Deep Learning and Applications

DSA 5204 • Lecture 8
Dr Low Yi Rui (Aaron)
Department of Mathematics

So Far

So far, we have

- Introduced some basic NN architectures
- Many techniques to improve the performance of NNs

However, the discussion has been focused on supervised learning

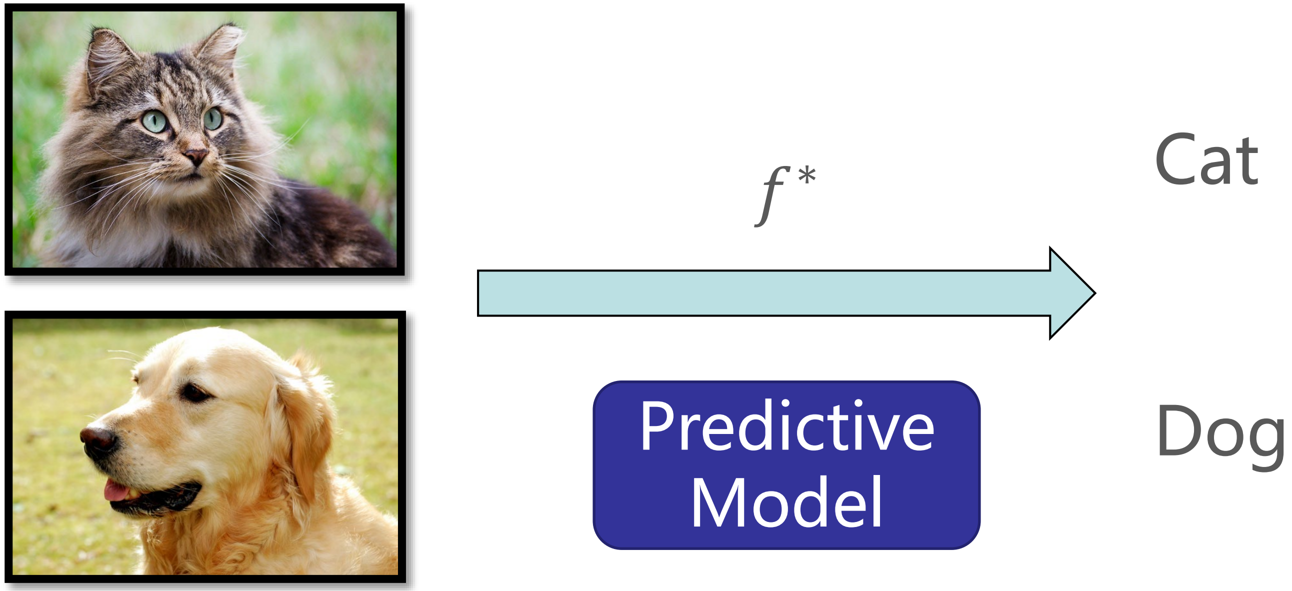
$$y = f^*(\boldsymbol{x}), \text{learn } \hat{f} \approx f^*$$

This lecture will venture out of this setting, into semi-supervised or unsupervised learning.

