

# Unsupervised learning

Fraida Fund

## Contents

Unsupervised learning . . . . .	3
The basic supervised learning problem . . . . .	3
The basic unsupervised learning problem . . . . .	3
Dimensionality reduction with PCA . . . . .	4
Dimensionality reduction problem . . . . .	4
Dimensionality reduction with PCA vs feature selection . . . . .	4
Projections . . . . .	5
PCA intuition (1) . . . . .	5
PCA intuition (2) . . . . .	5
PCA intuition (3) . . . . .	6
Sample covariance matrix (1) . . . . .	6
Sample covariance matrix (2) . . . . .	7
Directional variance . . . . .	7
Maximizing directional variance (1) . . . . .	7
Maximizing directional variance (2) . . . . .	8
Projections onto eigenvectors: uncorrelated features . . . . .	8
PCA intuition (5) . . . . .	8
PCA in summary (1) . . . . .	9
Approximating data . . . . .	10
Average approximation error . . . . .	10
Proportion of variance explained . . . . .	10
Clustering . . . . .	11
Clustering problem . . . . .	11
K-means clustering . . . . .	11
K-means algorithm . . . . .	11
K-means visualization . . . . .	12
K-means summary . . . . .	12
Dimensionality reduction with deep learning . . . . .	13
Dimensionality reduction using an autoencoder . . . . .	13
K-means as an autoencoder (1) . . . . .	13
K-means as an autoencoder (2) . . . . .	13
K-means as an autoencoder (3) . . . . .	13
PCA as an autoencoder (1) . . . . .	13
PCA as an autoencoder (2) . . . . .	14
Limits of PCA . . . . .	14
Neural autoencoder . . . . .	14
Overcomplete autoencoder . . . . .	15
Undercomplete autoencoder . . . . .	15
Sparse autoencoder (1) . . . . .	15
Sparse autoencoder (2) . . . . .	15
Autoencoder comparison . . . . .	16

What are autoencoders good for? . . . . .	16
Example: reconstruction of faces . . . . .	16
Example: MNIST visualization . . . . .	16
Density estimation . . . . .	17
Types of density estimation . . . . .	17
GAN: Generative adversarial networks . . . . .	17
GAN: basic idea (1) . . . . .	17
GAN: basic idea (2) . . . . .	17
Discriminator loss function (1) . . . . .	17
Discriminator loss function (2) . . . . .	17
Discriminator objective . . . . .	17
Generator objective (1) . . . . .	18
Overall objective . . . . .	18
Problem: gradient of cross-entropy loss . . . . .	18
Generator objective (2) . . . . .	18
Training: First, update discriminator . . . . .	18
Training: Then, update generator . . . . .	18
Illustration: training discriminator . . . . .	19
Illustration: training generator . . . . .	19

## Unsupervised learning

### The basic supervised learning problem

Given a **sample** with a vector of **features**

$$\mathbf{x} = (x_1, x_2, \dots, x_d)$$

There is some (unknown) relationship between  $\mathbf{x}$  and a **target** variable,  $y$ , whose value is unknown.

We want to find  $\hat{y}$ , our **prediction** for the value of  $y$ .

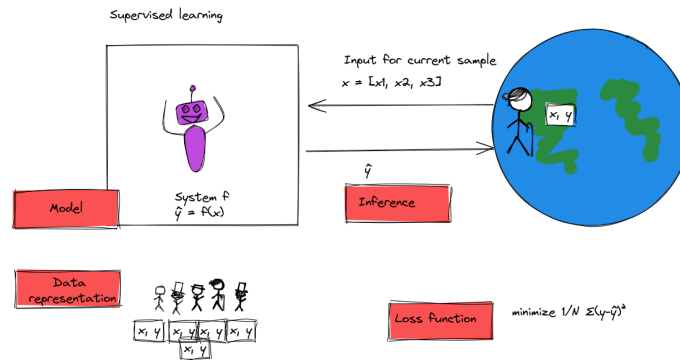


Figure 1: Basic supervised learning problem.

### The basic unsupervised learning problem

Given a **sample** with a vector of **features**

$$\mathbf{x} = (x_1, x_2, \dots, x_d)$$

We want to learn something about the underlying *structure* of the data.

No labels!

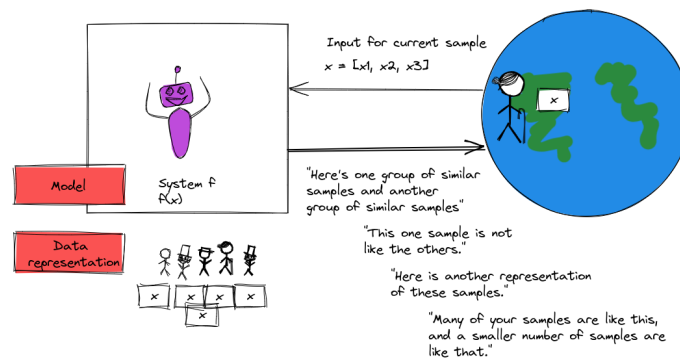


Figure 2: Unsupervised learning.

What are some things we might be able to learn about the structure of the data?

