Dimensionality Reduction

Sewoong Oh

CSE/STAT 416
University of Washington

Embedding high-dimensional data

- Embedding high-dimensional data into lower dimensional space helps us make sense of the data
- One prominent example is **data visualization**

- Each image is very high-dimensional (thousands or millions of pixels, each pixel taking 3 real values, each for Red, Green, Blue

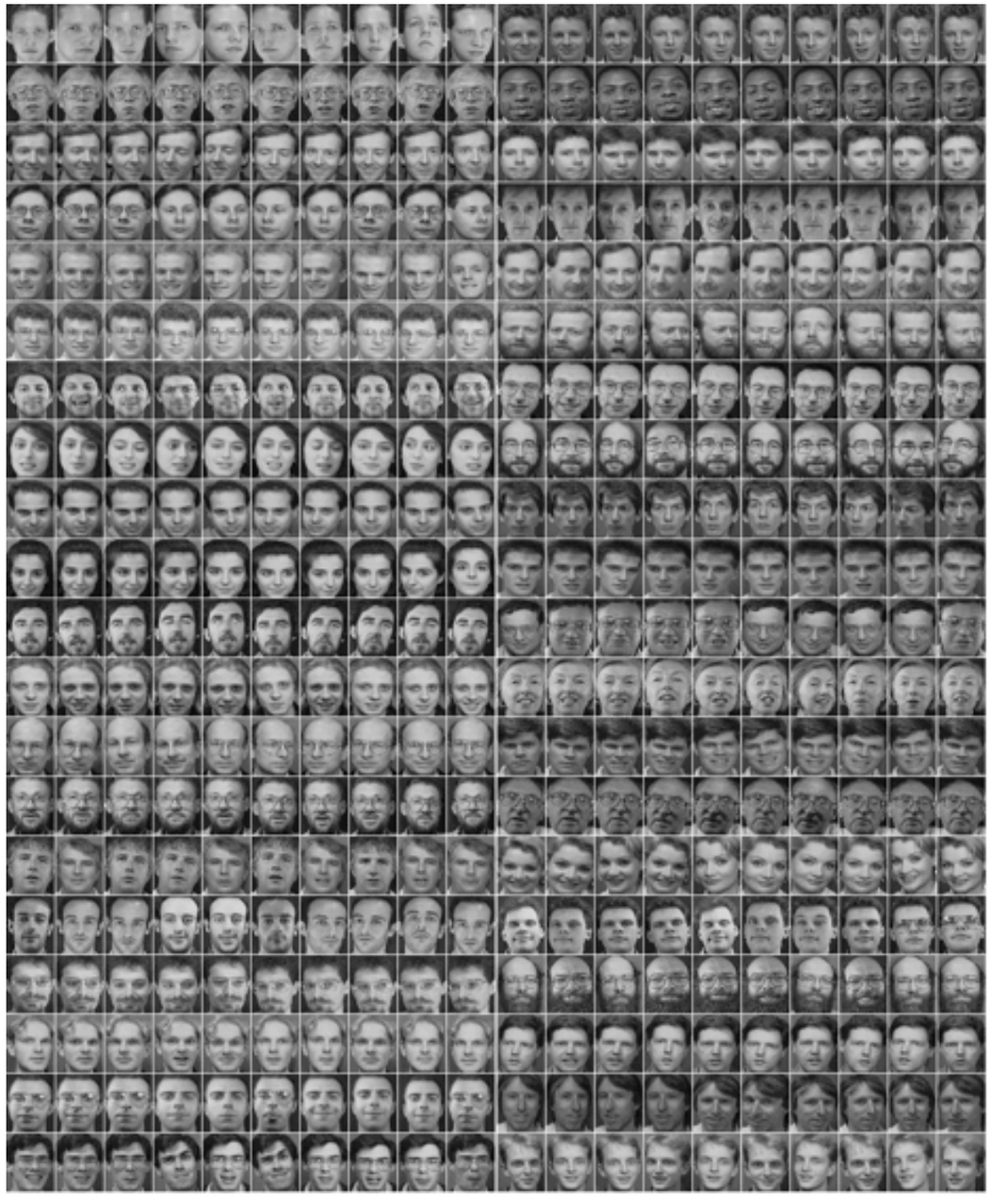
Can we give each image coordinates,
such that similar images are near each other?

Can we give each image coordinates,
such that each coordinate is meaningful?