Unsupervised Learning Overview

Supervised Learning

Supervised learning is about learning to make predictions



f^*

Cat

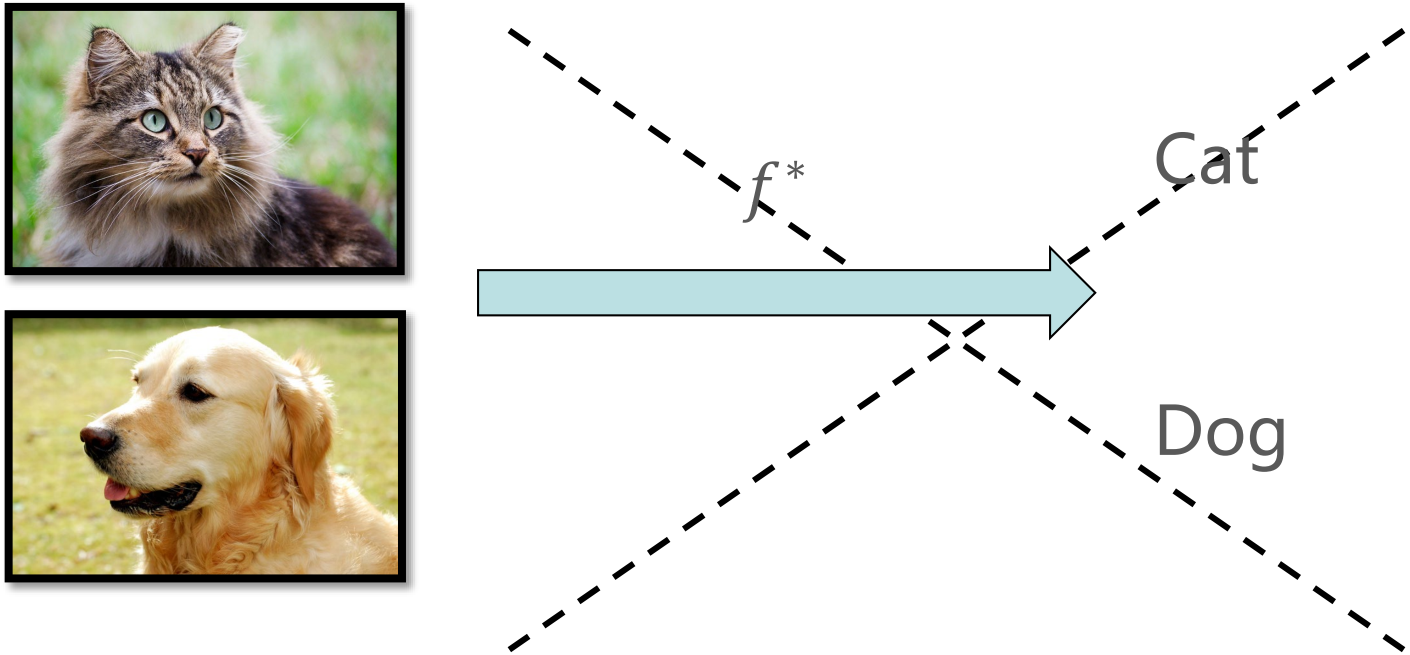
Predictive Model

Dog

Our goal: Using data, learn a predictive model that approximates $\boldsymbol{f^*}$