- dimensionality reduction
- feature representation
- embedding
- clustering
- anomaly detection
- density estimation

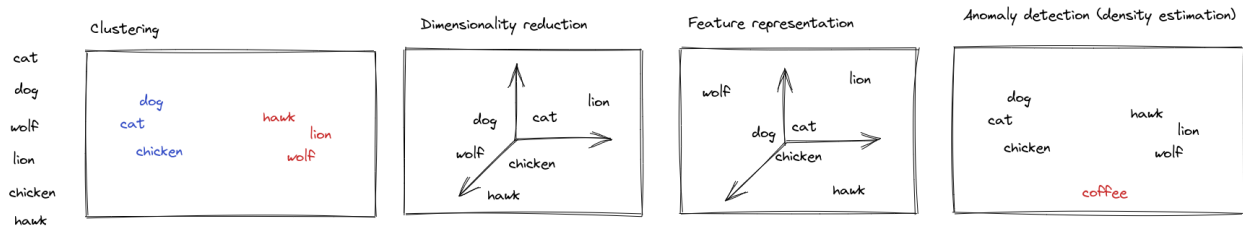


Figure 3: Unsupervised learning problems.

## Dimensionality reduction with PCA

Why?

- Supervised ML on small feature set
- Visualize data
- Compress data

### Dimensionality reduction problem

- Given  $N \times p$  data matrix  $X$  where each row is a sample  $x_n$
- **Problem:** Map data to  $N \times p'$  where  $p' \ll p$

### Dimensionality reduction with PCA vs feature selection

Previous feature selection:

- Choose subset of existing features
- Many features are somewhat correlated; redundant information

Now: *new* features, so we can get max information with min features.

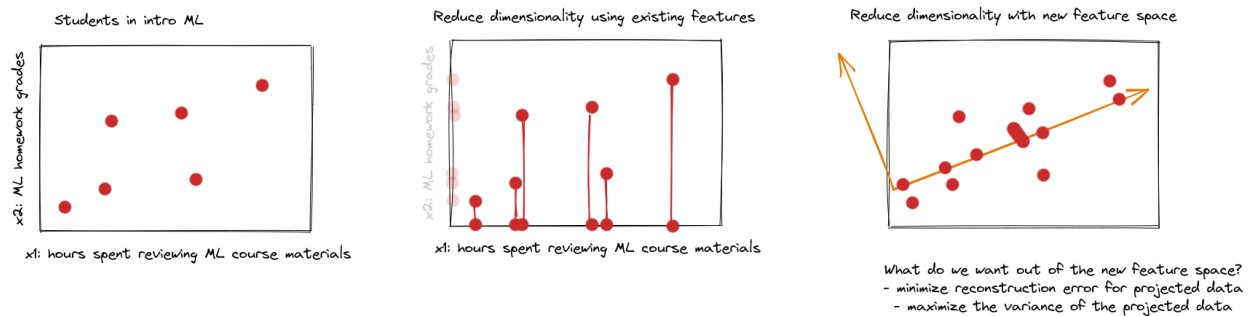


Figure 4: Instead of using existing features, we project the data onto a new feature space.

## Projections

Given vectors  $z$  and  $v$ ,  $\theta$  is the angle between them. Projection of  $z$  onto  $v$  is:

$$\hat{z} = \text{Proj}_v(z) = \alpha v, \quad \alpha = \frac{v^T z}{v^T v} = \frac{\|z\|}{\|v\|} \cos \theta$$

$V = \{\alpha v | \alpha \in \mathbb{R}\}$  are the vectors on the line spanned by  $v$ , then  $\text{Proj}_v(z)$  is the closest vector in  $V$  to  $z$ :  $\hat{z} = \underset{w \in V}{\text{argmin}} \|z - w\|^2$ .

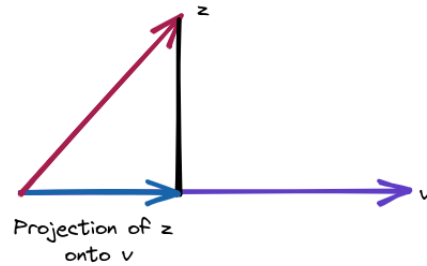


Figure 5: Projection of  $z$  onto  $v$ .

## PCA intuition (1)

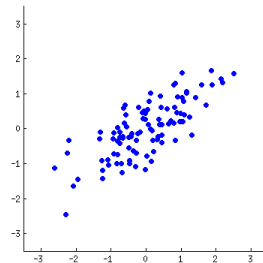


Figure 6: Data with two features, on two axes. Data is centered.

## PCA intuition (2)

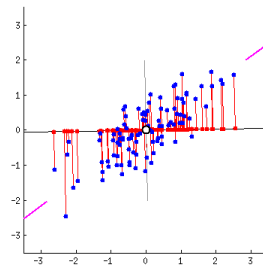


Figure 7: Construct a new feature by drawing a line  $w_1 x_1 + w_2 x_2$ , and projecting data onto that line (red dots are projections). [View animation here.](#)

### PCA intuition (3)

Project onto which line?

- Maximize average squared distance from the center to each red dot; **variance of new feature**
- Minimize average squared length of the corresponding red connecting lines; **total reconstruction error**

Can you convince yourself that these two objectives are related, and are achieved by the same projection?