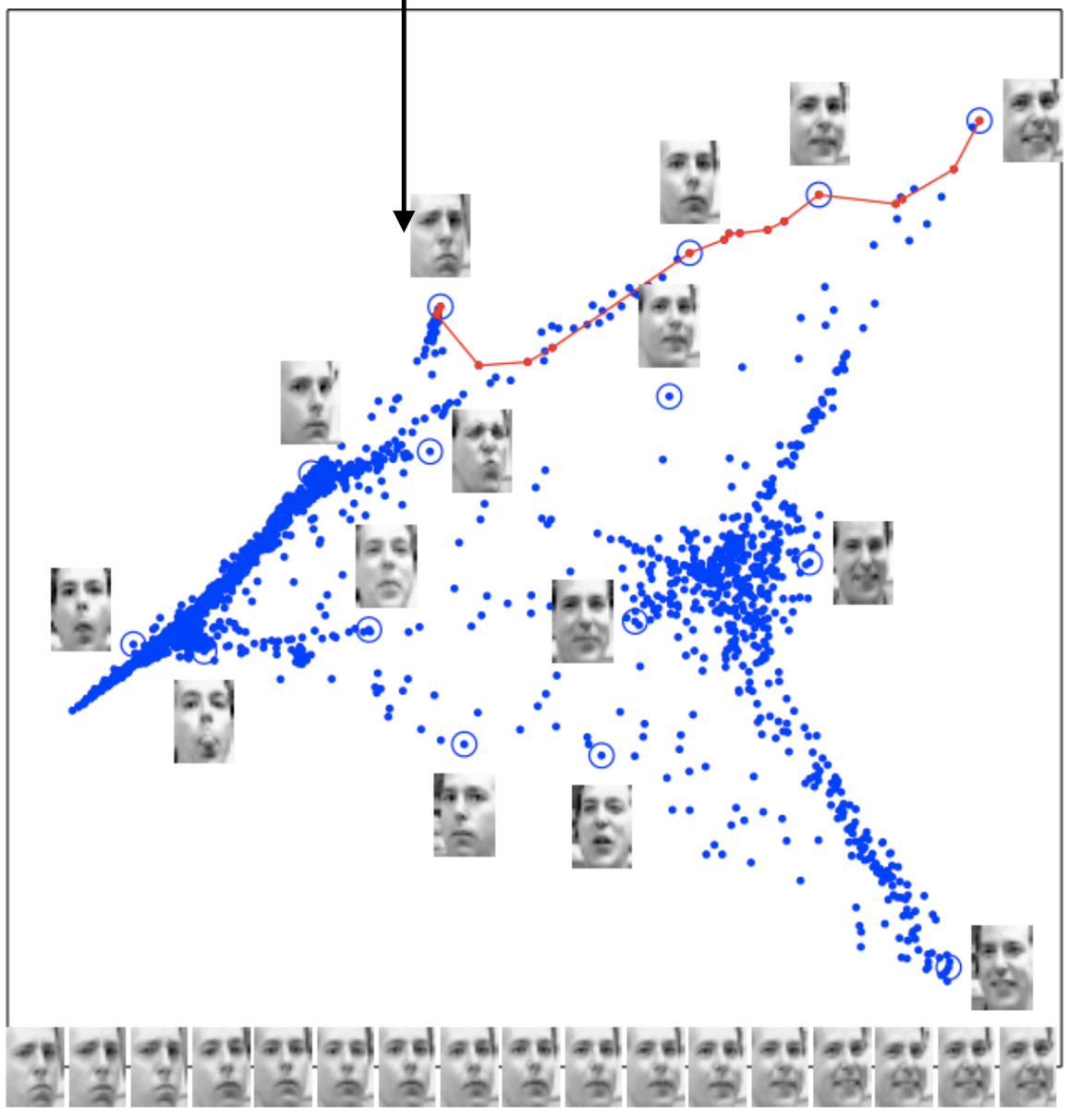
→ **(x,y,z,…)**

- Embed each data point (an image) in 2-dimensional space (embedding)

Input data image



2-dimensional embedding



- Similar looking images are close together
- Horizontal coordinate corresponds to “smile”, and vertical to “rotation”
- These two properties are learned automatically (because they are prominent in training data)

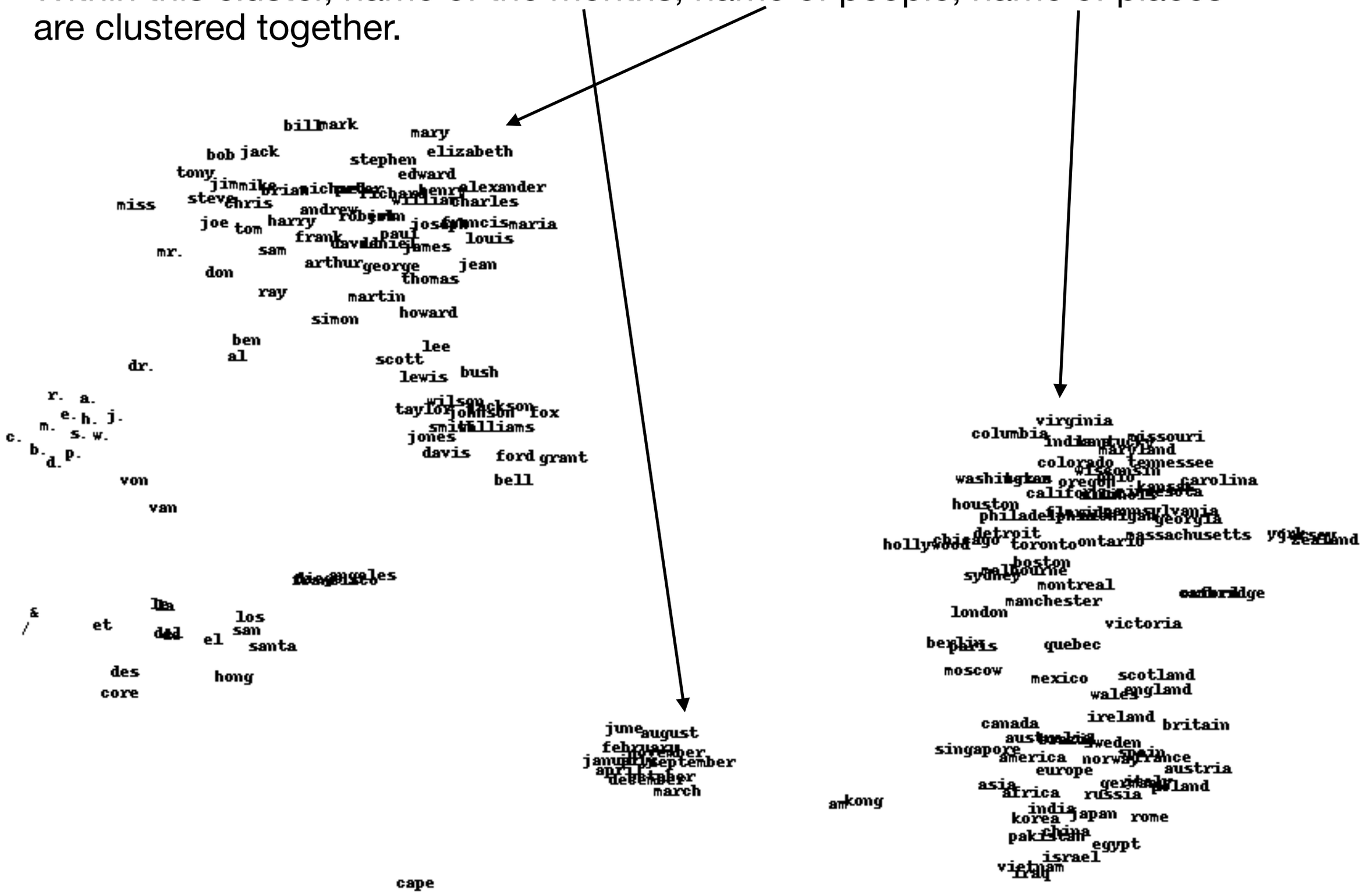
Word embedding

- Word embedding is extremely important application, because
  - Natural Language Processing (NLP) is one of the most important applications in machine learning with high impact (translation, Siri, Alexa, Google home, etc.)
  - Yet, words are in such a high dimensions, it is critical to find good embedding