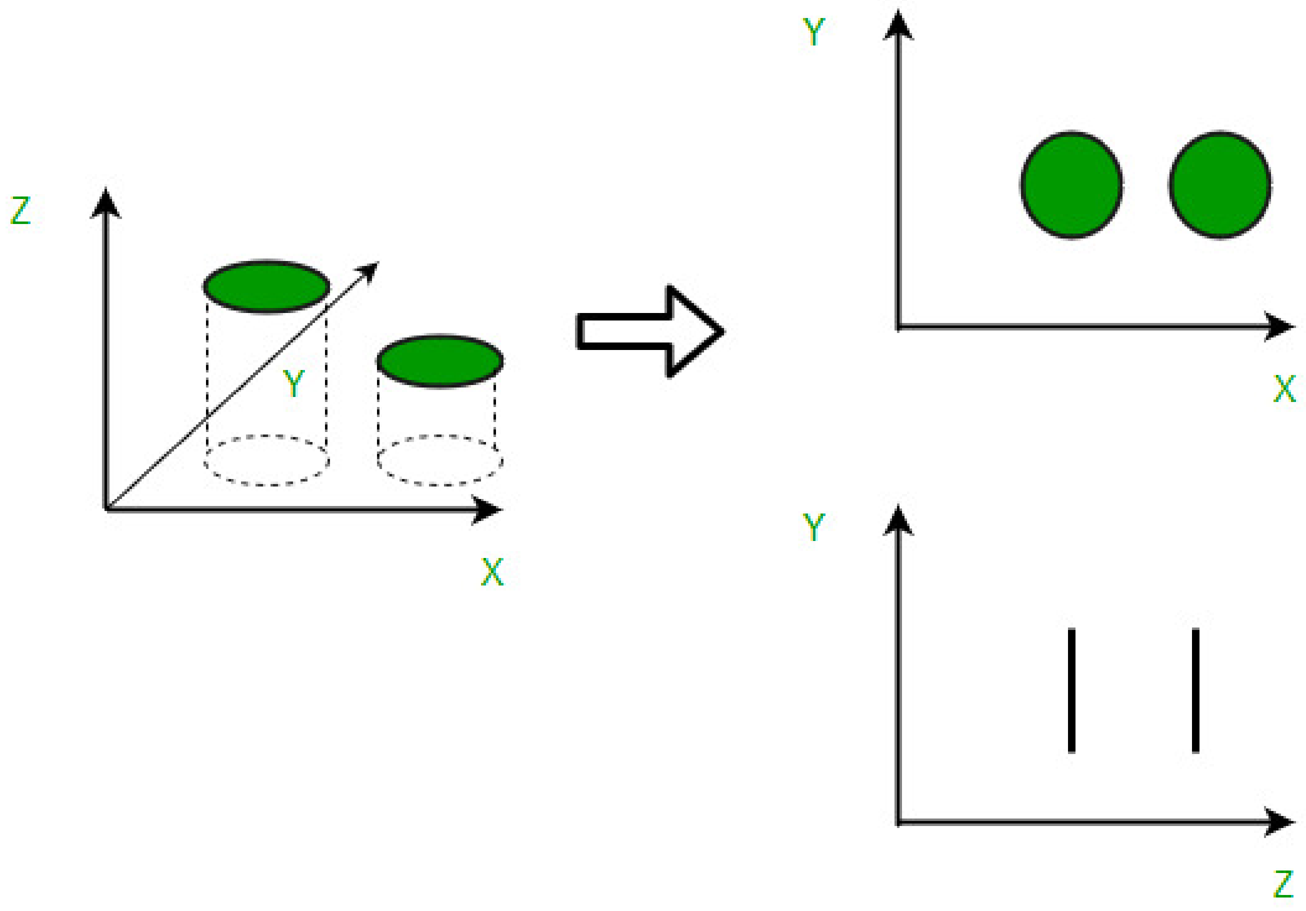
Unsupervised Learning

Unsupervised learning is for when we do not have labels



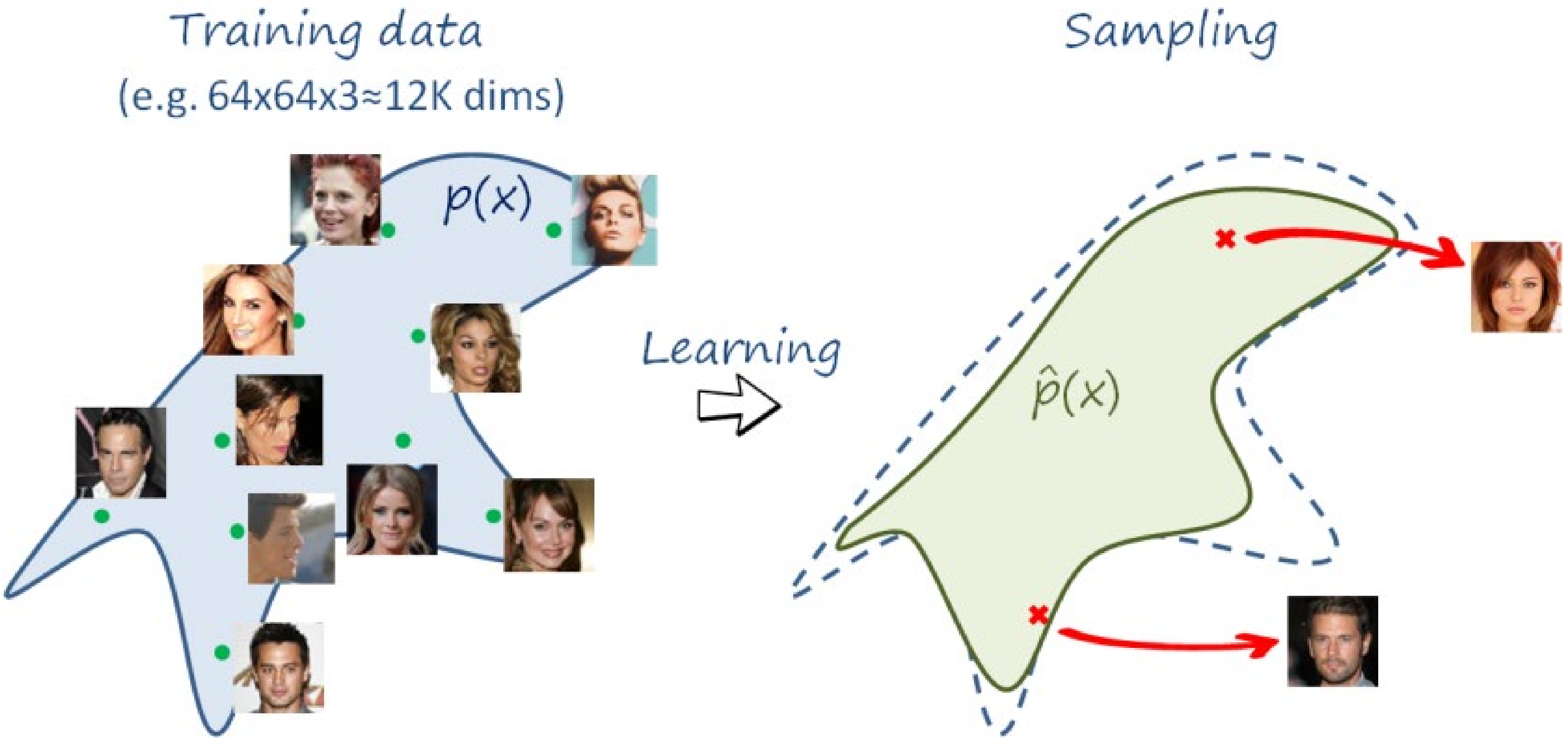
Example goal: learn some task-agnostic patterns from the inputs