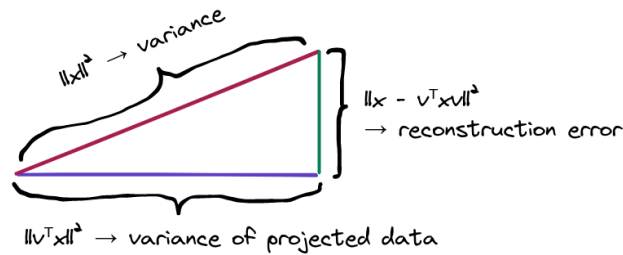


Figure 8: Pythagorean decomposition. Keeping reconstruction error minimized (on average) is the same as keeping variance of projection high (on average).

The intuition is that, by Pythagorean decomposition: the variance of the data (a fixed quantity) is equal to the variance of the projected data (which we want to be large) plus the reconstruction error (which we want to be small).

### Sample covariance matrix (1)

- sample variance  $s_x^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$
- sample covariance  $s_{xy} = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})$
- $\text{Cov}(x, y)$  is a  $p \times p$  matrix  $Q$  with components:

$$Q_{k,l} = \frac{1}{N} \sum_{i=1}^N (x_{ik} - \bar{x}_k)(x_{il} - \bar{x}_l)$$

Note:  $x$  and  $y$  in this notation are two different features, not a feature matrix and label.

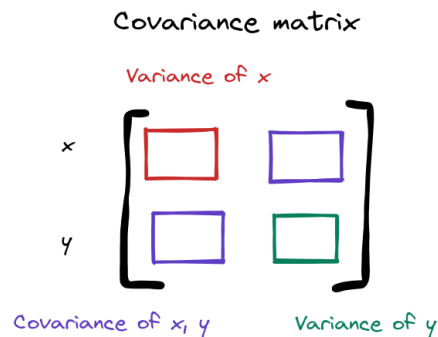


Figure 9: Illustration of covariance matrix.

## Sample covariance matrix (2)

Let  $\tilde{X}$  be the data matrix with sample mean removed, row  $\tilde{x}_i = x_i - \bar{x}$

Sample covariance matrix is:

$$Q = \frac{1}{N} \tilde{X}^T \tilde{X}$$

(compute covariance matrix by matrix product!)

## Directional variance

Projection onto a unit vector  $v$ :  $z_i = (v^T \tilde{x}_i)v$

- Sample mean:  $\bar{z} = v^T \bar{x}$
- Sample variance:  $s_z^2 = v^T Q v$

Now we have these mean-removed rows of data, and we want to project each row onto some vector  $v$ , where  $z$  is the projection of  $\tilde{x}_i$  onto  $v$ . And we want to choose  $v$  to maximize the variance of  $z$ ,  $s_z^2$ .

We will call this the *directional variance* - the variance of the projection of the row onto  $v$ .

## Maximizing directional variance (1)

Given data  $\tilde{x}_i$ , what directions of unit vector  $v$  ( $\|v\| = 1$ ) maximizes the variance of projection along direction of  $v$ ?

$$\max_v v^T Q v \quad \text{s.t.} \|v\| = 1$$

Important note:

- an eigenvector is a special vector that, when you multiply the covariance matrix by the eigenvector, the result is a shorter or longer eigenvector pointing in the *same direction*.
- the eigenvalue is the value by which eigenvector is scaled when multiplied by the covariance matrix.
- a  $p \times p$  matrix has  $p$  eigenvectors.
- the eigenvectors are orthogonal.

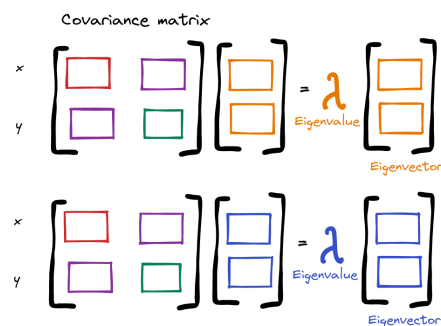


Figure 10: Eigenvectors and eigenvalues.

## Maximizing directional variance (2)

Let  $v_1, \dots, v_p$  be *eigenvectors* of  $Q$  (there are  $p$ ):

$$Qv_j = \lambda_j v_j$$

- Sort them in descending order:  $\lambda_1 \geq \lambda_2 \geq \dots \lambda_p$ .
- The largest one is the vector that maximizes directional variance, the next is direction of second most variance, etc.

**Theorem:** any eigenvector of  $Q$  is a local maxima of the optimization problem

$$\max_v v^T Q v \quad \text{s.t.} \|v\| = 1$$

**Proof:** Define the Lagrangian,

$$L(v, \lambda) = v^T Q v - \lambda [\|v\|^2 - 1]$$

At any local maxima,

$$\frac{\partial L}{\partial v} = 0 \implies Qv - \lambda v = 0$$

Therefore,  $v$  is an eigenvector of  $Q$ .

For a nice, detailed proof of this, I recommend [this set of notes](#) by Cosma Shalizi at CMU.

