Unsupervised Learning

Overview

Algorithms such as regression and classification "learn" a function $y = f(x)$

Unsupervised learning algorithms look at data (the X) and try to find patterns within it

Clustering and Principal Components Analysis are familiar examples

Supervised and Unsupervised Learning

Training the ML-based Classifier

Corpus

Classification

Feedback - loss function or cost function

Improves model parameters

Everything so far discussed really applied only to **Supervised Learning**

Types of ML Algorithms

Supervised

Labels associated with the training data is used to correct the algorithm

Unsupervised

The model has to be set up right to learn structure in the data

Supervised Learning

Input variable x and output variable y

Learn the mapping function $y = f(x)$

Approximate the mapping function so for new values of x we can predict y

Use existing dataset to correct our mapping function approximation

Unsupervised Learning

Only have input data x no output data

Model the underlying structure to learn more about data

Algorithms self discover the patterns and structure in the data

Unsupervised Learning does not have:

y variables

a labeled corpus

Why Look Within

In Life	In Machine Learning
To be emotionally self-sufficient	To make unlabelled data self-sufficient
To learn what values matter to you	Latent factor analysis
Identify others who share them...	Clustering
..and those who don't	Anomaly detection
Eliminate what does not matter	Quantisation
In general, to train yourself to navigate the outside world	Pre-training for supervised learning problems (classification, regression)

Unsupervised Learning Use-cases

ML Technique	Use-case
To make unlabelled data self-sufficient	Identify photos of a specific individual
Latent factor analysis	Find common drivers of 200 stocks
Clustering	Find relevant document in a corpus
Anomaly detection	Flag fraudulent credit card transactions
Quantisation	Compress true color (24 bit) to 8 bit
Pre-training for supervised learning problems (classification, regression)	All of the above!

Unsupervised Learning Use-cases

What	How
To make unlabelled data self-sufficient	Autoencoder
Latent factor analysis	Autoencoder
Clustering	Clustering
Anomaly detection	Autoencoder
Quantisation	Clustering
Pre-training for supervised learning problems (classification, regression)	All of the above!

Unsupervised ML Algorithms

Clustering

Identify patterns in data items e.g. K-means clustering

Autoencoding

Identify latent factors that drive data e.g. PCA

Clustering Algorithms

Unsupervised ML Algorithms

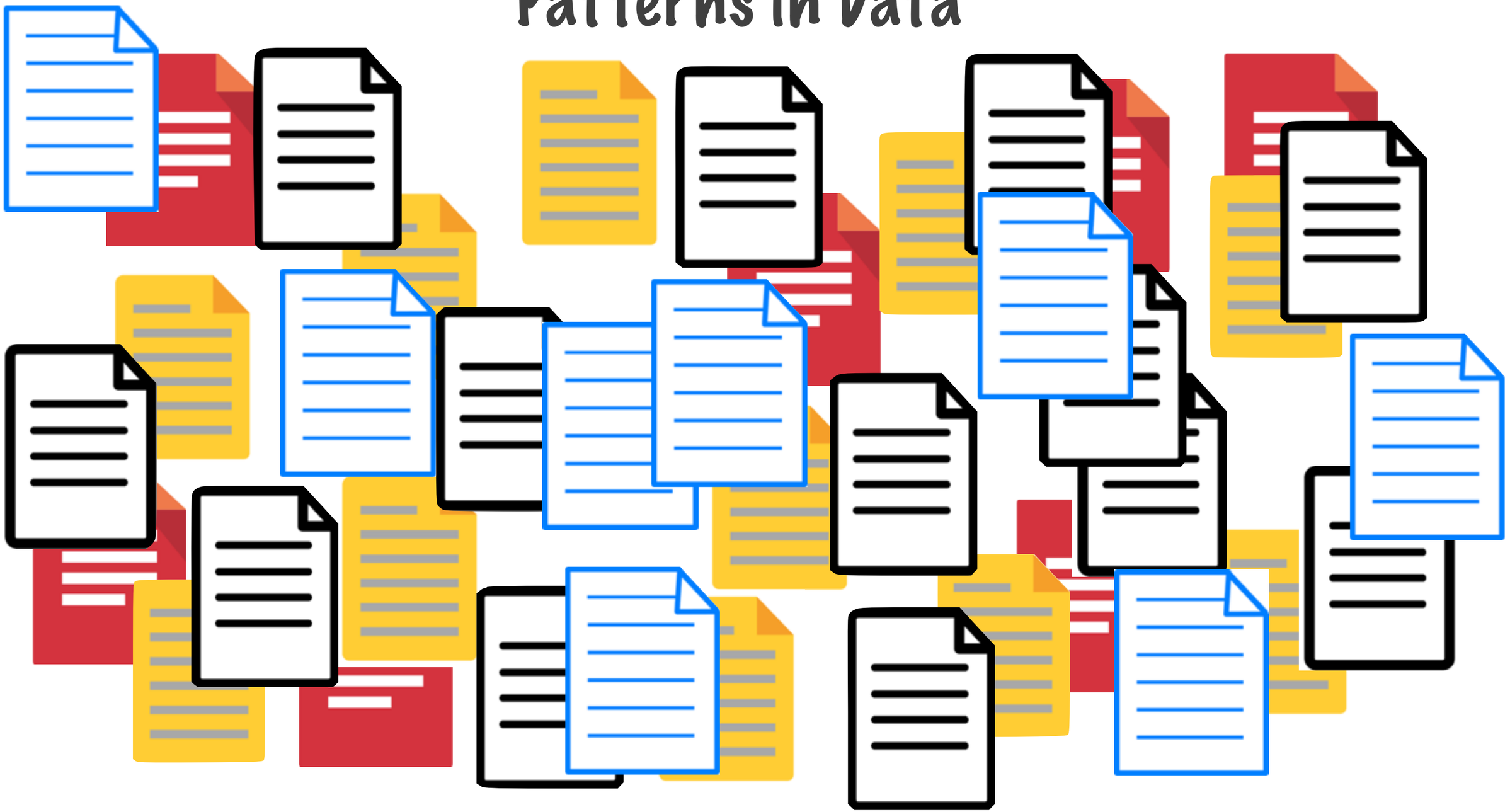
Clustering

Identify patterns in data items e.g. K-means clustering

Autoencoding

Identify latent factors that drive data e.g. PCA

Patterns in Data



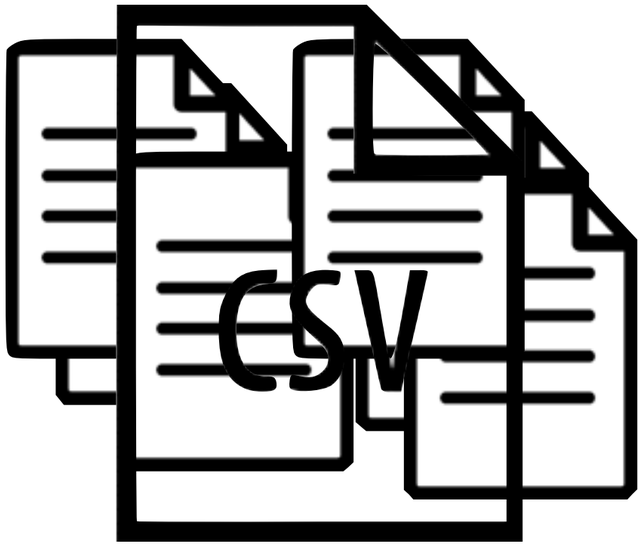
Patterns in Data

How do you make sense of this?

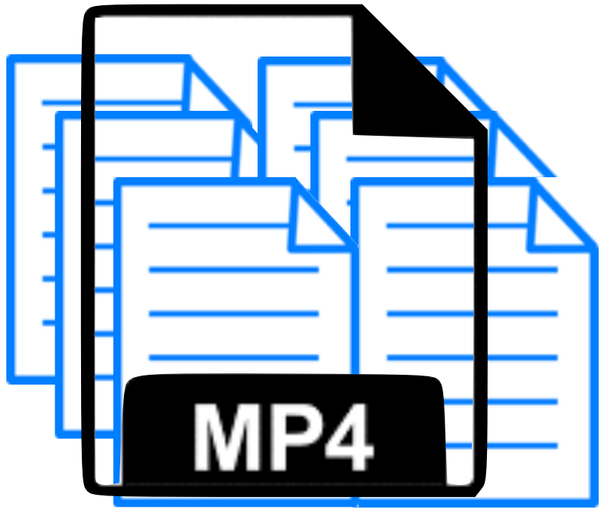
Patterns in Data

Group them based on some common attributes

Patterns in Data



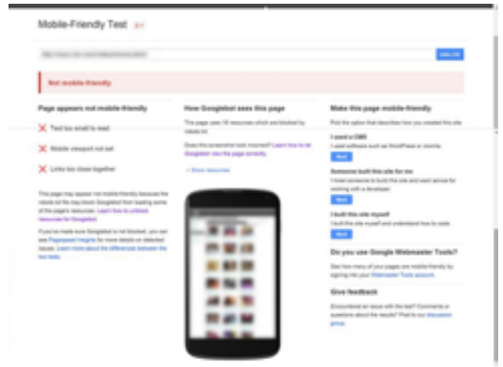
Clustering



Clustering

Products sold on Amazon

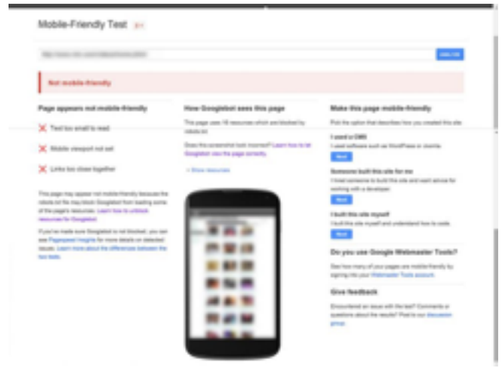
People on Facebook



Websites indexed by Google

What if you want to group more complex entities?

Clustering



Products sold on Amazon

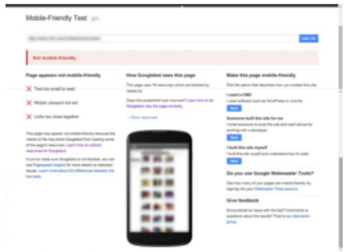
People on Facebook

Websites indexed by Google

Too many entities, too many attributes per entity

Huge complexity

Clustering

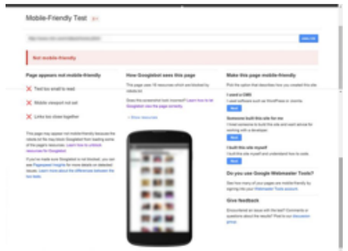


Anything can be represented by a set of numbers

Clustering

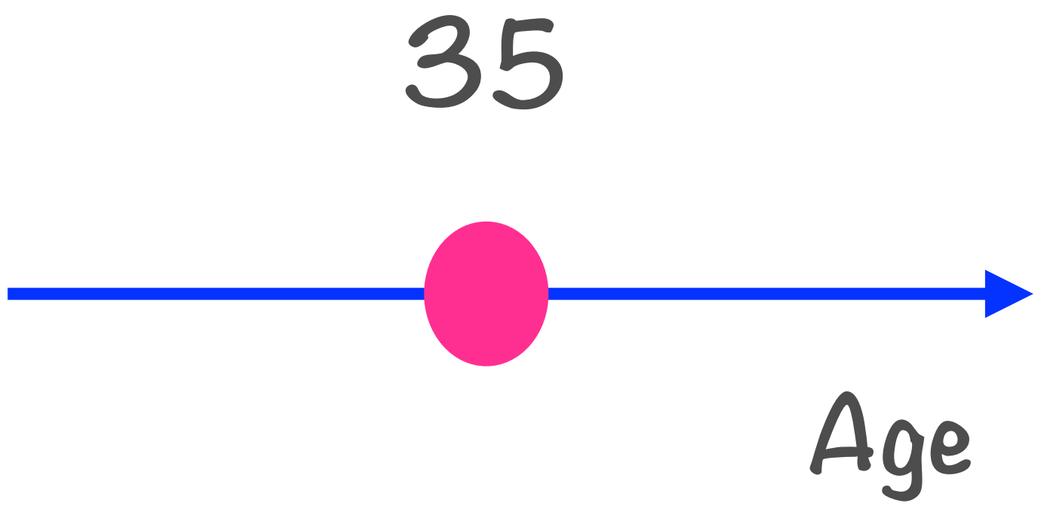
Product ID, Timestamp, Amount

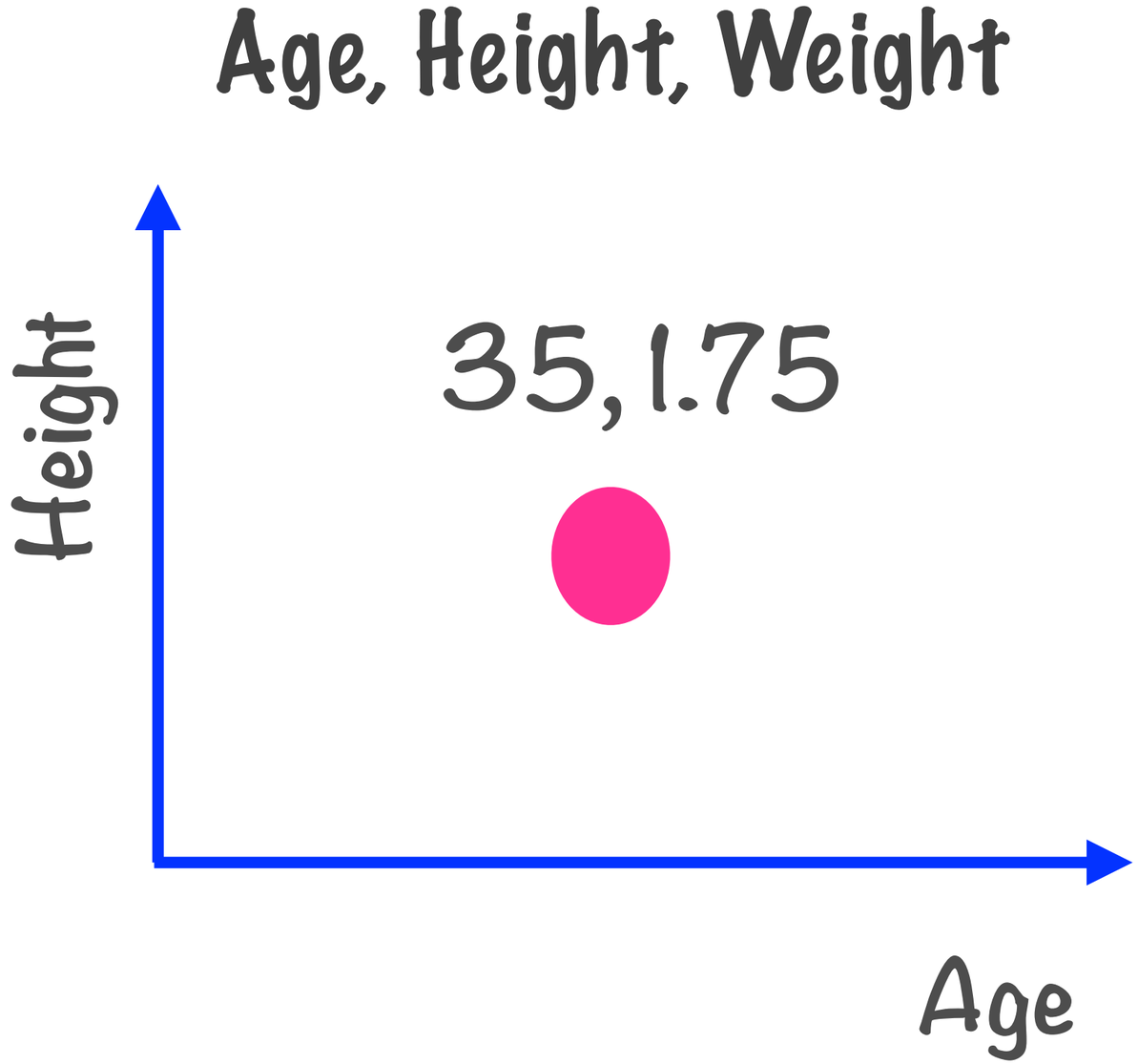
Age, Height, Weight

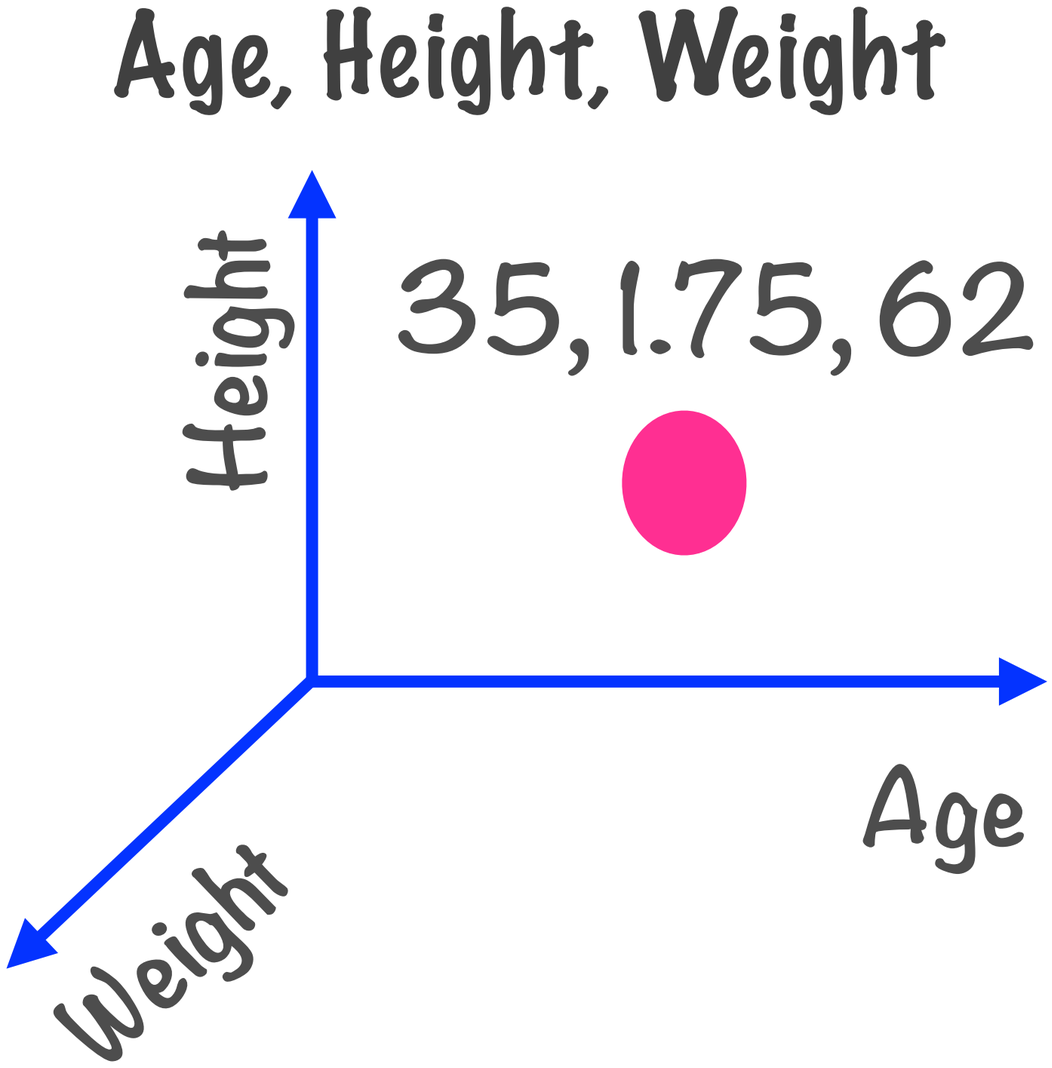


Length, word frequencies

Age, Height, Weight




Age, Height, Weight
Height
35, 1.75
Age

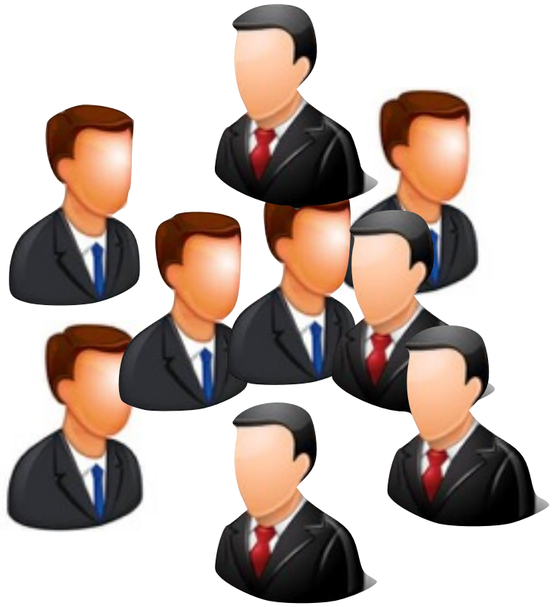

Age, Height, Weight
Height
35, 1.75, 62
Age
Weight

A set of N numbers represents a point in an N-dimensional Hypercube

Clustering

A set of points, each representing a Facebook user

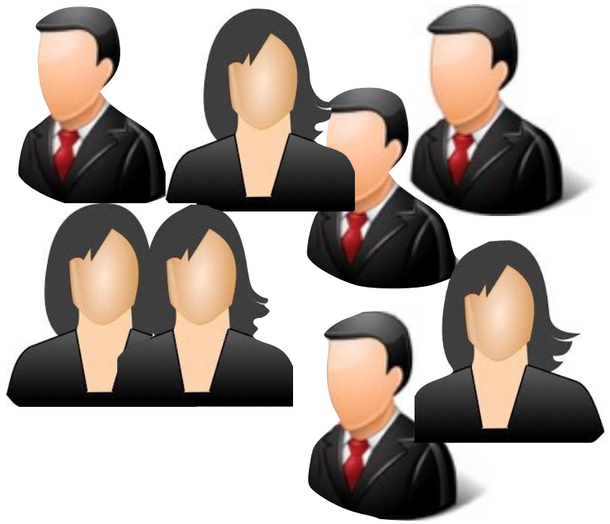
Clustering



Same group = similar

Different group = different

Clustering



Same group = similar

Different group = different

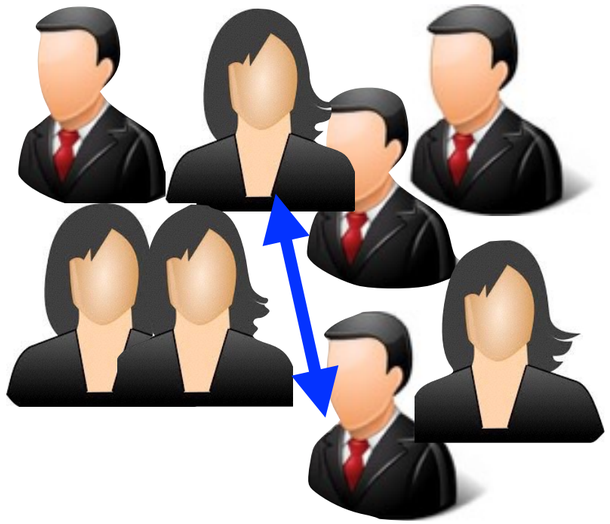
Users in a Cluster

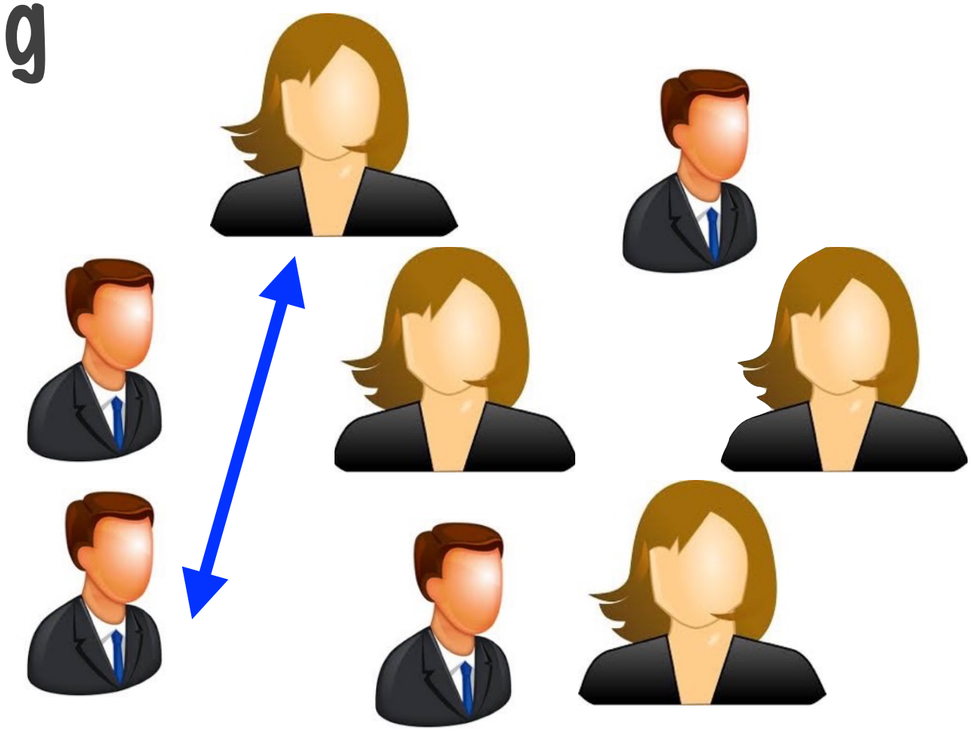
May like the same kind of music

May have gone to the same high school

May have kids of the same age

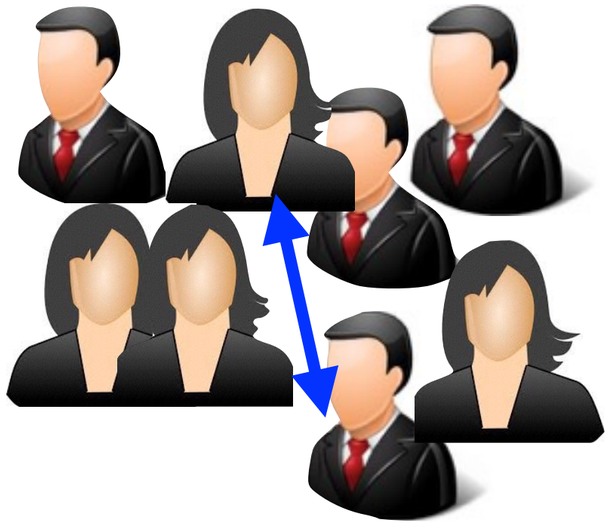
Clustering

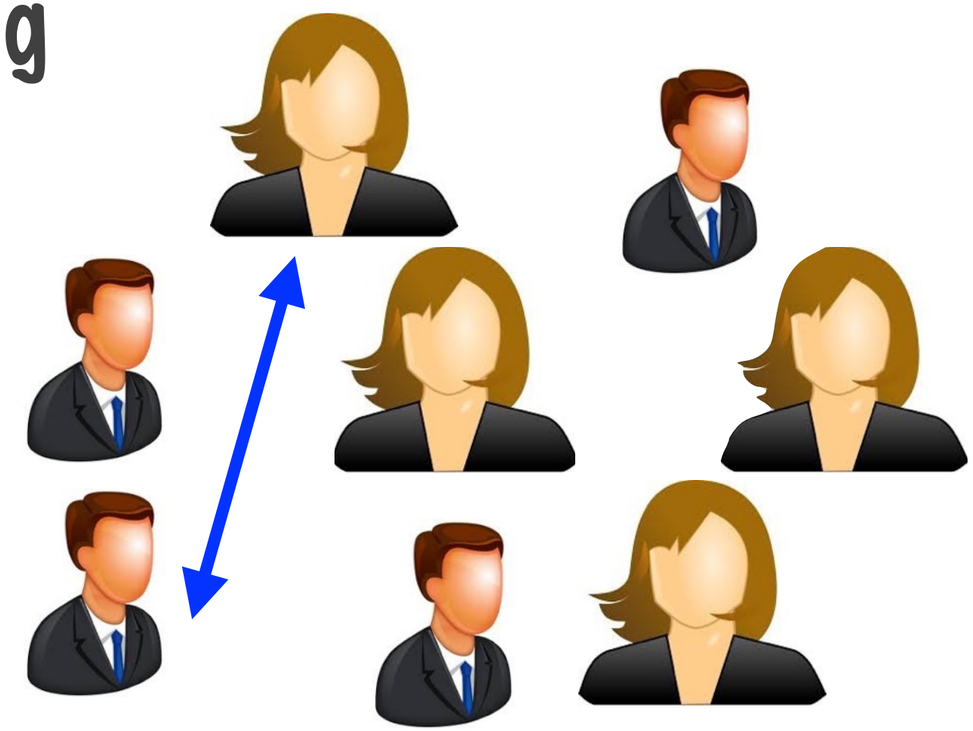




The distance between users in a cluster indicates how similar they are

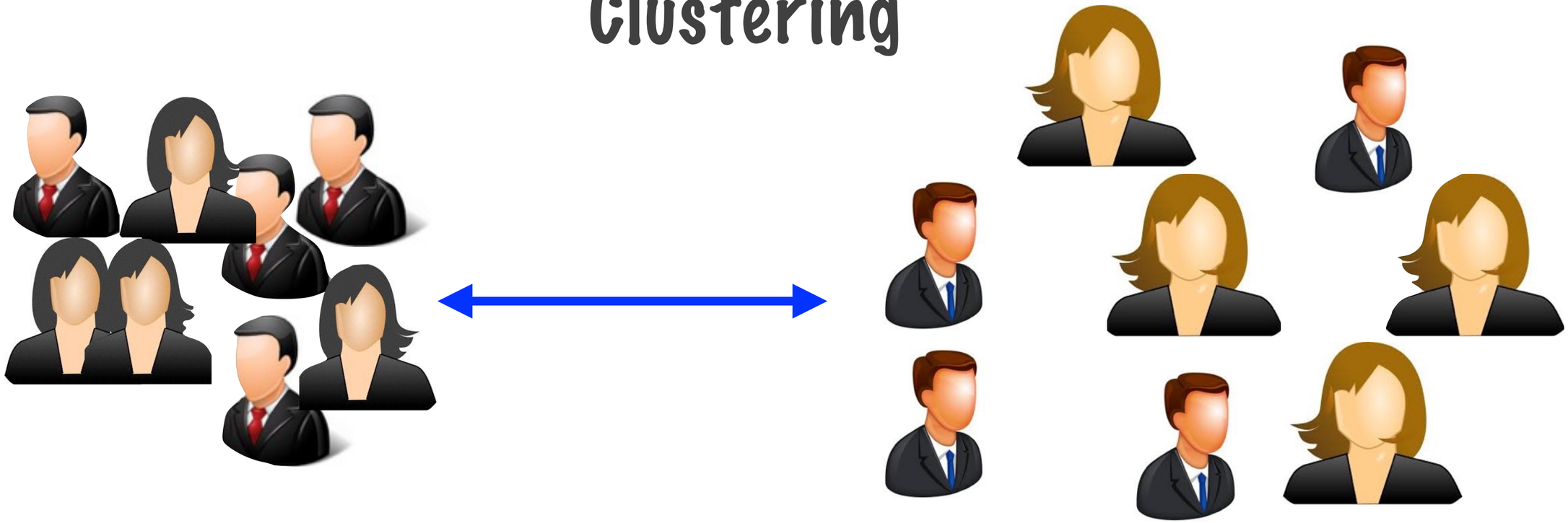
Clustering





Maximize intra-cluster similarity

Clustering

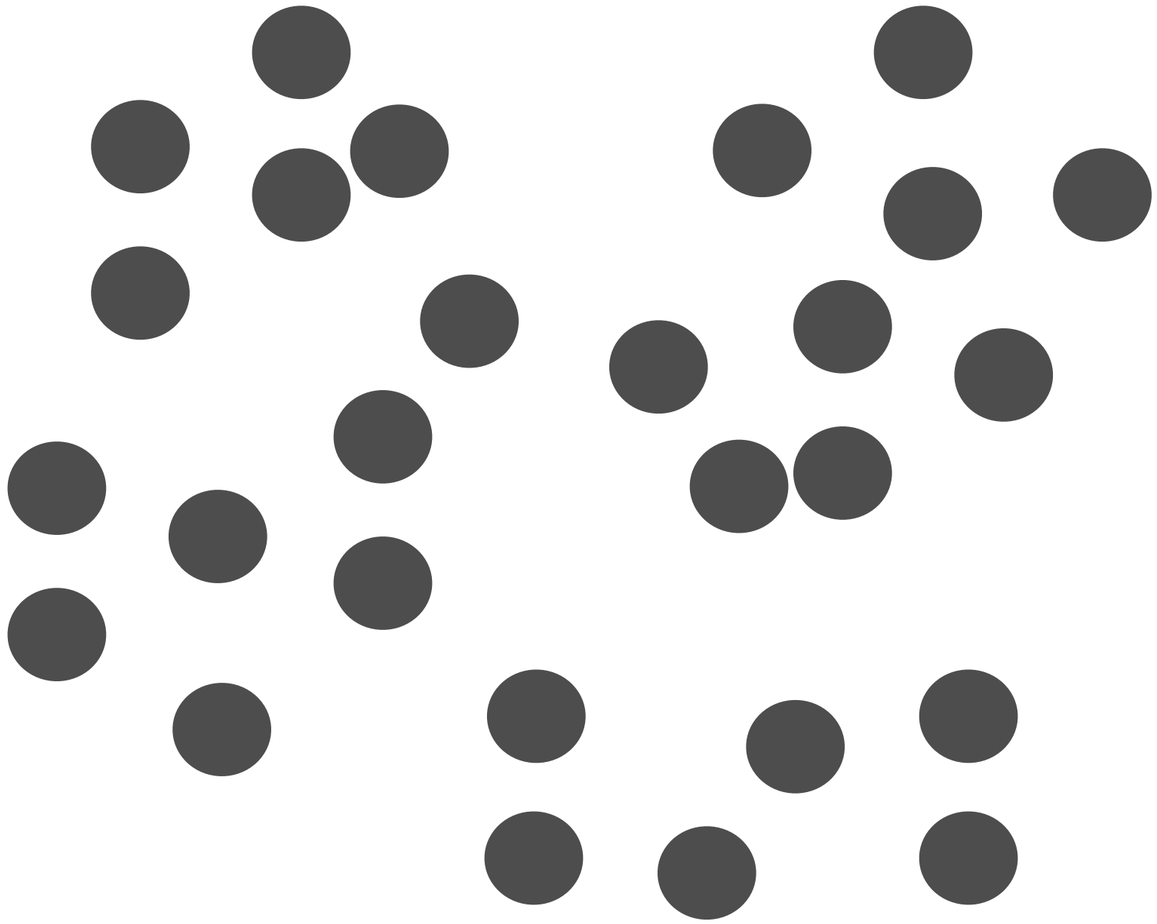


Minimize inter-cluster similarity

The **K-Means Clustering** algorithm is a famous Machine Learning algorithm to achieve this