Examples of Unsupervised Learning Tasks: Dimensionality Reduction



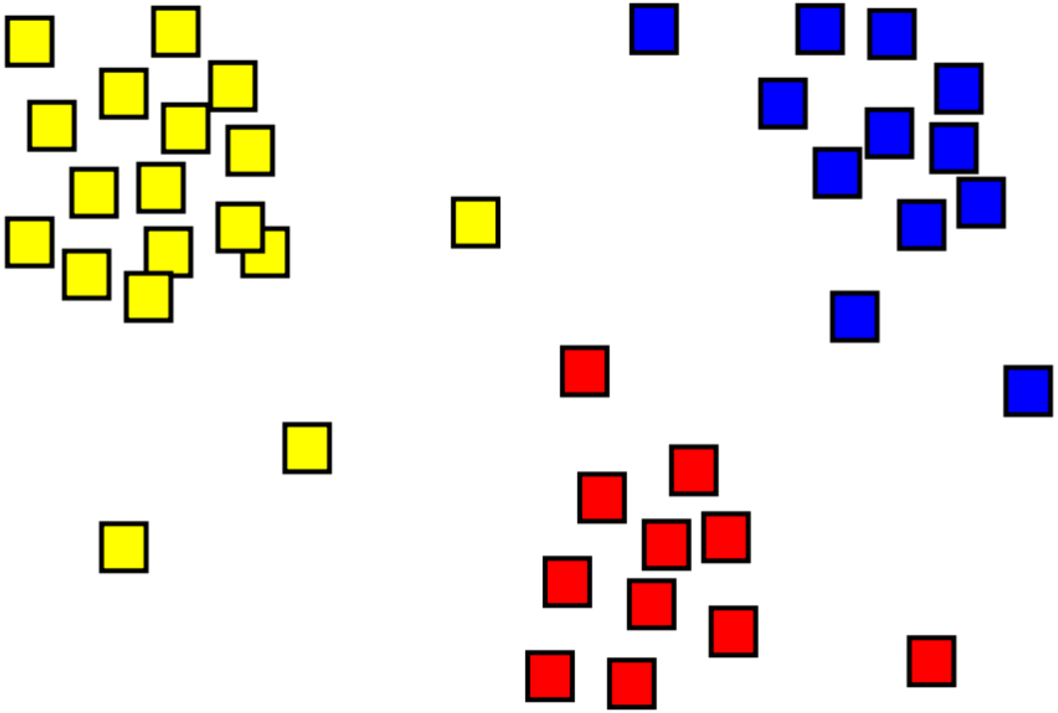
https://media.geeksforgeeks.org/wp-content/uploads/Dimensionality_Reduction_1.jpg