## Projections onto eigenvectors: uncorrelated features

- Eigenvectors are orthogonal:  $v_j^T v_k = 0$  if  $j \neq k$
- So the projections of the data onto eigenvectors are uncorrelated

These are called the *principal components*

## PCA intuition (5)

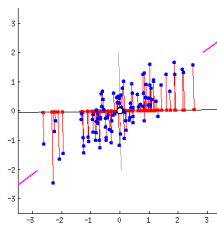


Figure 11: In the animation, gray and black lines form a rotating coordinate frame. When variance of projection is maximized, the black line points in direction of first eigenvector of covariance matrix (direction of maximum variance of the data), and grey line points toward second eigenvector (direction of second-most variance of the data). [View animation here.](#)



## PCA in summary (1)

Given high-dimensional data,

1. Center data (remove mean)
2. Get covariance matrix
3. Get eigenvectors, eigenvalues
4. Sort by eigenvalue
5. Choose  $p'$  eigenvectors with largest eigenvalues
6. Project data onto those eigenvectors

Now you have  $N \times p'$  data that maximizes info

Note: in practice, we compute PCA using singular value decomposition (SVD) which is numerically more stable.

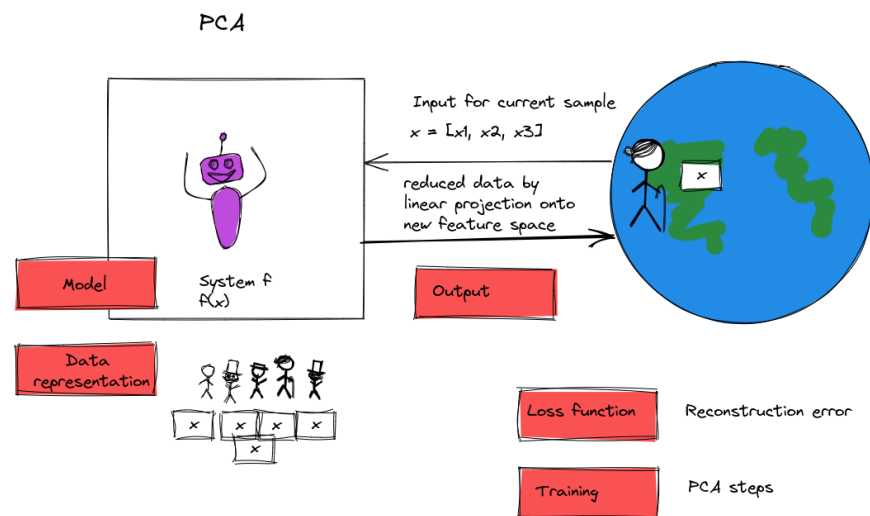


Figure 12: PCA summary.

### Approximating data

Given data  $\tilde{x}_i, i = 1, \dots, N$ , and PCs  $v_1, \dots, v_p$ , we can project + then reconstruct the data:

$$\tilde{x}_i = \sum_{j=1}^p (v_j^T \tilde{x}_i) v_j$$

Consider approximation with *first*  $d < p$  coefficients:

$$\hat{x}_i = \sum_{j=1}^d (v_j^T \tilde{x}_i) v_j$$

### Average approximation error

For sample  $i$ , error is:

$$\tilde{x}_i - \hat{x}_i = \sum_{j=d+1}^p (v_j^T \tilde{x}_i) v_j$$

The projection onto the first principal components carries the most information; the projection onto the last principal components carries the least. So the error due to missing the last PCs is small!

### Proportion of variance explained

The *proportion of variance* explained by  $d$  PCs is:

$$PoV(d) = \frac{\sum_{j=1}^d \lambda_j}{\sum_{j=1}^p \lambda_j}$$

where the denominator is variance of projected data:  $\frac{1}{N} \sum_{i=1}^N \|\tilde{x}_i\|^2 = \sum_{j=1}^p \lambda_j$

## Clustering

### Clustering problem

- Given  $N \times d$  data matrix  $X$  where each row is a sample  $x_n$
- **Problem:** Group data into  $K$  clusters
- More formally: Assign  $\sigma_n = \{1, \dots, K\}$  cluster label for each sample
- Samples in same cluster should be close:  $\|x_n - x_m\|$  is small when  $\sigma_n = \sigma_m$

### K-means clustering

We want to minimize

$$J = \sum_{i=1}^K \sum_{n \in C_i} \|x_n - \mu_i\|^2$$

- $\mu_i$  is the mean of each cluster
- $\sigma_n \in \{1, \dots, K\}$  is the cluster that  $x_n$  belongs to

### K-means algorithm

Start with random (?) guesses for each  $\mu_i$ . Then, iteratively:

- Update cluster membership (nearest neighbor rule): For every  $n$ ,

$$\sigma_n = \operatorname{argmin}_i \|x_n - \mu_i\|^2$$

- Update mean of each cluster (centroid rule): for every  $i$ ,  $\mu_i$  is average of  $x_n$  in  $C_i$

(Sensitive to initial conditions!)

Clustering (K means):  
1. Randomly place cluster means  
2. Assign each point to cluster (nearest neighbor rule)  
3. Update position of cluster means  
4. Repeat steps 2, 3  
5. Stop when the assignment of cluster labels doesn't change

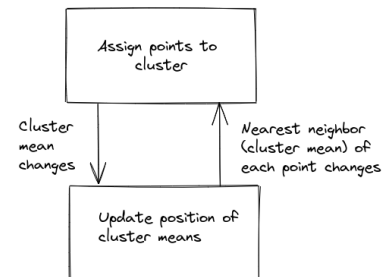
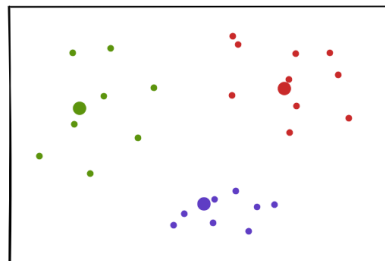


Figure 13: K-means clustering.