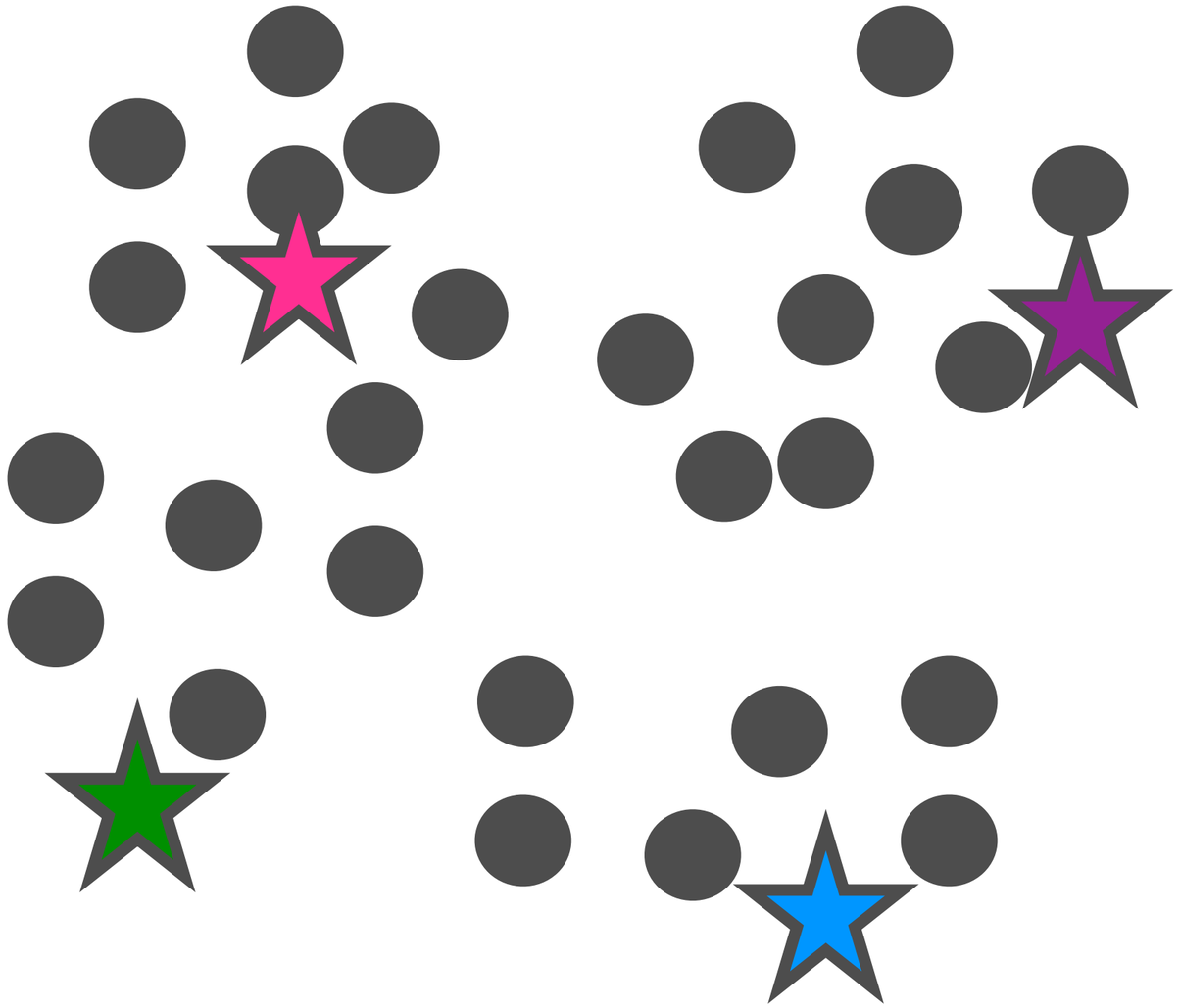
K-Means Clustering

Initialize K centroids i.e means



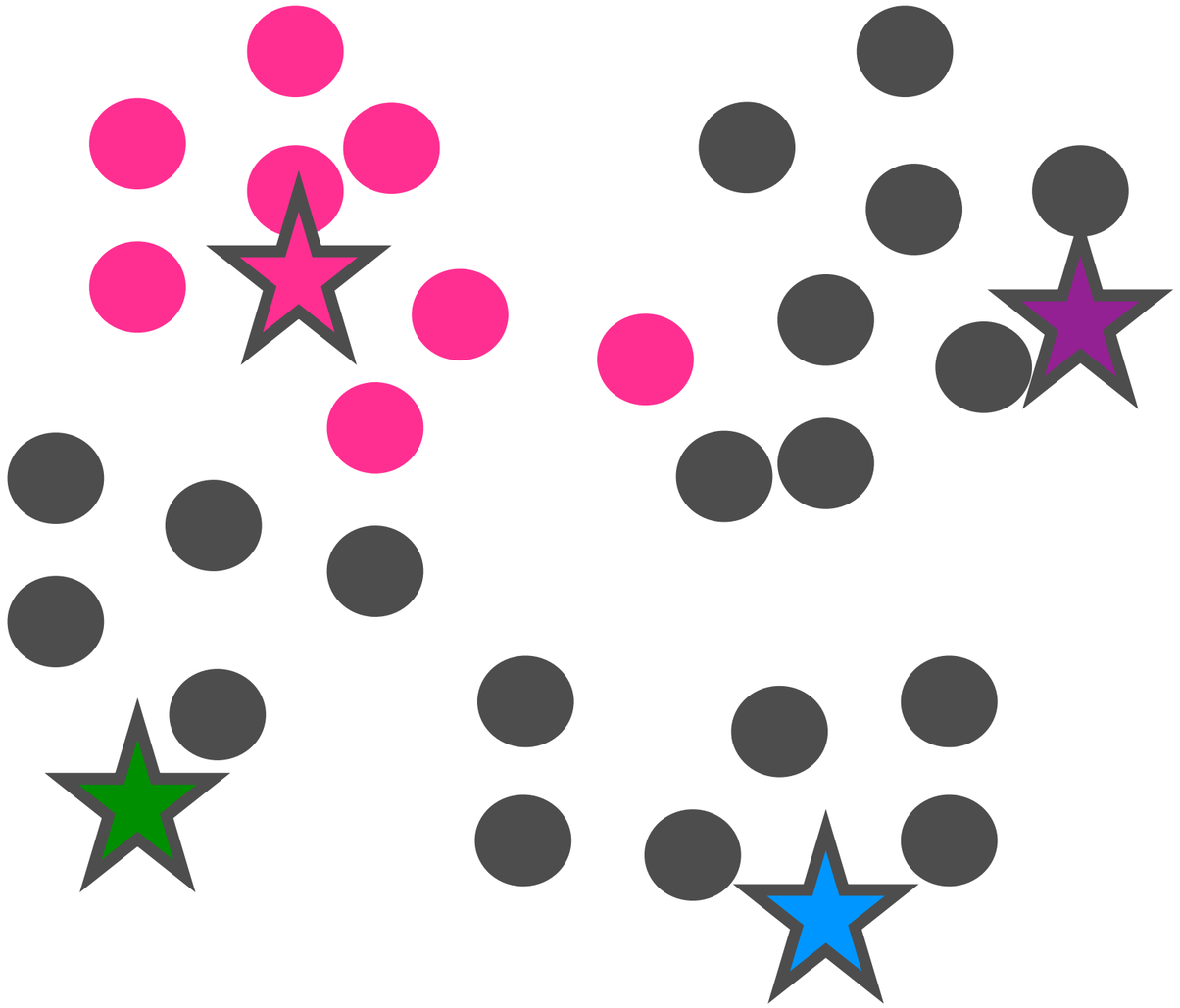
K-Means Clustering

Assign each point to a cluster



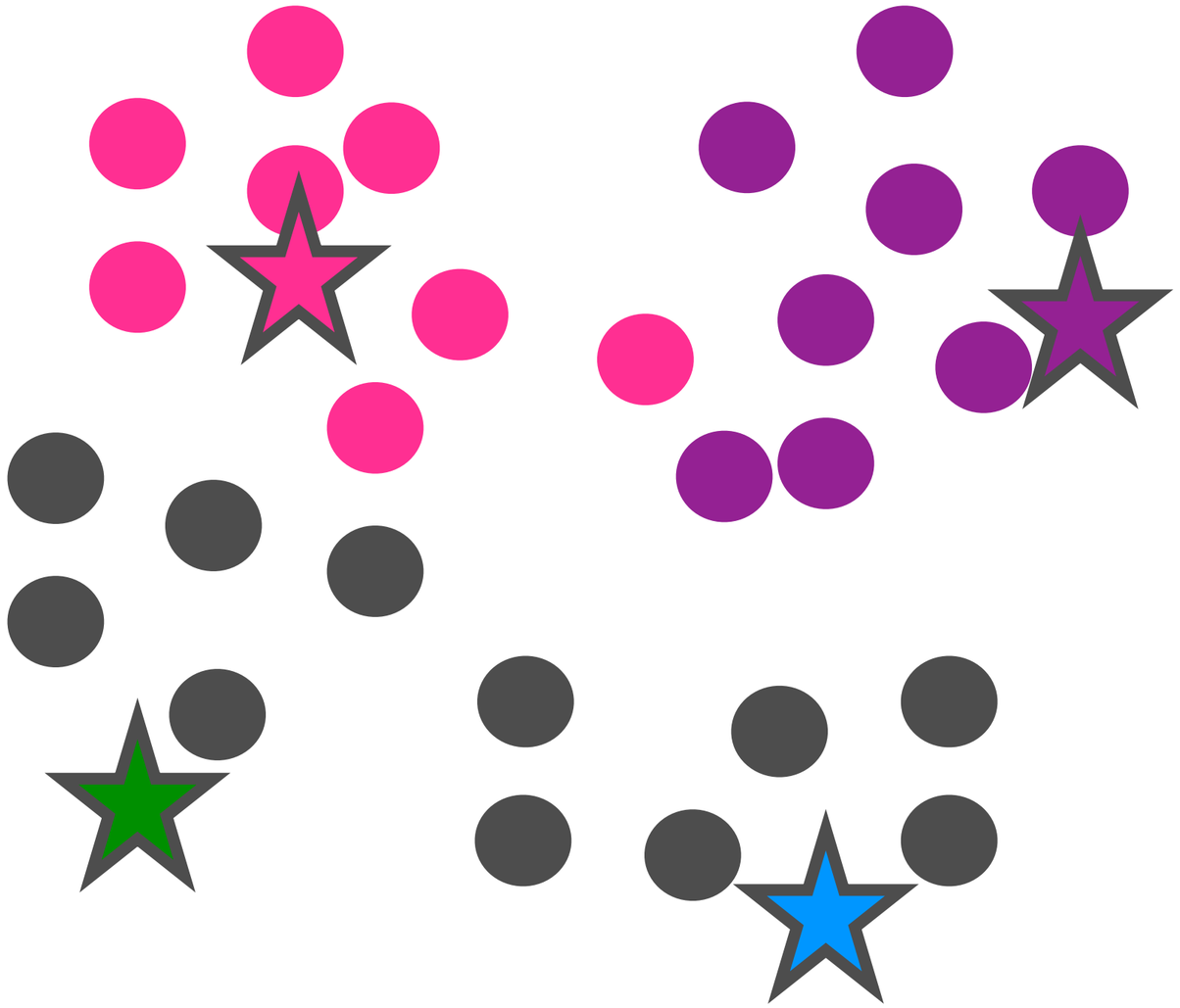
K-Means Clustering

Assign each point to a cluster



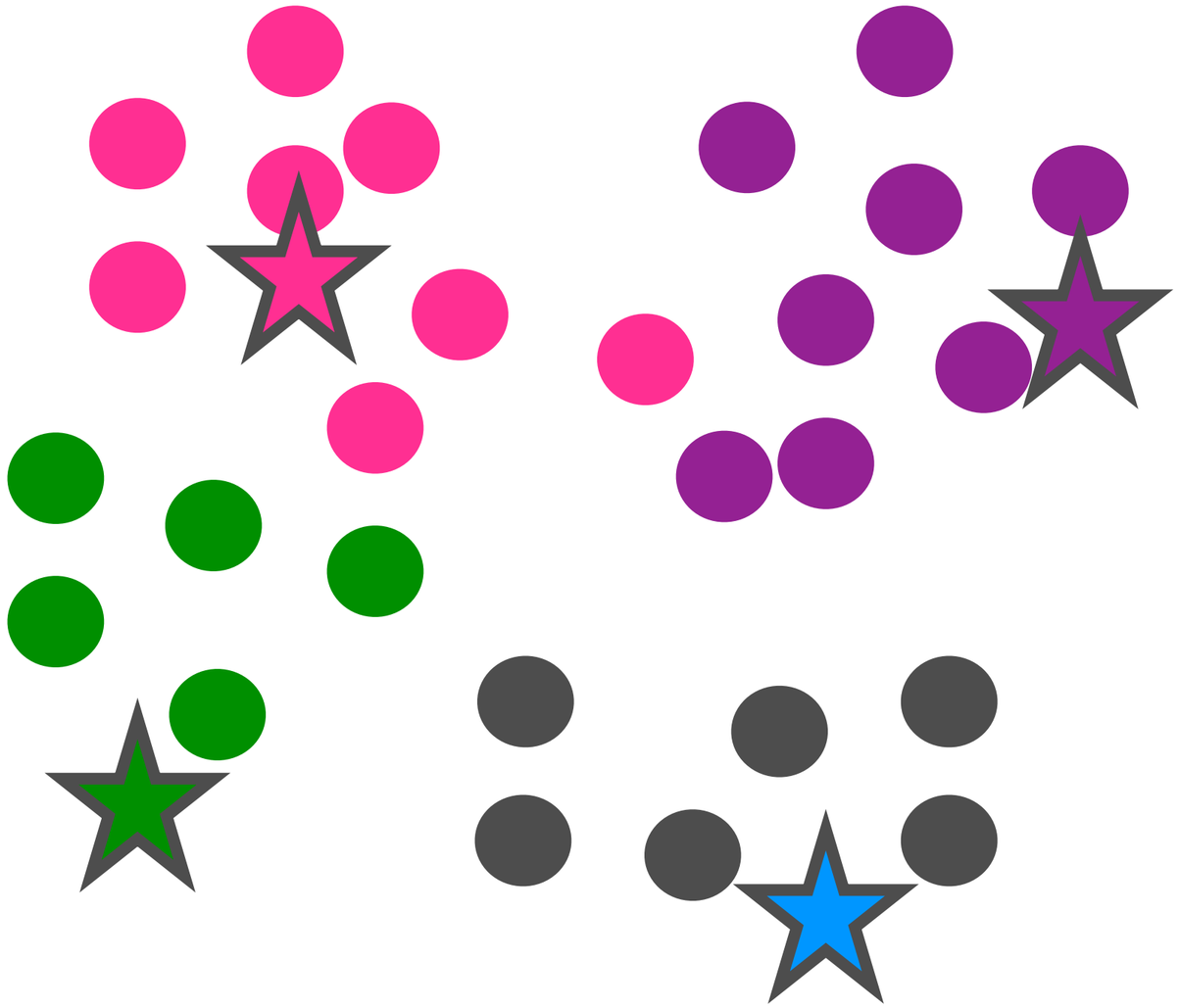
K-Means Clustering

Assign each point to a cluster



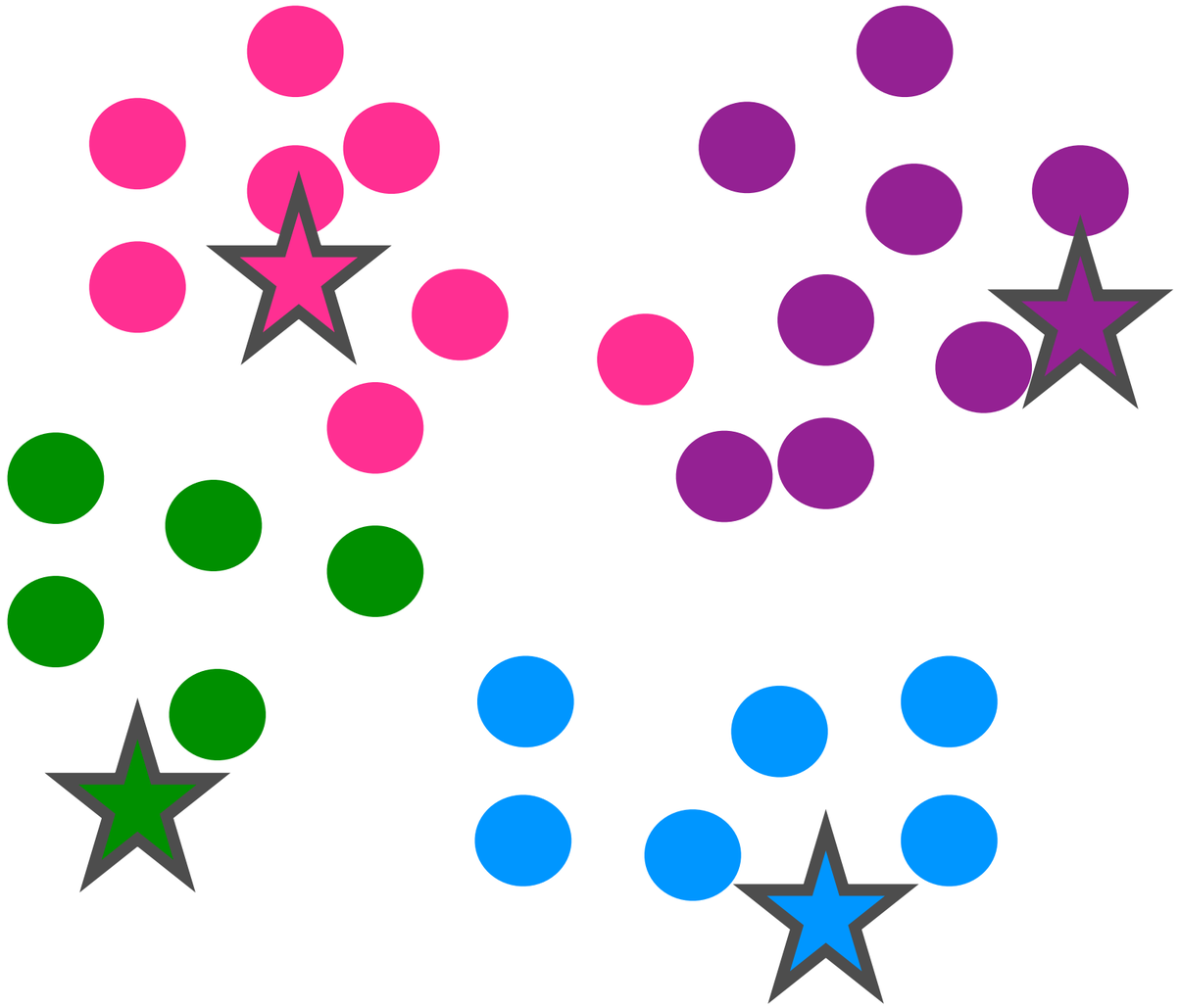
K-Means Clustering

Assign each point to a cluster



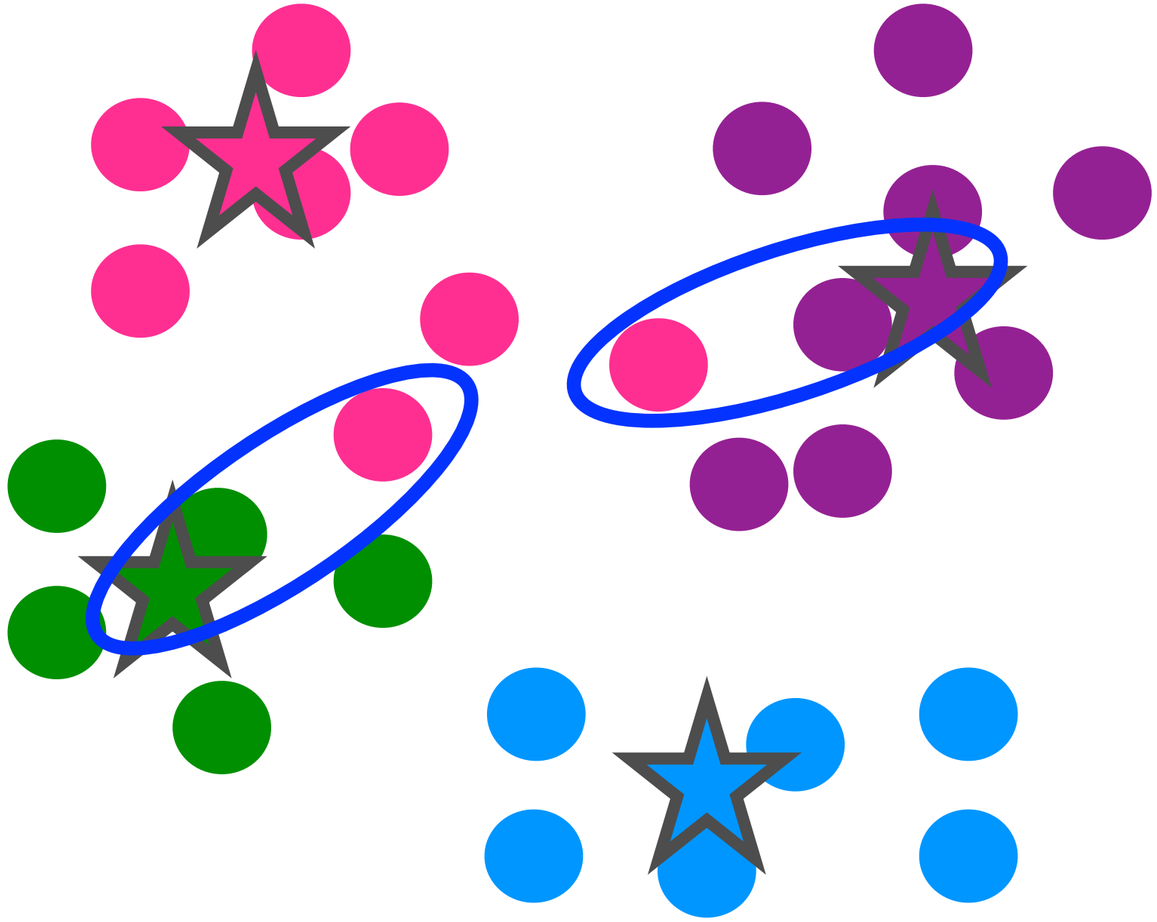
K-Means Clustering

Recalculate the mean for each cluster



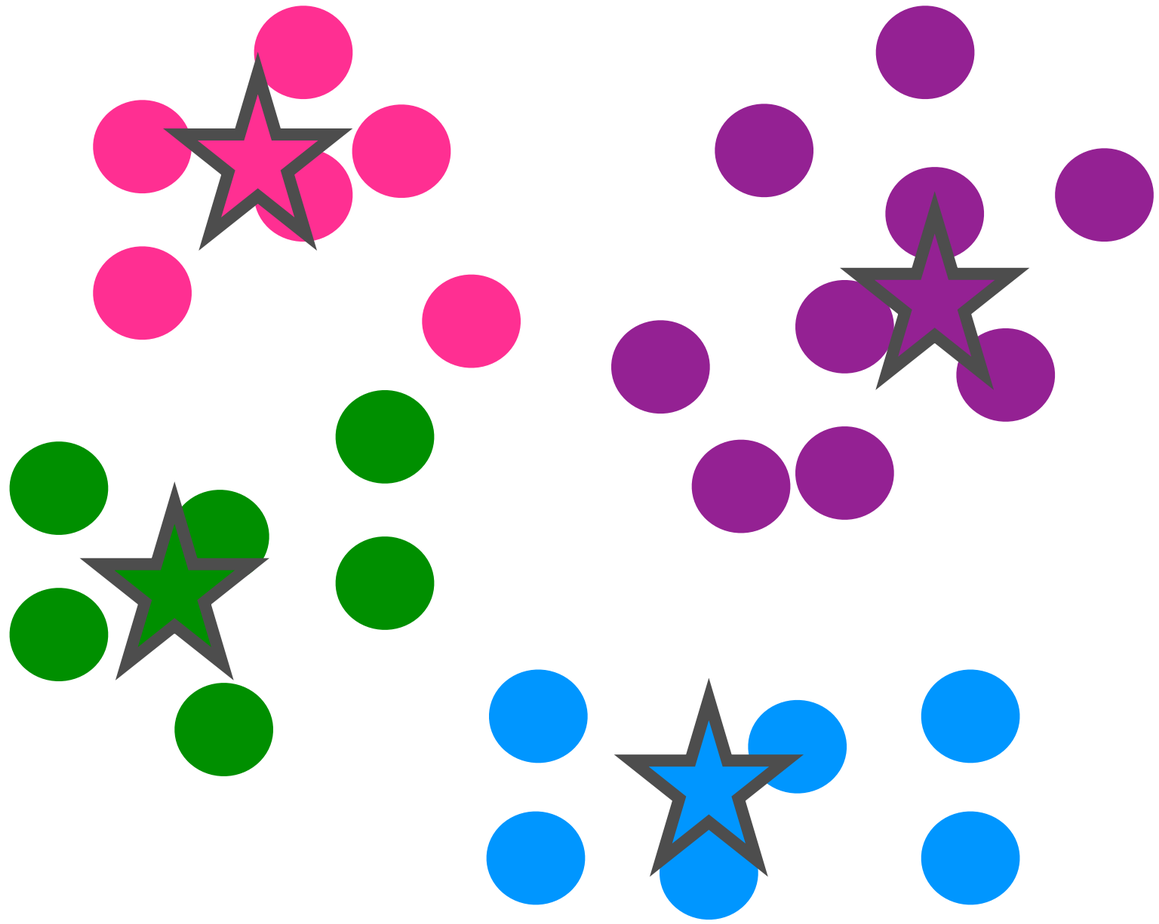
K-Means Clustering

Re-assign the points to clusters

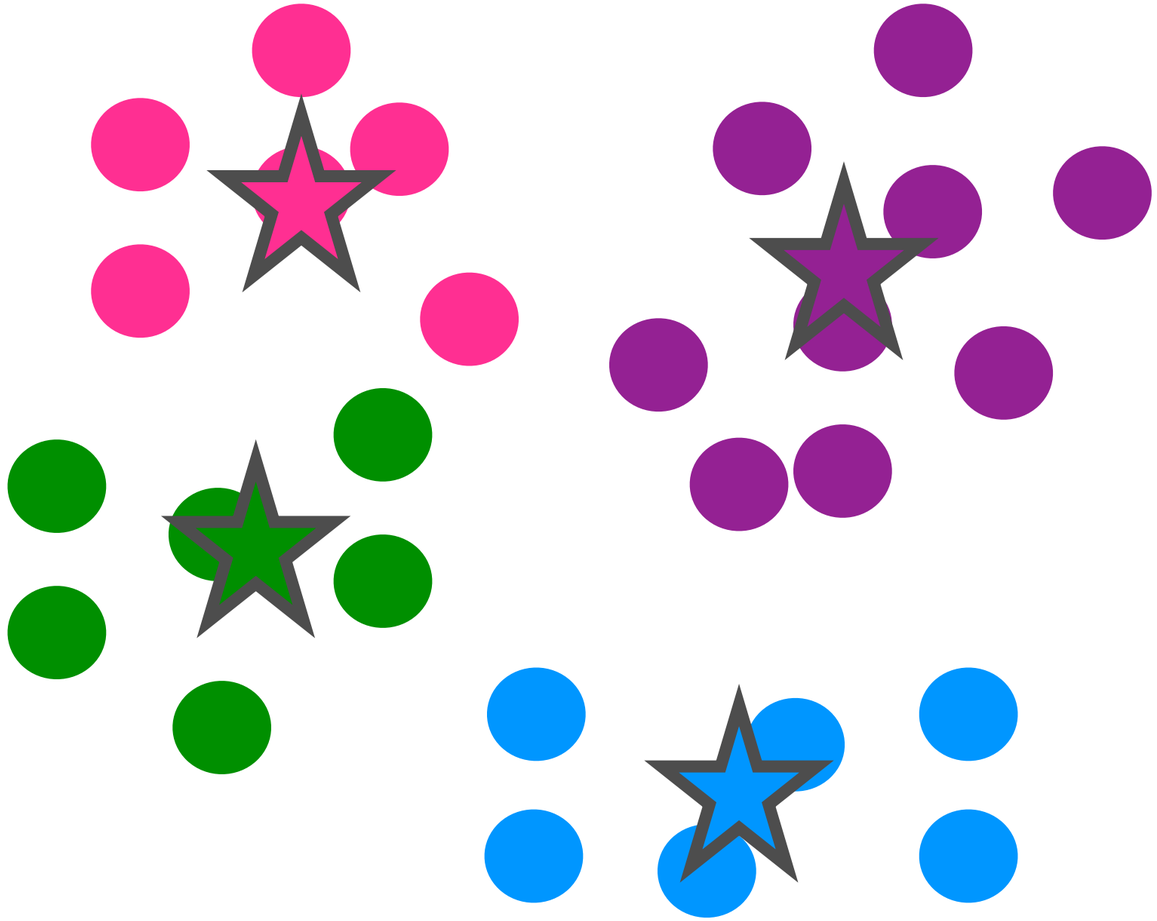


K-Means Clustering

Iterate until points are in their final clusters

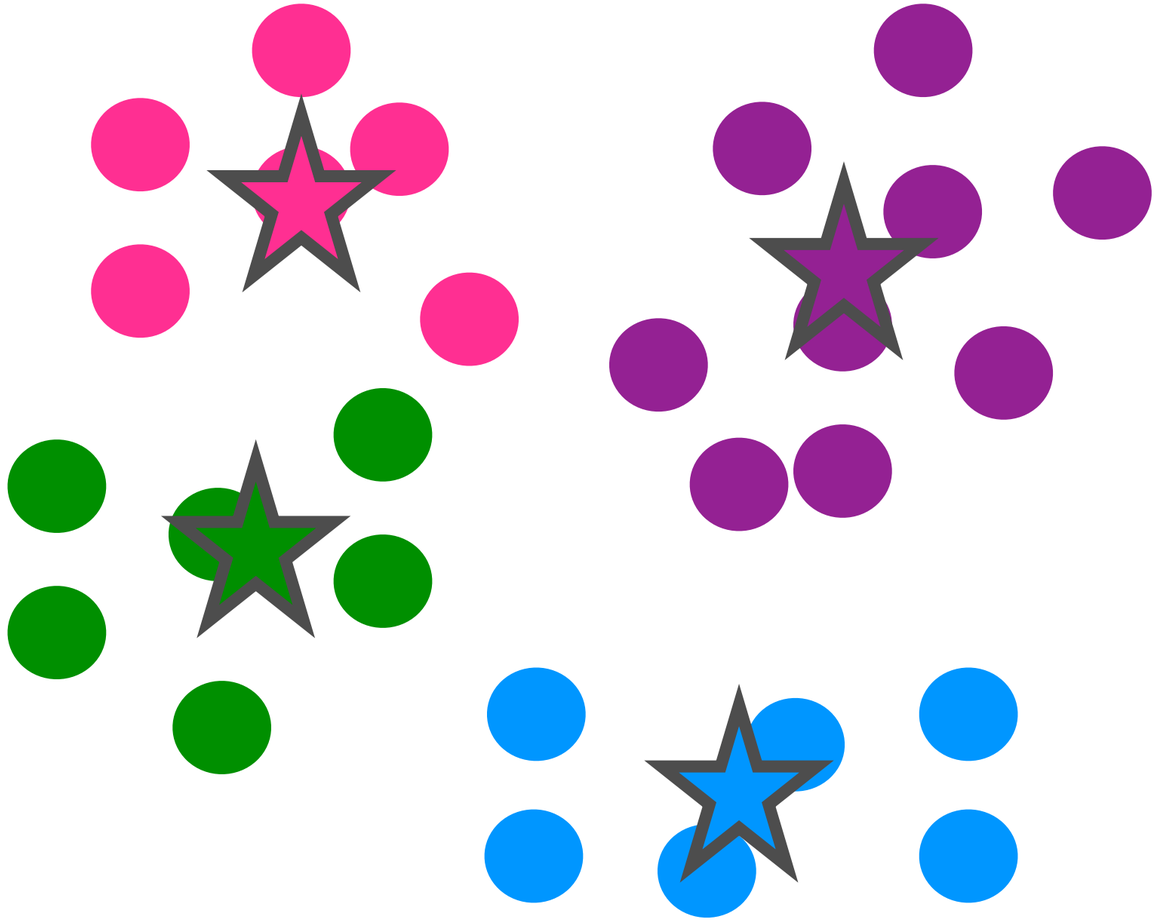


K-Means Clustering

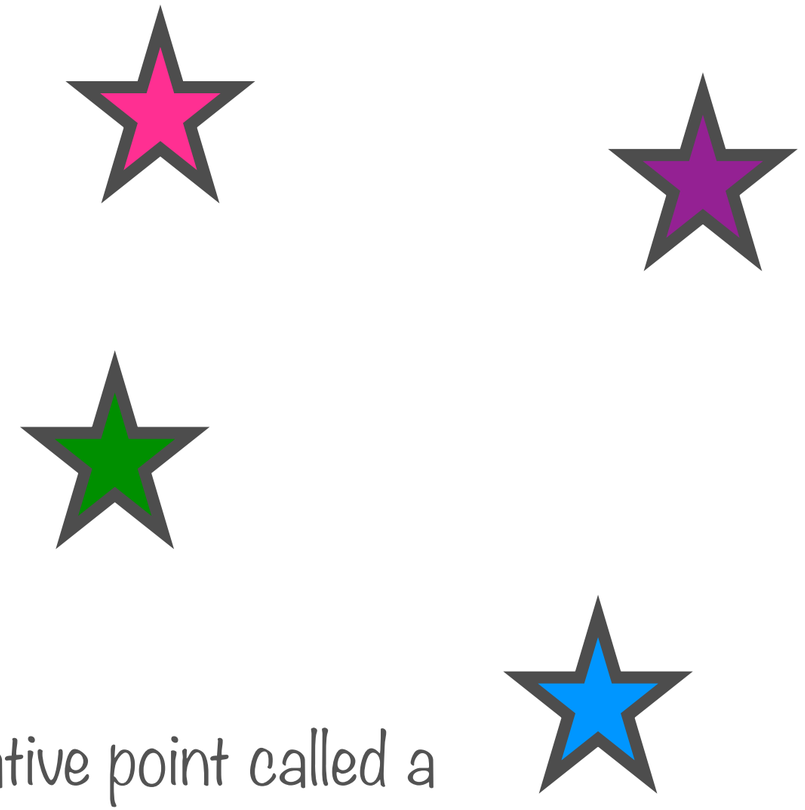


Convergence

K-Means Clustering



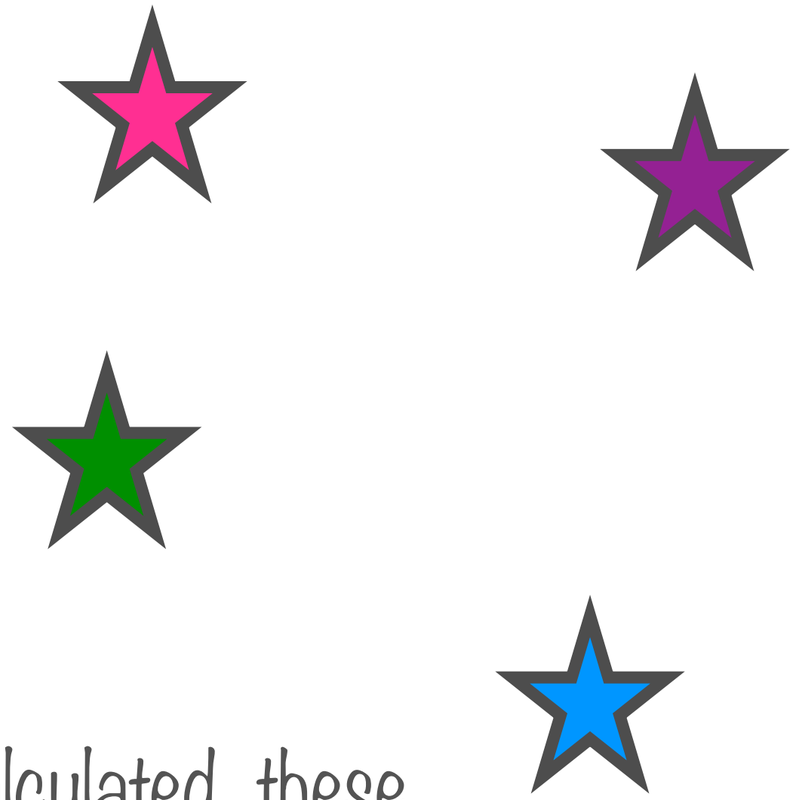
K-Means Clustering



Each cluster has a representative point called a

reference vector

K-Means Clustering



Because of how they are calculated, these reference vectors are often called centroids

Initialise K centroids

Repeat:

For each data point:

Assign to “nearest” cluster

For each centroid:

Update coordinates

Have centroids converged?