Examples of Unsupervised Learning Tasks: Generative Models



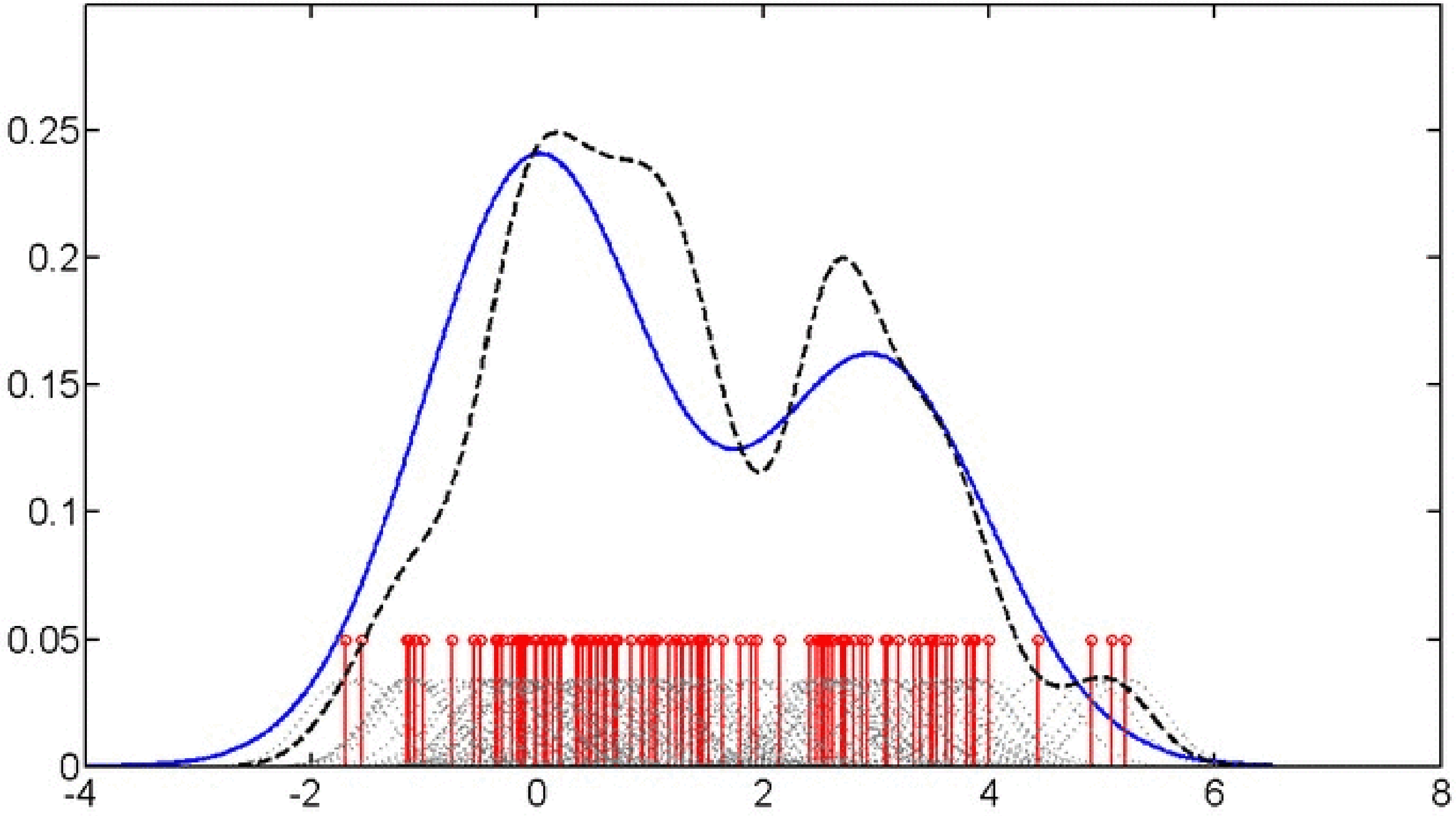
http://www.lherranz.org/wp-content/uploads/2018/07/blog_generativesampling.png

Examples of Unsupervised Learning Tasks: Clustering



https://upload.wikimedia.org/wikipedia/commons/thumb/c/c8/Cluster-2.svg/1200px-Cluster-2.svg.png

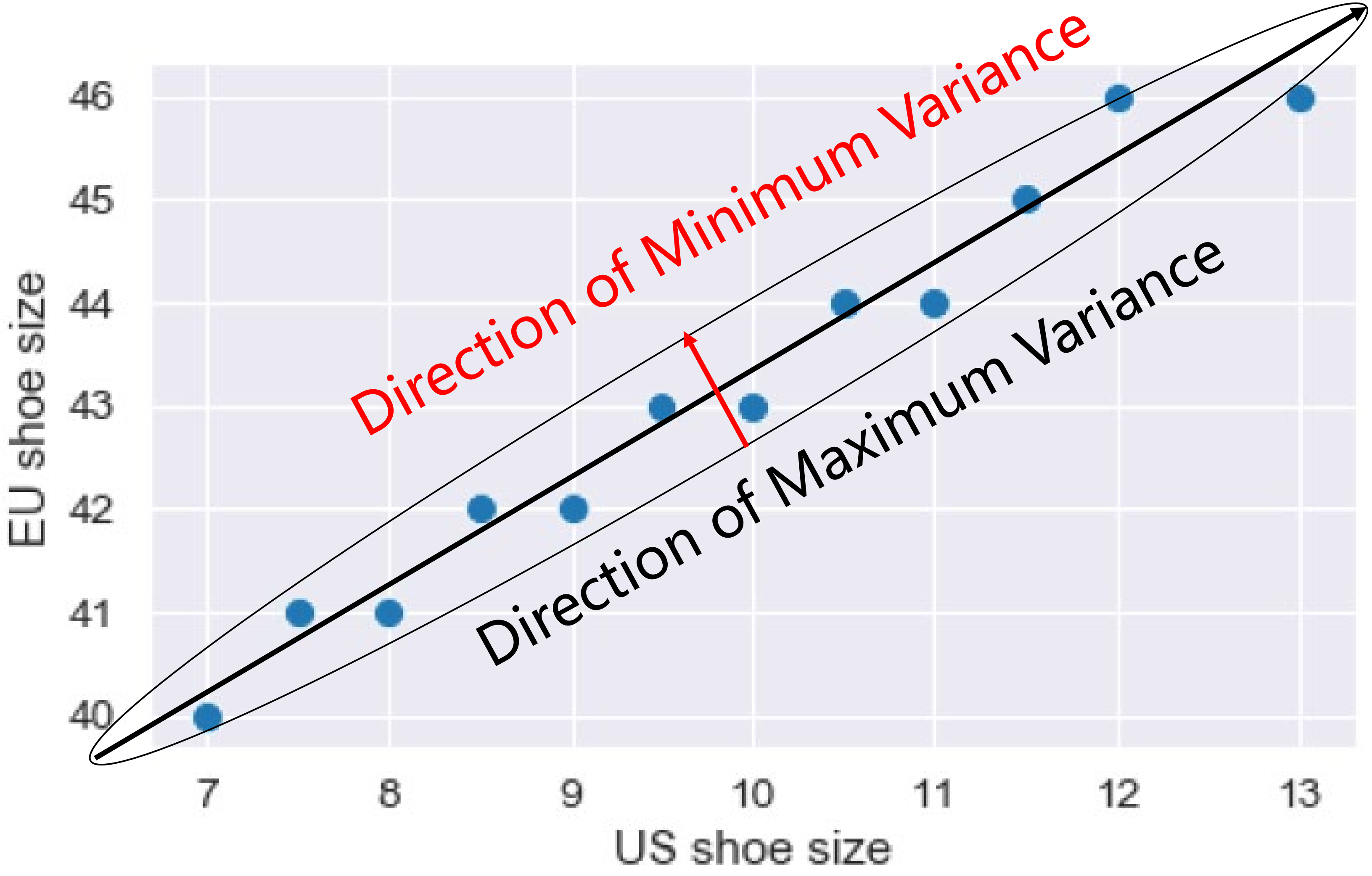
Examples of Unsupervised Learning Tasks: Density Estimation