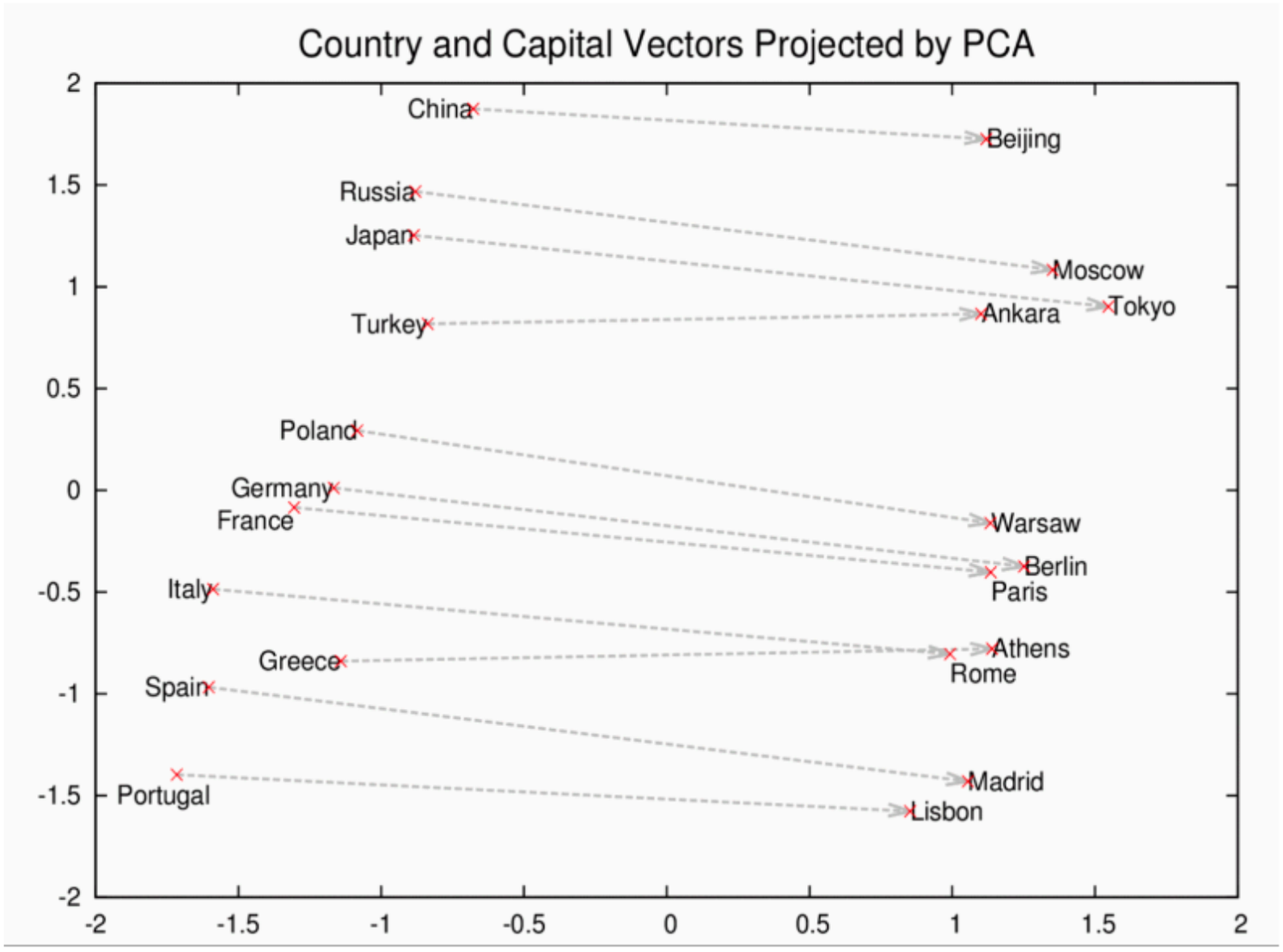
Seattle ⟶ (x,y,z,…)

Word embedding zoomed in shows similar words clustered together

- This shows one cluster which contain **names**
- Within this cluster, name of the months, name of people, name of places are clustered together.



Word embedding captures analogies between words, as directions in the embedded space



https://skymind.ai/wiki/word2vec

- Word Analogy
  - king:queen::man:? woman
  - house:roof::castle:? dome
  - building:architect::software:? programmer

Dimensionality reduction

- High dimensional input data such as books, images, videos go through **dimensionality reduction,** which represents each data in a much smaller dimensions
- We want a good representation, that captures the pattern in the original data such as relations, clusters, distances

- 1. This can be used in training,
  as lower dimensional data is faster to train
- 2. Can be used to visualize the data for human assessment
- 3. Can be used to discover the intrinsic dimensionality of data
  e.g. for MNIST hand written digits, we really only need a five or ten real values per data to represent it

Popular idea: linear projection

- **Linear projection**:
  - mathematical term from linear algebra
  - e.g. given x=(x[1],x[2],…,x[d]), an example of a linear projection onto 2-D might be