Yes: Stop, we’re done

No: Keep iterating

- Pick an initial solution (algorithms exist to pick well)
- Iterate until convergence
  - Update assignments of points to clusters
  - Update coordinates of reference vectors
- Keep iterating until we converge

Initialise K centroids

Repeat:

For each data point:

Assign to "nearest" cluster

For each centroid:

Update coordinates

Have centroids converged?

Yes: Stop, we're done

No: Keep iterating

- Hyperparameters
  - Number of clusters
  - Initial values of centroids

Initialise K centroids

Repeat:

For each data point:

Assign to "nearest" cluster

For each centroid:

Update coordinates

Have centroids converged?

Yes: Stop, we're done

No: Keep iterating

- Design choice #1:
  - Distance measure between point, cluster
  - Euclidean distance often used

Initialise K centroids

Repeat:

For each data point:

Assign to "nearest" cluster

For each centroid:

Update coordinates

Have centroids converged?

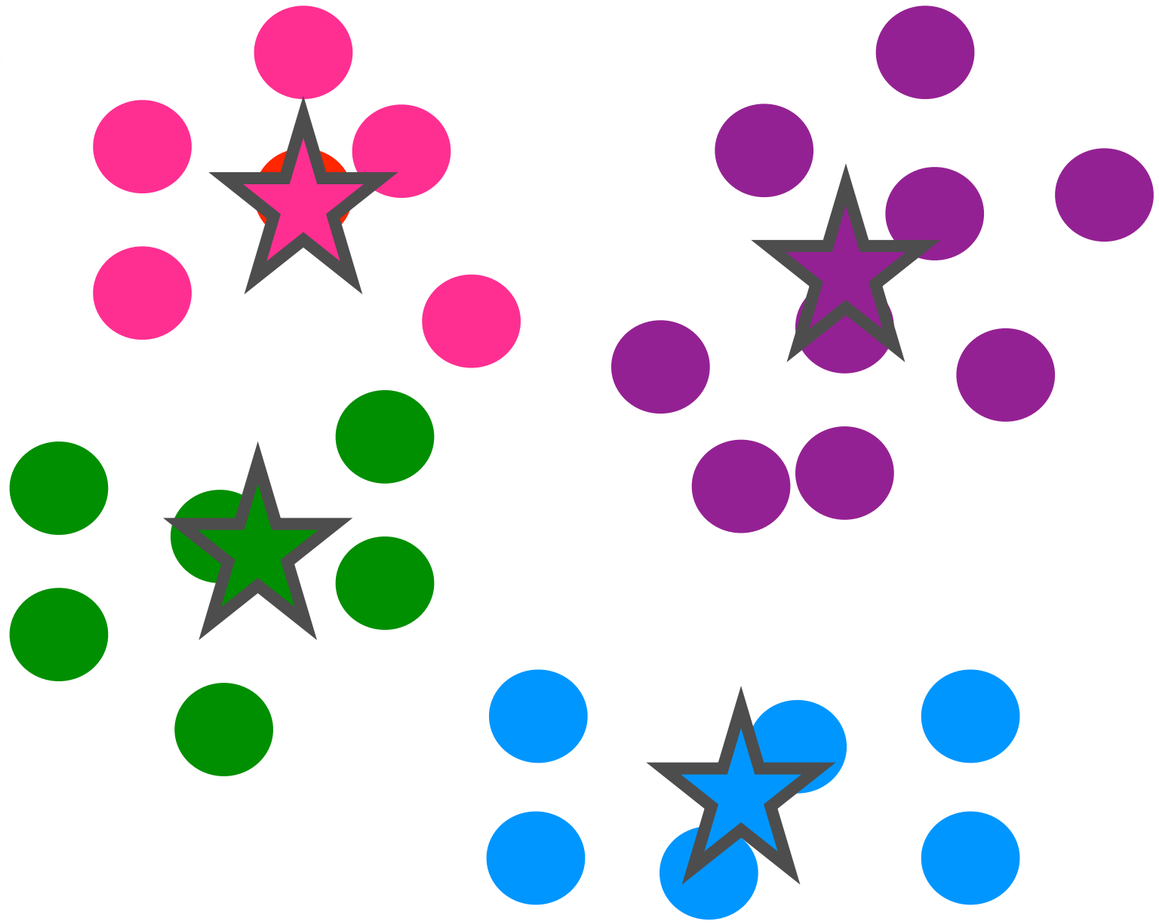
Yes: Stop, we're done

No: Keep iterating

- Design choice #2:
  - Calculating cluster center from points in cluster
  - Centroid (simple average) often used

Total Reconstruction Error

...Minimising total reconstruction error

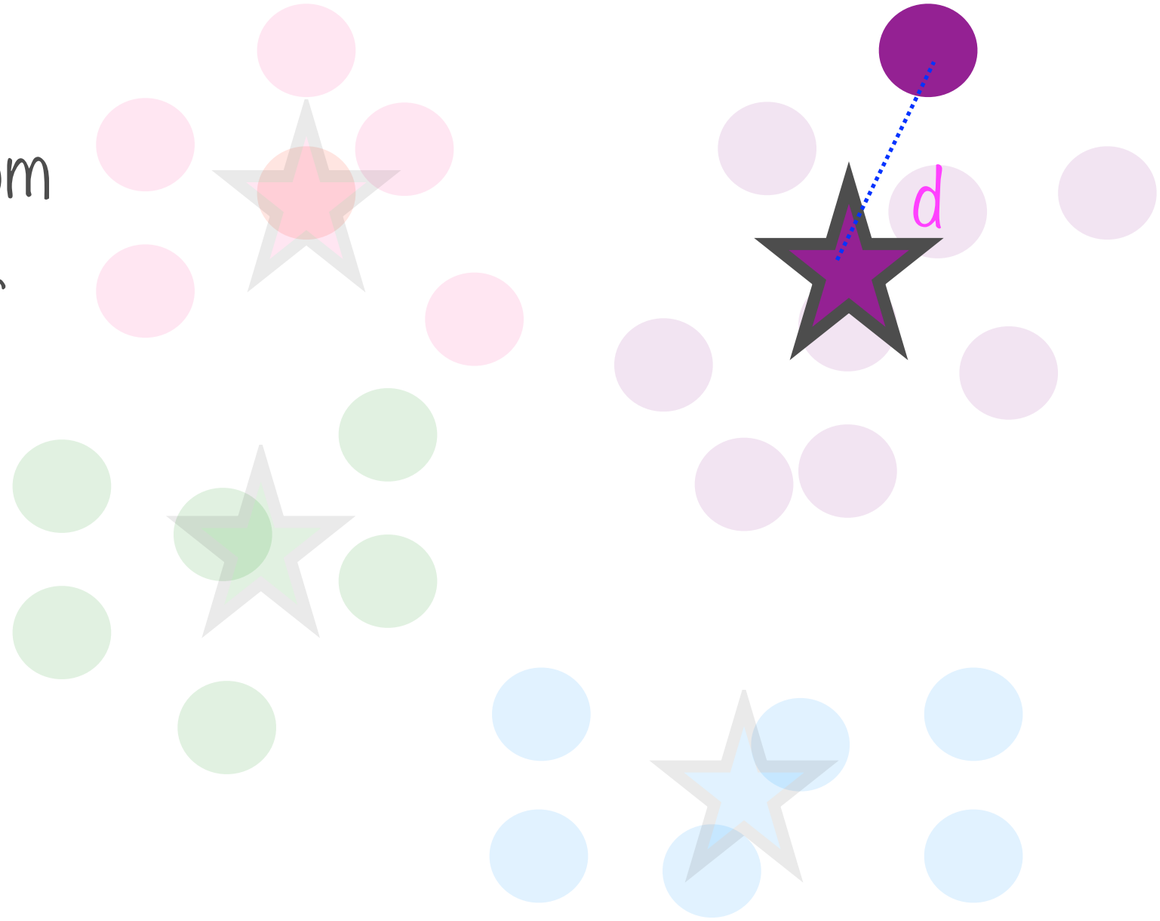


The lower the total reconstruction error, the better the fit

Individual Reconstruction Error

Square of Euclidean distance of each point from nearest reference vector

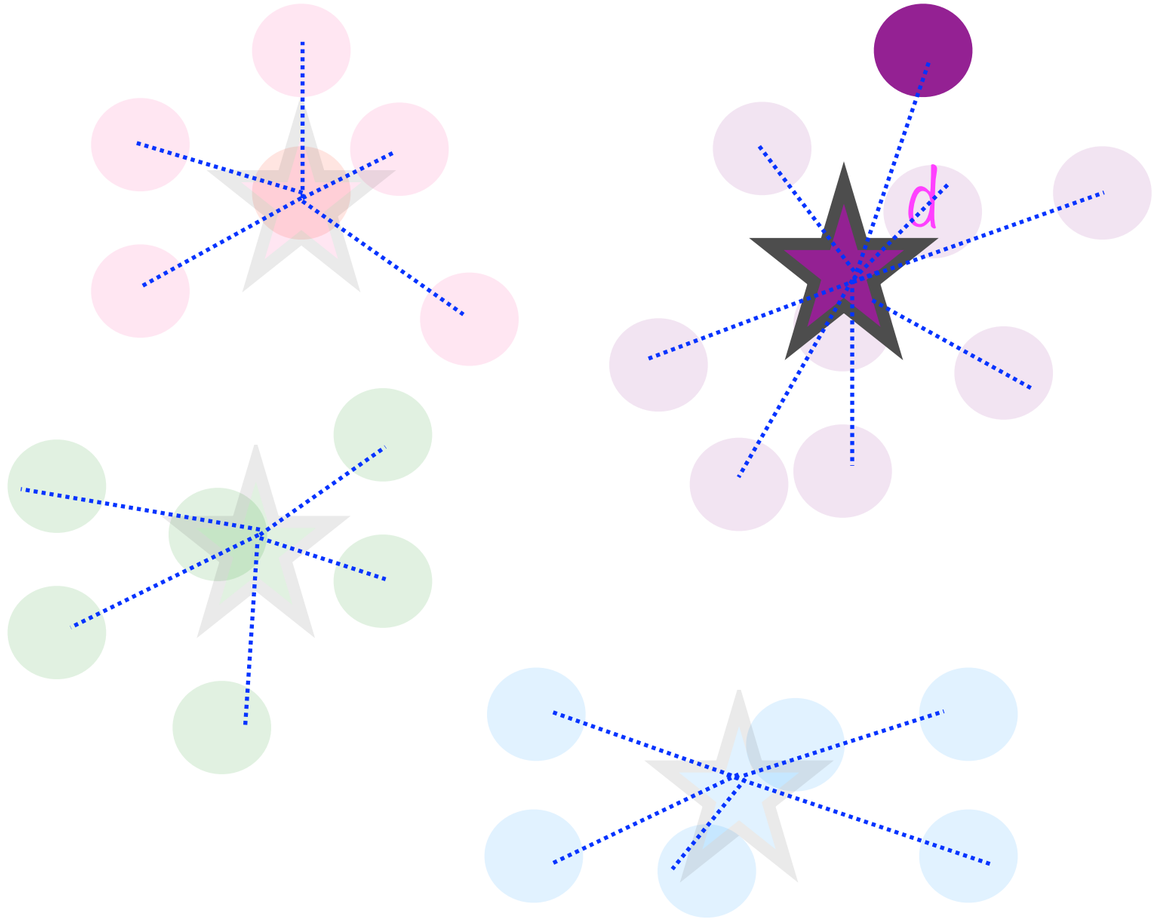
d^2



Total Reconstruction Error

Sum over all points

and reference vectors



Hyperparameters

Model configuration properties that define a model, and remain constant during the training of the model

Understanding Hyperparameters

Model Inputs

Model Parameters

Model Hyperparameters

Understanding Hyperparameters

Model Inputs

Input data points, training dataset

Model Parameters

Model Hyperparameters

Understanding Hyperparameters

Model Inputs

Input data points, training dataset

Model Parameters

Reference vectors, i.e. centroids of each cluster

Model Hyperparameters

Understanding Hyperparameters

Model Inputs

Input data points, training dataset

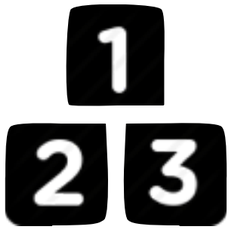
Model Parameters

Reference vectors, i.e. centroids of each cluster

Model Hyperparameters

Number of clusters, initial values, distance measure

Hyperparameters in K-Means Clustering



Number of clusters

Initial values

Distance measures

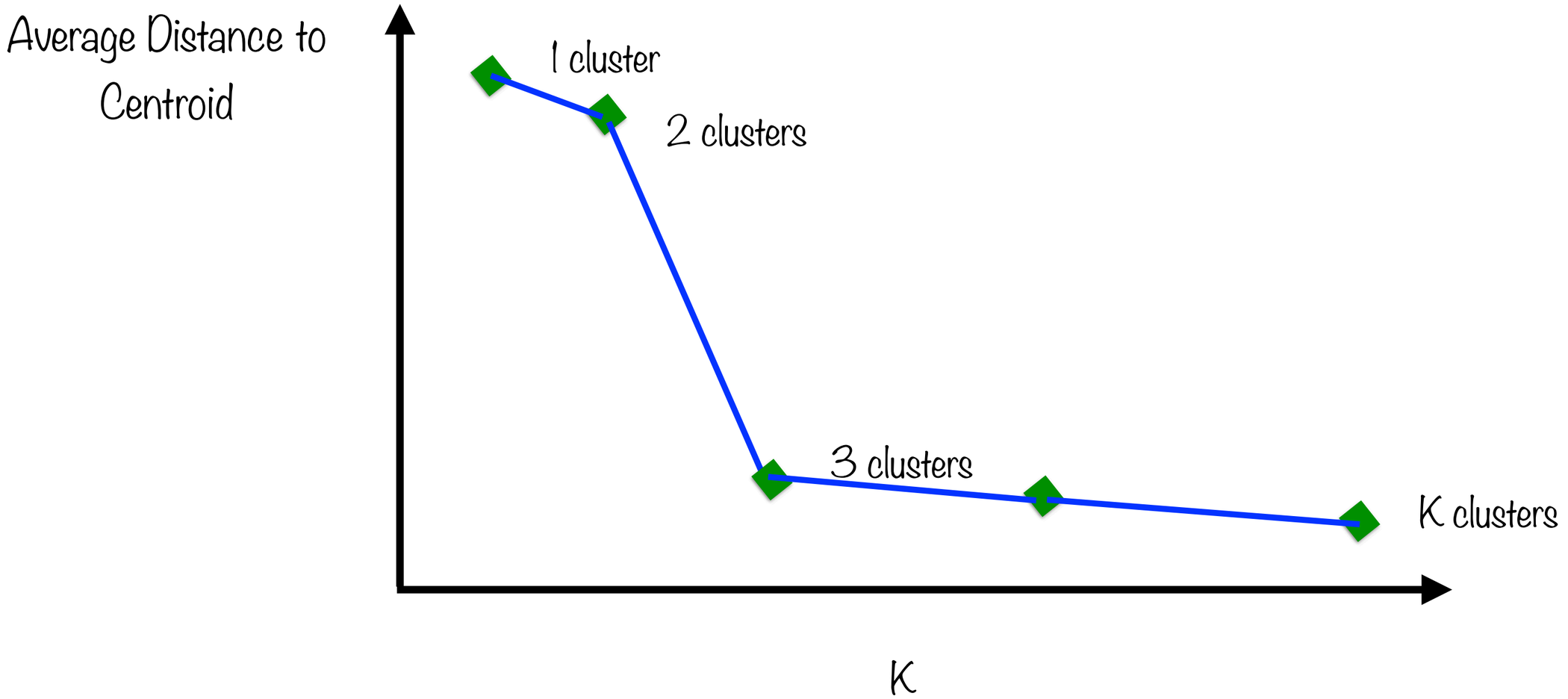
Number of Clusters

K is the most important hyperparameter

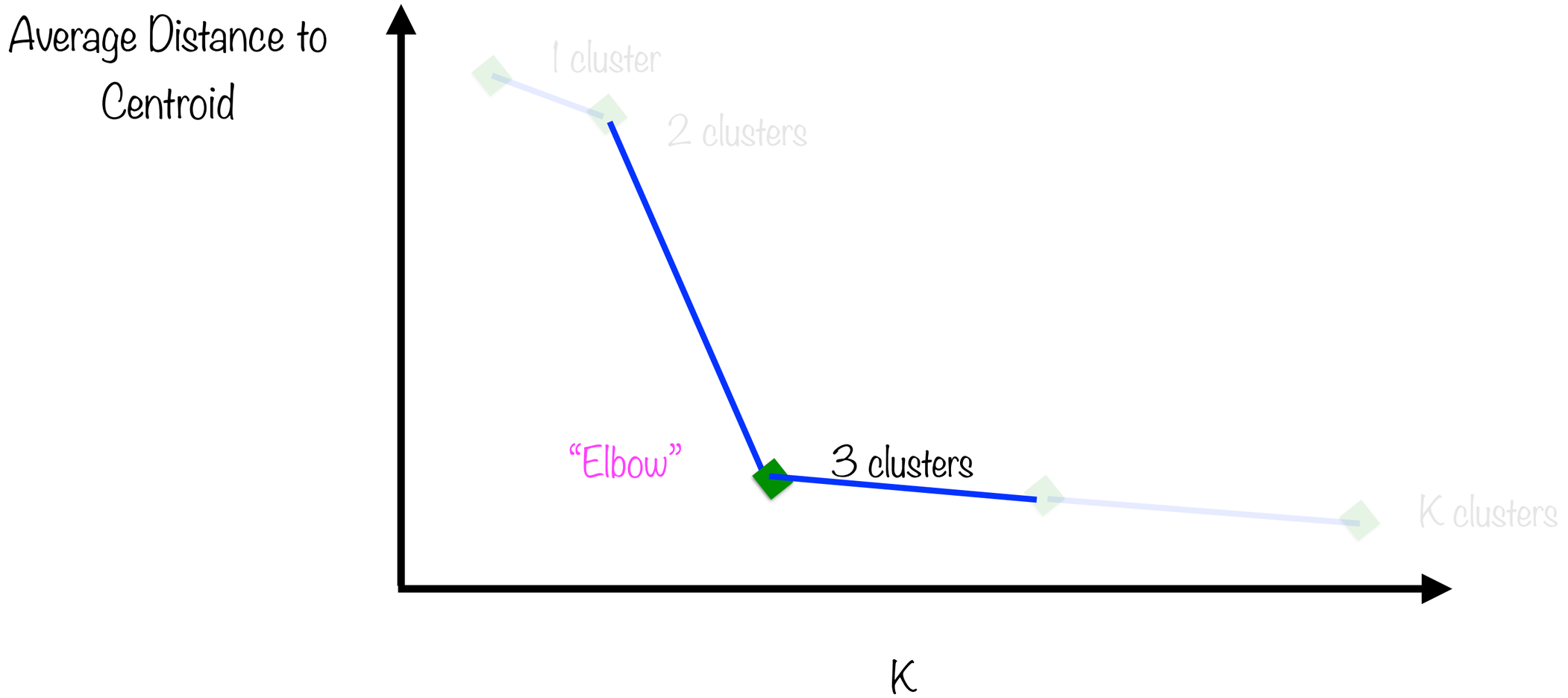
Sometimes obvious e.g. 10 in MNIST digit classification

Else plot average reconstruction error against k, identify elbow

Choosing K


Average Distance to Centroid
1 cluster
2 clusters
3 clusters
K clusters
K

Choosing K

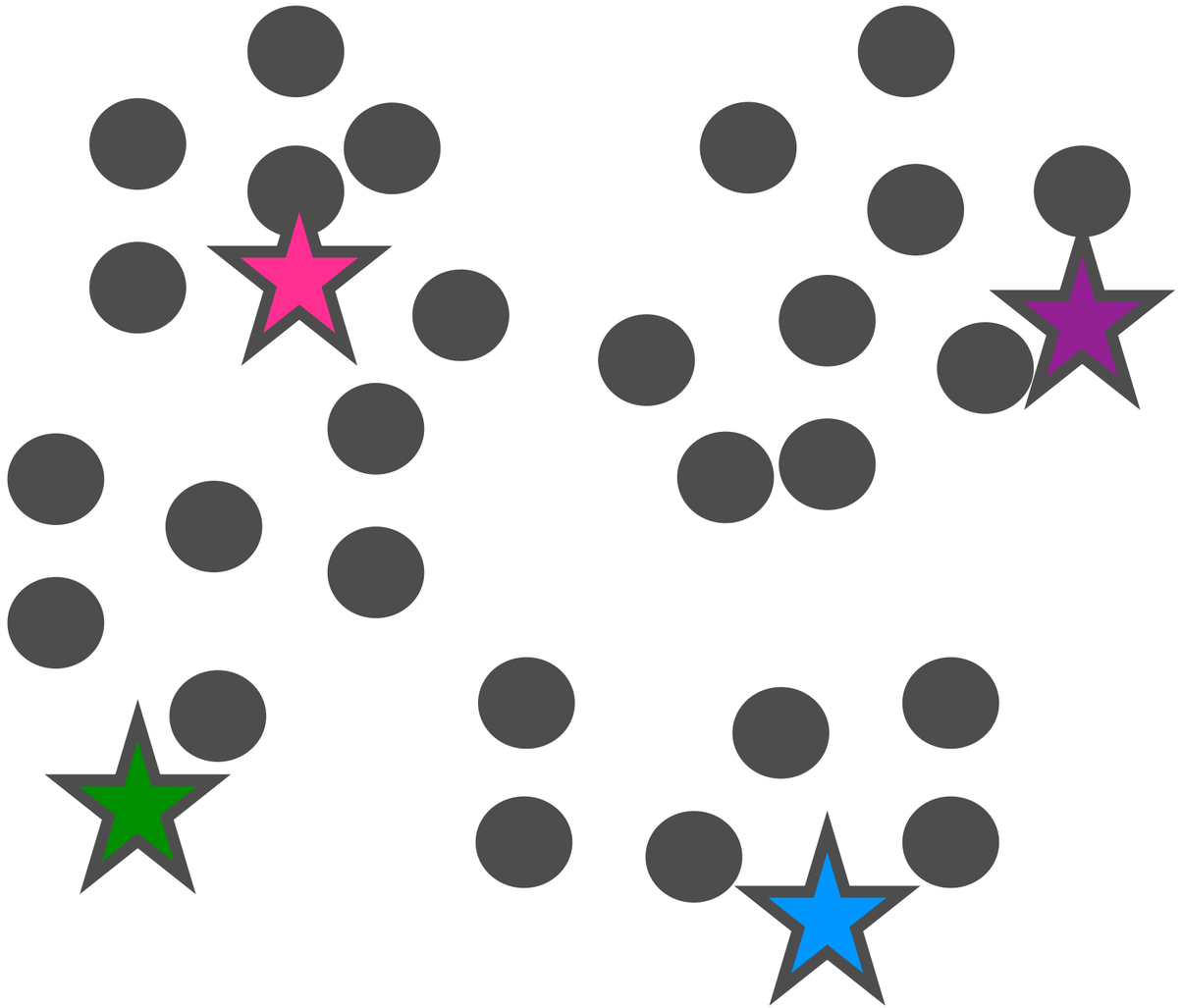

Average Distance to Centroid
1 cluster
2 clusters
"Elbow"
3 clusters
K clusters
K

Initial Values

Final reference vector values sensitive to initial values

Random initialization might not work - examine data carefully

Initial Values



Initial Values

Final reference vector values sensitive to initial values

Random initialization might not work - examine data carefully

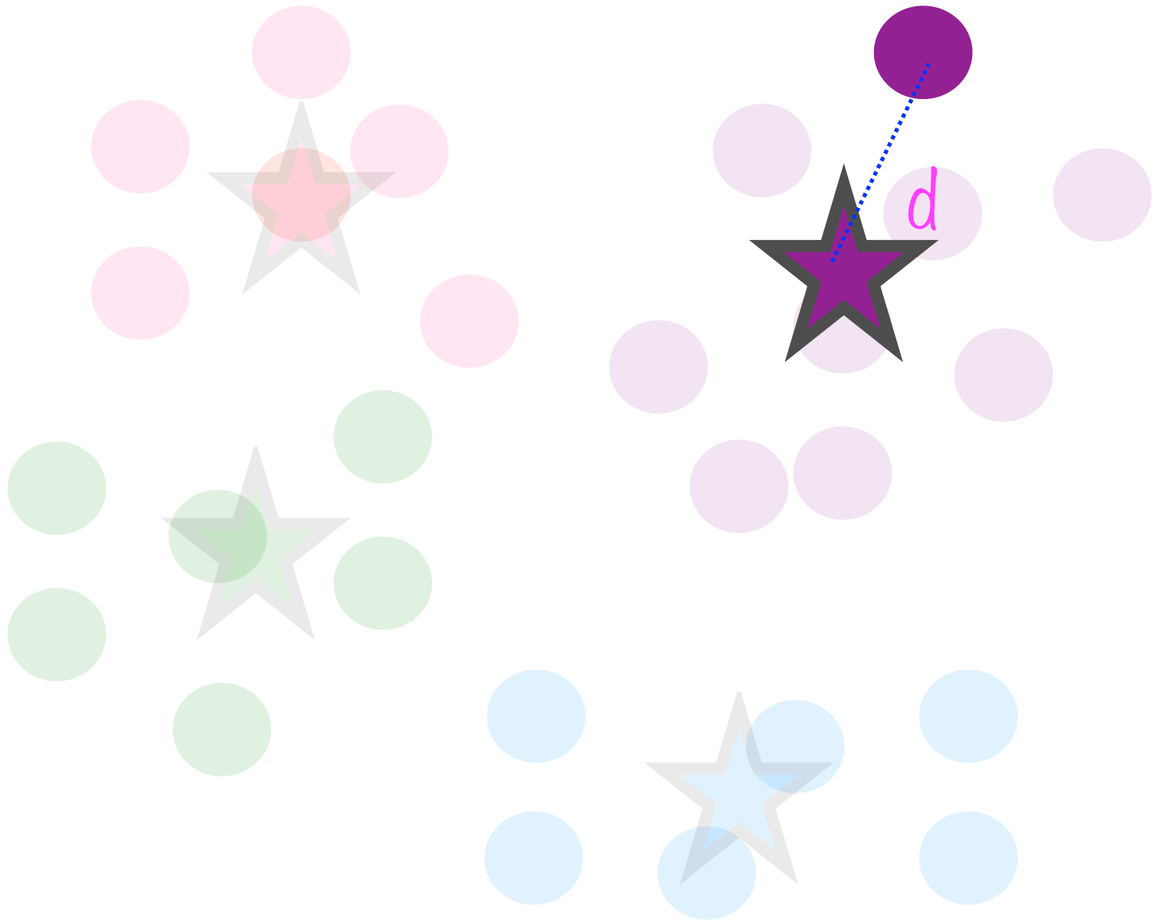
- Can perform PCA of data
- Divide range of normalised PCs into K
- Take average of each

Distance Measures

Can choose multiple distance measures:

Distance Measures

Distance from each point to the center



Distance Measures

Can choose multiple distance measures:

- Euclidean distance - centroid might not be actual data point
- Mahalanobis distance - normalise each dimension to have equal variance
- Cosine distance - cosine of angle between point and centroid

Autoencoders and Dimensionality Reduction

Unsupervised ML Algorithms

Clustering

Identify patterns in data items e.g. K-means clustering

Autoencoding

Identify latent factors that drive data e.g. PCA

Memorizing Numbers

40 65 73 81 36 68 22 15

50 25 76 38 19 58 29 88 44 22 11 34 17 52

Memorizing Numbers

40 65 73 81 36 68 22 15

Completely random, only brute force memorization works

50 25 76 38 19 58 29 88 44 22 11 34 17 52

Memorizing Numbers

40 65 73 81 36 68 22 15

50 25 76 38 19 58 29 88 44 22 11 34 17 52

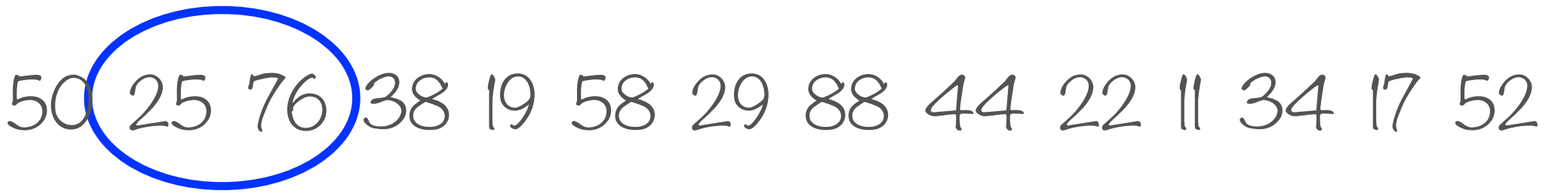
There is a pattern here

Memorizing Numbers

50 25 76 38 19 58 29 88 44 22 11 34 17 52

$$n_2 = n_1 / 2$$

Memorizing Numbers



$$n_3 = 3n_2 + 1$$

Memorizing Numbers

50 25 76 38 19 58 29 88 44 22 11 34 17 52

$$n_2 = n_1 / 2$$

$$n_3 = 3n_2 + 1$$

Memorizing Numbers

50 25 76 38 19 58 29 88 44 22 11 34 17 52

$$n_2 = n_1 / 2$$
$$n_3 = 3n_2 + 1$$

This is an encoding to remember this long sequence

Autoencoders find patterns in data so it can remember the data using a a more compact representation

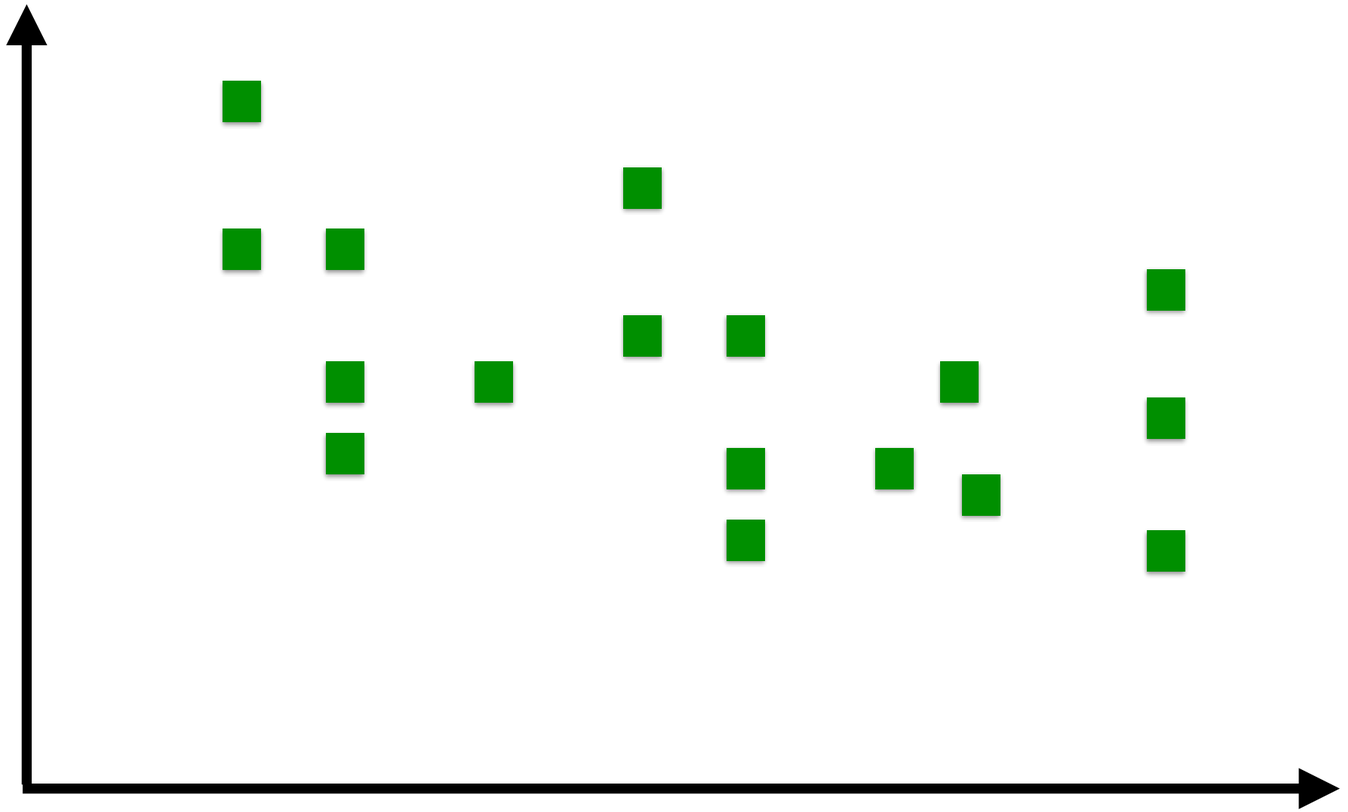
An encoding

Latent Factor Analysis: Principal Component Analysis

Data in One Dimension

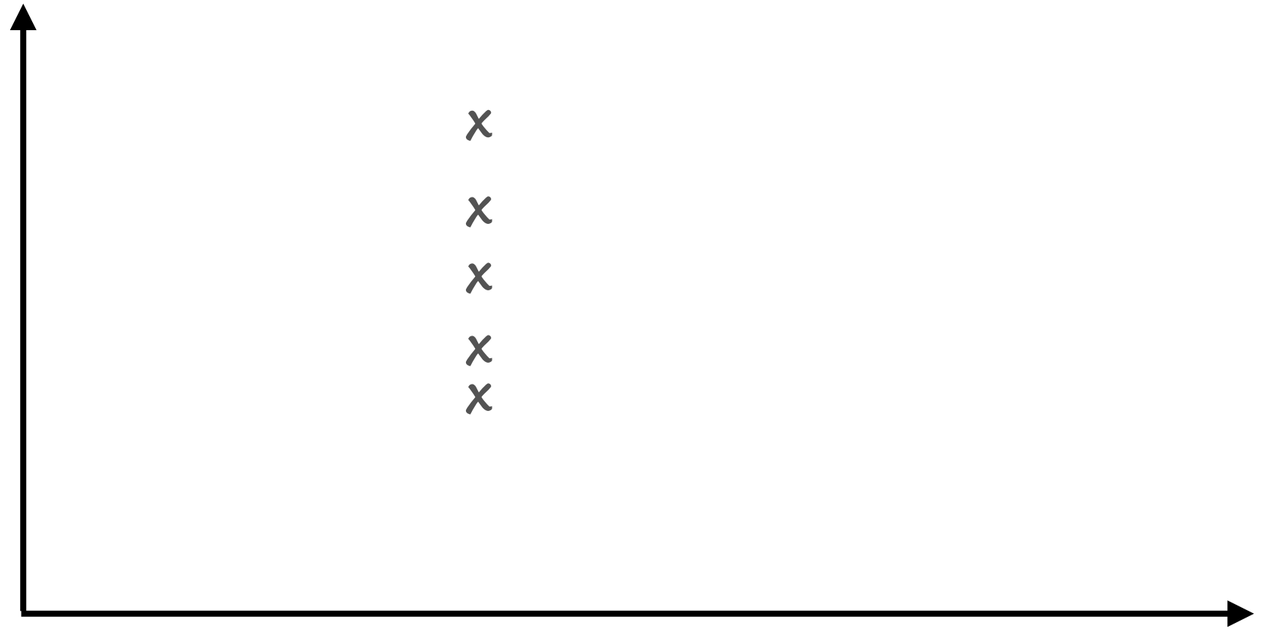
Unidimensional data points can be represented using a line, such as a number line