## K-means visualization

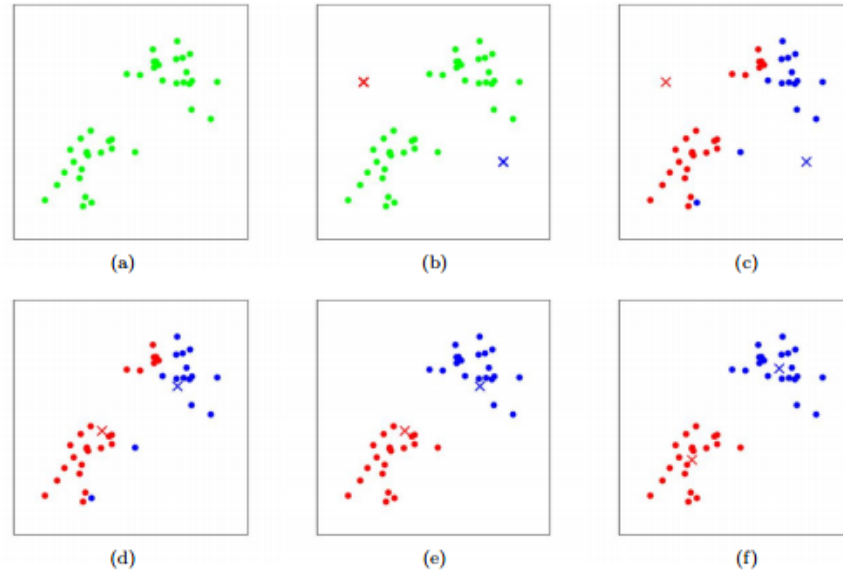


Figure 14: Visualization of k-means clustering.

## K-means summary

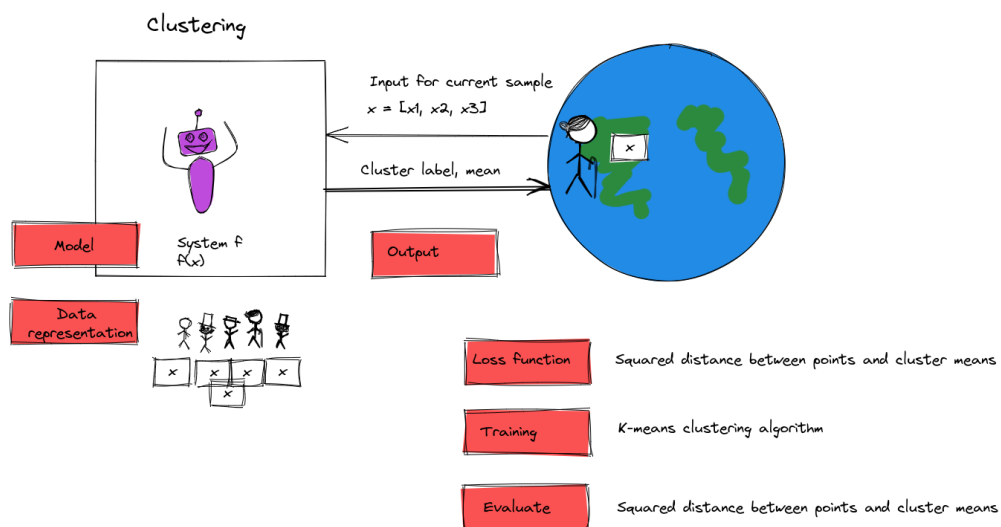


Figure 15: Clustering summary.

## Dimensionality reduction with deep learning

### Dimensionality reduction using an autoencoder

An *autoencoder* is a learner that includes:

- Encoder: produces low-dimensional representation of input,  $x \rightarrow z$
- Decoder: reconstructs an estimate of input from the low-dimensional representation,  $z \rightarrow \hat{x}$
- $z$  known as *latent variables*, *latent representation*, or *code*

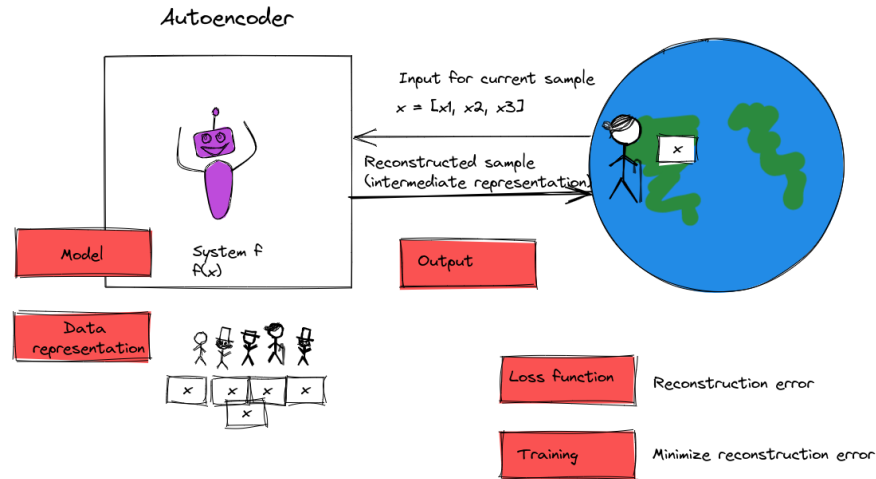


Figure 16: Autoencoder.