https://commons.wikimedia.org/w/index.php?curid=24309466

Review: Principal Component Analysis

PCA: Fitting Ellipsoids to the Data



The PCA Algorithm

Data matrix: $X \in \mathbb{R}^{N \times d}$

Covariance matrix: $S = \frac{1}{N} X^T X$

Perform eigenvalue decomposition: $S = U \Lambda U^T$

We obtain:

- U's columns are principal components of the data
- $\Lambda = \mathrm{diag}\{\lambda_1, \lambda_2, \dots, \lambda_d\}$ are the explained variances
- We usually arrange $\{\lambda_i\}$ in descending order
- We can compute the PC scores

$$Z = XU$$

where we also have the reverse transformation $X = ZU^T$

PCA as a Compression Algorithm
<Lecture Notebook>

PCA is a natural way to do compression of the data.

Let us instead only take the first m principal components

$$Z_m = XU_m \quad \left(U_m \in \mathbb{R}^{d\times m}, \text{first } m \text{ eigenvectors}\right)$$

Z_m can be regarded as a m-dimensional compressed version of X

We can attempt to reconstruct X using

$$X \approx X_m = Z_m U_m^T$$

What is the error of this reconstruction?

We have

$$X_m = XU_mU_m^T$$

For the i^{th} data point, we have

$$\boldsymbol{x}^{(i)} = \sum_{j=1}^{d} (\boldsymbol{u}_j^T \boldsymbol{x}^{(i)}) \boldsymbol{u}_j \quad \text{and} \quad \boldsymbol{x}_m^{(i)} = \sum_{j=1}^{m} (\boldsymbol{u}_j^T \boldsymbol{x}^{(i)}) \boldsymbol{u}_j$$