Data in Two Dimensions



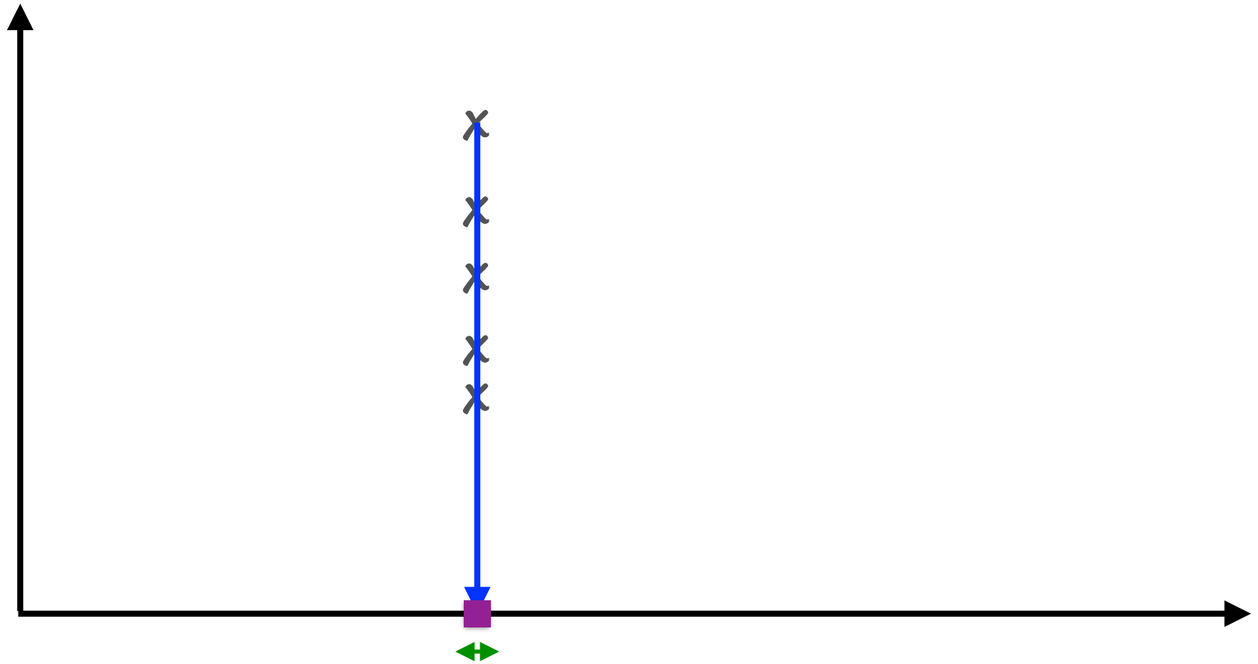
It's often more insightful to view data in relation to some other, related data

A Question of Dimensionality



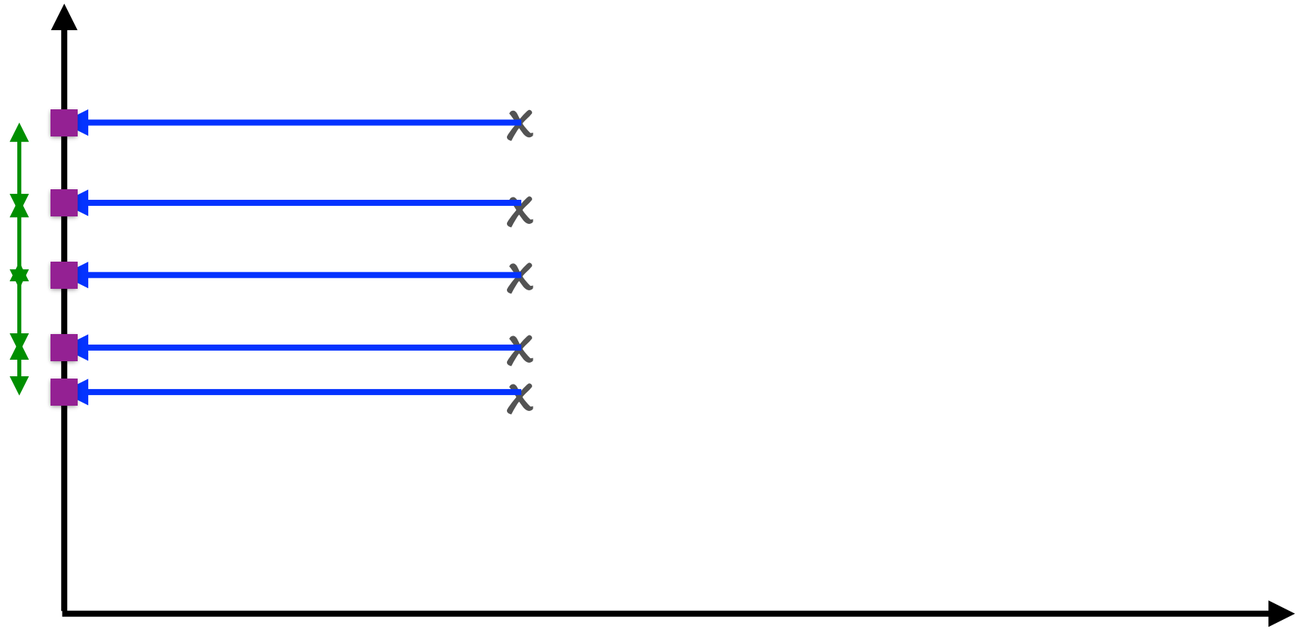
Pop quiz: Do we really need two dimensions to represent this data?

Bad Choice of Dimensions



If we choose our axes (dimensions) poorly then we do need two dimensions

Good Choice of Dimensions

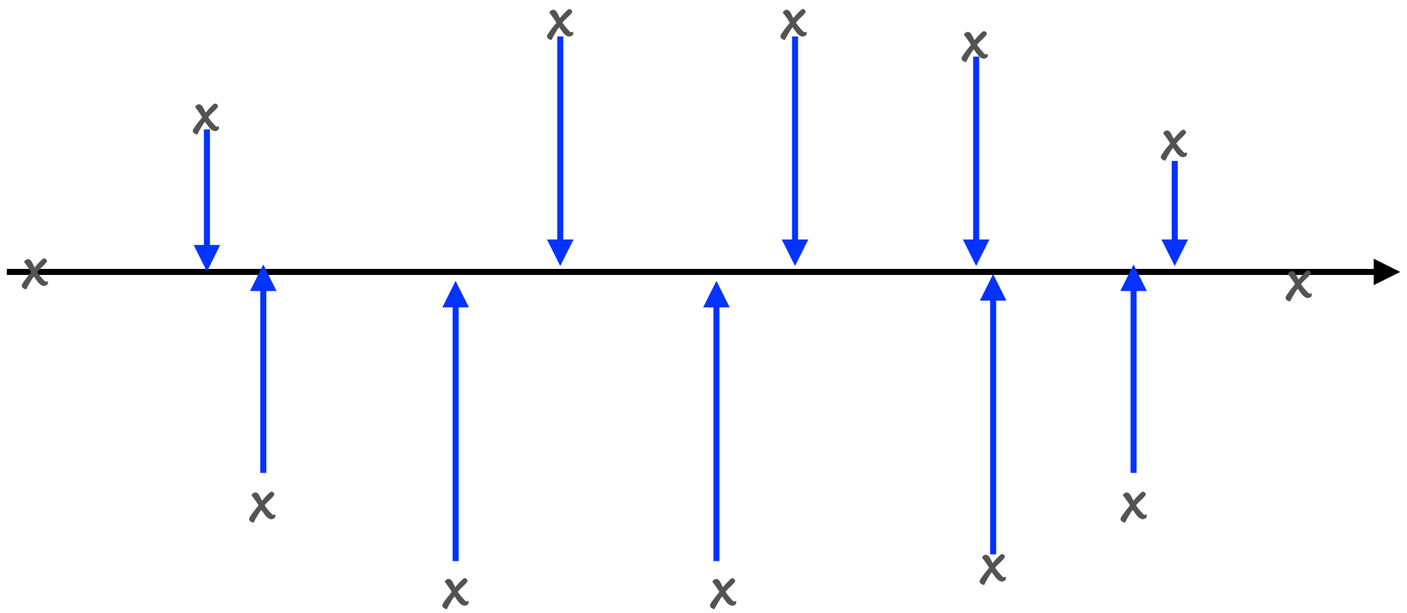


If we choose our axes (dimensions) well then one dimension is sufficient

Intuition Behind PCA

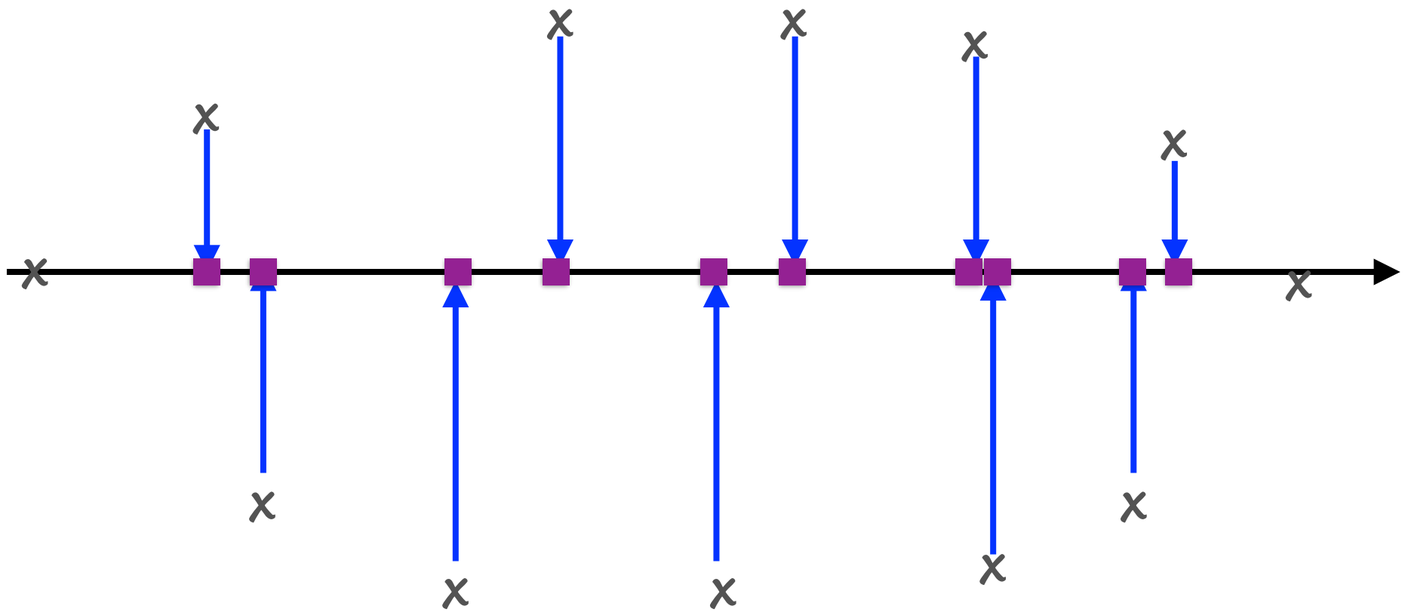
Objective: Find the "best" directions to represent this data

Intuition Behind PCA



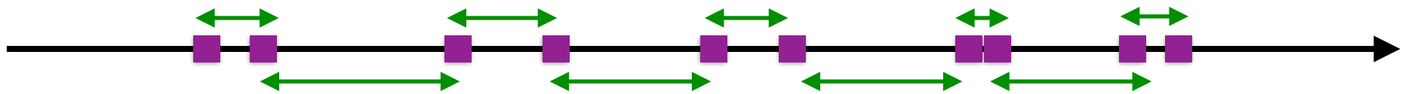
Start by "projecting" the data onto a line in some direction

Intuition Behind PCA



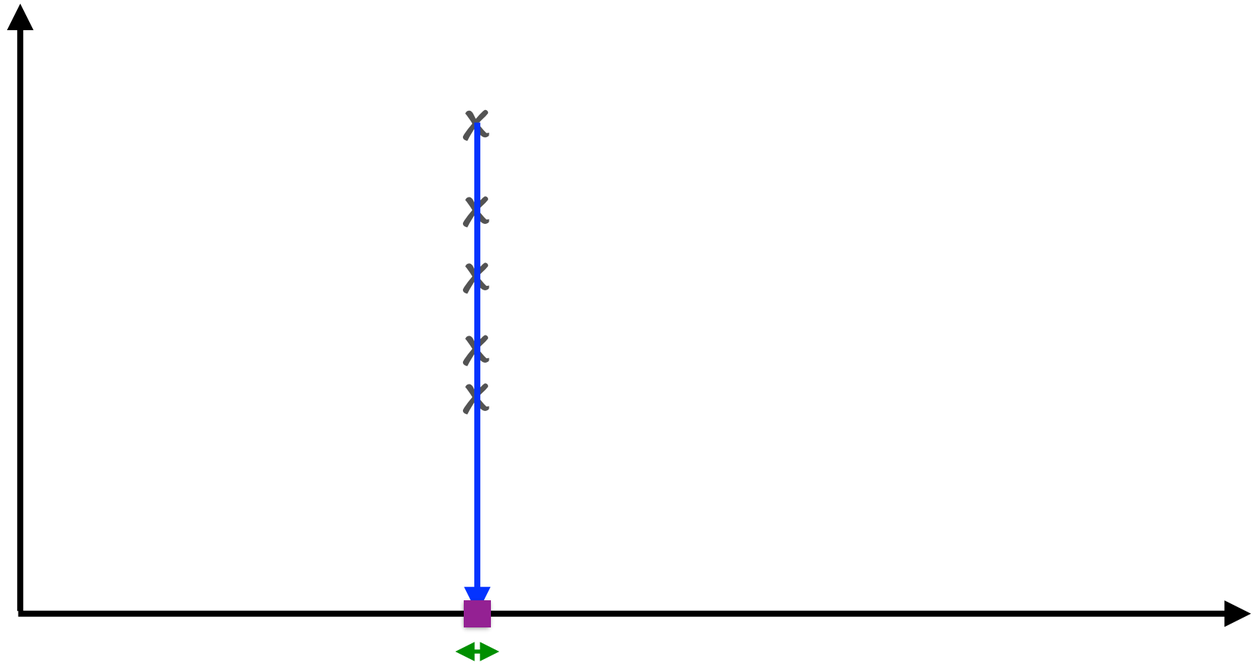
Start by "projecting" the data onto a line in some direction

Intuition Behind PCA



The greater the distances between these projections, the "better" the direction

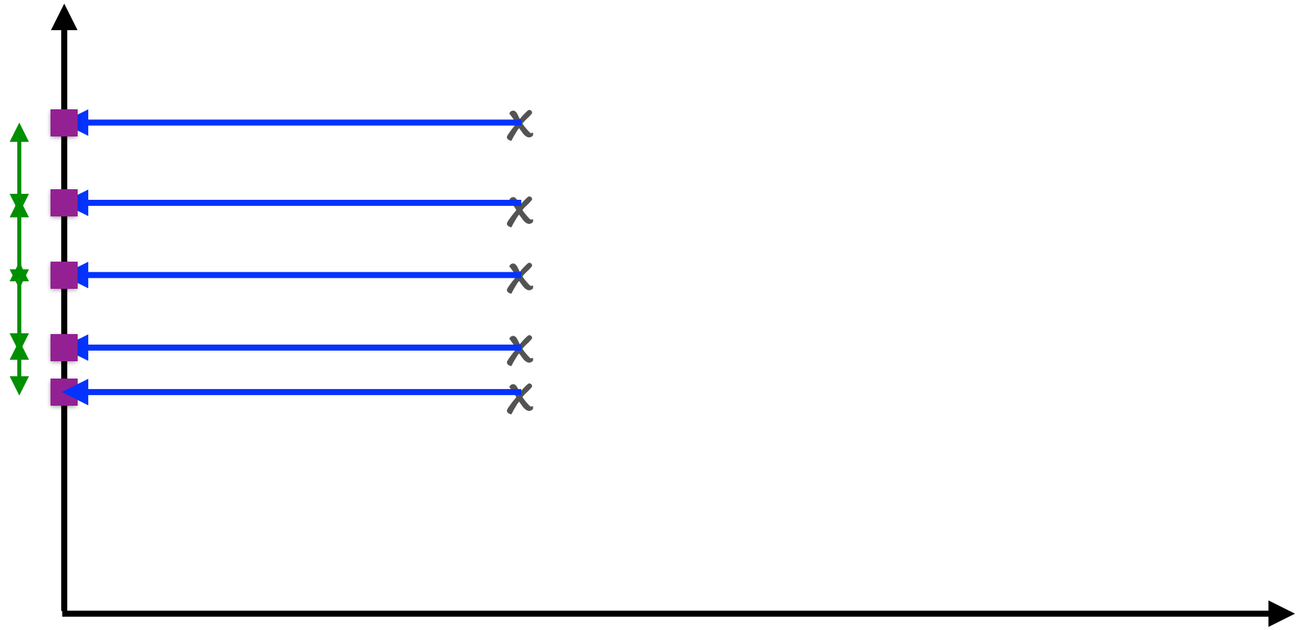
Bad Projection



A projection where the distances are minimised is a bad one -

information is lost

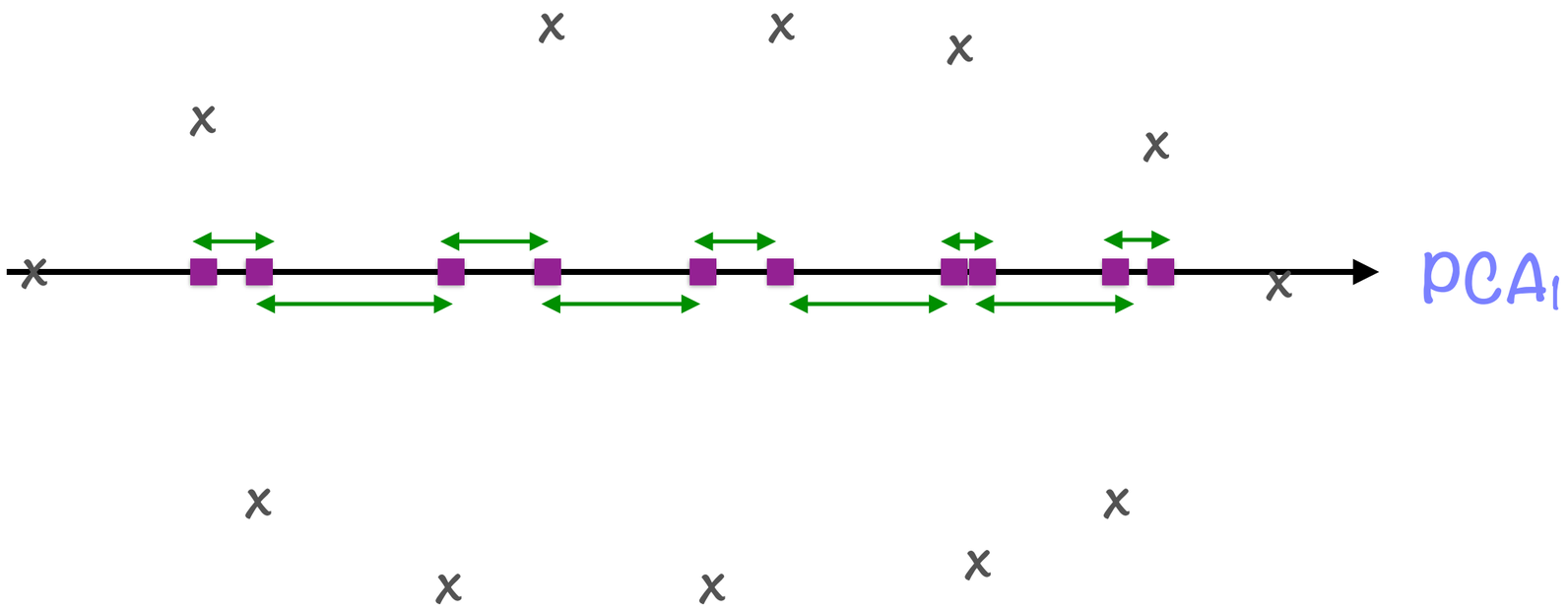
Good Projection



A projection where the distances are maximised is a good one

- information is preserved

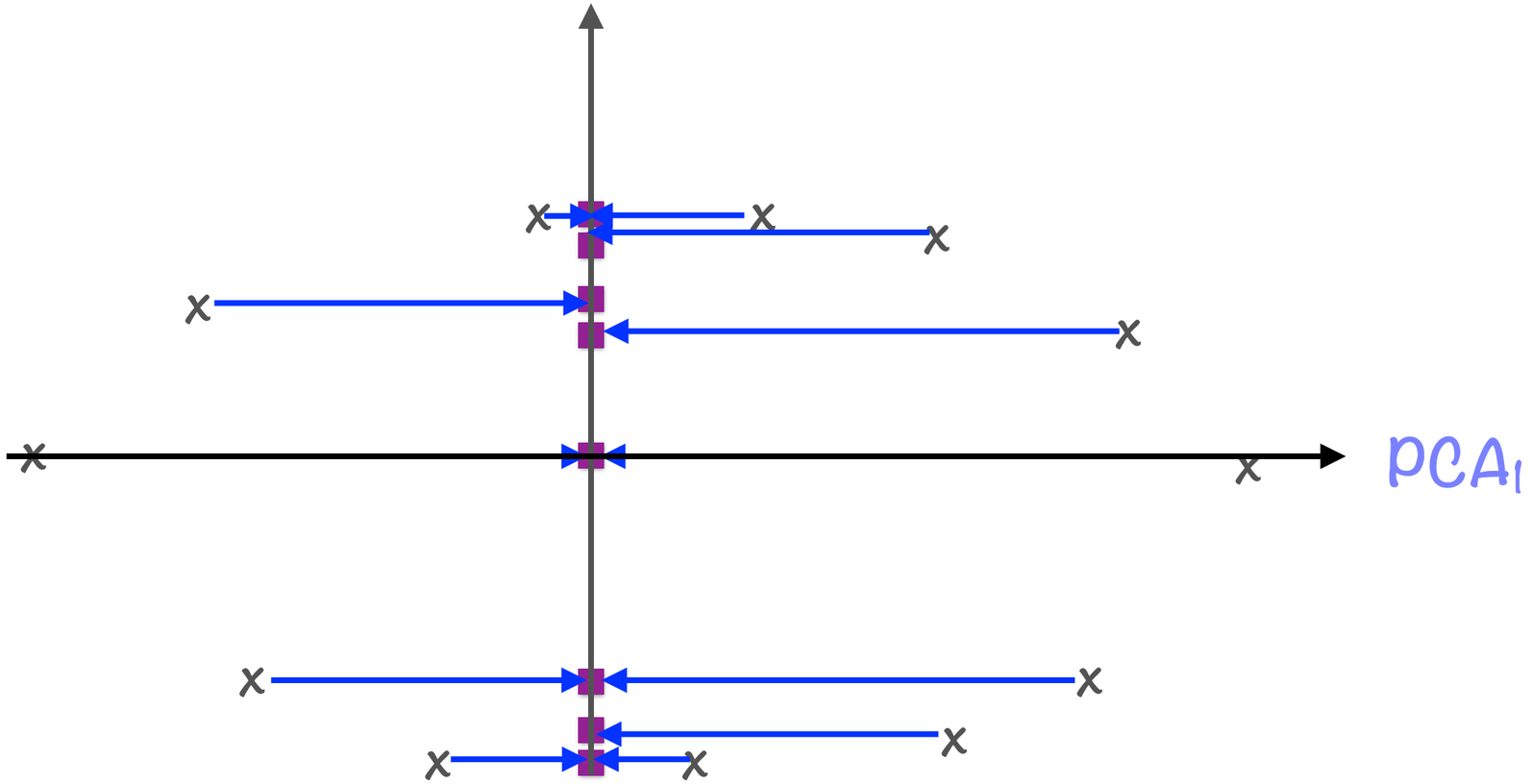
Intuition Behind PCA



The direction along which this variance is maximised is the

first principal component of the original data

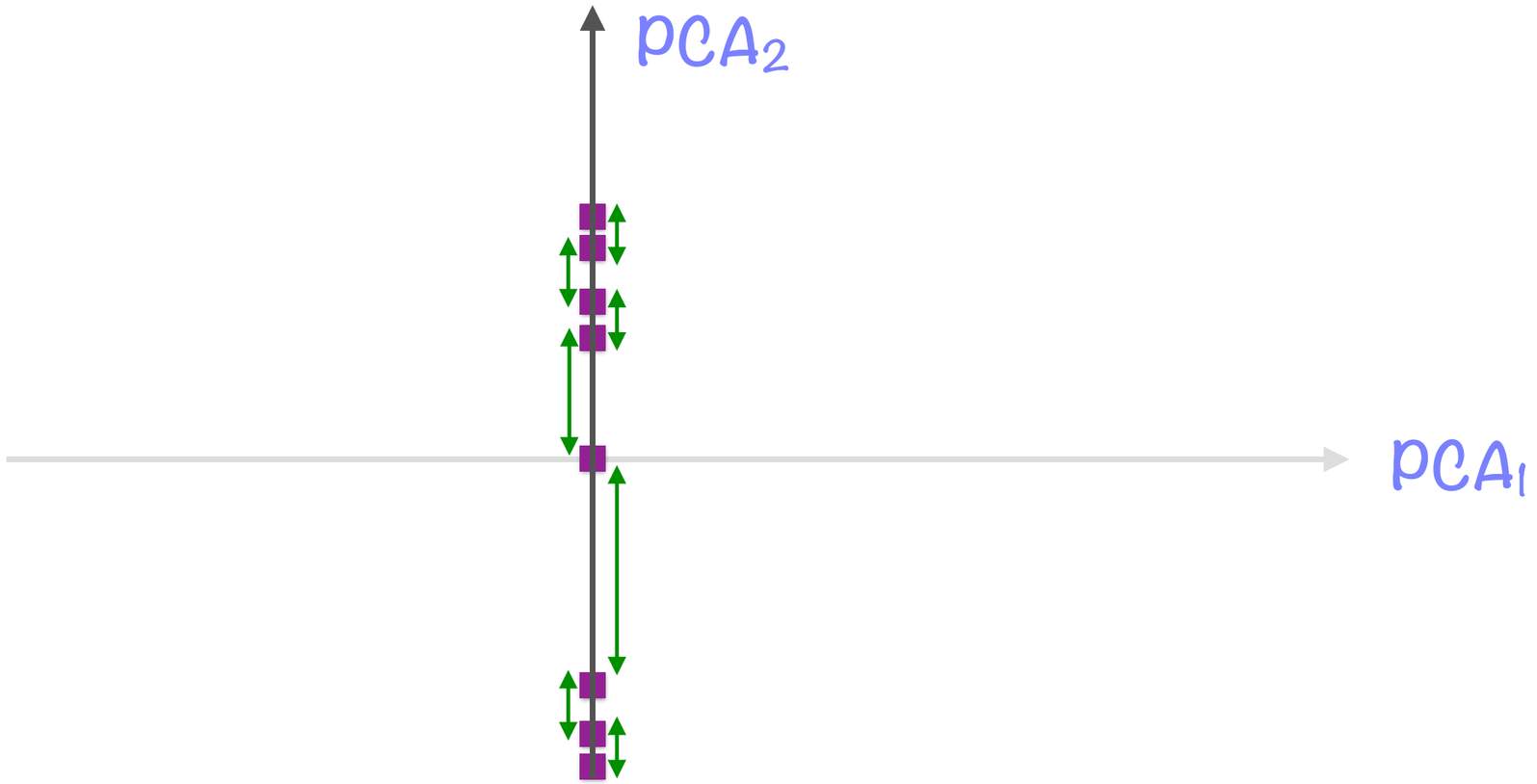
Intuition Behind PCA



Find the next best direction, the second principal component,

which must be at right angles to the first

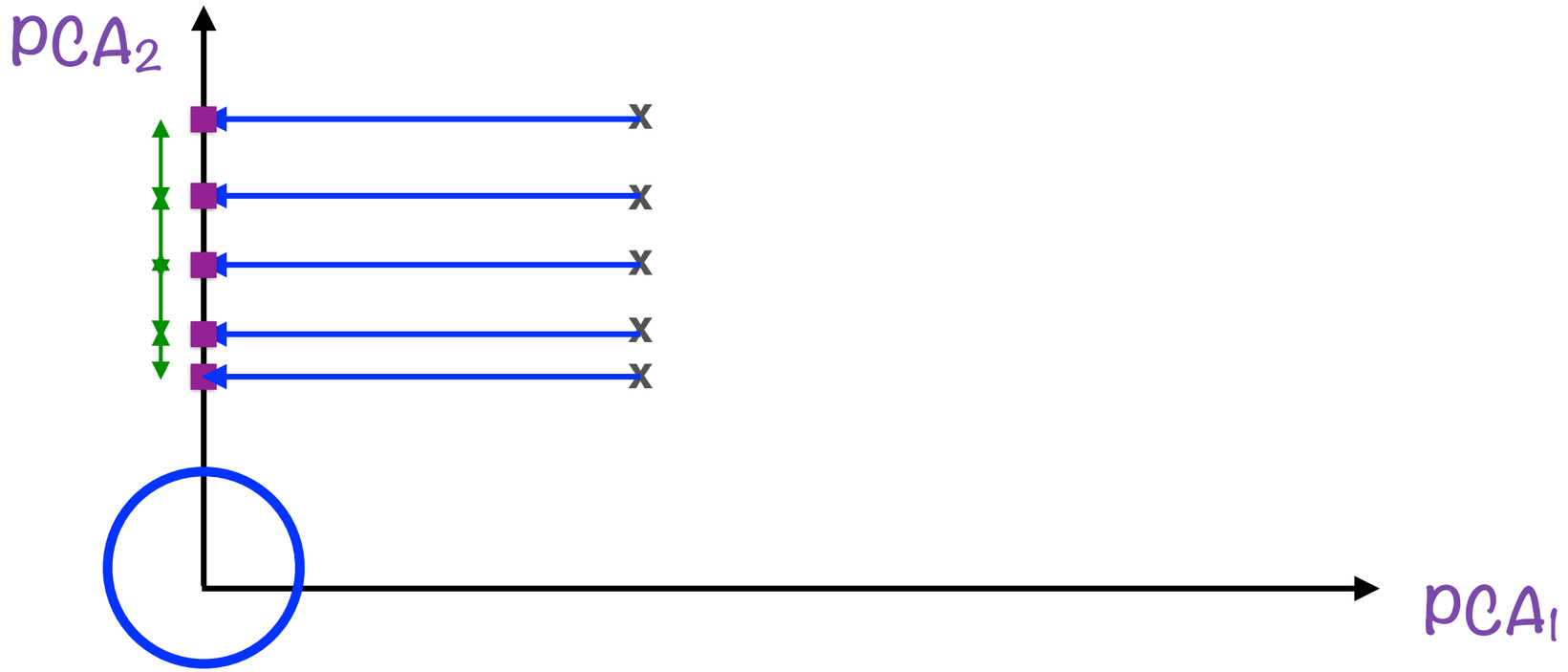
Intuition Behind PCA



Find the next best direction, the second principal component,

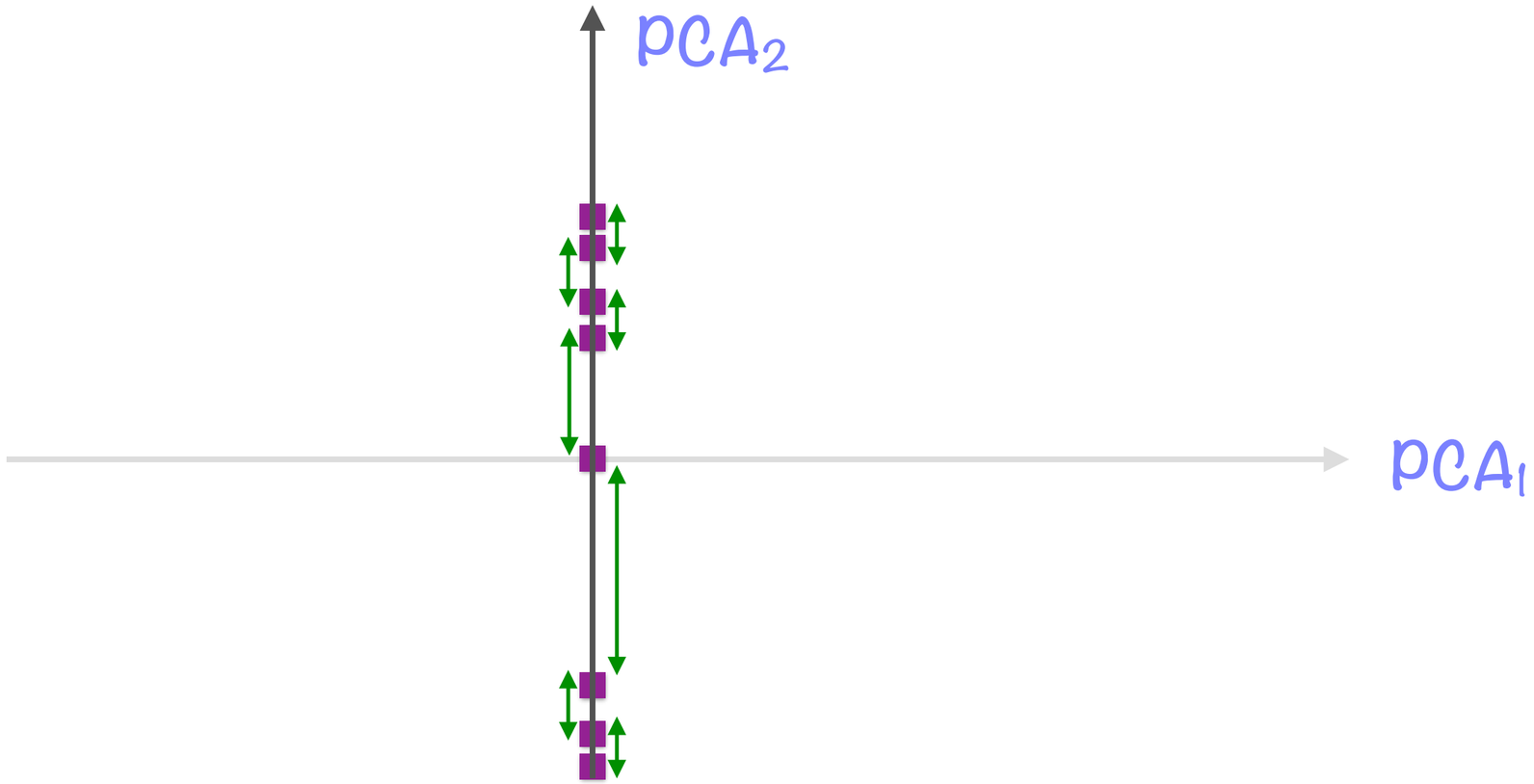
which must be at right angles to the first