$$z_i[1] = .2x_i[1] + 0.3x_i[2] + \cdots + 0.6x_i[d]$$

$$z_i[2] = .5x_i[1] - 0.1x_i[2] + \cdots + 0.3x_i[d]$$

- In linear algebraic terminology,

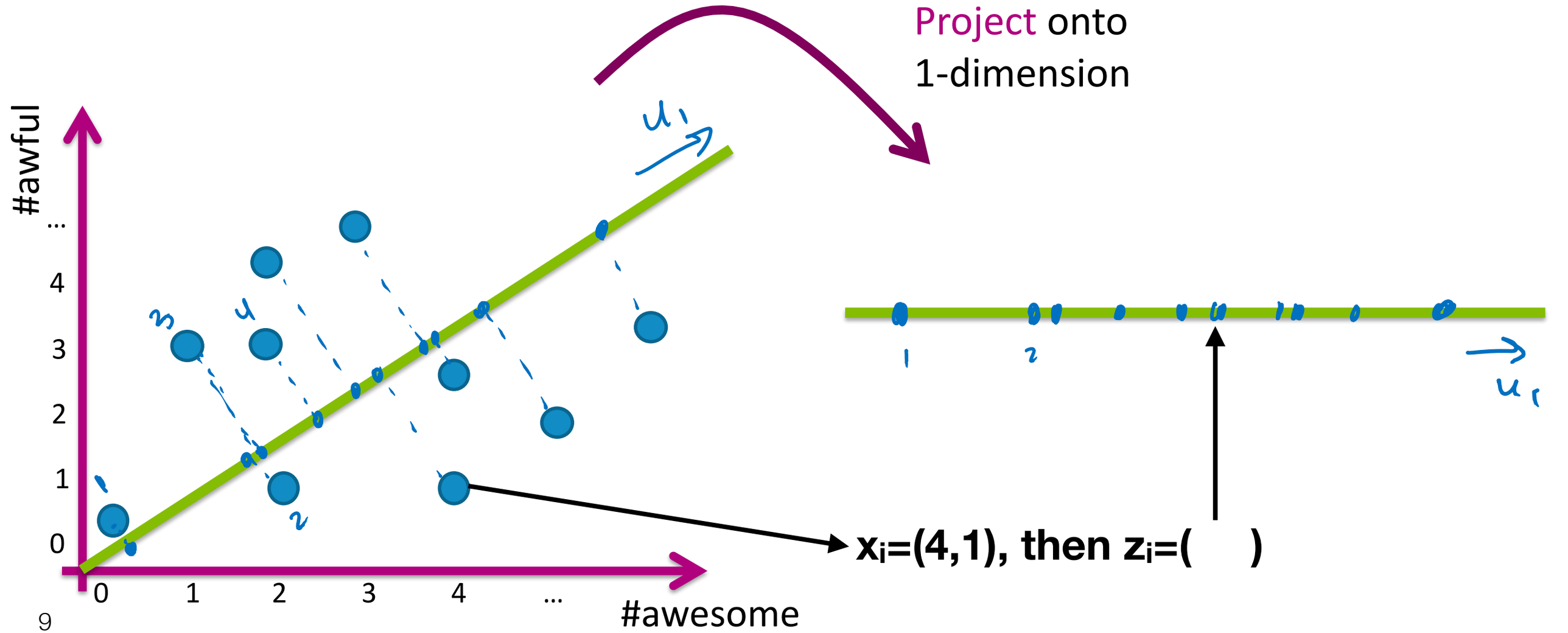
projection is defined by a projection matrix U, such that when a sample point x_i is projected to z_i, we write as

$$z_i = Ux_i$$

$$z_i = \begin{bmatrix} z_i[1] \\ z_i[2] \end{bmatrix} \quad U = \begin{bmatrix} 0.2 & 0.2 & \cdots & 0.6 \\ 0.5 & -0.1 & \cdots & 0.3 \end{bmatrix} \quad x_i = \begin{bmatrix} x_i[1] \\ \vdots \\ x_i[d] \end{bmatrix}$$

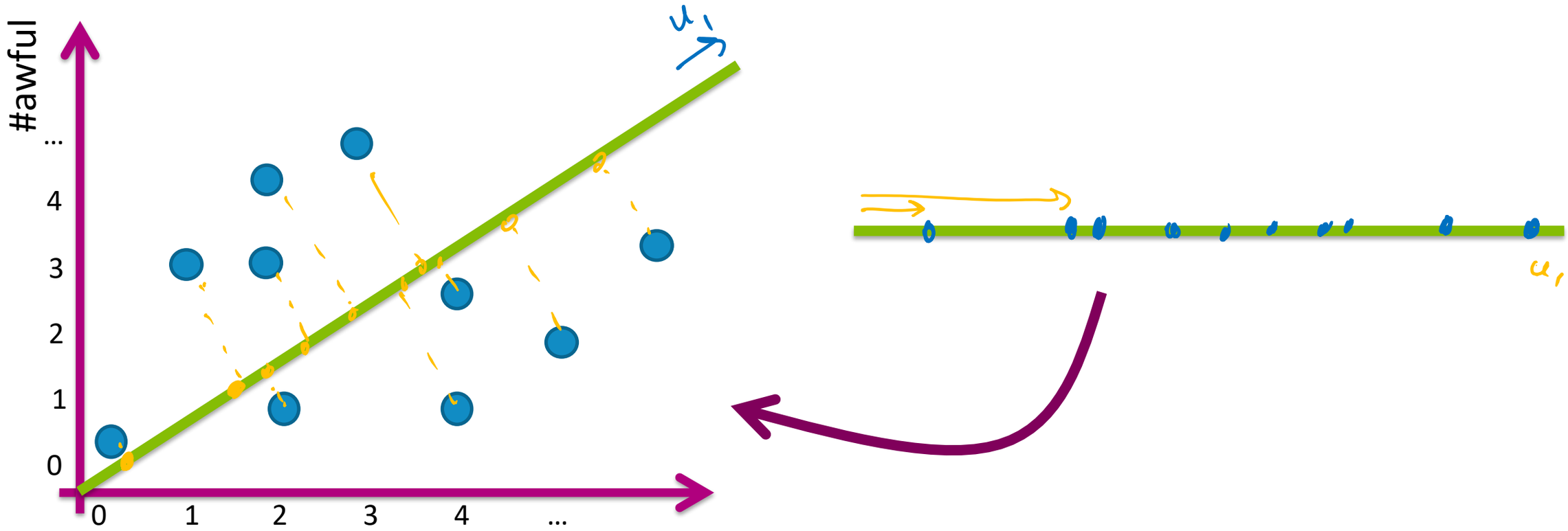
Linear projection

- In this example, a data point is in 2-D, and we choose to project it to 1-D
- The projection coefficients can be visualized as a line, and projection operation of projecting ***x*** onto ***z*** amounts to finding the closest point on the line
- This example shows: u[1] = [4, 3]
- $z = u[1]^T x = 4*x[1] + 3*x[2]$



