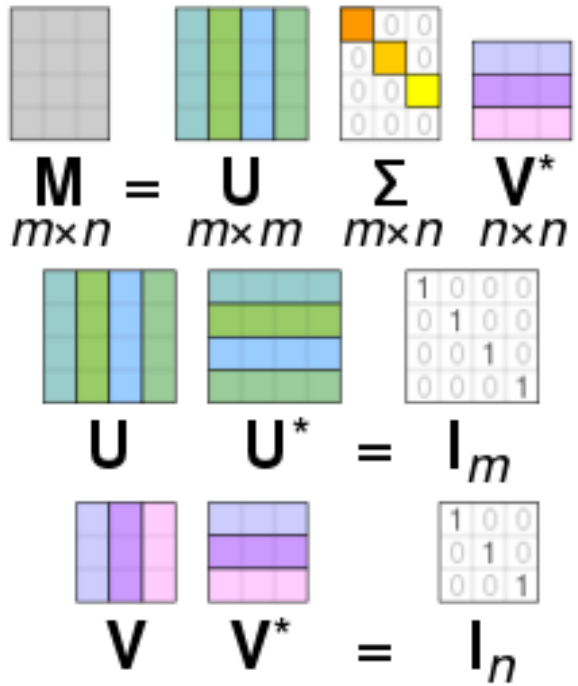

- *Has time complexity* $O(m^2n + mn^2)$.
- $\boldsymbol{\Sigma}$ *is unique (if in descending order), but* $\boldsymbol{V}$ *and* $\boldsymbol{U}$ *are generally not: e.g. sign flips.*
- *(Eigendecomposition is unique if all eigenvalues are unique)*
- *If M is square+symmetric, yields the spectral decomposition.*

https://en.wikipedia.org/wiki/Singular_value_decomposition

# Singular Value Decomposition (SVD)

- Columns in $U$ are the eigenvectors of $MM^T$, called the *left singular vectors of M* ($MM^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma^2 U^T$).
- Columns in $V$ are the eigenvectors of $M^T M$, called the *right singular vectors of M* ($M^T M = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$).
- Both spectral decompositions at once!
- Eigenvalues are the same, given by $\lambda_i = \Sigma_{i,i}^2$ ($\Sigma_{\mathbf{i}}, \mathbf{i}$ are the *singular values* of $M$):
  - Since $v_i$ is an eigenvector for $M^T M$, it follows that $M^T M v_i = \lambda_i v_i$. It follows that…
  - ….$(MM^T)Mv_i = \lambda_i(Mv_i)$ thus $Mv_i$ is an eigenvector for $MM^T$ with eigenvalue $\lambda_i$!

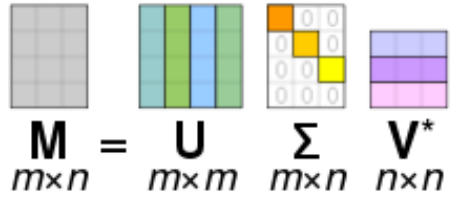
Can be applied to any matrix $M$.



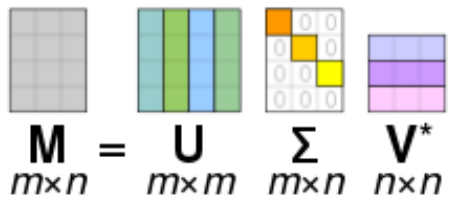
# Singular Value Decomposition (SVD)

$X \in \mathbb{R}^{n \times d}$


- Recall this example with $d \gg n$ ,e.g. $d = 10^4$ pixels, $n = 1000$ images.
- How can we make use of what we just learned to do PCA faster than the eigendecomposition $O(d^3)$?
- Instead of spectral decomposition of $X^T X$…
- …directly use SVD of the data matrix: $SVD(X) = U \Sigma V^T$
- SVD has time complexity $O(dn^2)$.
- $V \in \mathbb{R}^{d \times d}$ are the needed eigenvectors for $X^T X \in \mathbb{R}^{d \times d}$.
- $\lambda_i = \Sigma_{i,i}^2$ are needed eigenvalues.



# Singular Value Decomposition (SVD)




For PCA we want projections onto top $k$ PCs.