Principal Components at Right Angles



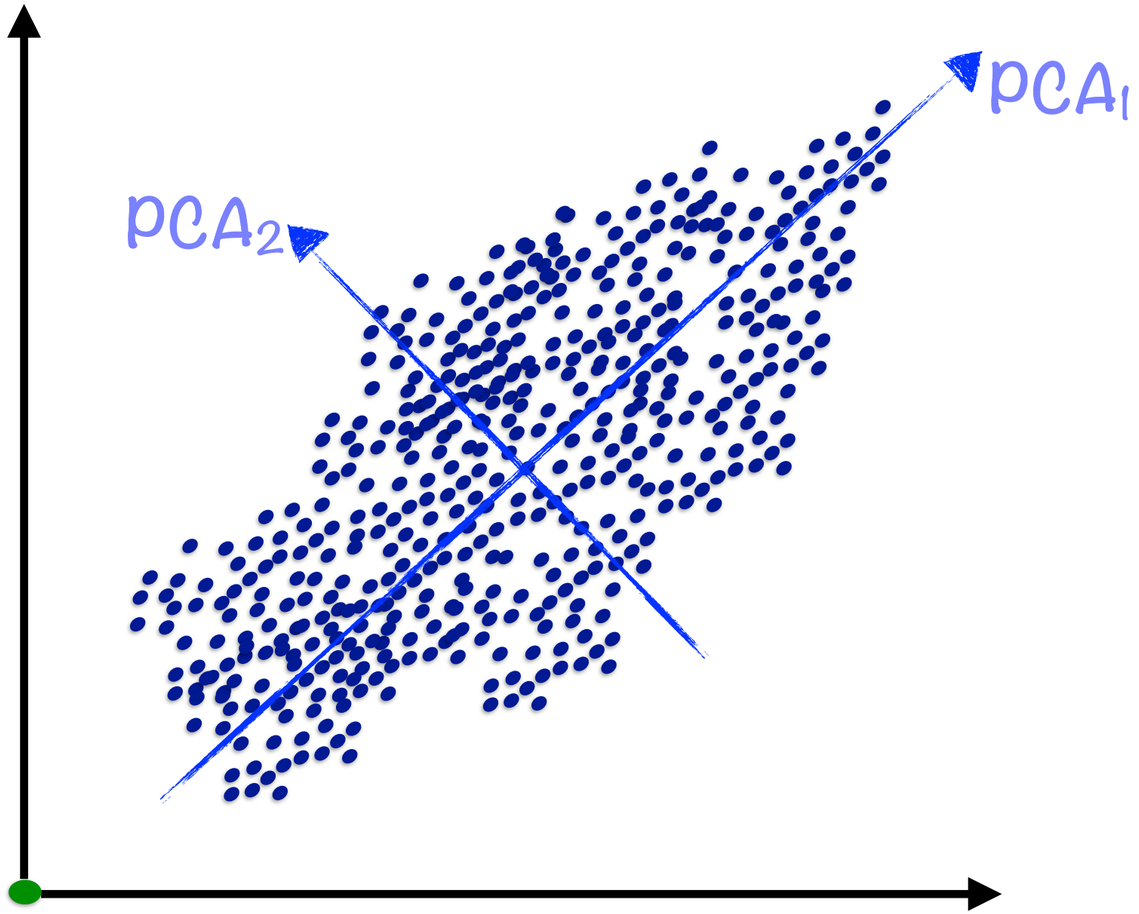
Directions at right angles help express the most variation with the smallest number of directions

Intuition Behind PCA



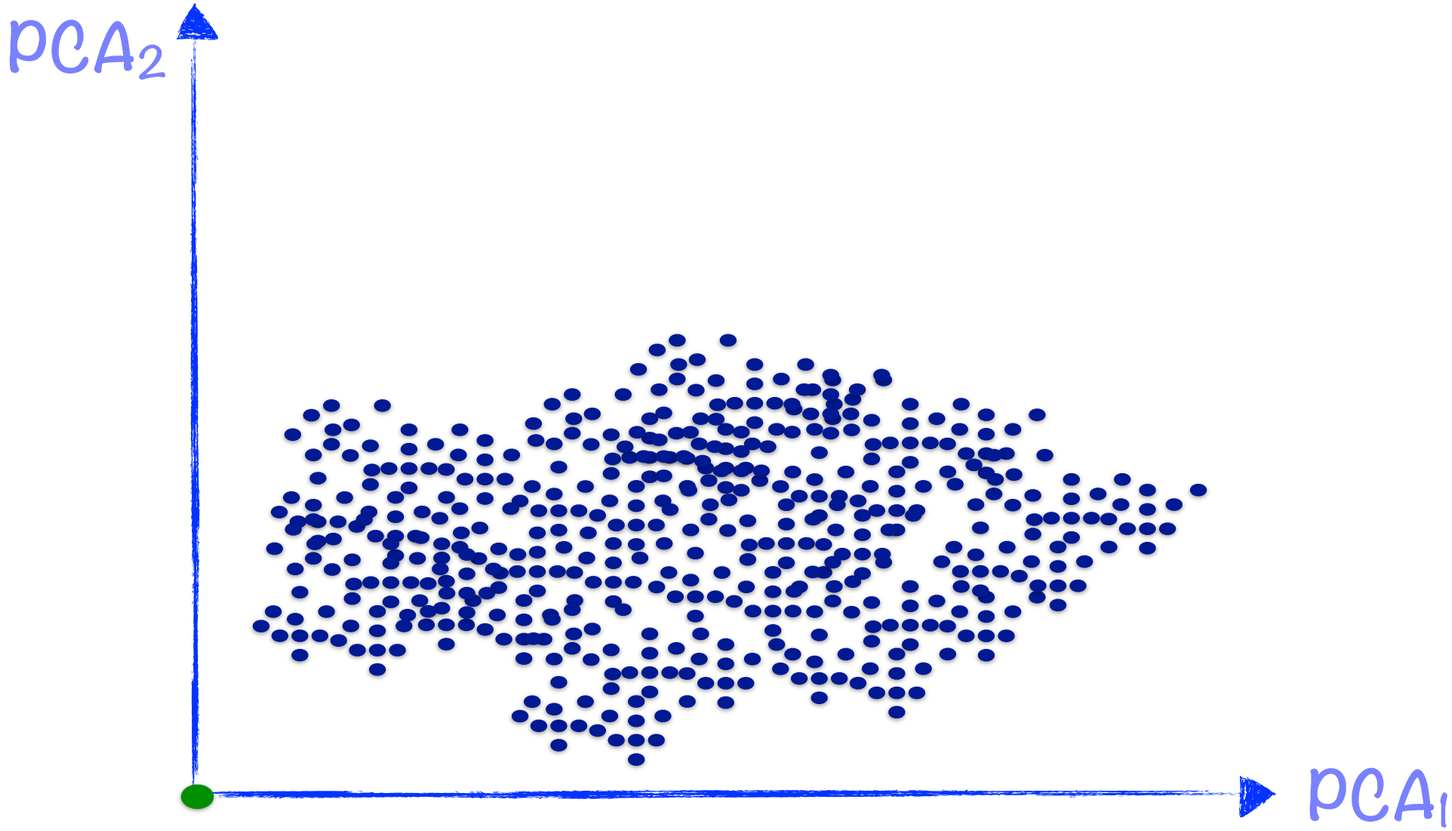
The variances are clearly smaller along this second principal component than along the first

Intuition Behind PCA



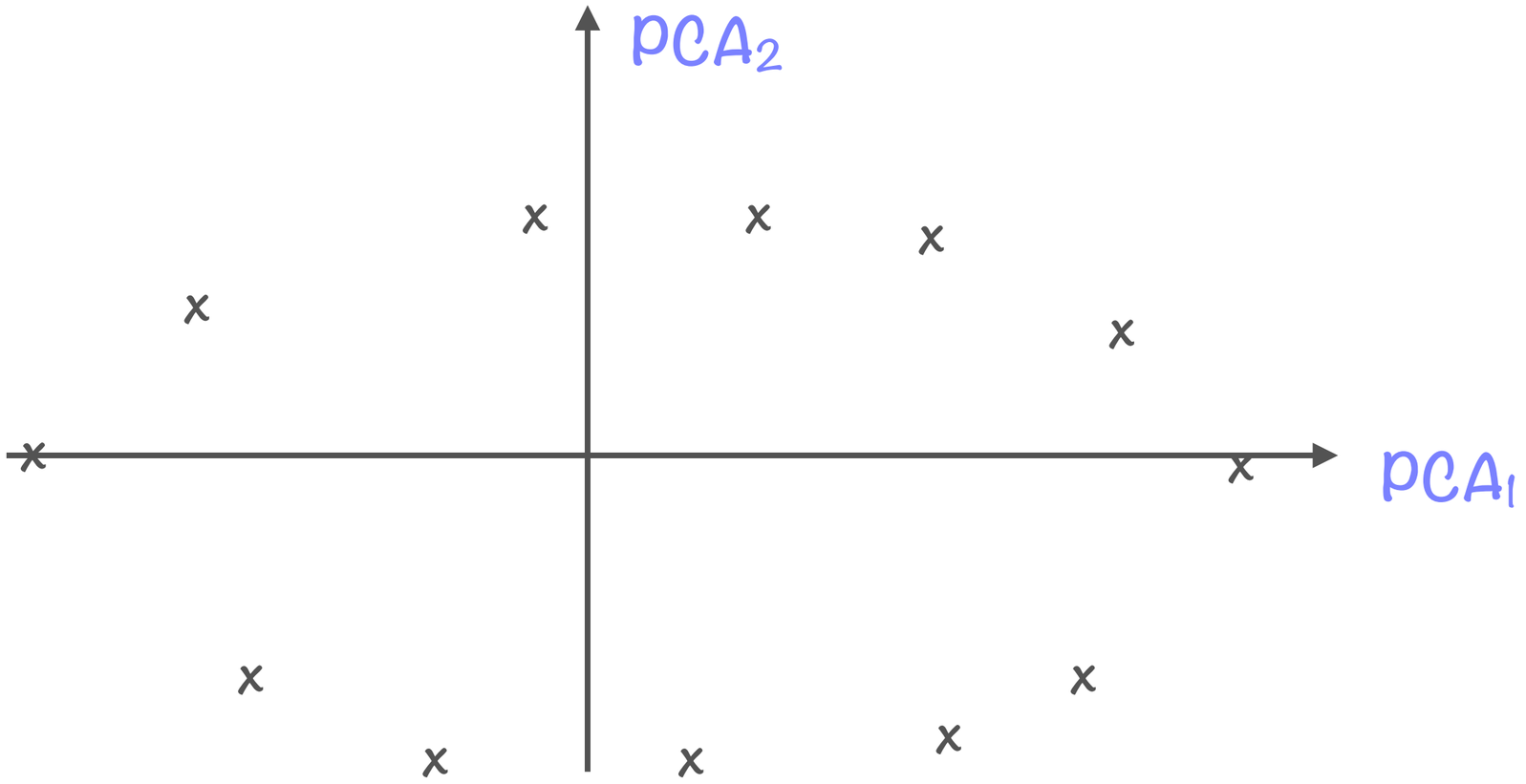
In general, there are as many principal components as there are dimensions in the original data

Intuition Behind PCA



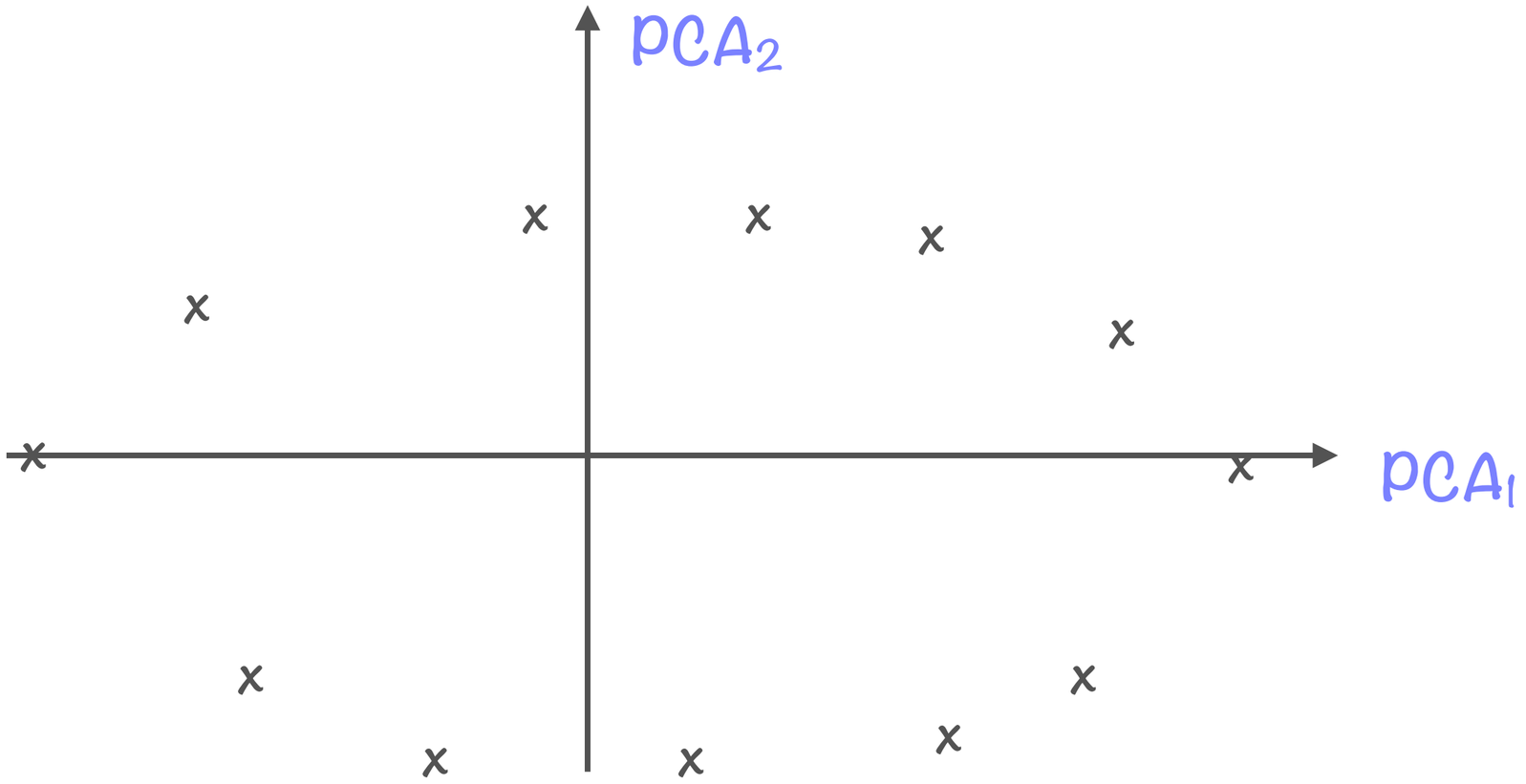
Re-orient the data along these new axes

Dimensionality Reduction



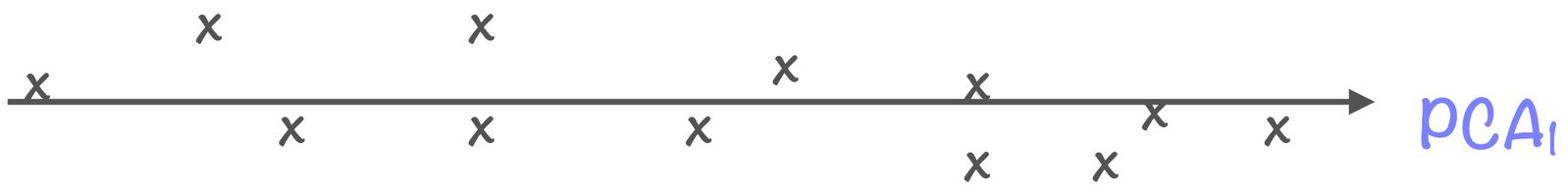
If the variance along the second principal component is small enough,
we can just ignore it and use just 1 dimension to represent the data

Dimensionality Reduction



Variation along 2 dimensions: 2 principal components required

Dimensionality Reduction



Variation along 1 dimension: 1 principal component is sufficient

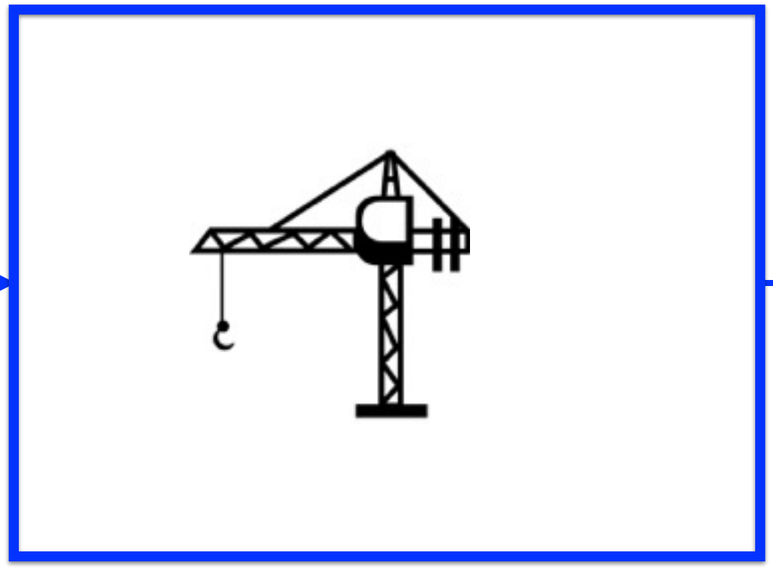
PCA is used for dimensionality reduction i.e. use fewer attributes to represent the same information

Choose the most important attributes

Dimensionality Reduction

$$\begin{bmatrix} x_{11} & \cdots & x_{1k} \\ x_{21} & \cdots & x_{2k} \\ x_{31} & \cdots & x_{3k} \\ \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nk} \end{bmatrix}$$

k columns



PCA Factor Reduction

$$\begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \\ f_{31} & f_{32} \\ \cdots & \cdots \\ f_{n1} & f_{n2} \end{bmatrix}$$

2 columns

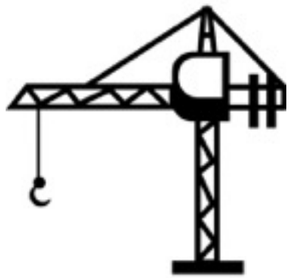
Principal Components Analysis

k columns

$[\quad X_1 \quad X_2 \quad X_3 \quad \dots \quad X_k \quad]$

n rows

X_i are highly correlated with each other



PCA

F_i are completely uncorrelated with each other

$[\quad F_1 \quad F_2 \quad F_3 \quad \dots \quad F_k \quad]$

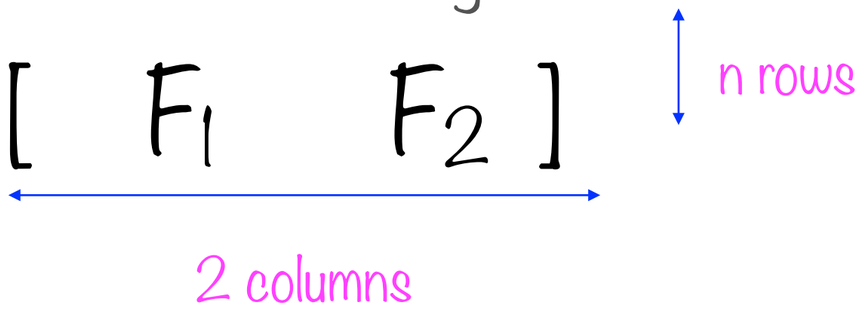
n rows

k columns

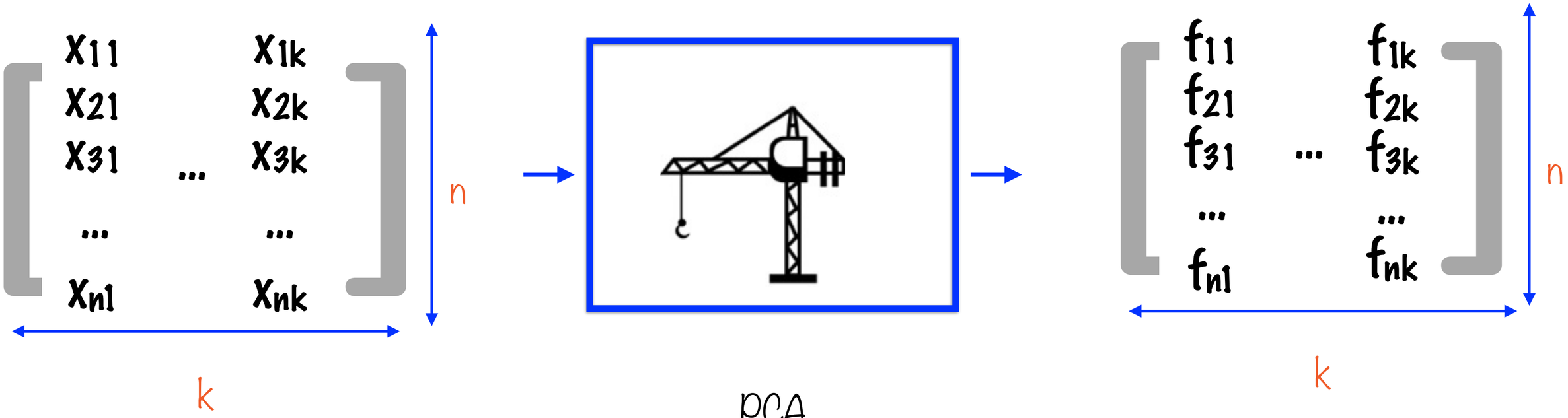
Principal Components Analysis

Keep F_1 and F_2, discard the rest

These 2 principal components explain the vast majority of the total variance in the original data



Principal Components Analysis

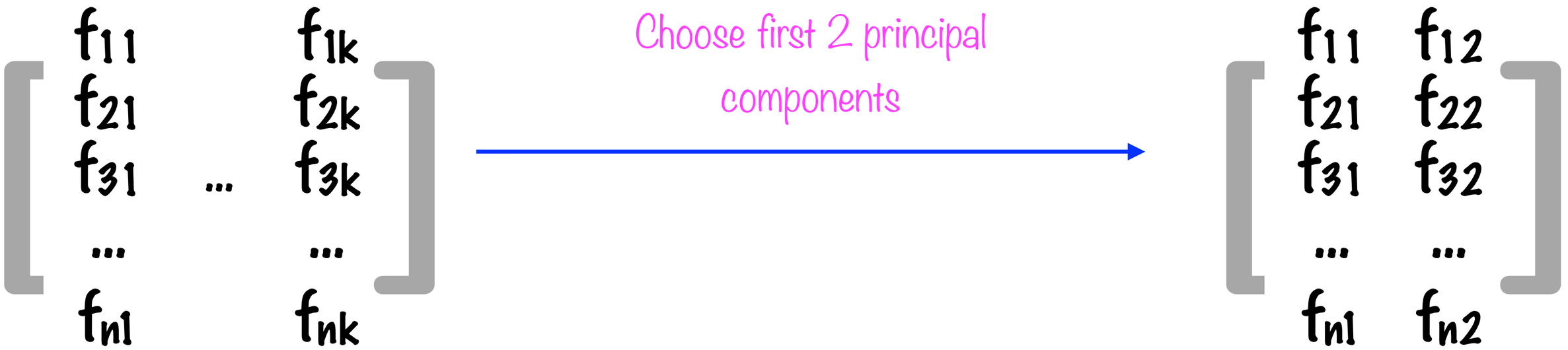


Original Data

Same number of columns

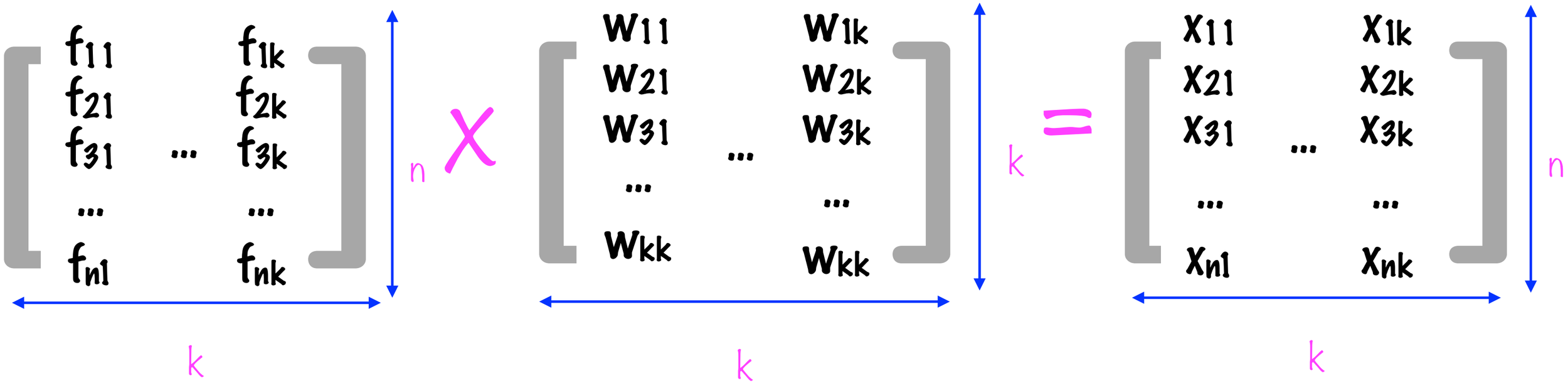
Principal Components

Dimensionality Reduction



Principal Components

Reconstruct Original Data



Principal Components

Weight Vectors

Original Data

Autoencoders

Autoencoders are Neural Networks that learn efficient representations of data (e.g. PCA)

$$y = f(x)$$

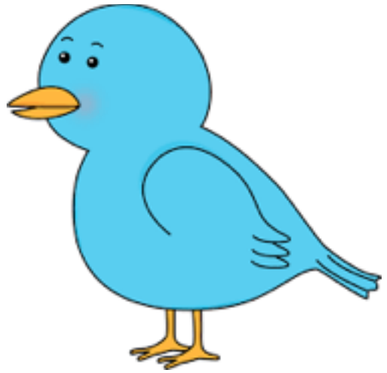
Supervised Machine Learning

Most machine learning algorithms seek to “learn” the function f that links the features and the labels

Supervised Machine Learning

Horse

Bird



ML Algorithm

Unicorn?

Bat?

≠

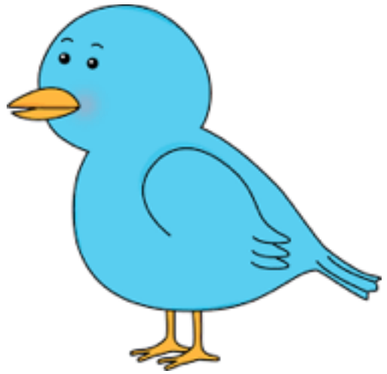
Horse

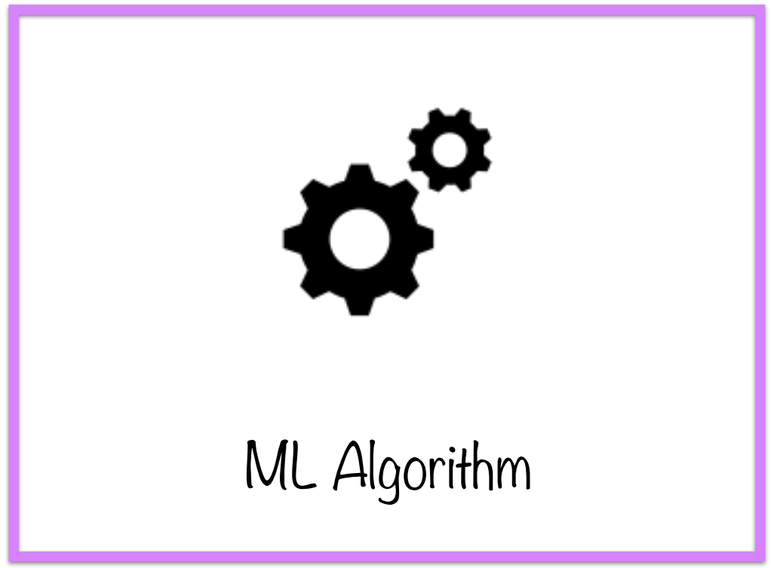
Bird

Supervised Machine Learning

Horse

Bird





Unicorn?

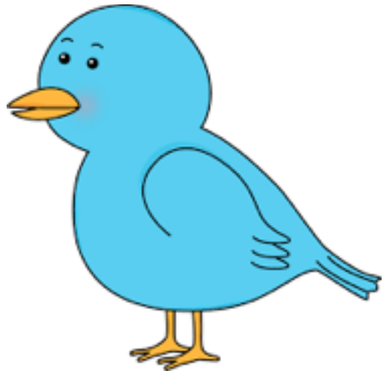
Bat?

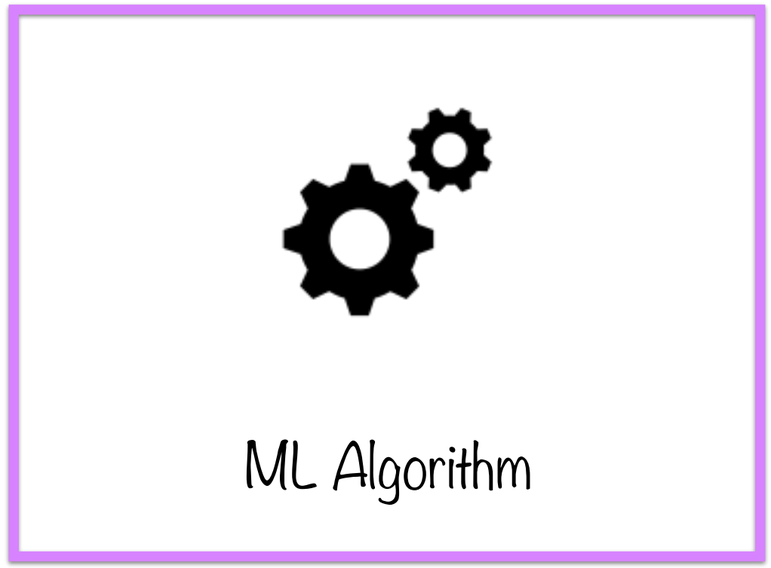
Feedback (cost function) trains the model

Supervised Machine Learning

Horse

Bird





Horse

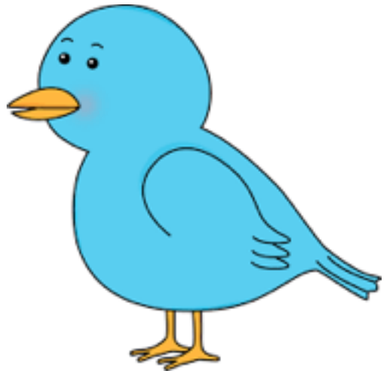
Bird

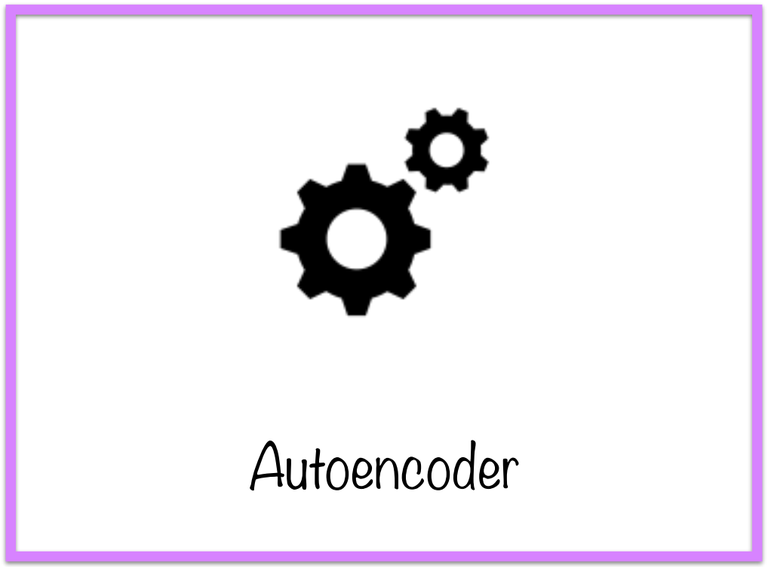
$x = f(x)$

Autoencoders Learn the Input!