#### K-means as an autoencoder (1)

- Encoder: map each data point to one of  $K$  clusters
- Decoder: “reconstruct” data point as center of its cluster

#### K-means as an autoencoder (2)

- Let  $X \in \mathbb{R}^{n \times d}$  be the data matrix containing  $n$   $d$ -dimensional data points.
- Let  $Z$  be a  $n \times k$  matrix (if  $k$  clusters) where each entry is all zeros, except for one 1
- Let  $D$  be a  $k \times d$  matrix of cluster centers.

#### K-means as an autoencoder (3)

- Encoder performs non-linear mapping, expresses result as one-hot vector in  $Z$ .
- Decoder is linear:

$$X \approx \hat{X} = ZD$$

Note:  $Z$  was  $n \times k$ ,  $D$  was  $k \times d$ , so  $ZD$  will be  $n \times d$ .

#### PCA as an autoencoder (1)

- Let  $X \in \mathbb{R}^{n \times d}$  be the (mean-removed) data matrix containing  $n$   $d$ -dimensional data points.
- Let  $V$  be a  $d \times k$  matrix of  $k$  eigenvectors with highest eigenvalues
- $Z = XV$  is the  $n \times k$  matrix of PCA projections
- Then  $X \approx \hat{X} = ZV^T$

## PCA as an autoencoder (2)

- Encoder: linear projection using  $k$  best principal components
- Decoder: also linear projection

## Limits of PCA

- PCA learns linear projection
- Neural network with non-linear activation function can learn complex non-linear features
- Use neural network to do something like PCA?

## Neural autoencoder

- Neural network with  $d$  inputs,  $d$  outputs
- Use input as target
- (Self-supervised: creates its own labels)
- Train network to learn approximation of identity function

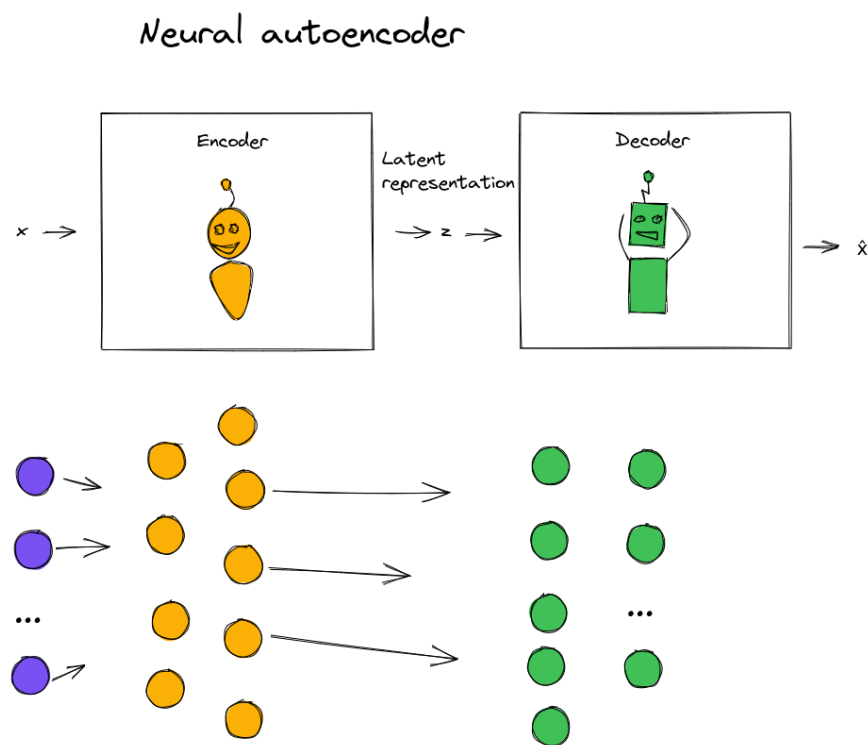


Figure 17: Neural autoencoder.

What should the architecture of the network be?

### Overcomplete autoencoder

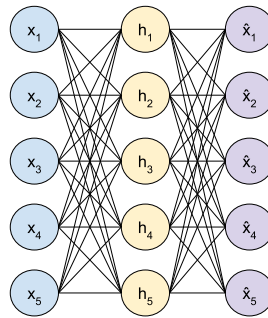


Figure 18: If we train this network to minimize reconstruction loss, it may literally learn the identity function - not useful.

### Undercomplete autoencoder

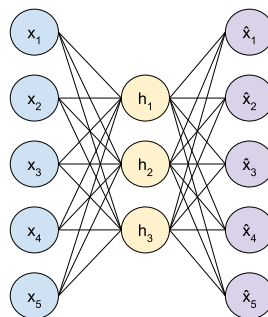


Figure 19: Is this network forced to learn a low-dimensional representation?

### Sparse autoencoder (1)

- Does a small “bottleneck” force autoencoder to learn useful latent features?
- Even if “bottleneck” is very small, can still memorize data by encoding index
- Instead of limiting number of hidden nodes, add a penalty function on *activations*

### Sparse autoencoder (2)

Allow many hidden units, but for a given input, most of them must produce a very small activation.