- <u>When we used a spectral decomposition, $X^T X = QDQ^T$</u>, we compute: $X_k = XQ_k \in \mathbb{R}^{n\times k}$ ($Q$ are eigvecs of $X^T X$).
- <u>When using the SVD of $X$</u>, we can instead get this from:
- $X_k = XV_{:,1:k} = U_{:,1:k}\Sigma_{1:\mathrm{k},1:k} \in \mathbb{R}^{n\times k}$ ("scores" in PCA basis).
- We don't need to compute covariance matrix, or do the projections, we just need $SVD(X)$!

$$Xv_i = \sigma_i u_i$$

# “Eckart Young theorem” 1936

- The SVD “k-reconstruction” produces the best $k$-rank approximation by the matrix norm, $\|X - X_{recon-k}\|_F$.
- First proven by Schmidt (of Gram-Schmidt fame) in 1907 for Froebenius norm.
- Later rediscovered by Eckart & Young 1936, also generalized to other norms..
- <u>Thus, PCA provides the best low rank approximation to the data matrix.</u>

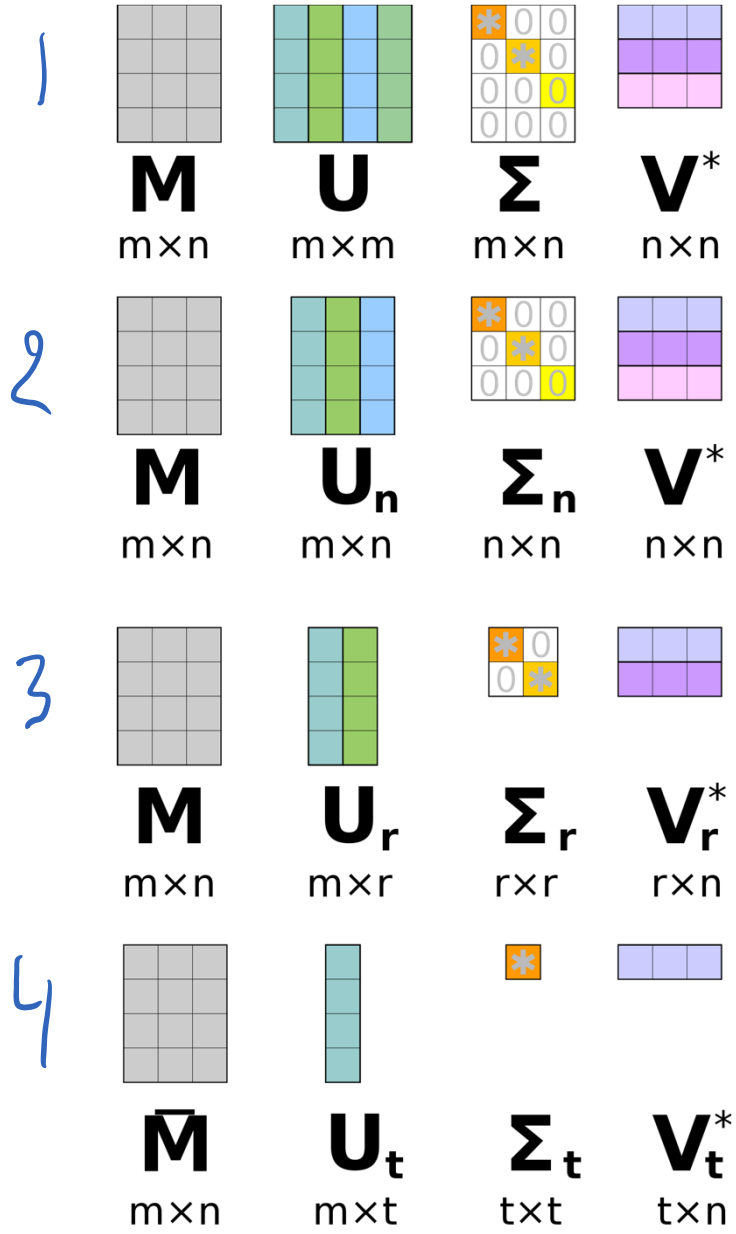
$$\|\mathsf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

https://en.wikipedia.org/wiki/Low-rank_approximation#Proof_of_Eckart%E2%80%93Young%E2%80%93Mirsky_theorem_(for_spectral_norm)