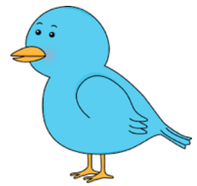
The process is inherently unsupervised, but cleverly uses the input itself to train an algorithm

Autoencoder

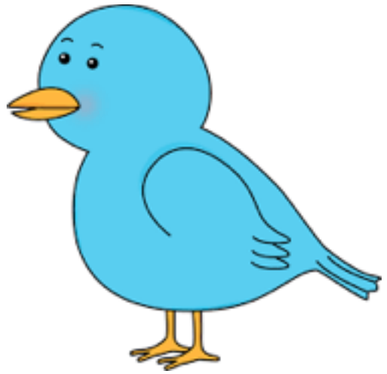


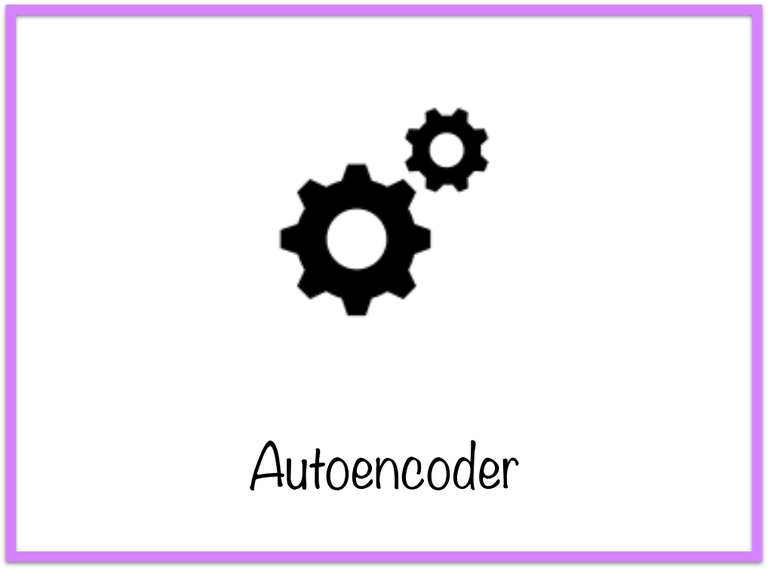


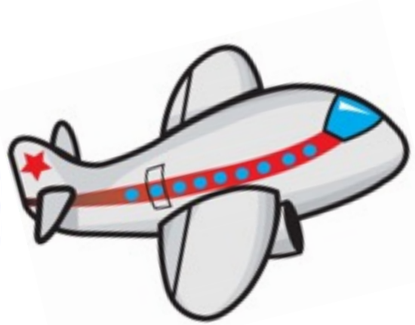
≠



Autoencoder

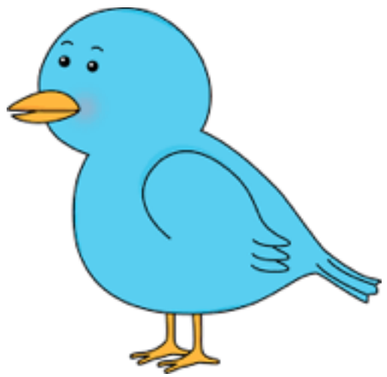


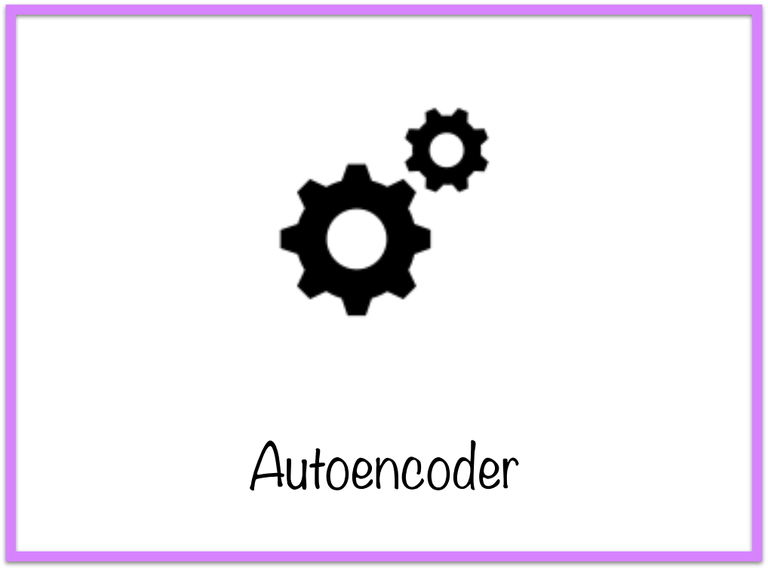


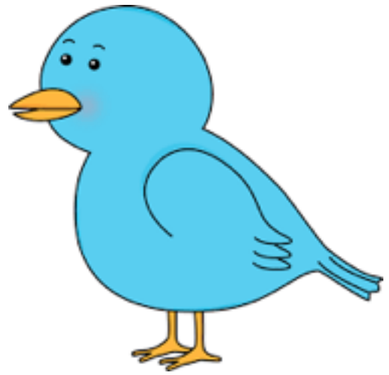


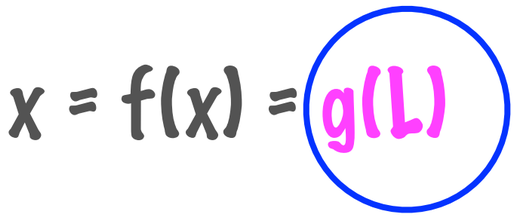
Feedback (cost function) trains the model

Autoencoder









Uncover Hidden Patterns in Data

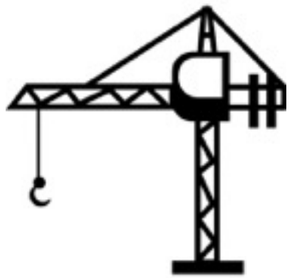
The function f is just the identity, not very interesting. Autoencoders uncover latent factors L that actually drive our data

Autoencoding

k columns

$[\ X_1 \quad X_2 \quad X_3 \quad \dots \quad X_k\ ]$

n rows



Autoencoder

$[\ F_1 \quad F_2\ ]$

n rows

2 columns

Unsupervised learning is often a preparatory step before a supervised learning step (classification, regression)

Autoencoders

Autoencoders are the ultimate “look-within” unsupervised ML technique

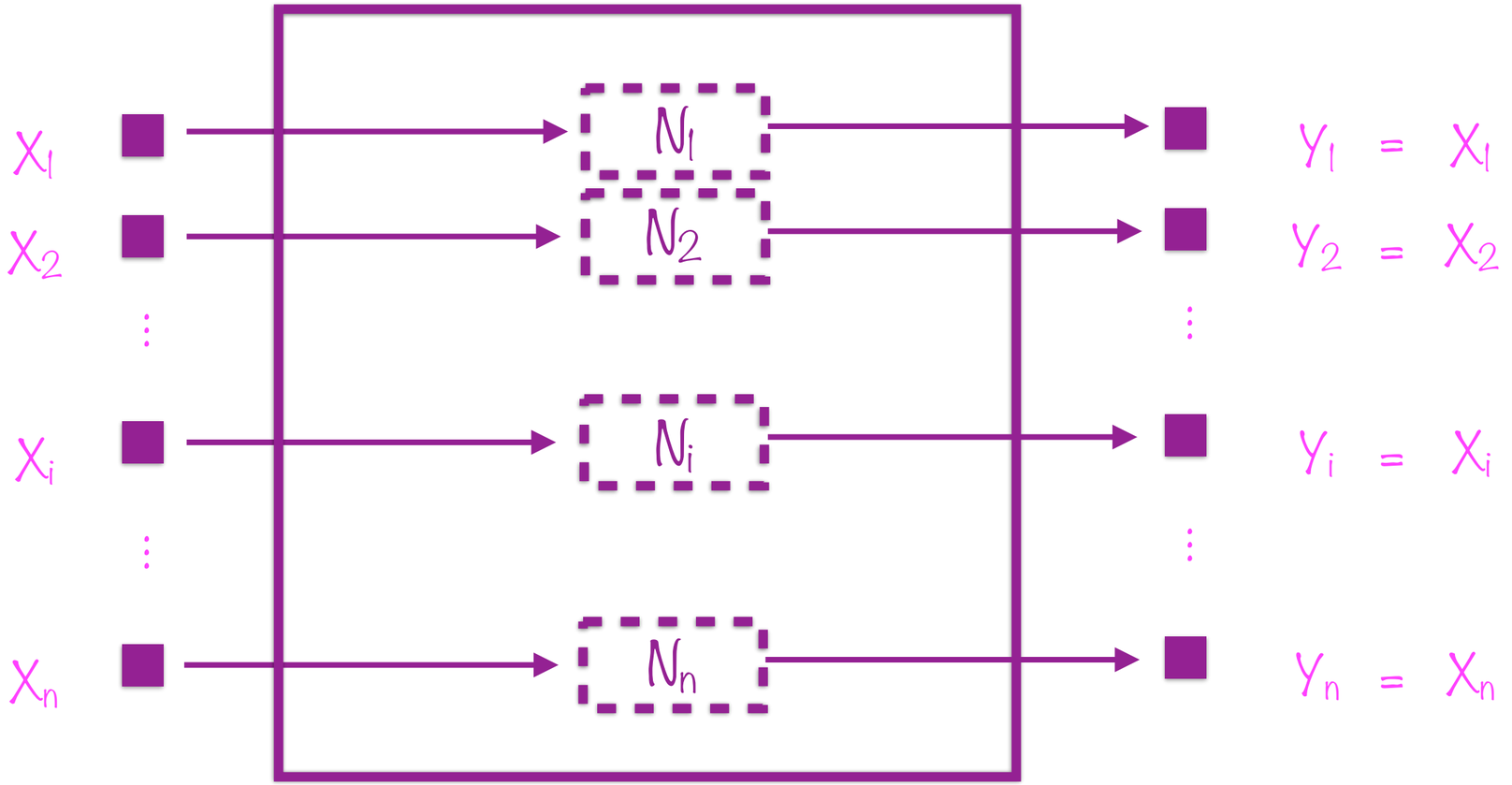
Simply neural networks that try to output exactly what is input

Autoencoders

Sounds trivial...

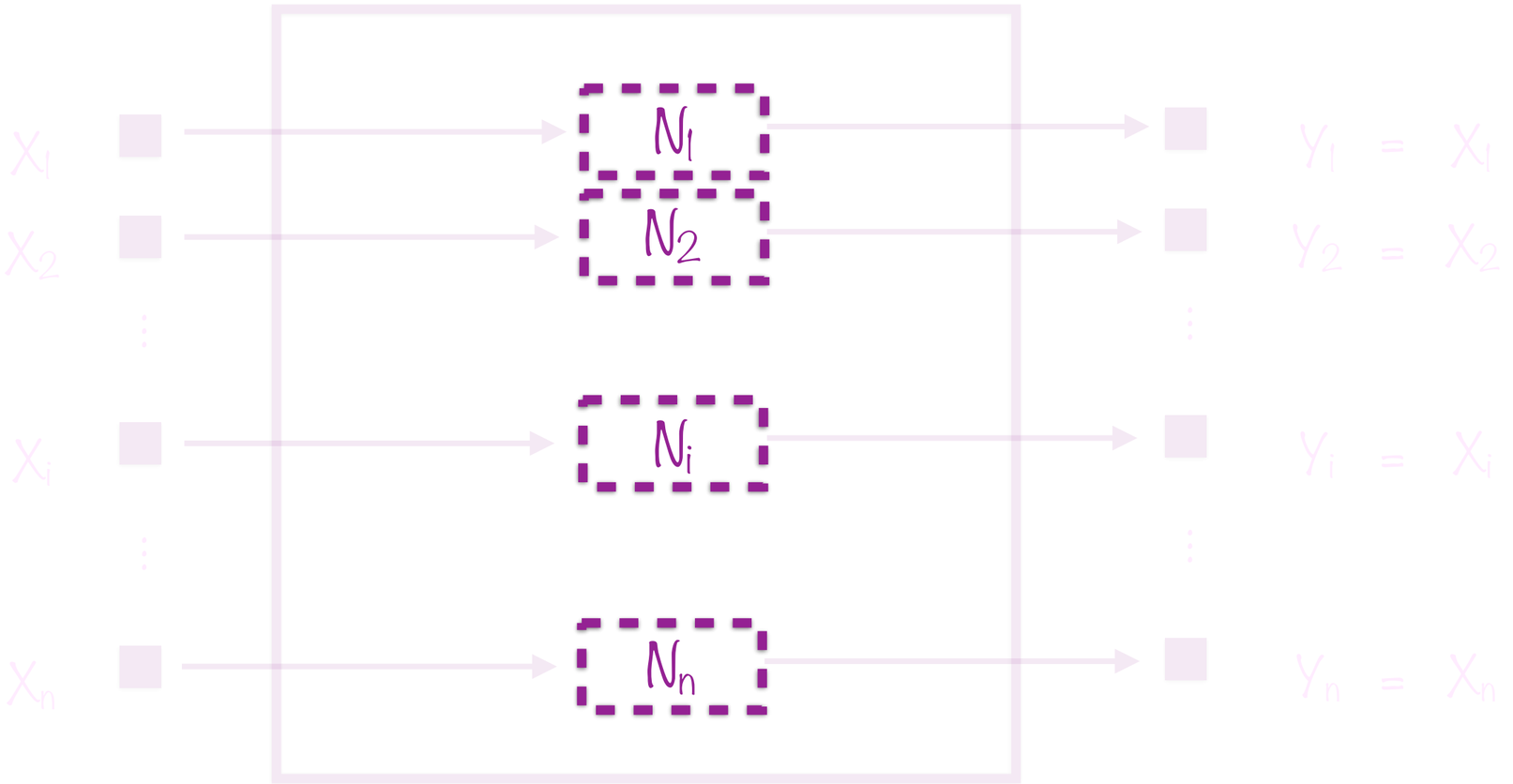
...But we constrain the NN architecture to force real learning

Trivial Autoencoder



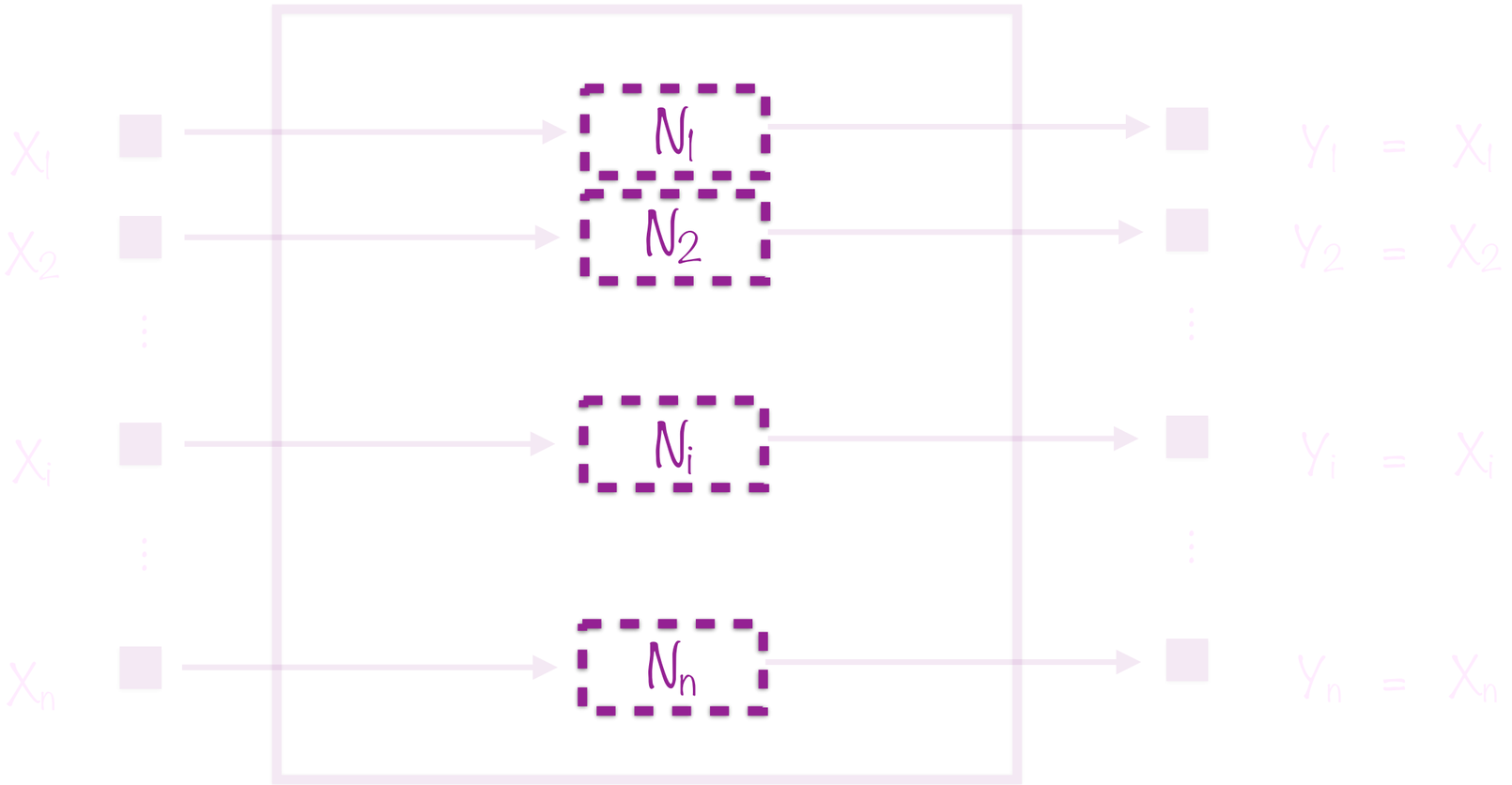
This Neural Network trivially "learns" the input

Trivial Autoencoder



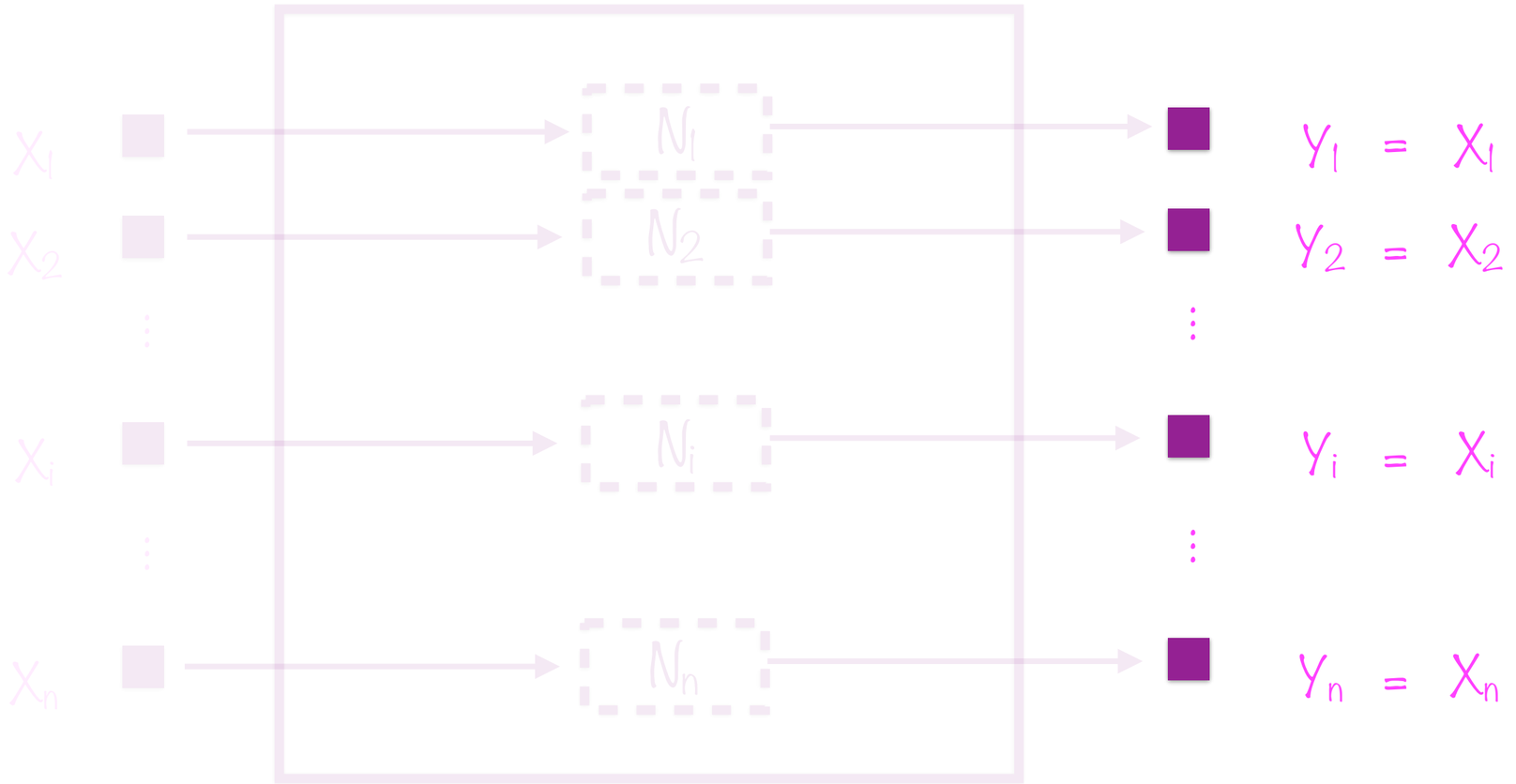
Just one layer, which serves as both input and output layer

Trivial Autoencoder



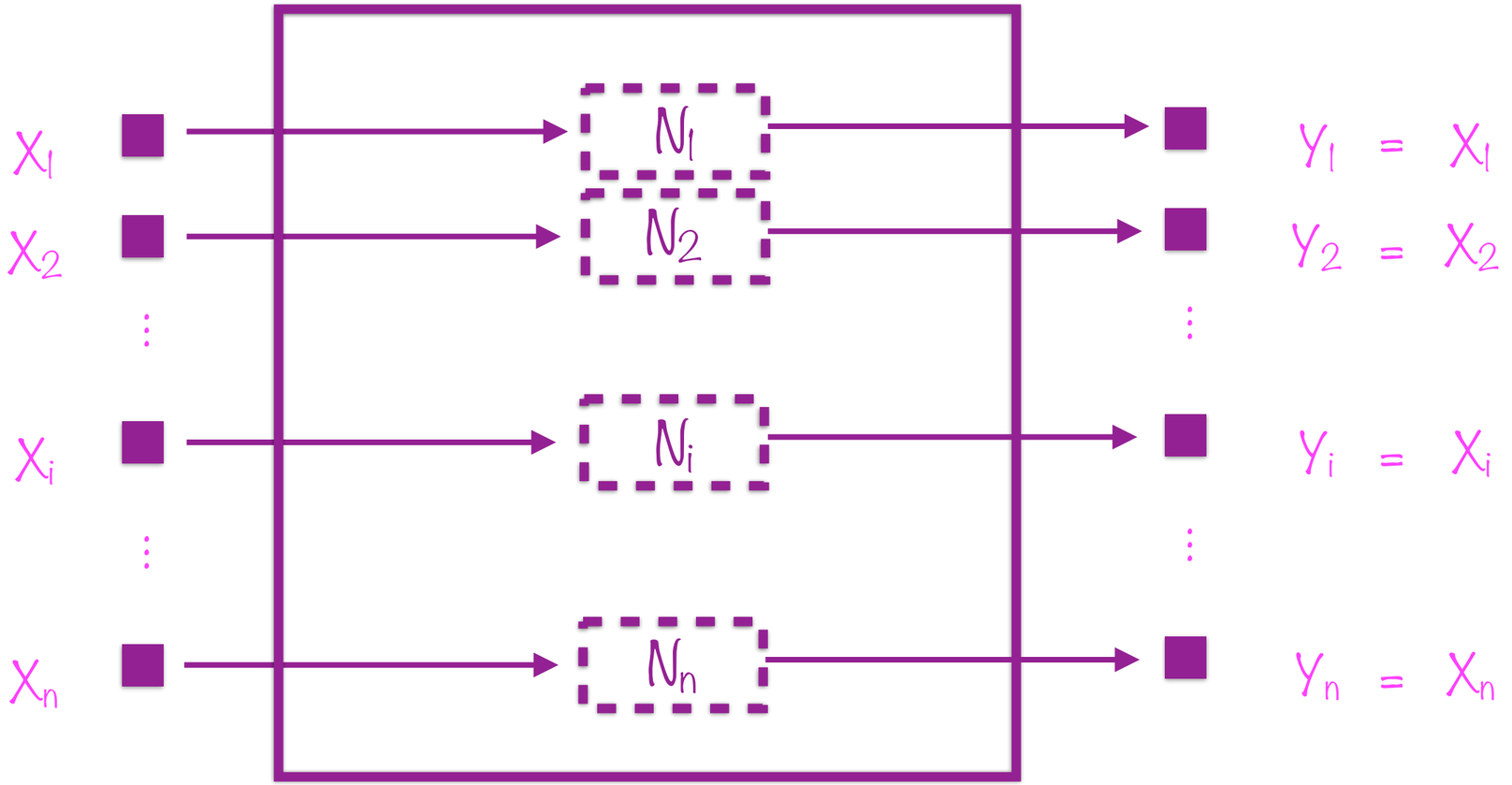
In an autoencoder, the input and output layer must have same dimensionality as the input data

Trivial Autoencoder



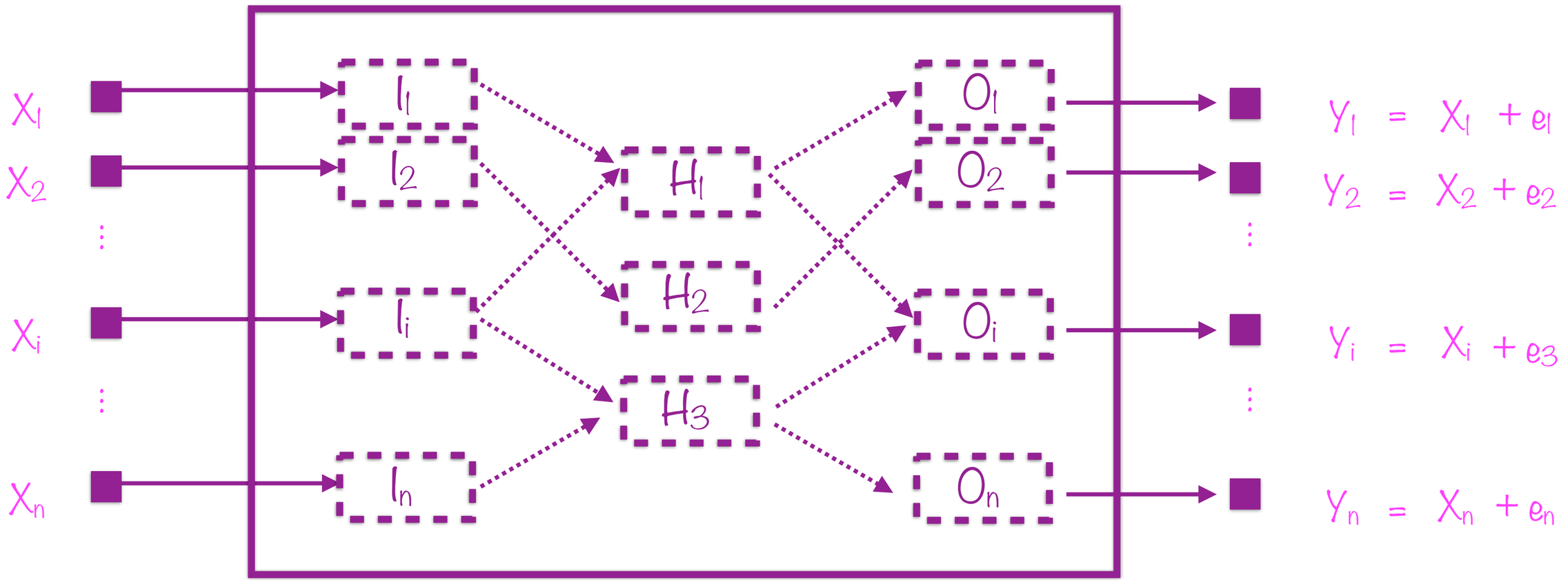
The autoencoder just passes the input through, so output is exactly equal to input

Trivial Autoencoder



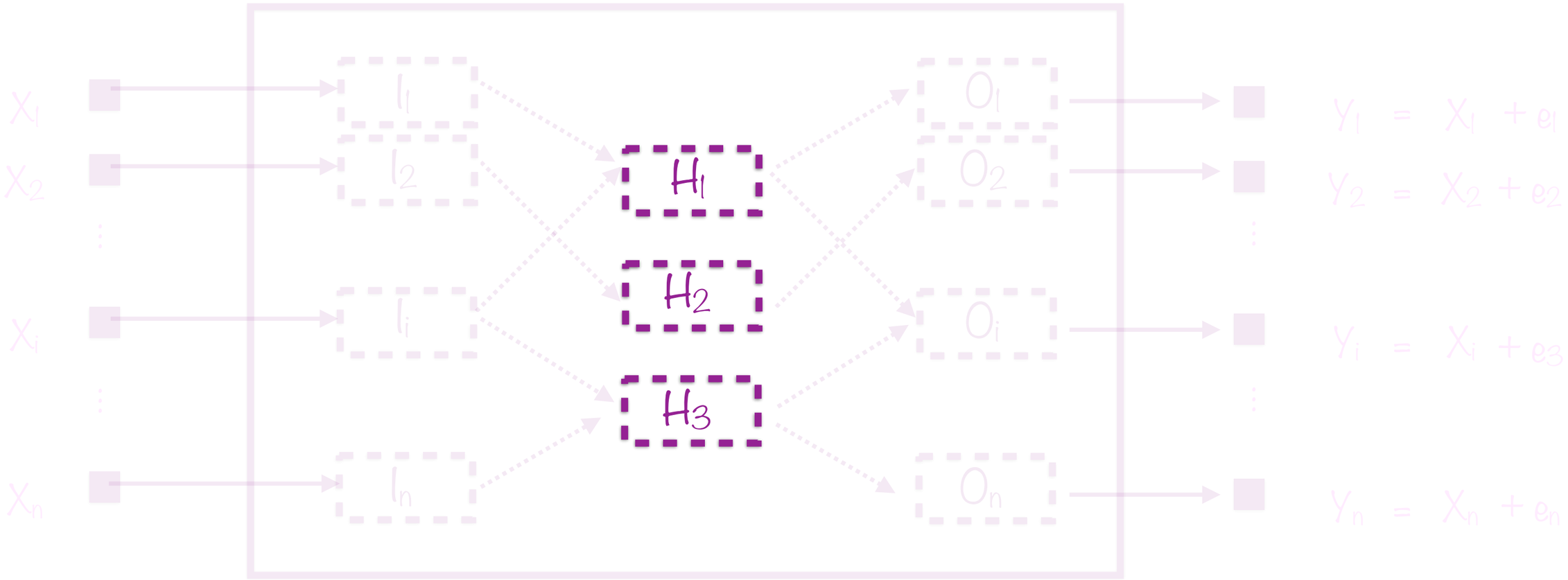
Now, let's constrain the network to force dimensionality reduction

Undercomplete Autoencoder



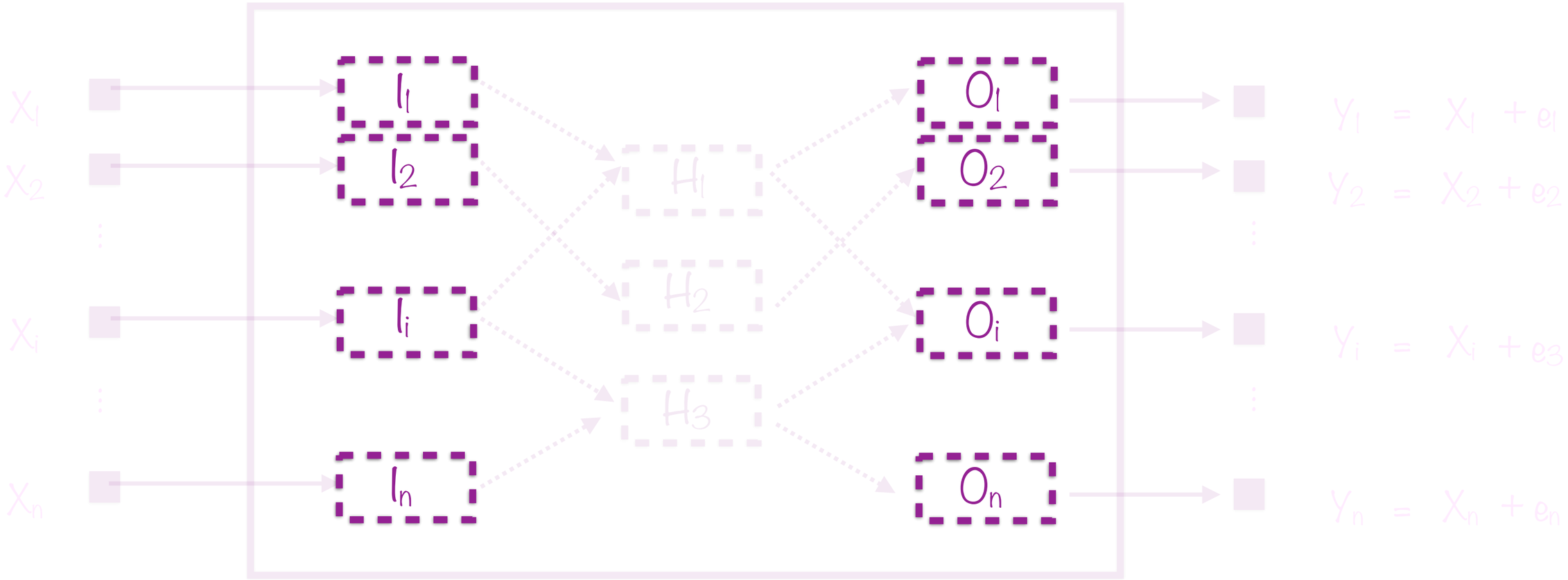
Dimensionality of the NN is now lower than that of input data

Undercomplete Autoencoder



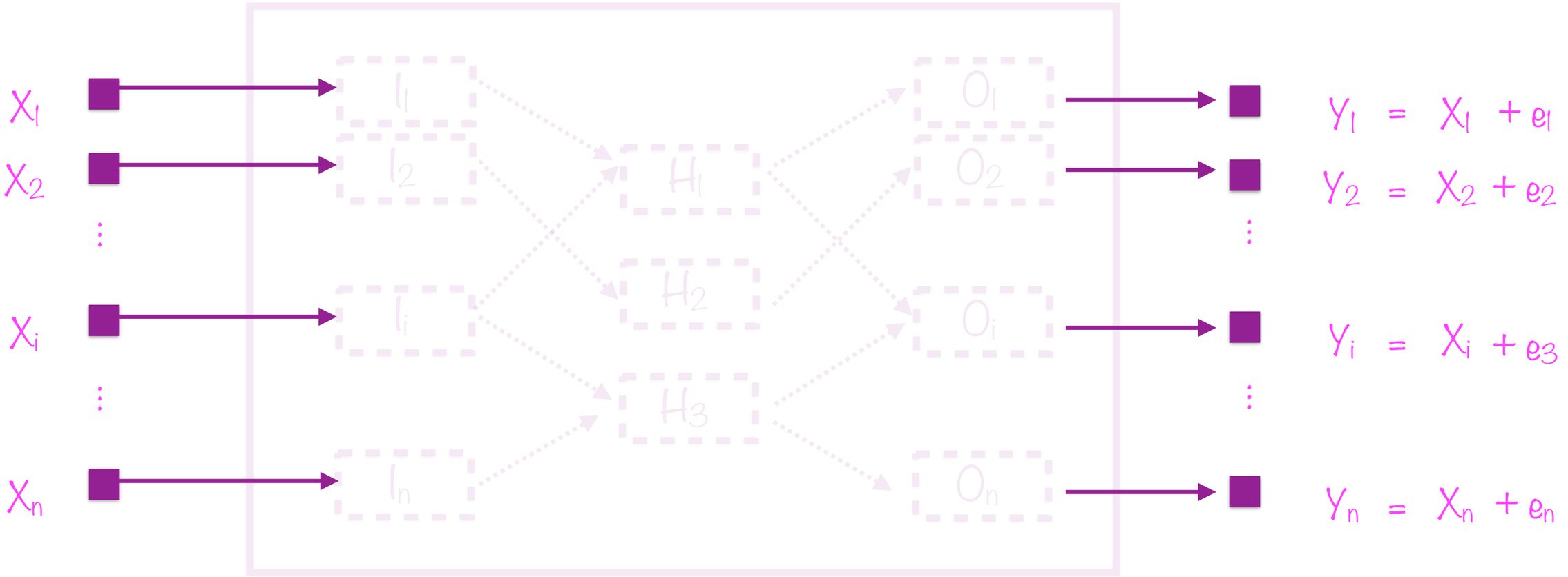
Add a middle, hidden layer with just three neurons ($3 < N$)