Linear projection and reconstruction

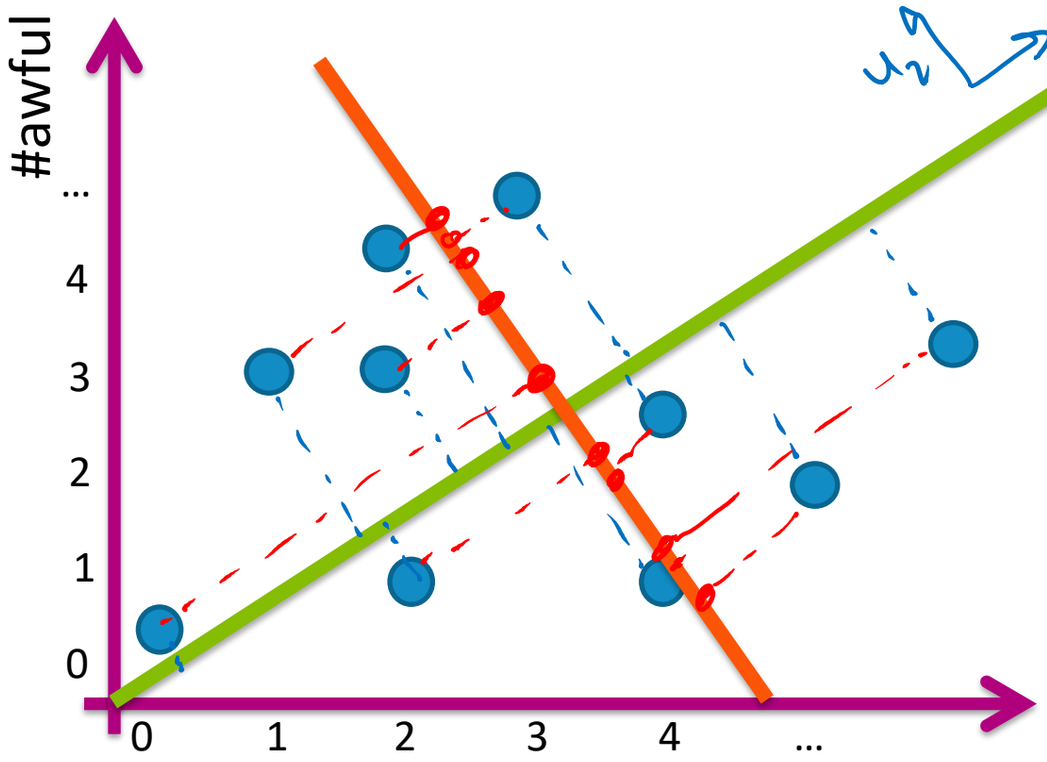
- z= u[1]^{T}x = 4*x[1] + 3*x[2]
- Can we get back x[1],x[2] from z ?
  Why?

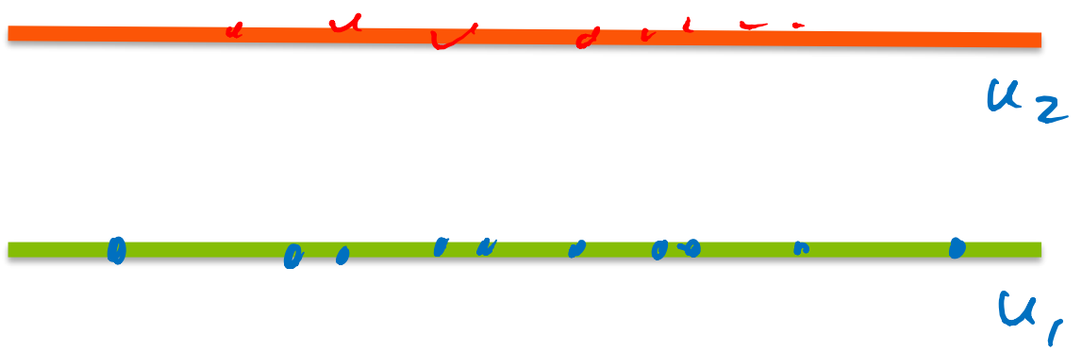


If z = 11, then what is x?

What if we project onto 2-D?

- As long as those two projections are different (i.e. linearly independent), we can recover the original coordinates back
- z[1] = 4*x[1] + 3*x[2] u[1] = [4, 3]
- z[2] = 3*x[1] - 4*x[2] u[2] = [3, -4]



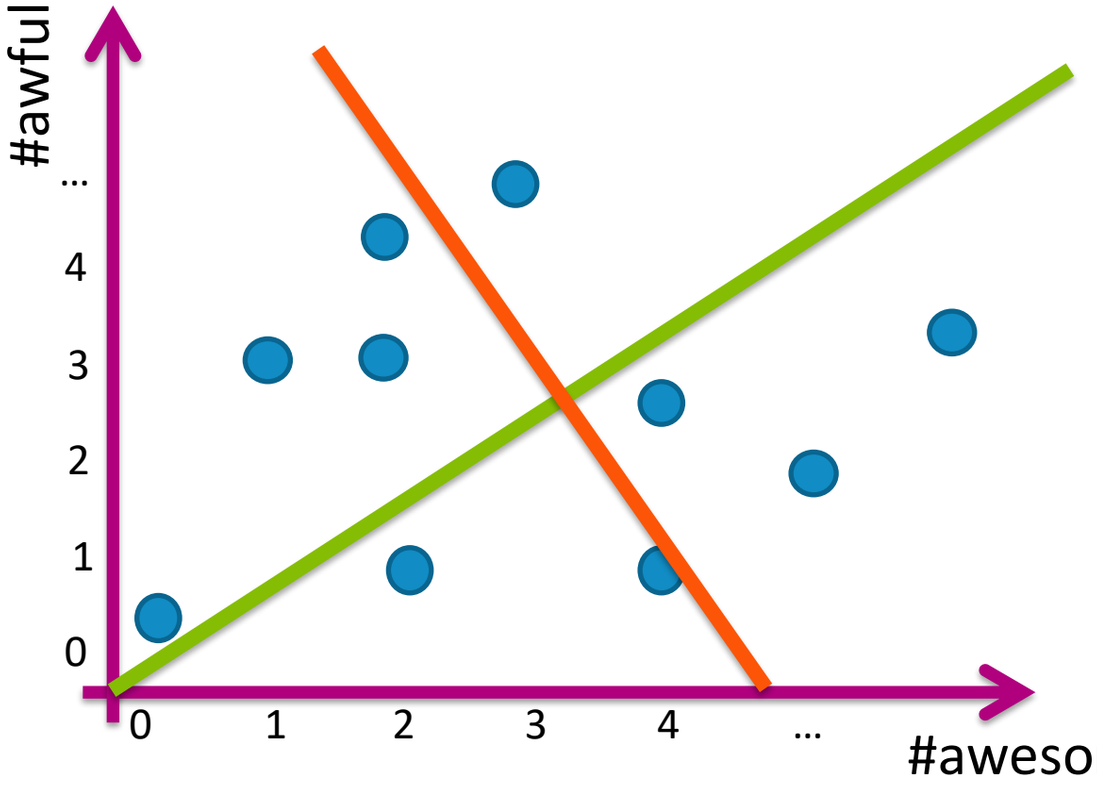


Perfect reconstruction!