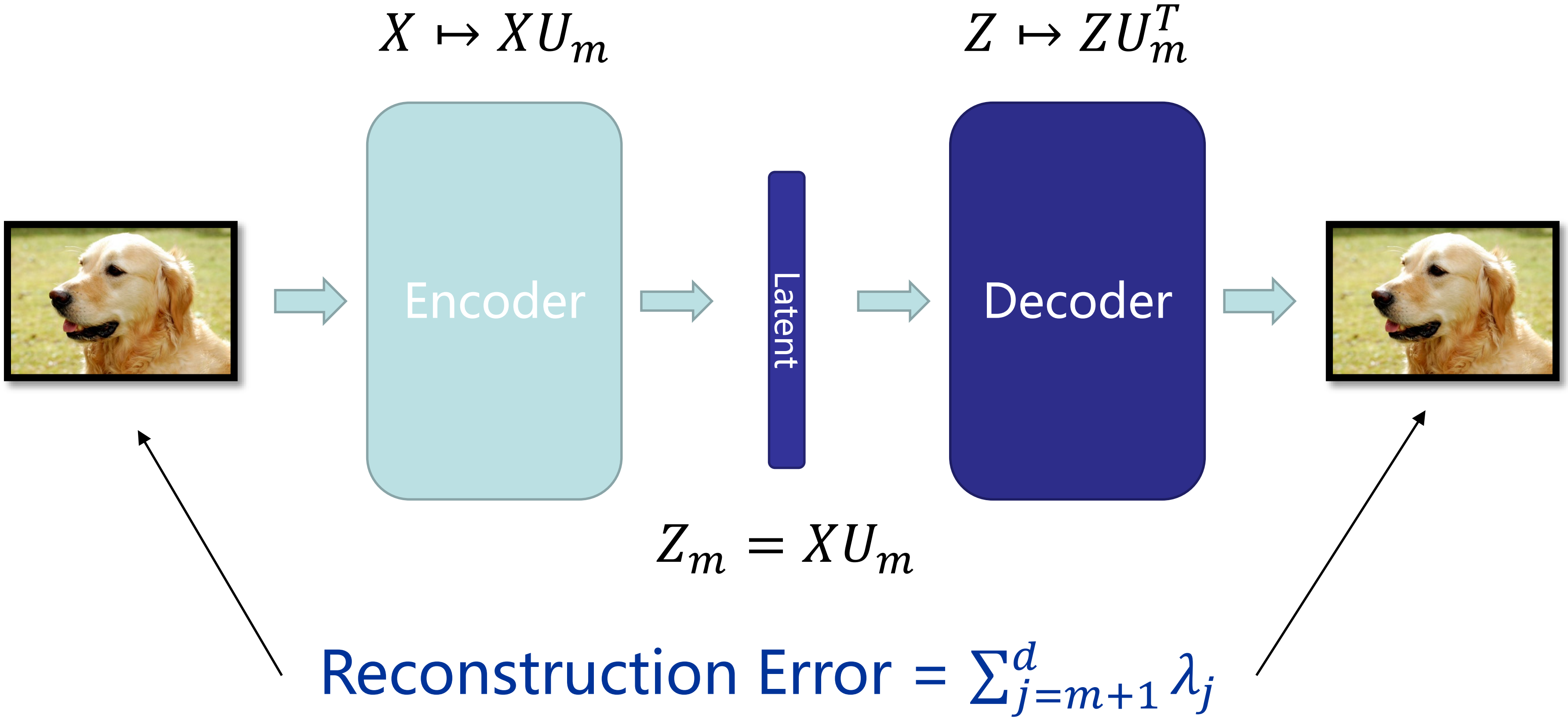
So

$$\boldsymbol{x}^{(i)} - \boldsymbol{x}_m^{(i)} = \sum_{j=m+1}^{d} (\boldsymbol{u}_j^T \boldsymbol{x}^{(i)}) \boldsymbol{u}_j$$

Giving

$$\frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{x}_m^{(i)} \right\|^2 = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=m+1}^{d} \left(\boldsymbol{u}_j^T \boldsymbol{x}^{(i)} \right)^2 = \sum_{j=m+1}^{d} \lambda_j$$

Illustration of PCA as Compression Algorithm



$X \mapsto XU_m$

$Z \mapsto ZU_m^T$

$Z_m = XU_m$

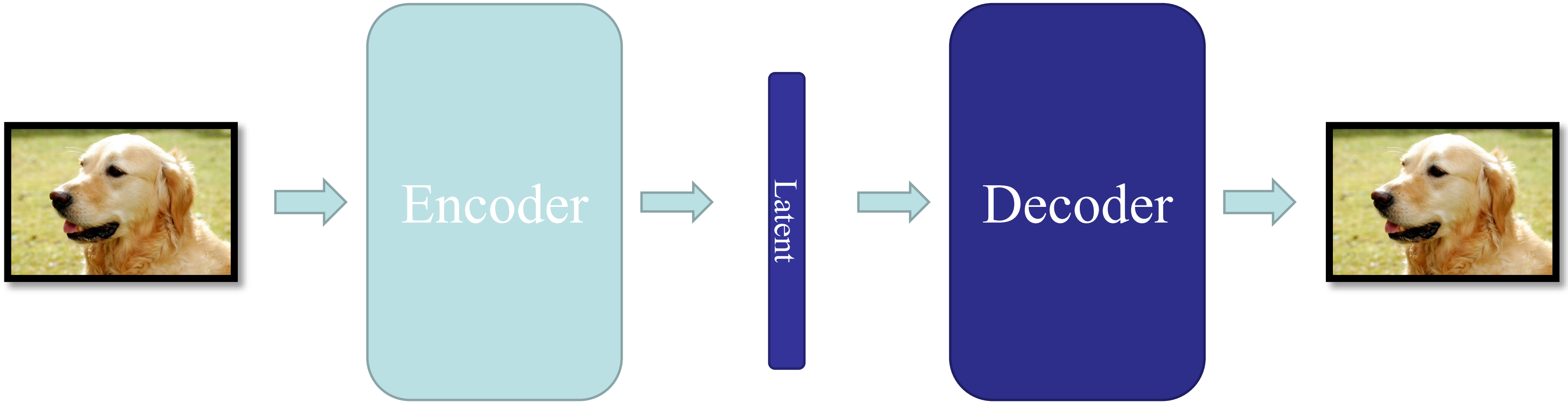
Reconstruction Error = $\sum_{j=m+1}^{d} \lambda_j$