Undercomplete Autoencoder



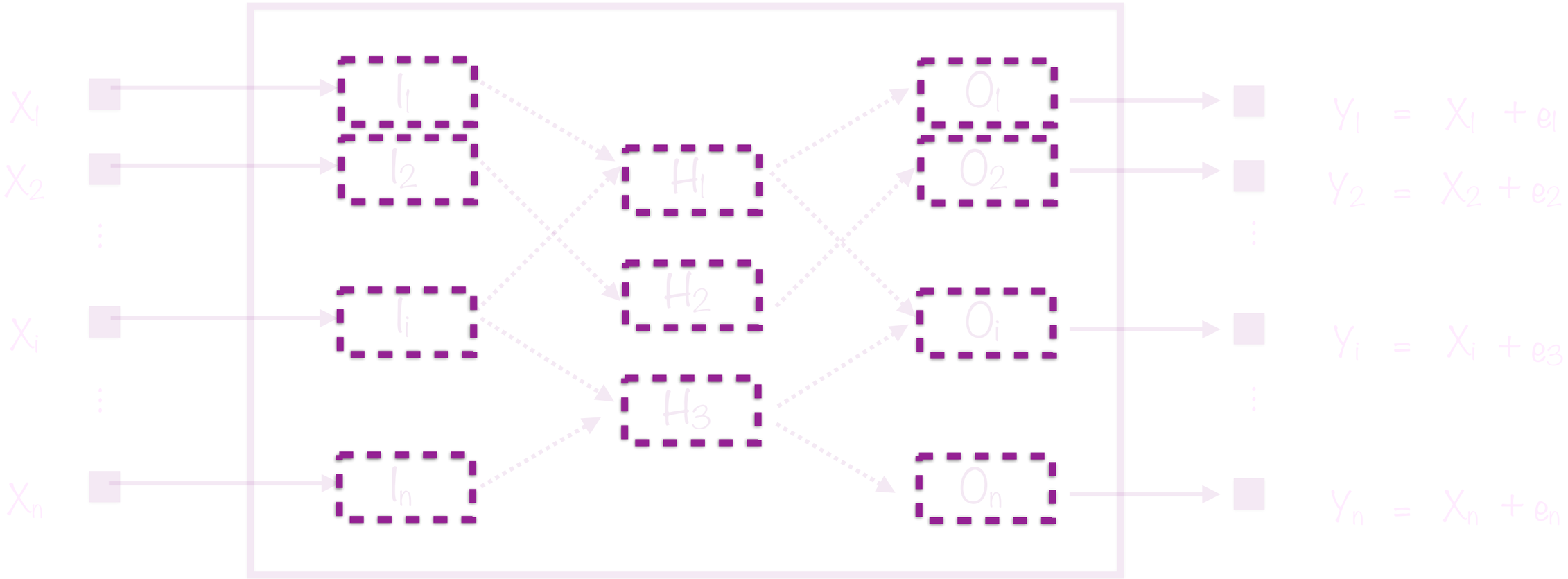
The input and output layers must now be separated, since each must still have same dimensionality as the input

Undercomplete Autoencoder



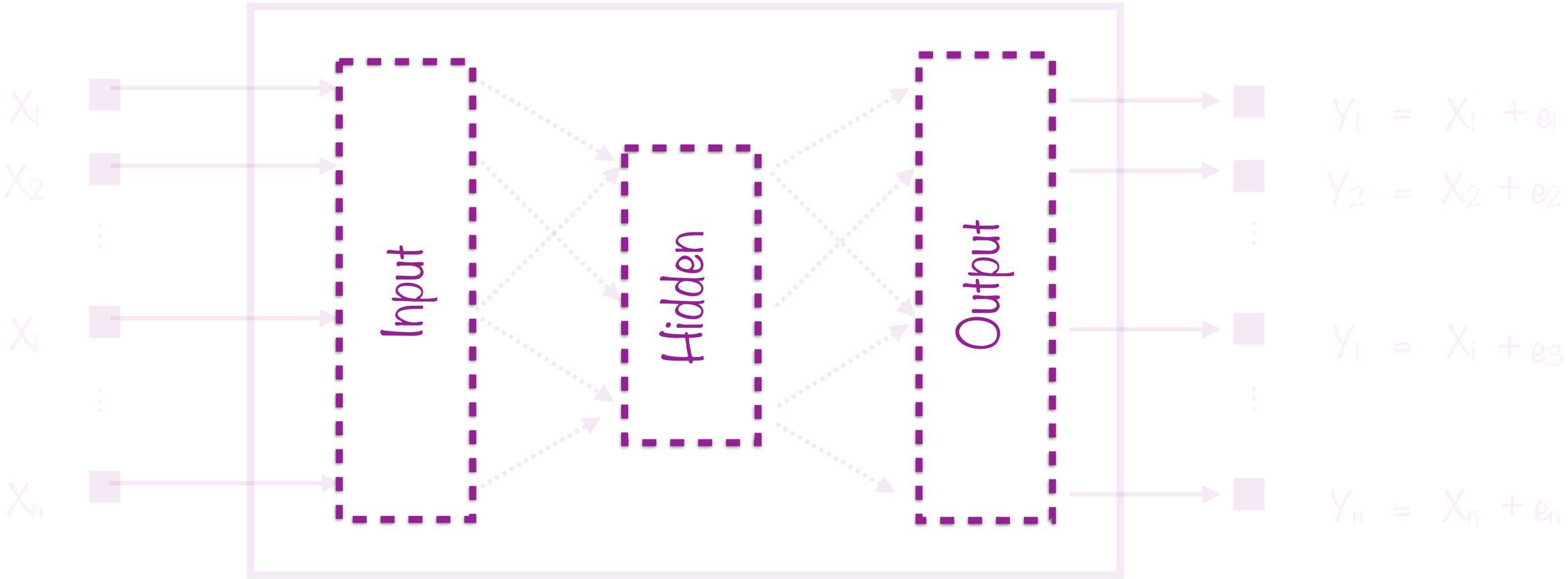
Why? Because autoencoder seeks to reconstruct input

Undercomplete Autoencoder



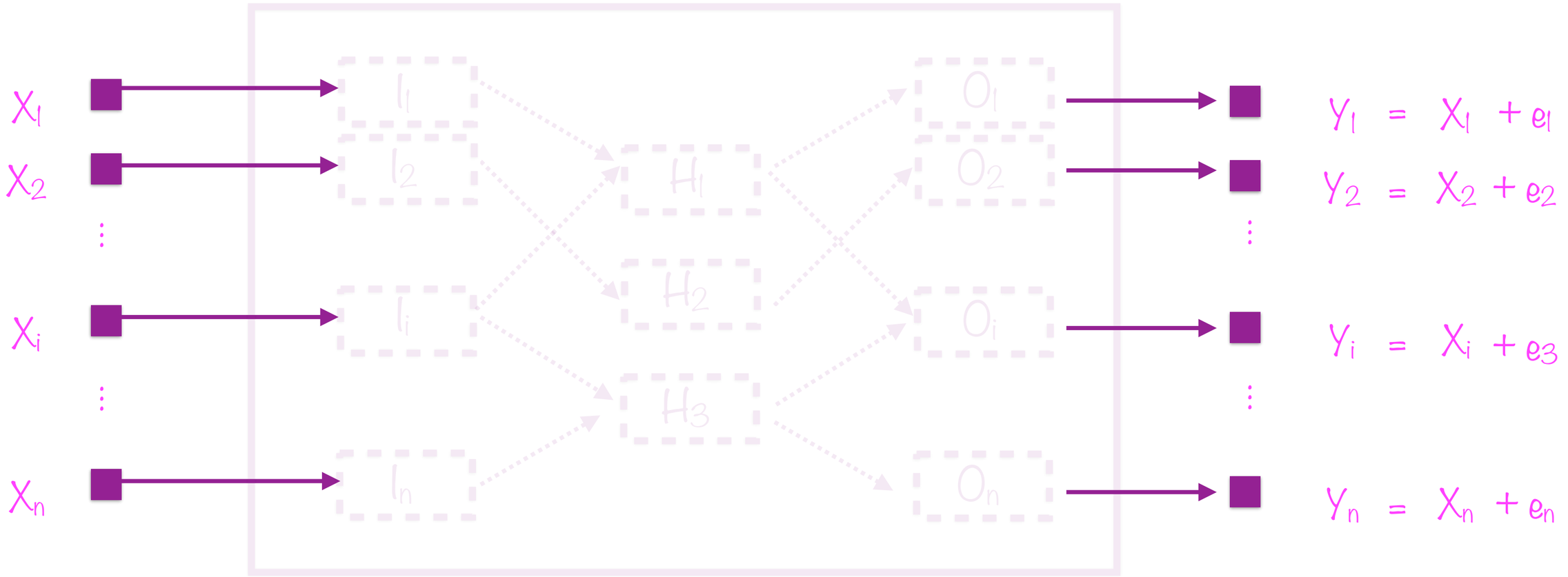
This gives undercomplete autoencoders a characteristic sandwich-like appearance

Undercomplete Autoencoder



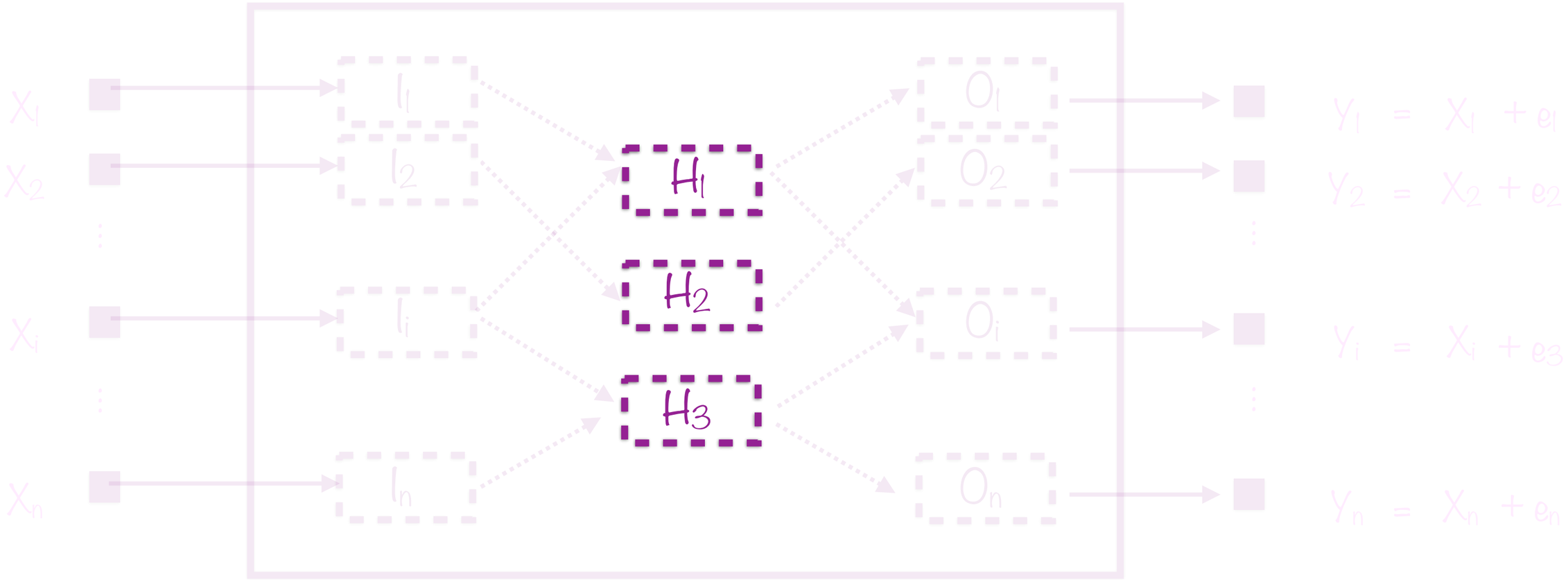
This gives undercomplete autoencoders a characteristic sandwich-like appearance

Undercomplete Autoencoder



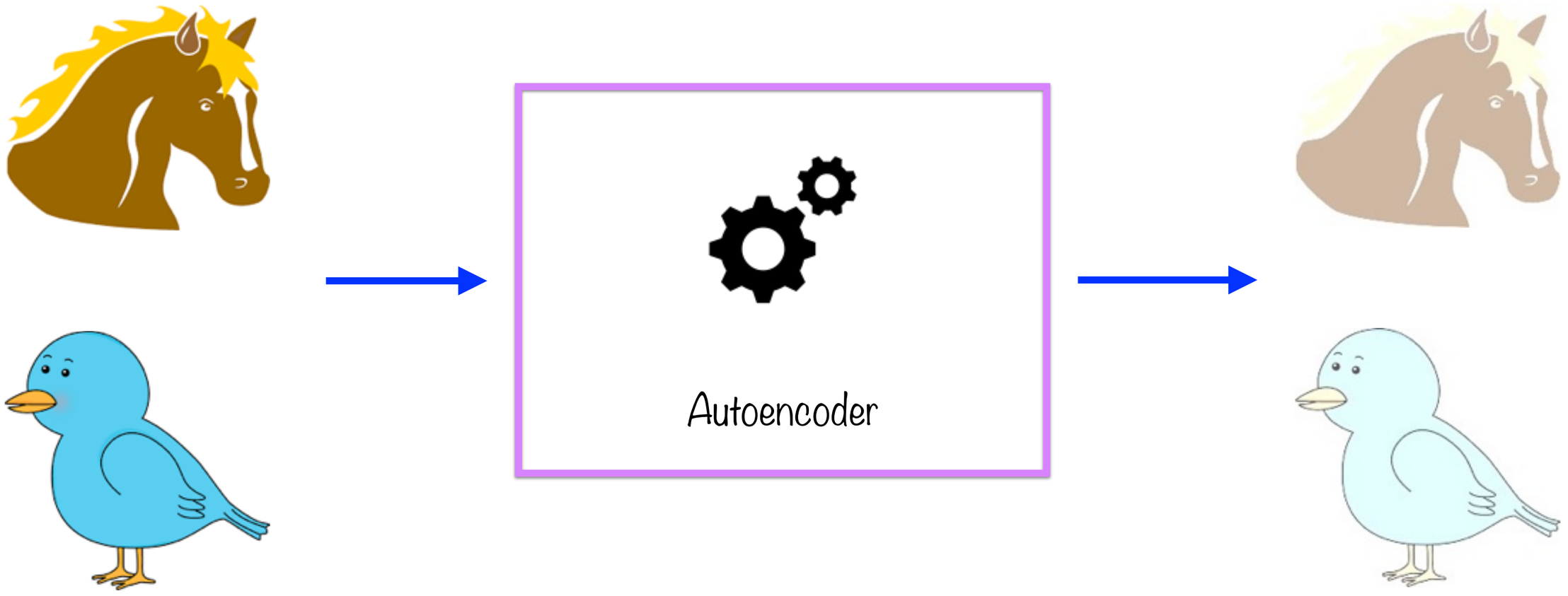
The undercomplete autoencoder will try to exactly match the input, but it will likely not succeed completely

Undercomplete Autoencoder

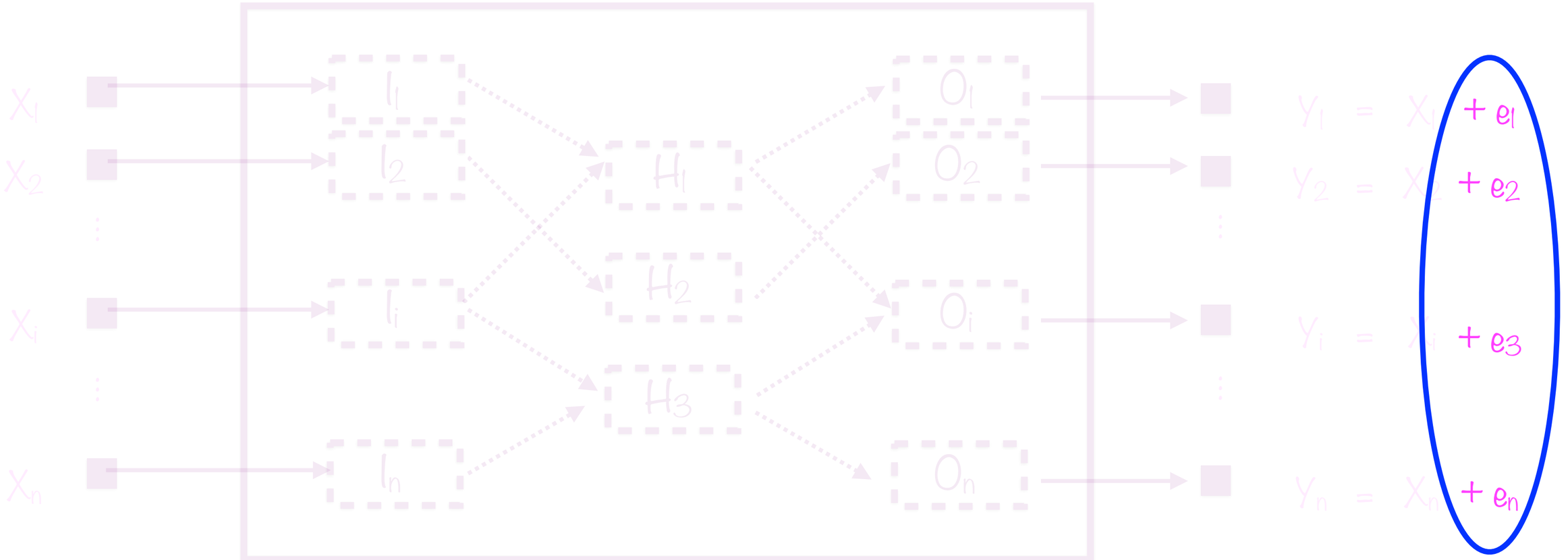


Now, because of the dimensionality reduction, output will not be exactly same as input

Undercomplete Autoencoder

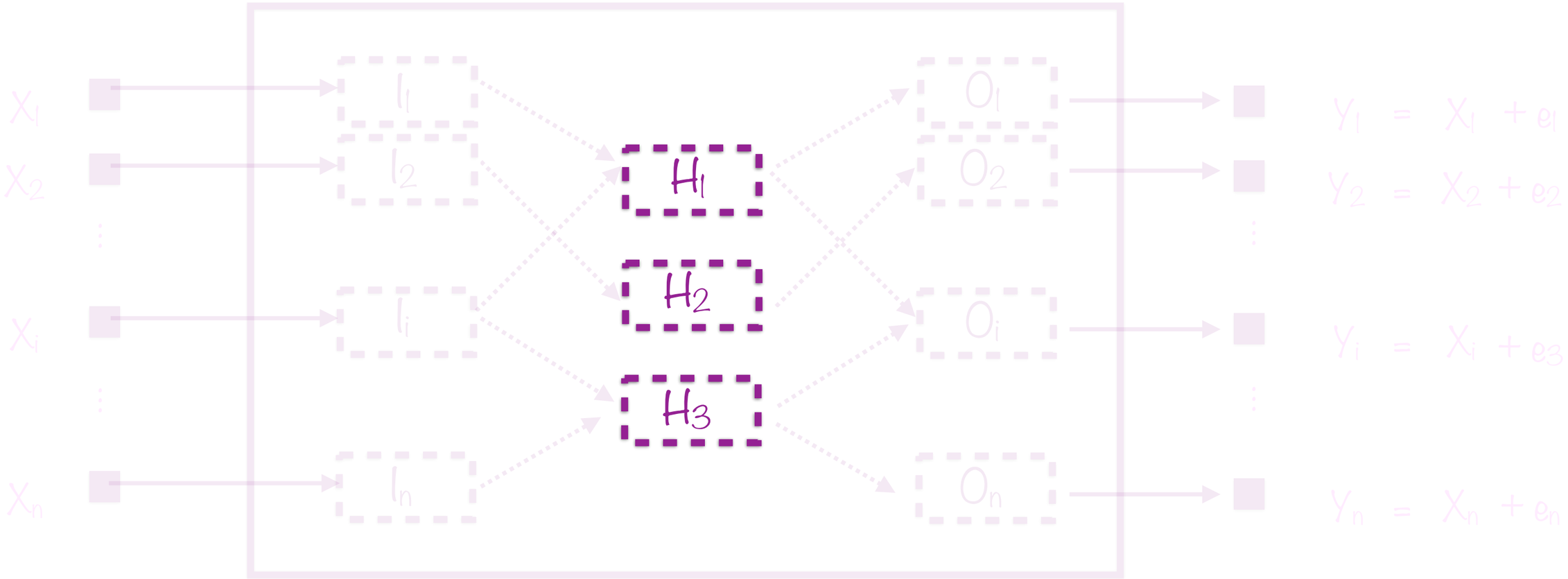


Undercomplete Autoencoder



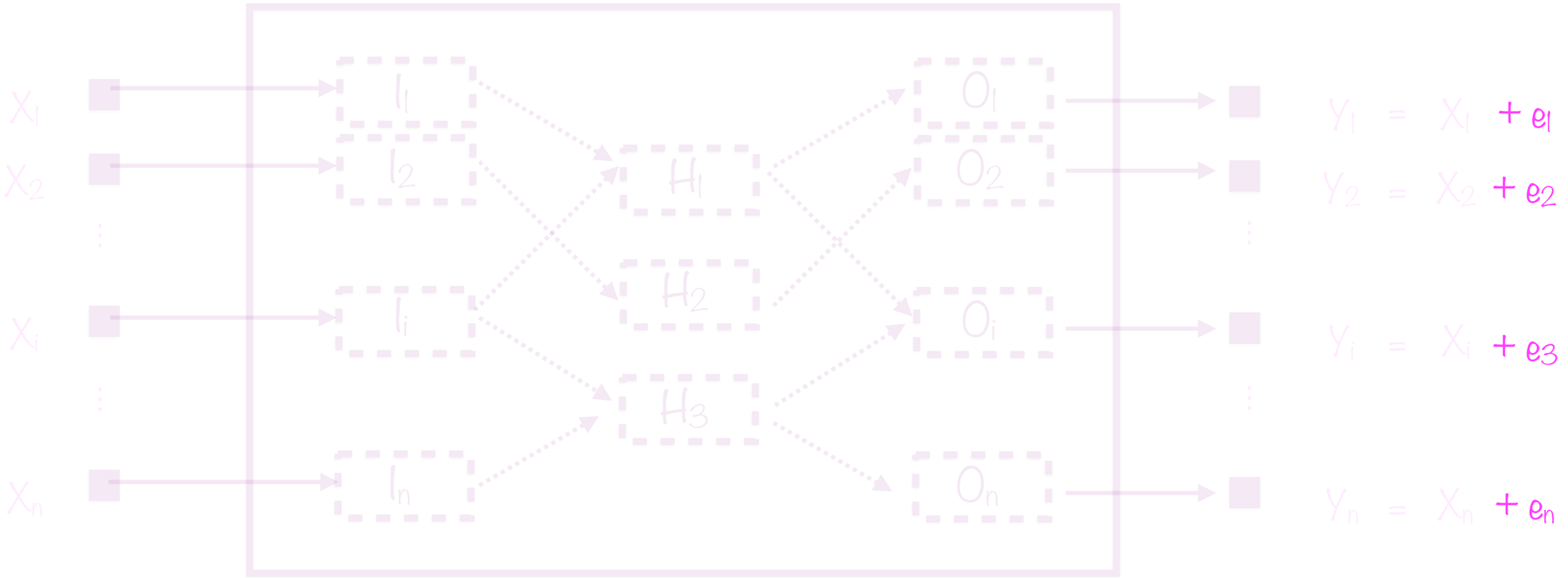
A reconstruction error will now exist

Undercomplete Autoencoder



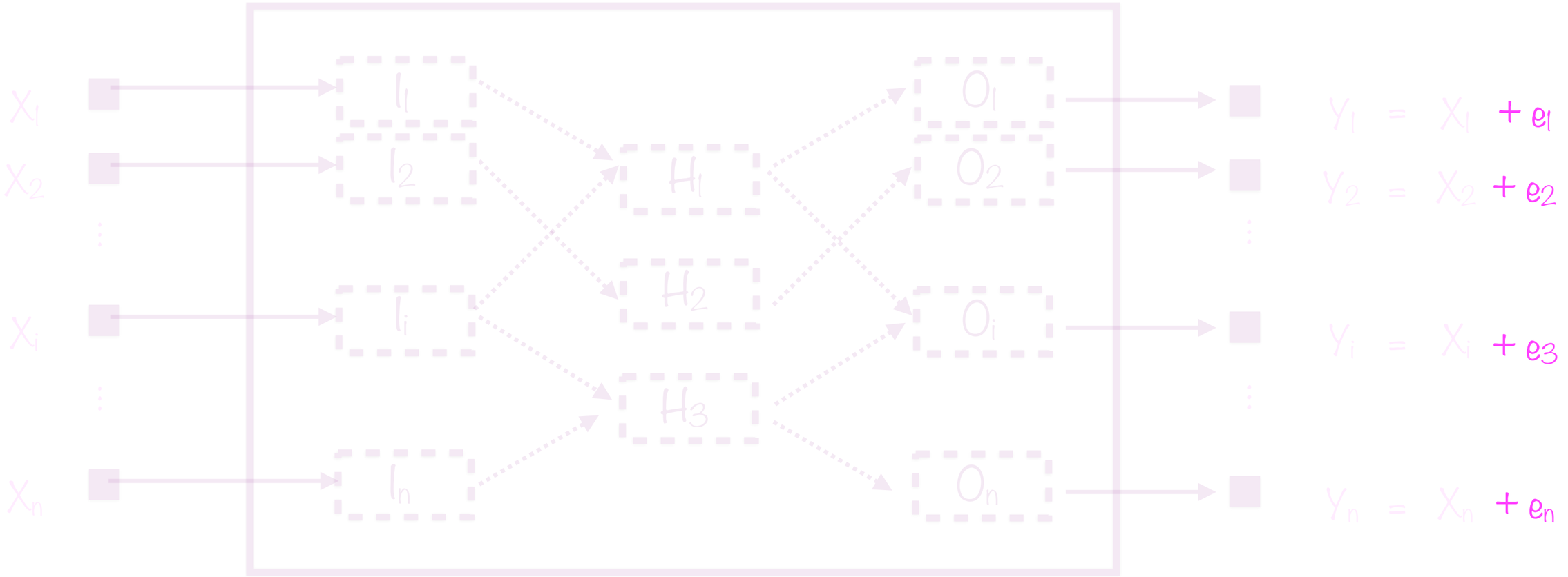
The autoencoder will be forced to learn the most significant characteristics of the data i.e. latent factors

Undercomplete Autoencoder



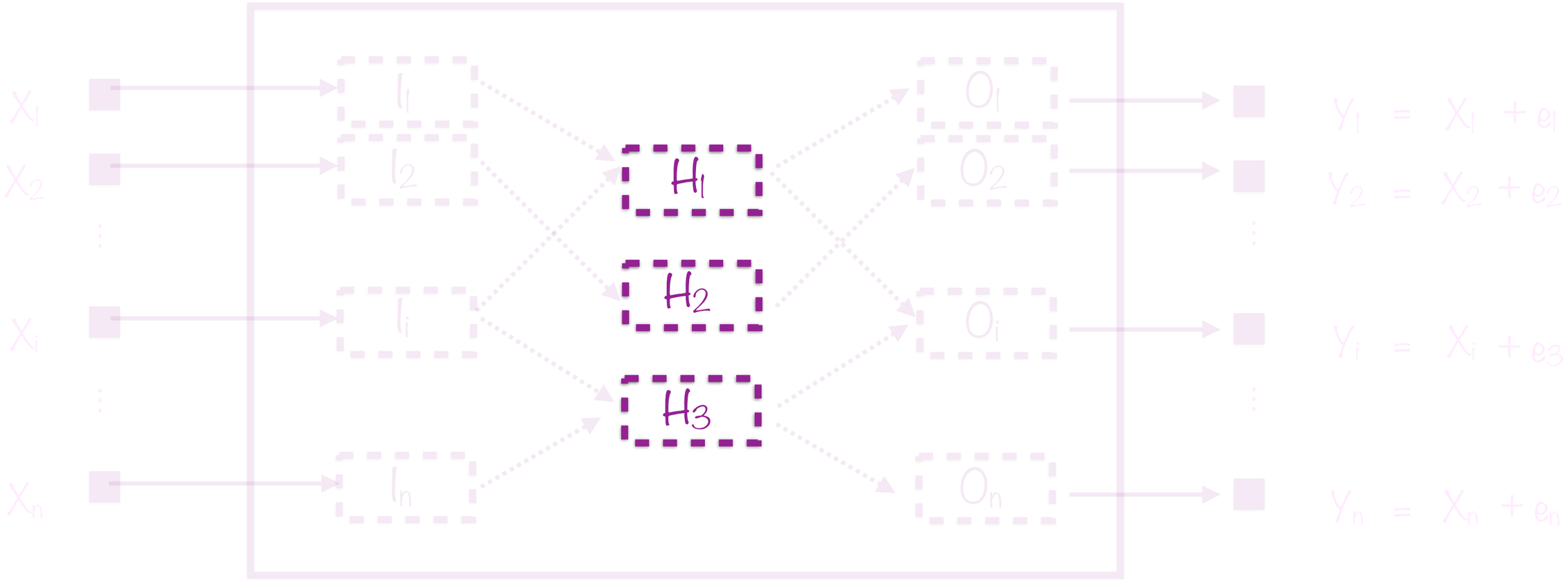
Design choice: What cost function to use in training to minimise reconstruction error?

Undercomplete Autoencoder



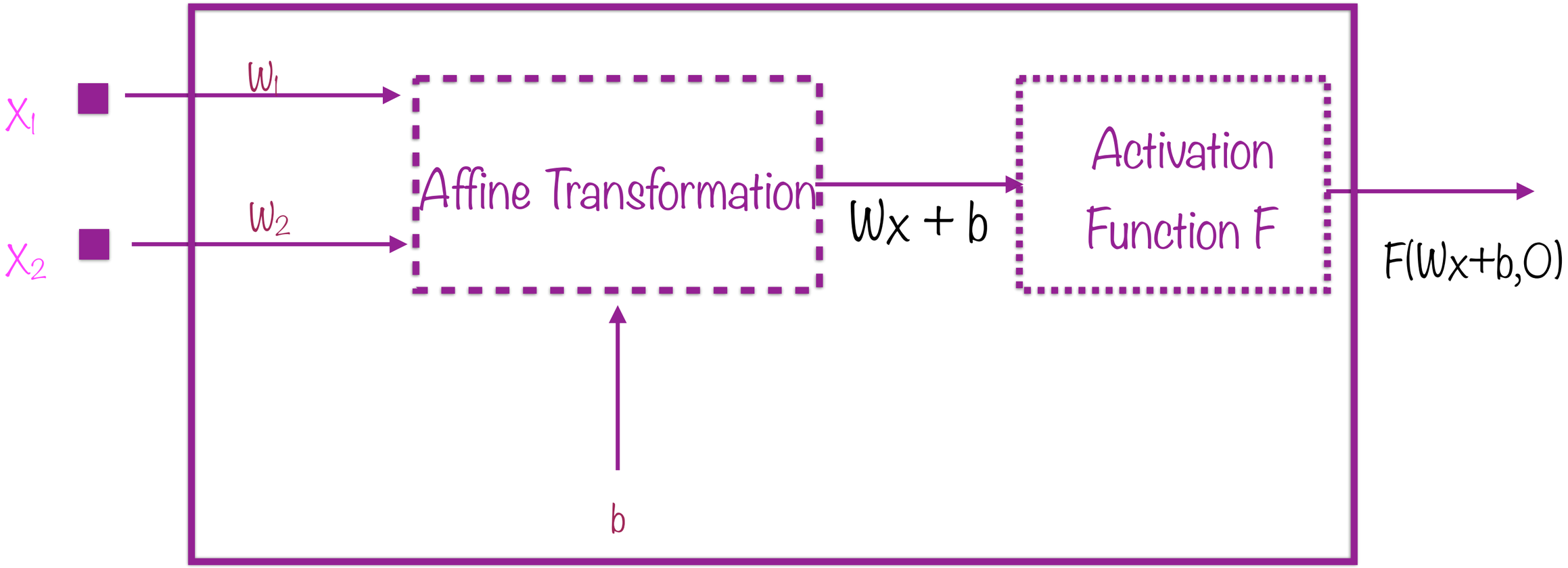
Possible choice: Minimise MSE (mean-square-error)

Undercomplete Autoencoder



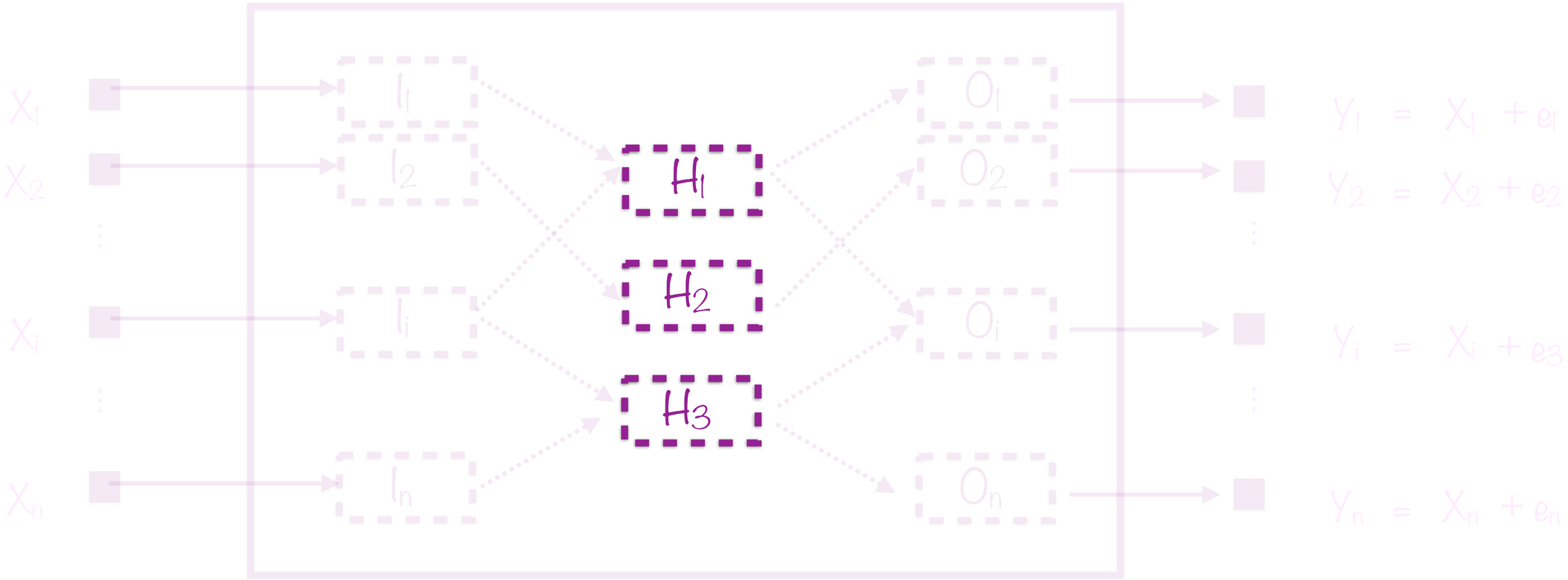
Design choice: What activation function to use for the hidden layer neurons?