- Add penalty term to loss function, like regularization, but not on weights!
- Penalty is on average activation value (over all the training samples)

## Autoencoder comparison

- **Undercomplete autoencoder:** uses entire network for each sample. Limits capacity to memorize, but also limits capacity to extract complex features.
- **Sparse autoencoder:** different parts of network can “specialize” depending on input. Limits capacity to memorize, but can still extract complex features.

## What are autoencoders good for?

- Not typically useful for compression - too data-specific
- Can use to initialize supervised learning model - throw away decoder, fine-tune with classifier
- Can use for dimensionality reduction for data visualization (often in combination with other unsupervised learning methods)
- Can use for data denoising

## Example: reconstruction of faces

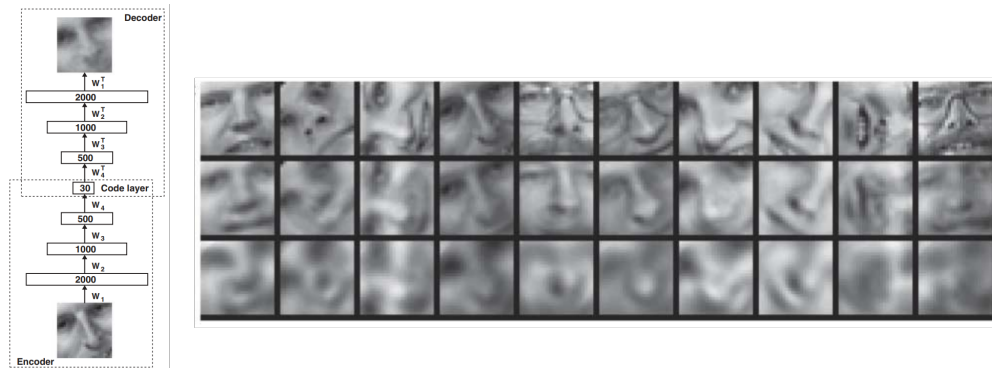


Figure 20: Reconstruction of faces (top) by 30-D neural autoencoder (middle) and 30-D PCA (bottom). Image via Hinton et al “Reducing the dimensionality of data with neural networks”, Science, 2006.

## Example: MNIST visualization

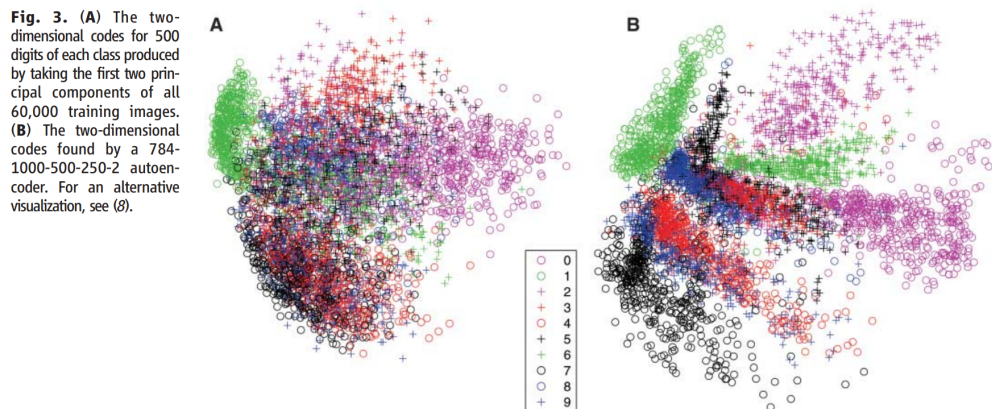


Figure 21: Image via Hinton et al “Reducing the dimensionality of data with neural networks”, Science, 2006.



## Density estimation

### Types of density estimation

- Explicit: define and solve for density (then sample from it if you want)
- Implicit: sample from density without defining it

### GAN: Generative adversarial networks

- From [Goodfellow et al 2014](#)
- Unsupervised, generative, implicit density estimation: Given training data, generate new samples from same distribution

### GAN: basic idea (1)

Two neural networks play a “game”:

Generator:

- takes random noise  $z$  drawn from  $p_z$  as input,
- generates samples, tries to trick “discriminator” into believing they are real,
- learns parameters  $\theta$ .

### GAN: basic idea (2)

Discriminator:

- takes samples  $x$  drawn from  $p_{\text{data}}$  as input,
- produces classification  $y$  (1=real, 0=fake),
- learns parameters  $\phi$ .

### Discriminator loss function (1)

Discriminator wants to update its parameters  $\phi$  so

- $D_\phi(x)$  (output for real data) is close to 1
- $D_\phi(G_\theta(z))$  (output for generated data) is close to 0

### Discriminator loss function (2)

Binary cross-entropy loss:

$$-\sum_{i=1}^N y_i \log D_\phi(x_i) - \sum_{i=1}^N (1 - y_i) \log(1 - D_\phi(x_i))$$

Left side is for “true” samples and the right side is for “fake” samples...