# Practicalities: Reduced SVDs

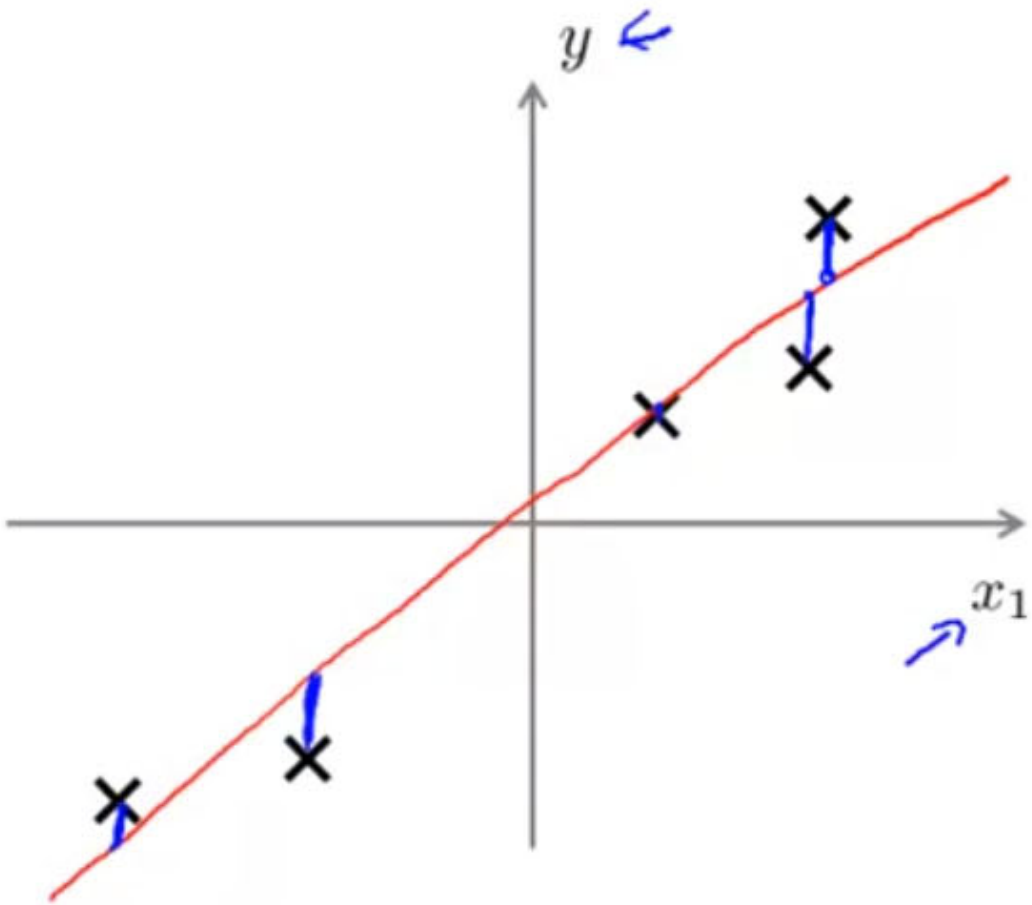
For PCA and other applications, don't need the entire SVD, and can make do with "trimmed down" versions:

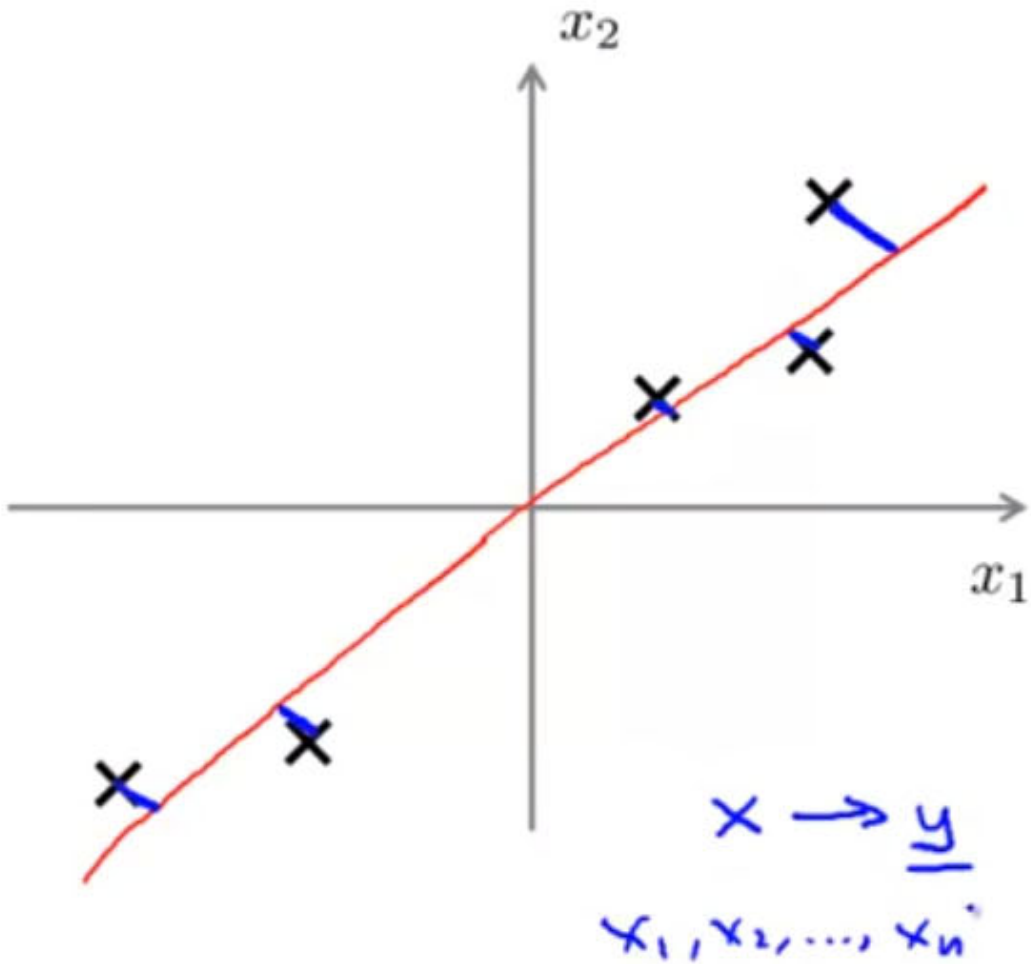
1. Full SVD
2. Thin SVD (remove columns of U not corresponding to rows of V*)
3. Compact SVD (remove vanishing singular values and corresponding columns/rows in U and V*),
4. Truncated SVD (keep only largest t singular values and corresponding columns/rows in U and V*)




https://en.wikipedia.org/wiki/Singular_value_decomposition

# PCA is not linear regression

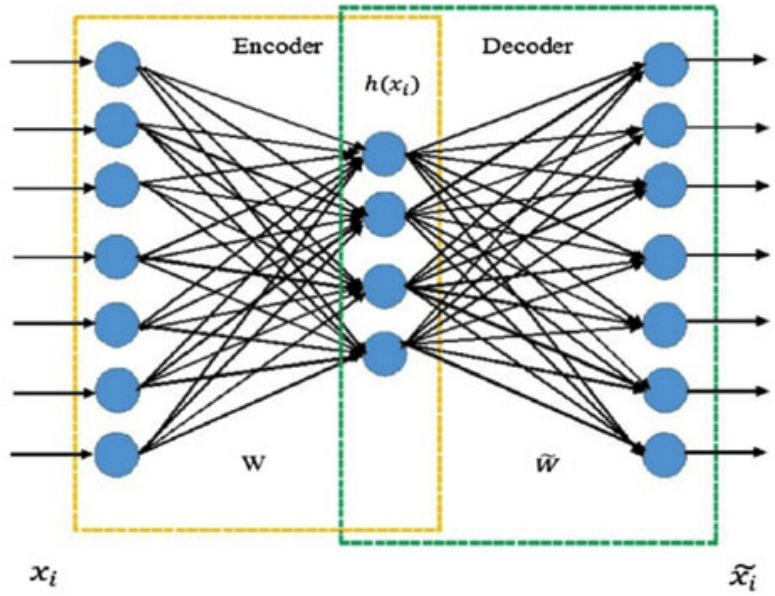



Andrew Ng

# PCA from neural networks!

- Special kind of neural network, called an *autoencoder* recovers the same subspace as PC-k.
- Autoencoder tries to compress a data set by trying to predict itself back after going through a bottleneck.




- For a linear autoencoder with one hidden layer containing $k$ nodes, using squared loss, the hidden layer representation is equivalent to that found from PCA-k (although $W$ may correspond to different eigenvectors that span the same space).
- Can generalize by making non-linear transfers, and more layers, etc.

*https://towardsdatascience.com/understanding-pca-autoencoders-algorithms-everyone-can-understand-28ee89b570e2*