Activation Function



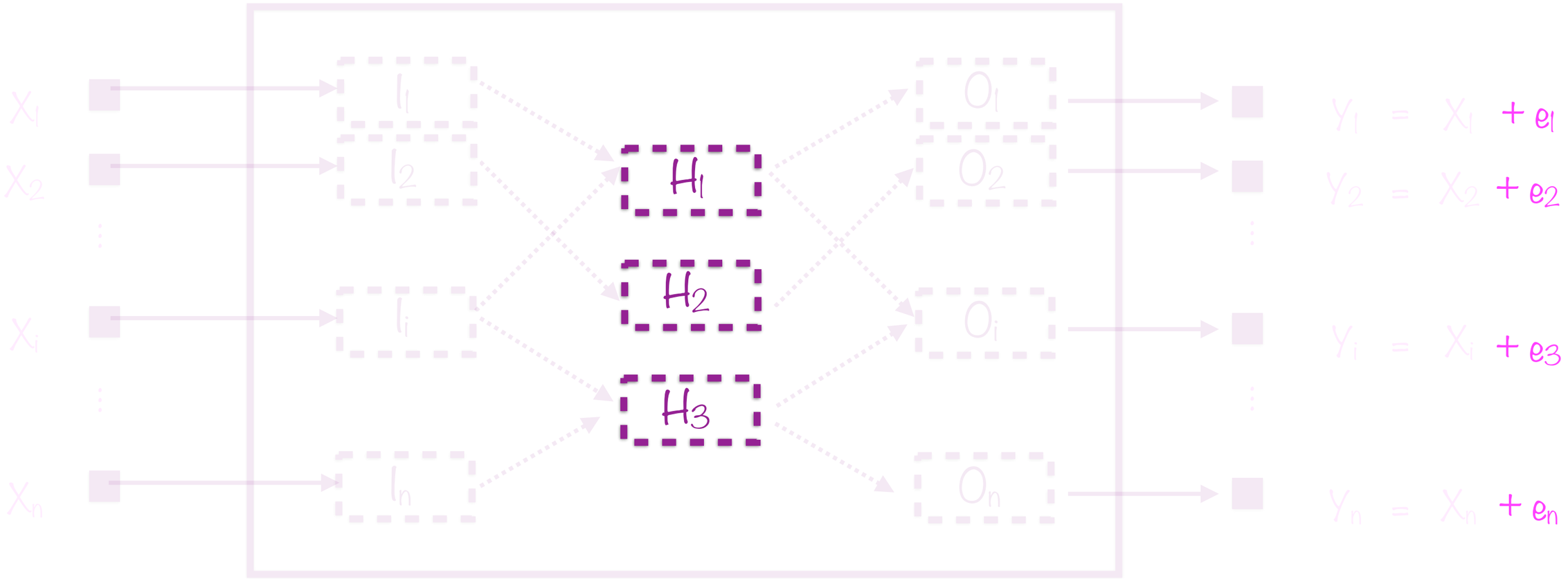
Recall: Each neuron performs two simple operations, an affine transformation and an activation function

Undercomplete Autoencoder



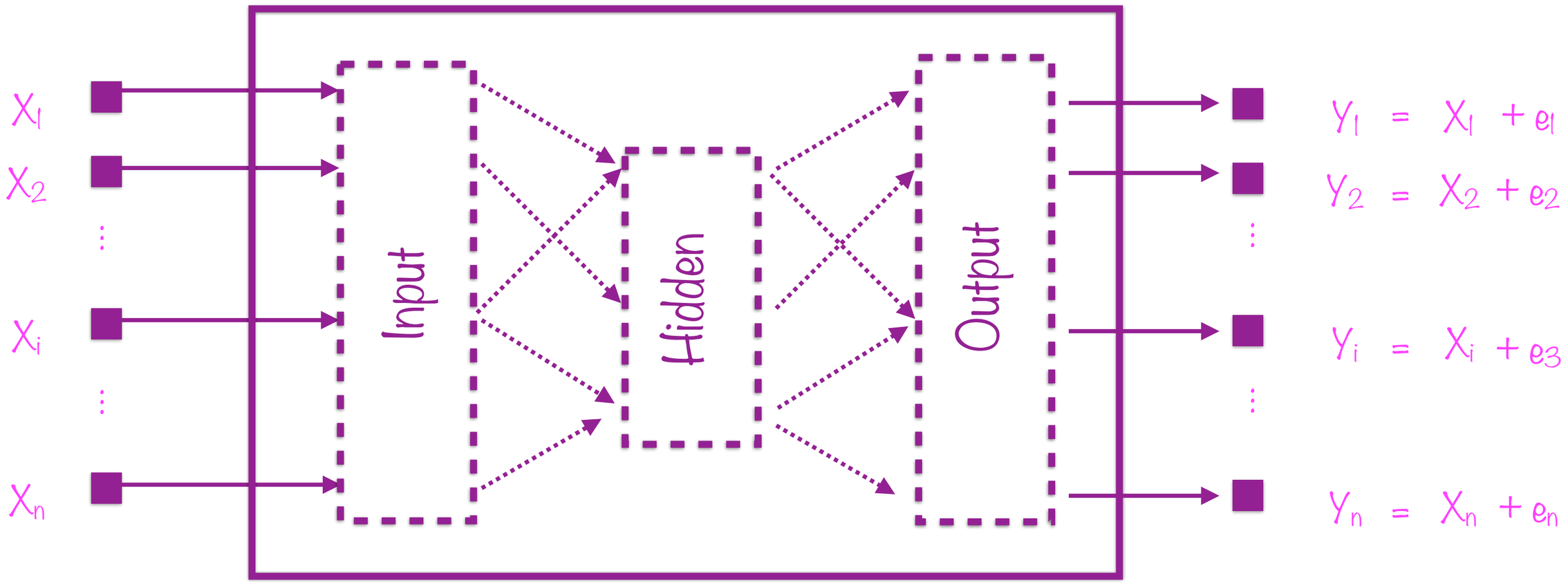
Possible choice: Use linear neuron (no activation function)

Linear Neurons + Min MSE = PCA



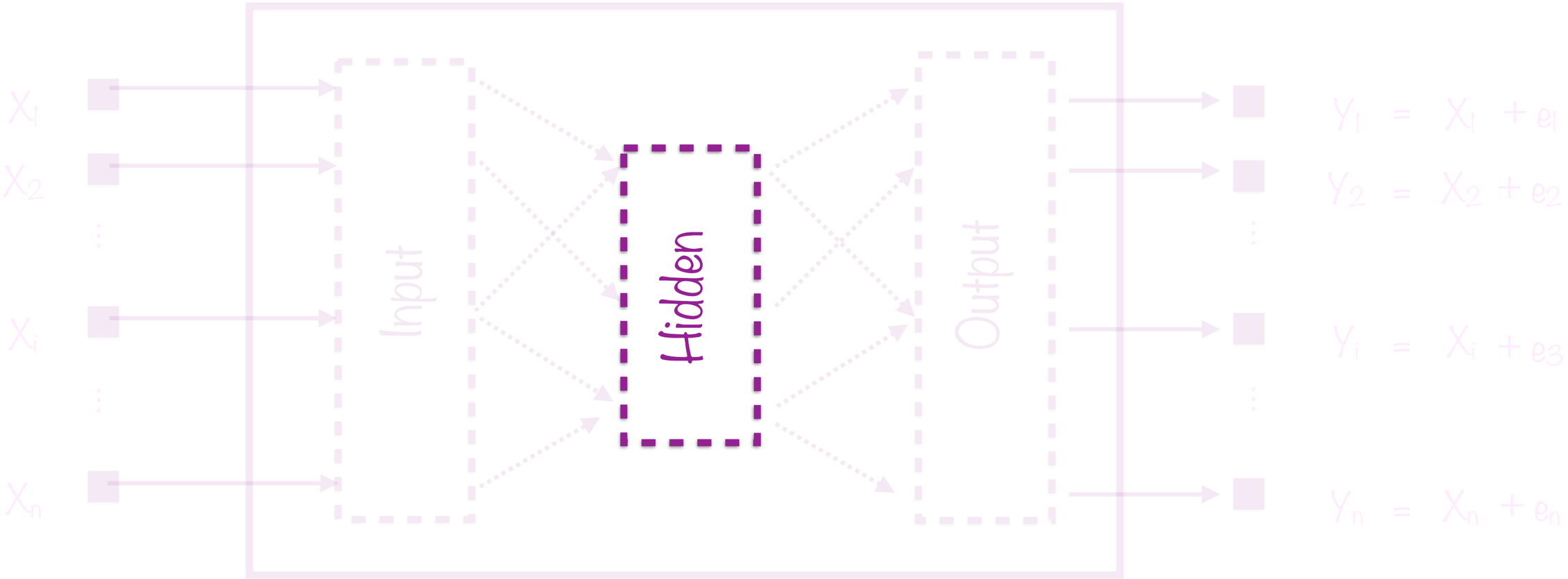
Autoencoders with linear neurons that are trained to minimise MSE will simply perform PCA

Undercomplete Autoencoder



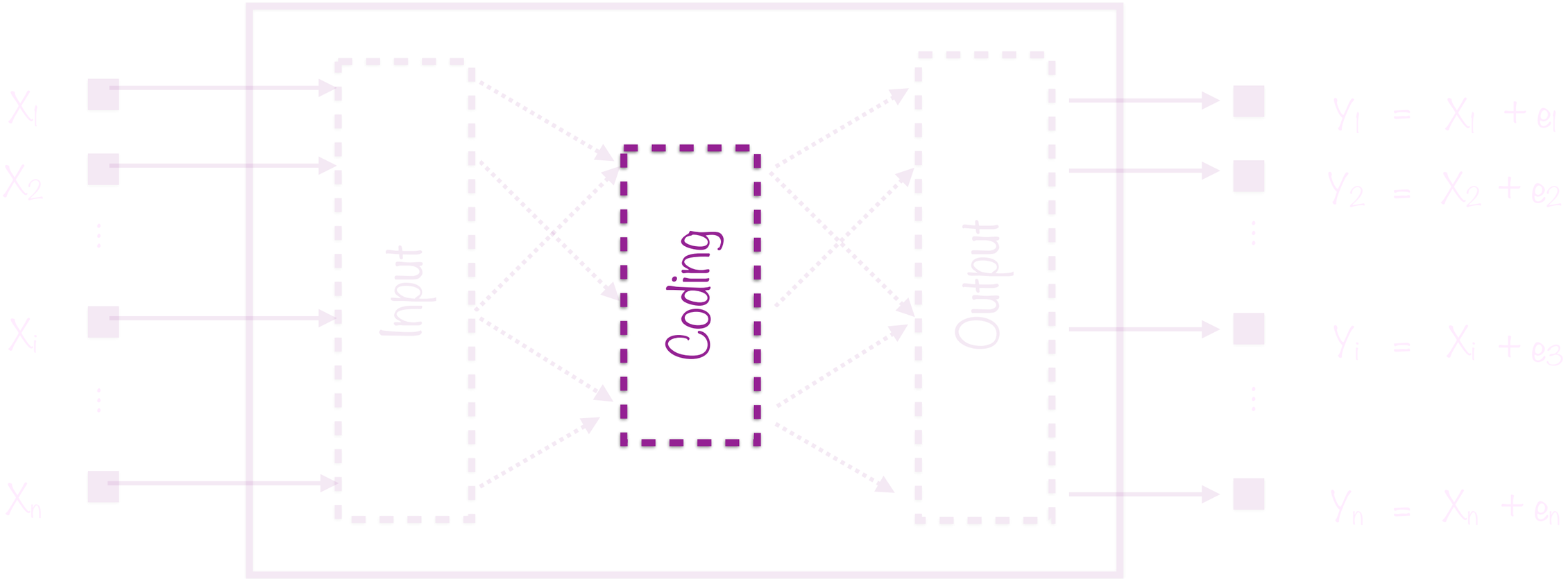
The central hidden layer is called the coding layer

Undercomplete Autoencoder



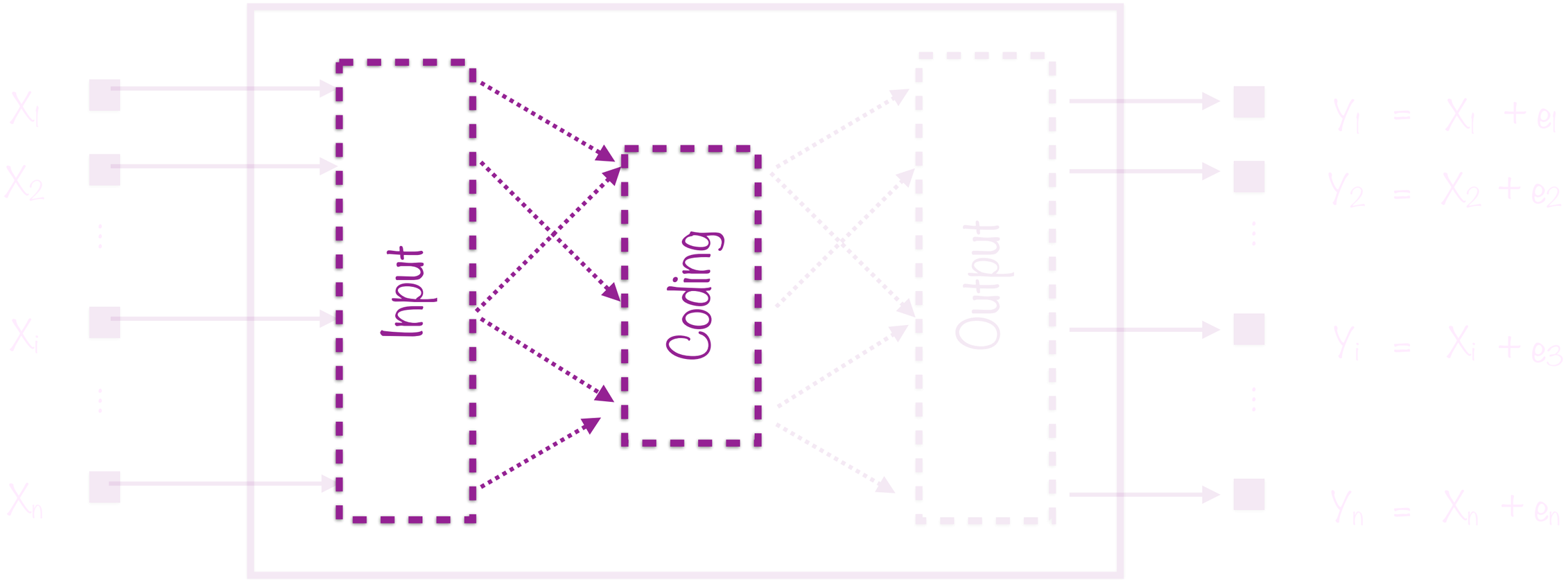
The central hidden layer is called the coding layer

Undercomplete Autoencoder



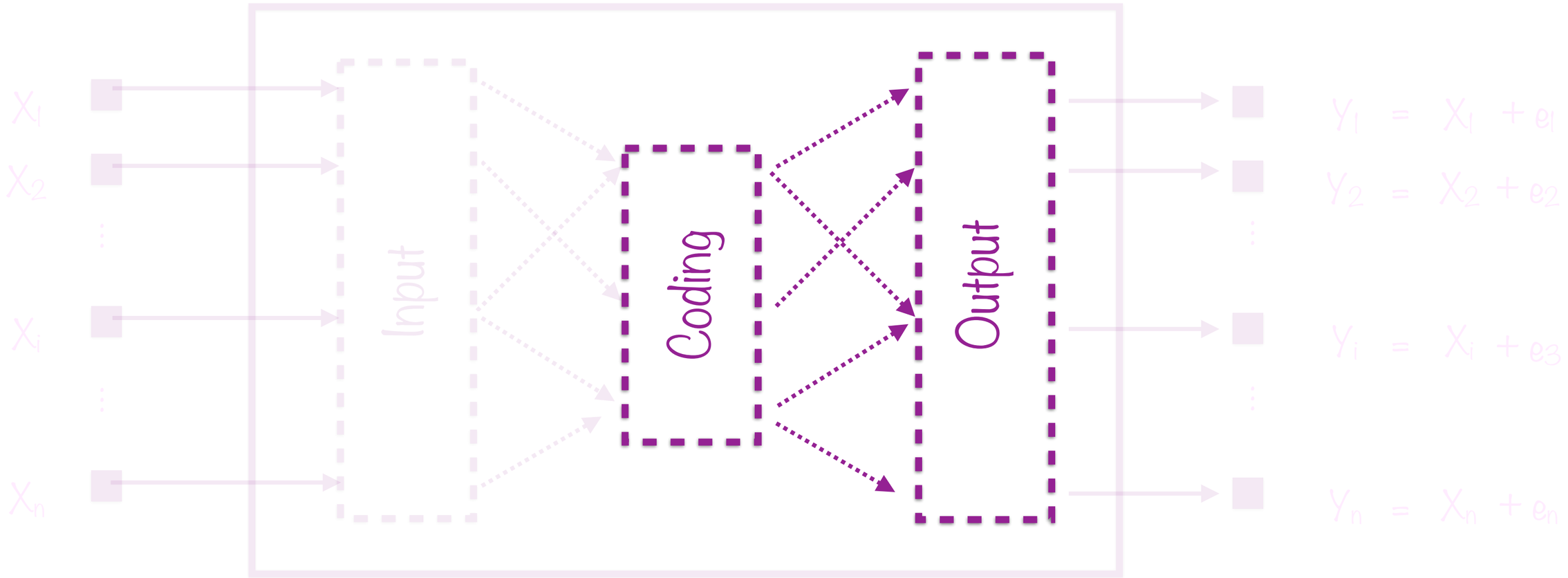
The central hidden layer is called the coding layer

Undercomplete Autoencoder



The first phase - from input to coding - is called encoding

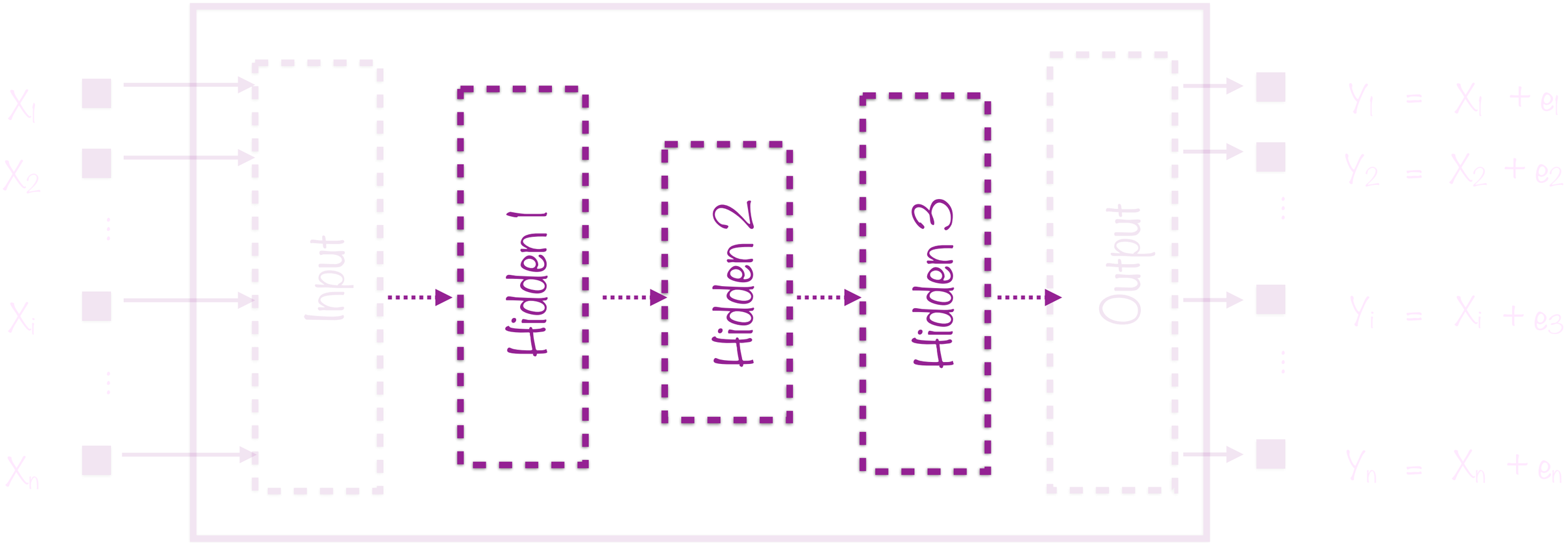
Undercomplete Autoencoder



The second phase - from coding to output - is called decoding

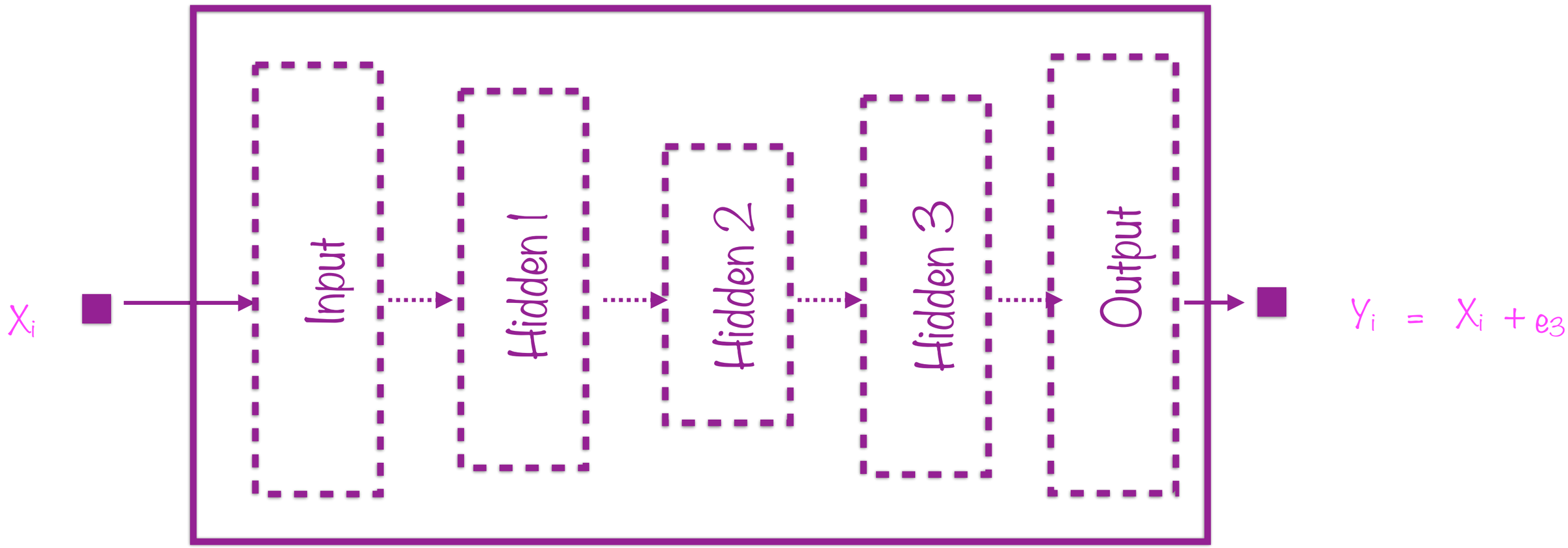
Autoencoders are a way to learn what features really matter

Stacked Autoencoder

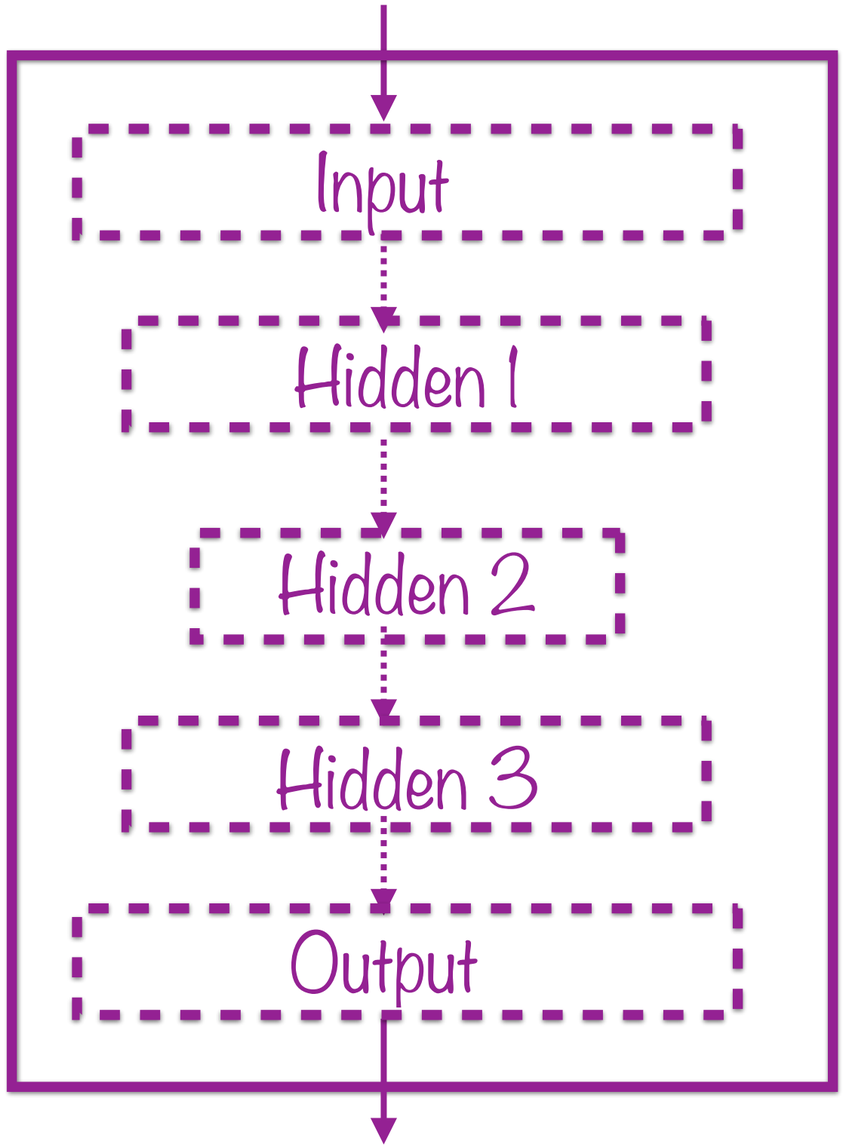


Add multiple hidden layers to learn more complex internal patterns

Stacked Autoencoder



Add multiple hidden layers to learn more complex internal patterns



Stacked Autoencoders

Overfitting is a serious concern with stacked autoencoders

One solution is to "tie" the weights of symmetric hidden layers to be the same

Another is to train individual hidden layers independently

```
y = doSomething(really_complicated_x)
```

Supervised Machine Learning

Here the feature vectors are very complex, making it hard for us to even figure out what features matter

really_complicated_x = g(simple_L)

Unsupervised Pretraining

Use autoencoders to find hidden patterns in the complicated feature data and to discover the latent factors L that matter

```
y = doSomething(g(simple_L))
```

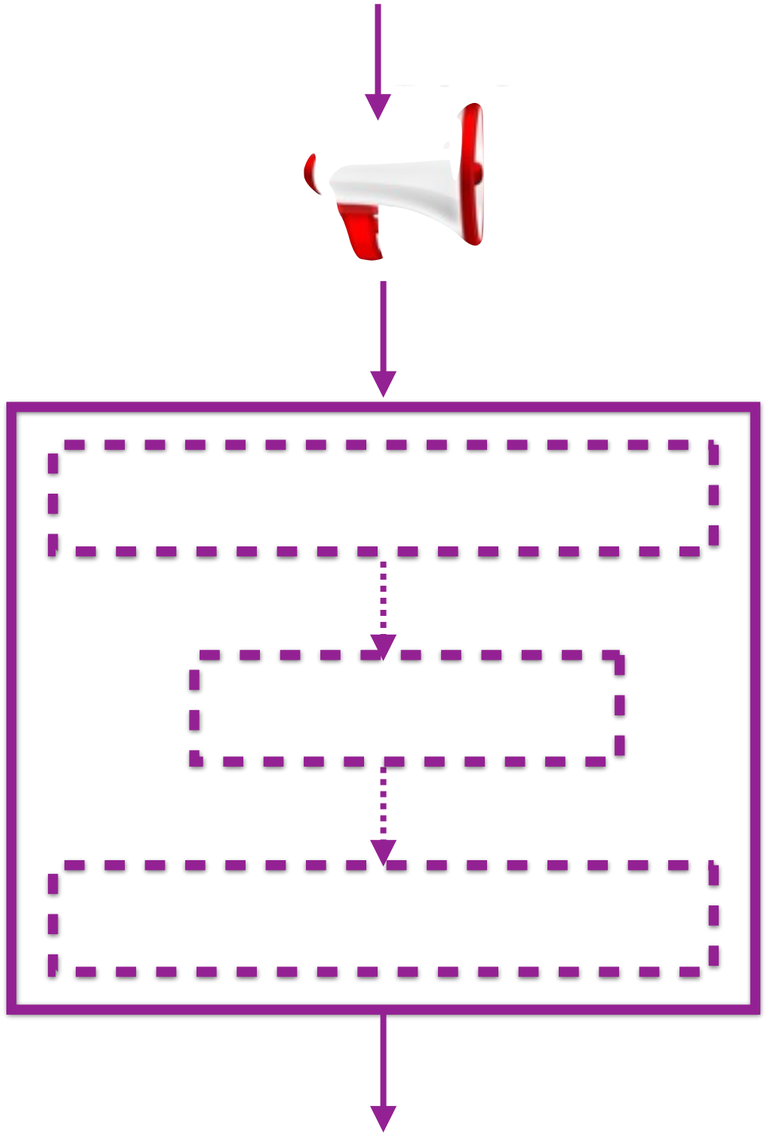
Supervised Learning Made Simpler

The latent factors output by the autoencoder make the supervised learning easier and less reliant on human expertise in feature engineering

$$x = f(x + \text{random\_noise})$$

Denoising Autoencoders

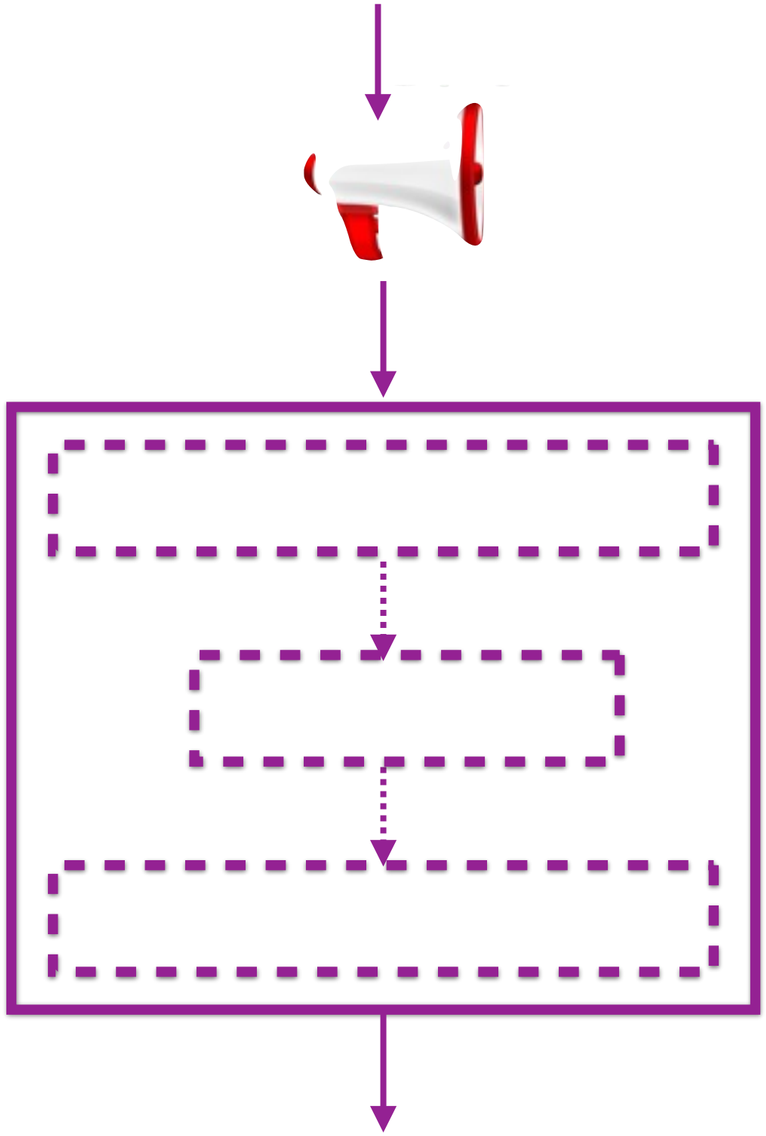
A powerful trick is to add random noise to the inputs and force the autoencoder to recover the signal from the noise



Denoising Autoencoder

Forces autoencoder to discern signal from noise

Can't trivially copy over inputs



Denoising Autoencoder

Two simple mechanisms

- add white noise (Gaussian random numbers)

- Can also switch off inputs e.g. dropout

Types of Learning

$y = f(x)$

Supervised algorithms seek to learn external relationships

$f(x) = ?$

Unsupervised algorithms seek to learn internal relationships

Identifying Specific Individual

Images of people are common - millions of data items

Images of a specific individual are hard to get

Build classifier to tell if two photos are of same individual or not

Use large number of unlabelled images for this

Use image itself as label (autoencoder!)

Now use this classifier on few available photos of specific individual

Identifying Common Drivers

Portfolios of financial assets have hundreds of stocks, bonds

Many common drivers of returns (e.g. state of economy)

Measuring risk of complex portfolio is hard

Identify few, uncorrelated drivers shared by most assets

These factors explain most variance in stock returns - PCA

Now use Principal Components to measure risk

Scenario-based stress tests now relatively easy

Fraudulent Card Transactions

Patterns of card usage vary by individual

More idiosyncratic than spam emails

Hard to say what is fraudulent, what is not

Use autoencoders to find patterns of each individual

No labelled instances required at all

Run autoencoder on new card transactions

Flag instances where reconstruction loss is high