### Discriminator objective

Replace sums with expectations, then discriminator wants to *maximize*

$$\mathbb{E}_{x \sim p_{\text{data}}} [\log D_\phi(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D_\phi(G_\theta(z)))]$$

### Generator objective (1)

Generator wants to update its parameters  $\theta$  so that:

- $D_\phi(G_\theta(z))$  (output for generated data) is close to 1
- Minimize  $\mathbb{E}_{z \sim p_z} [\log(1 - D_\phi(G_\theta(z)))]$

### Overall objective

$$\min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{\text{data}}} [\log D_\phi(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D_\phi(G_\theta(z)))]$$

### Problem: gradient of cross-entropy loss

- Cross-entropy loss designed to accelerate learning (steep gradient) when classifier is wrong
- Gradient is flat when classifier is correct, when generator needs to improve!

### Generator objective (2)

- Instead, generator can do gradient *ascent* on the objective

$$\log(D_\phi(G_\theta(z^{(i)})))$$

- Instead of minimizing likelihood of discriminator being correct, now maximizing likelihood of discriminator being wrong.
- Can still learn even when discriminator is successful at rejecting generator samples

### Training: First, update discriminator

1. Get mini-batch of size  $m$  from data:  $x^{(1)}, \dots, x^{(m)} \sim p_{\text{data}}$
2. Get mini-batch of size  $m$  from noise input:  $z^{(1)}, \dots, z^{(m)} \sim p_z$
3. Forward pass: get  $G_\theta(z^{(i)})$  for each noise input, get  $D_\phi(x^{(i)})$  for each real sample, get  $D_\phi(G_\theta(z^{(i)}))$  for each fake sample.
4. Backward pass: gradient *ascent* on discriminator parameters  $\phi$ :

$$\frac{1}{m} \sum_{i=1}^m [\log D_\phi(x^{(i)}) + \log(1 - D_\phi(G_\theta(z^{(i)})))]$$

### Training: Then, update generator

5. Get mini-batch of size  $m$  from noise input:  $z^{(1)}, \dots, z^{(m)} \sim p_z$
6. Forward pass: get  $G_\theta(z^{(i)})$  for each noise input, get  $D_\phi(G_\theta(z^{(i)}))$  for each fake sample.
7. Backward pass: gradient *ascent* on generator parameters  $\theta$ :

$$\frac{1}{m} \sum_{i=1}^m \log(D_\phi(G_\theta(z^{(i)})))$$

### Illustration: training discriminator

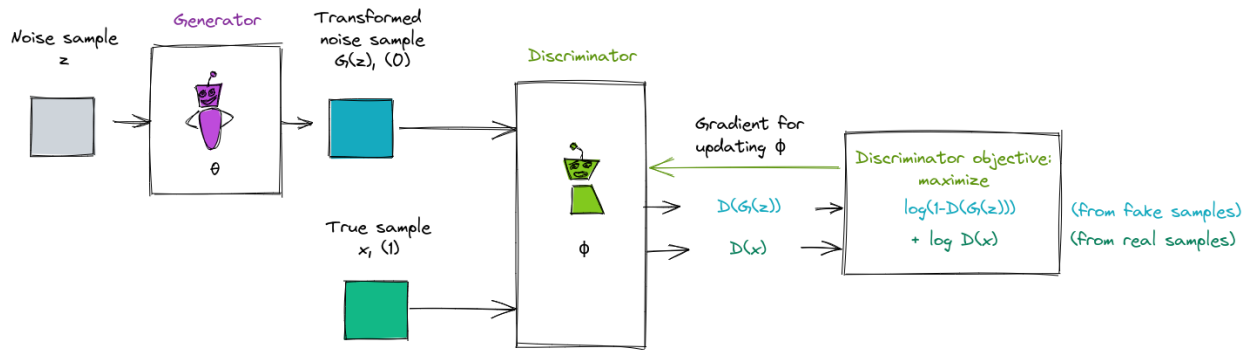


Figure 22: Training the discriminator.

### Illustration: training generator

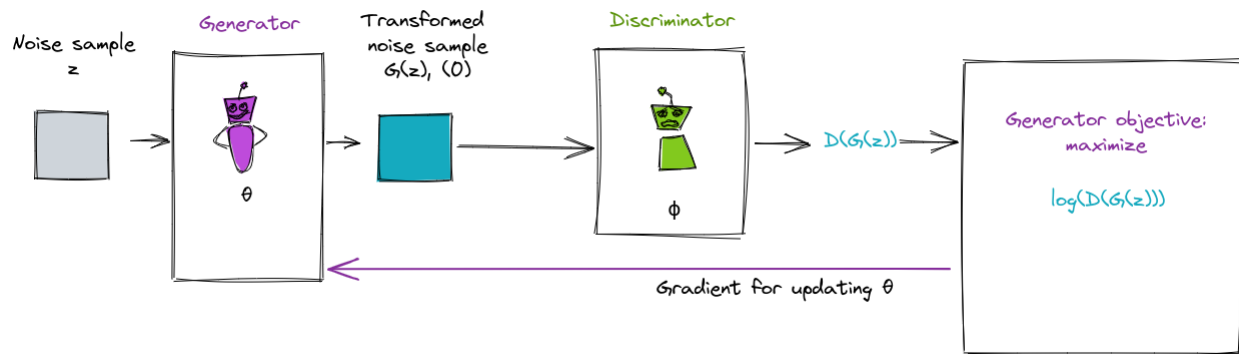


Figure 23: Training the generator.