If z = (11,2), then what is x?

- But using 2 lines does not give us dimensionality reduction that we want (the dimension is still 2-D)

- We ask the following fundamental question:
  if I want to choose one line, which one should I choose?
- We should choose one with less reconstruction error

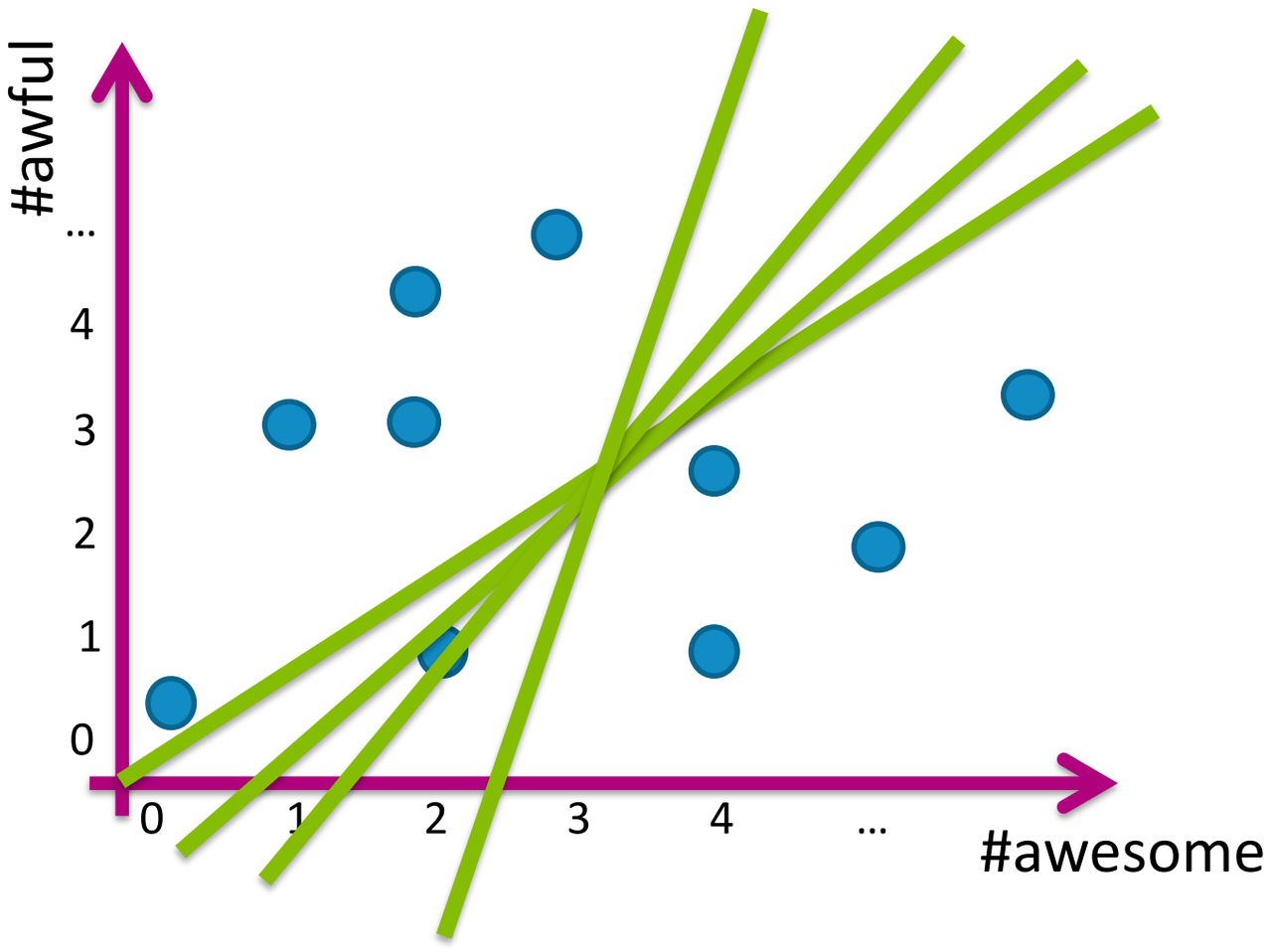


- What is a reconstruction of **z = 11** for the green line **z= 4*x[1] + 3*x[2]** ?
  - x=(2,1), **x=(33/28,44/21)**, (0,11/3), etc.
- What is the reconstruction error of **x=(2,1)**?
  -
- Which one has less reconstruction error?
  - Green
- **If the variation on the linear projection is large, then size of the error is small**

If I had to choose one of these vectors, which do I prefer?

- Over all possible lines?

- Extreme example
  on a lower dimensional subspace:



Principal component analysis (PCA) – Basic idea

Project d-dimensional data into k-dimensional space while preserving as much information as possible:

- e.g., project space of 10000 words into 3-dimensions

Main idea:

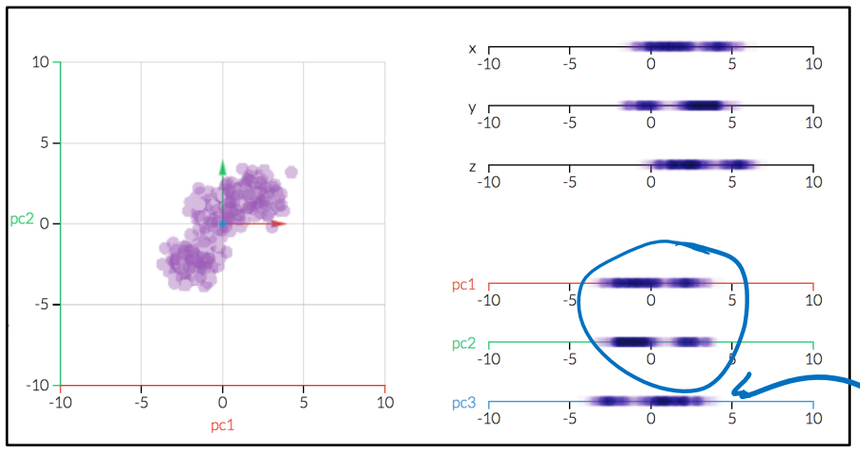
Choose projection with minimum reconstruction error

Visual explanation of PCA

http://setosa.io/ev/principal-component-analysis/

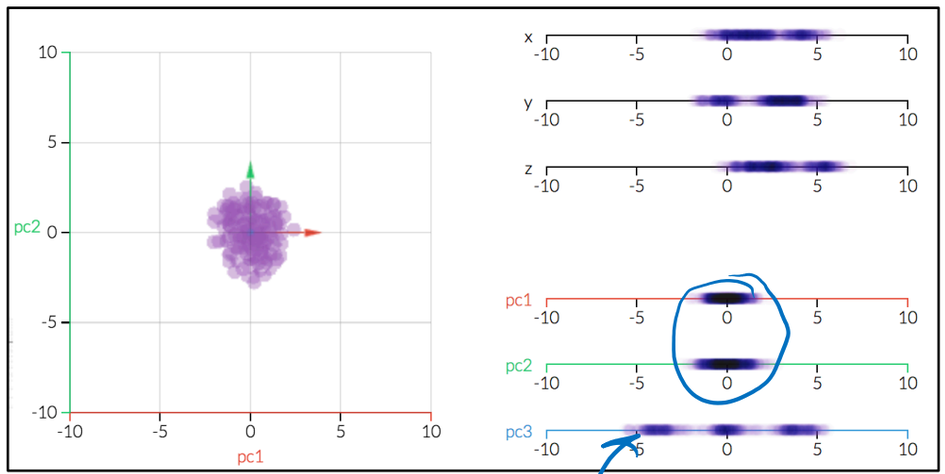
Choosing principal components

Which of the following pictures show the top two principal components (pc1, pc2) of the dataset?

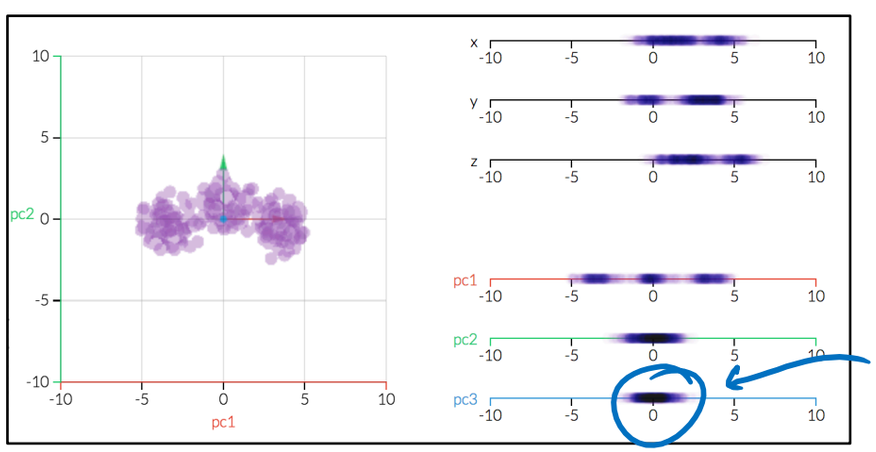


Option 1

Option 3



Option 2



Basic PCA algorithm

- Form data matrix **X**
  - Each row is a different data point…like our typical data tables

- **Recenter:** subtract the mean from each row of **X** → $\mathbf{X_c}$

- **Spread/orientation calculation:** Compute the covariance matrix Σ:

$$\Sigma_{ts} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x_{c,i}}[t]\mathbf{x_{c,i}}[s]$$

- **Find basis:**
  - Compute eigendecomposition of Σ
  - Select $(u_1, u_2, \cdots, u_k)$ to be eigenvectors with largest eigenvalues

- **Project data:** Project each data point onto each vector

$$z_i[1] = u_1^T x_i = u_1[1]x_i[1] + \cdots u_1[d]x_i[d]$$
$$z_i[2] = u_2^T x_i = u_2[1]x_i[1] + \cdots u_2[d]x_i[d]$$
$$\vdots$$
$$z_i[k] = u_k^T x_i = u_k[1]x_i[1] + \cdots u_k[d]x_i[d]$$

Reconstruction

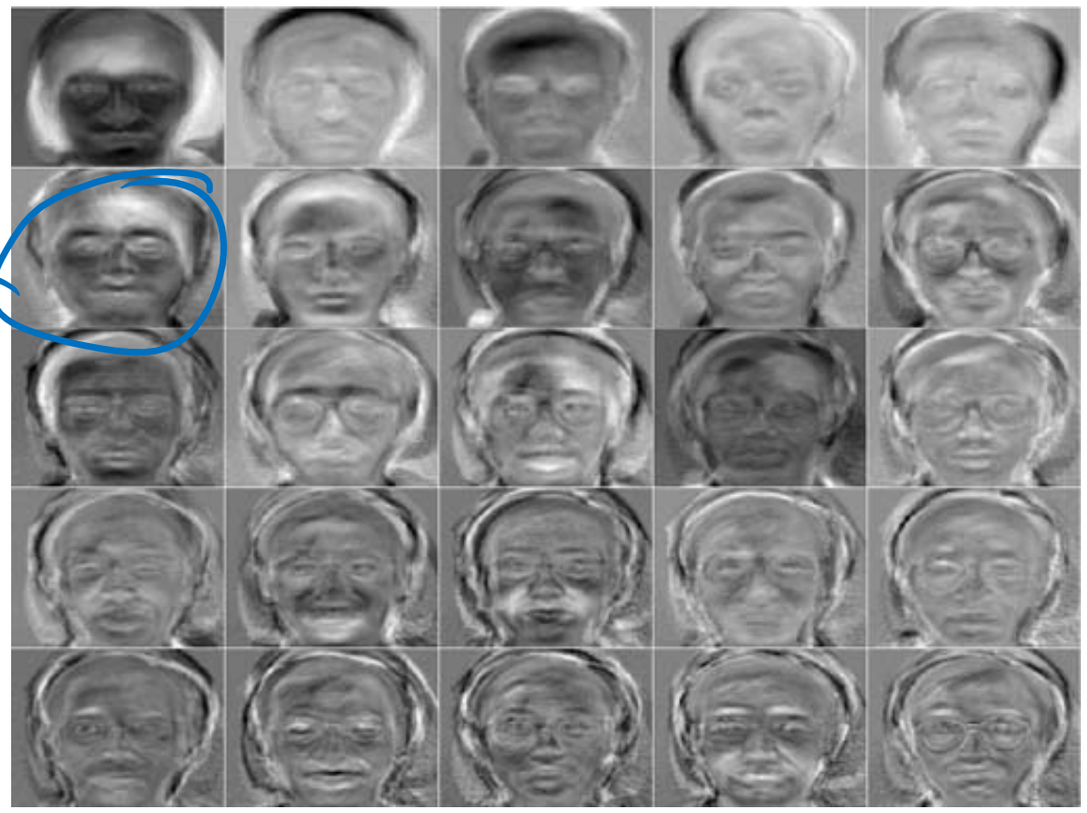
Using our principal components, reconstruct observation in original domain:

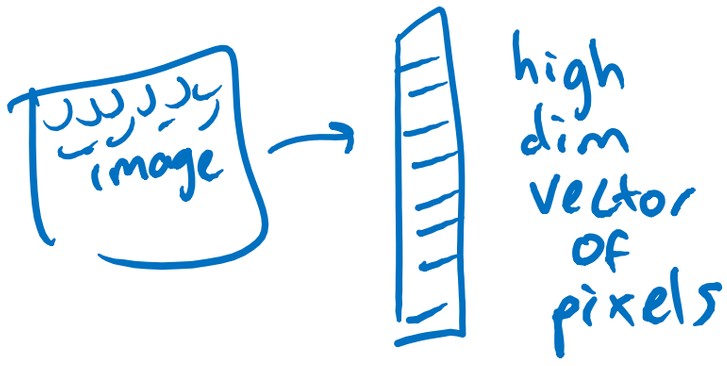
$$\hat{x}_i[1:d] = \underbrace{\bar{x}[1:d]}_{\text{add back subtracted mean}} + \sum_{j=1}^{k} \underbrace{z_i[j]}_{\text{amount}} \underbrace{u_j}_{\text{each principal direction}}$$

Eigenfaces [Turk, Pentland ’91]

Input images:

Principal components:





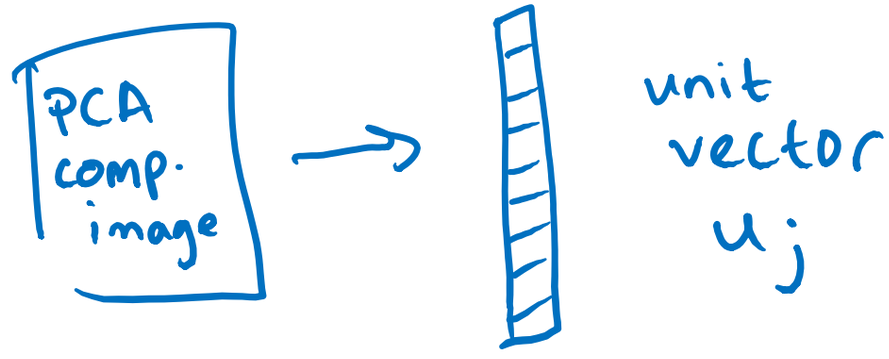
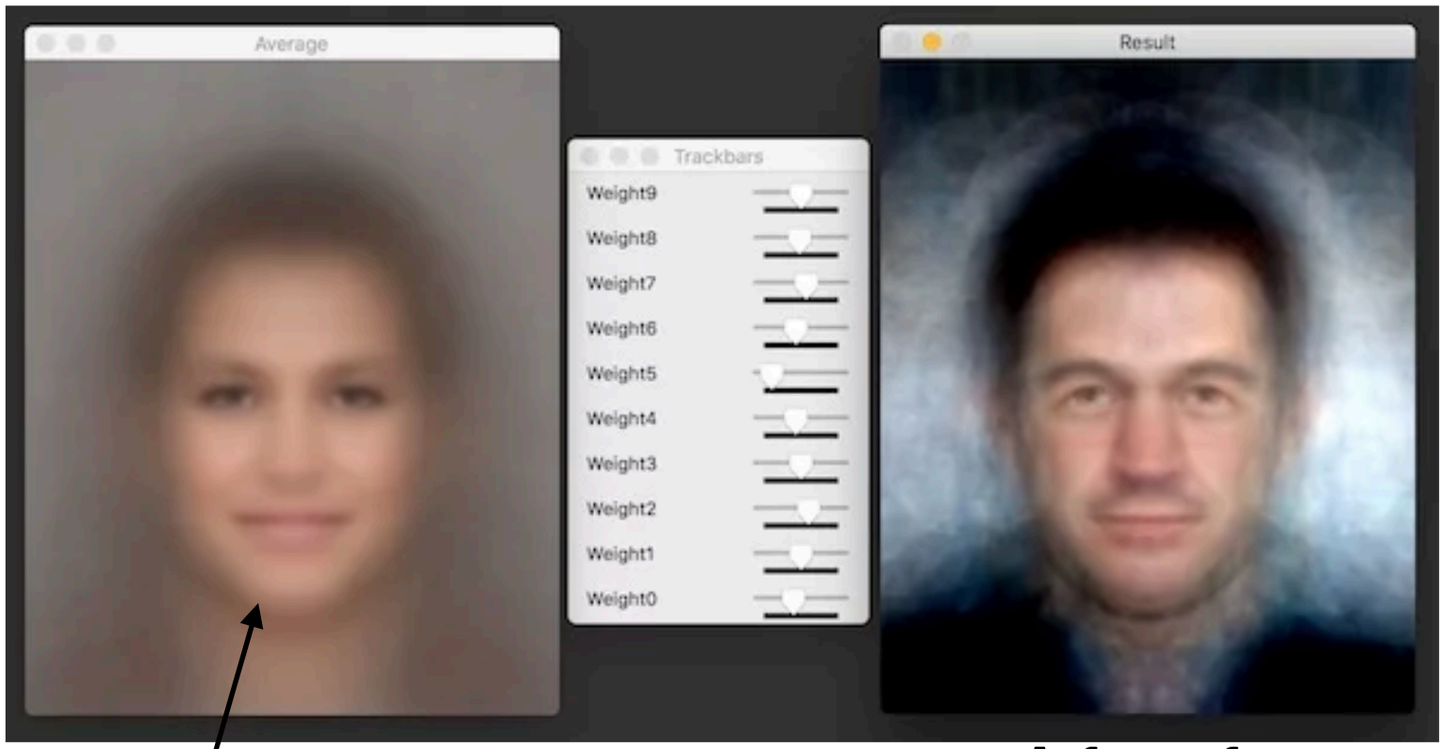


Image generation from eigenfaces



Average face

A face from weighted addition of principal components

Image reconostruction from eigenfaces

Real face

10 principal components give a pretty good reconstruction of the face