Autoencoders

Autoencoders

In this sense, the autoencoder is a nonlinear counter-part of PCA based compression.

PCA: $\boldsymbol{z} = U_m^T \boldsymbol{x}$ $\boldsymbol{x}' = U_m \boldsymbol{z}$



AE: $\boldsymbol{z} = f(\boldsymbol{x})$ $\boldsymbol{x}' = g(\boldsymbol{z})$

Neural Network Autoencoders

Neural networks are natural candidates for the encoding function f and the decoding function g.

Suppose that

$$f: \mathbb{R}^d \to \mathbb{R}^m$$
$$g: \mathbb{R}^m \to \mathbb{R}^d$$

Then,

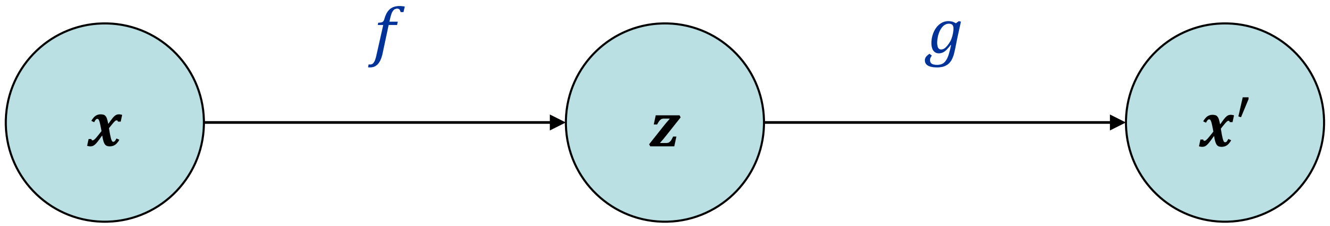
- $\mathbb{R}^m$ is called the **latent space**
- Given an input $\boldsymbol{x}$, the vector $\boldsymbol{z} = f(\boldsymbol{x})$ belonging to the latent space is called the **latent variable** or **latent state**

Learning Autoencoders

Learning autoencoders amounts to minimizing the loss

$$L(x, x') = L\left(x, g\big(f(x)\big)\right)$$

Hence, this is like a supervised learning problem, except that the label is also the input!



Undercomplete Autoencoders

The first case is when the latent dimension is small ($m \ll d$).

These are known as undercomplete autoencoders,

- Used for compression and feature extraction: $\boldsymbol{z} = f(\boldsymbol{x})$ is the compressed input/signal, which is a reduced description of the data
- If f, g are linear functions and L is the mean square loss then this is equivalent to PCA-based compression.

Regularized Autoencoders

There are cases where the autoencoder can memorize the input-output pair by just copying, e.g.

- The latent dimension m is too large
- The functions f, g are too complex

This is sometimes called overcomplete autoencoders, which can still learn some salient features if we regularize them:

- Replace $L\left(\boldsymbol{x}, g\left(f(\boldsymbol{x})\right)\right)$ by $L\left(\boldsymbol{x}, g\left(f(\boldsymbol{x})\right)\right) + \Omega(\boldsymbol{x}, f, g)$
- Example: sparse autoencoders uses the L^1 regularizer on the latent variables $\Omega(\boldsymbol{x}, f, g) = \alpha \|f(\boldsymbol{x})\|_1$
- Example: contractive autoencoders uses the regularizer $\Omega(x, f, g) = \alpha \|\nabla f(\boldsymbol{x})\|_2^2$

Denoising Autoencoders

Other than compression, we can also use autoencoders to perform denoising.

In this case, instead of minimizing

$$L\left(\boldsymbol{x}, g\left(f\left(\boldsymbol{x}\right)\right)\right)$$

during training, we minimize

$$L\left(\boldsymbol{x}, g\left(f\left(\widetilde{\boldsymbol{x}}\right)\right)\right)$$

where $\widetilde{x}$ **is a noise-perturbed version of** x**.**

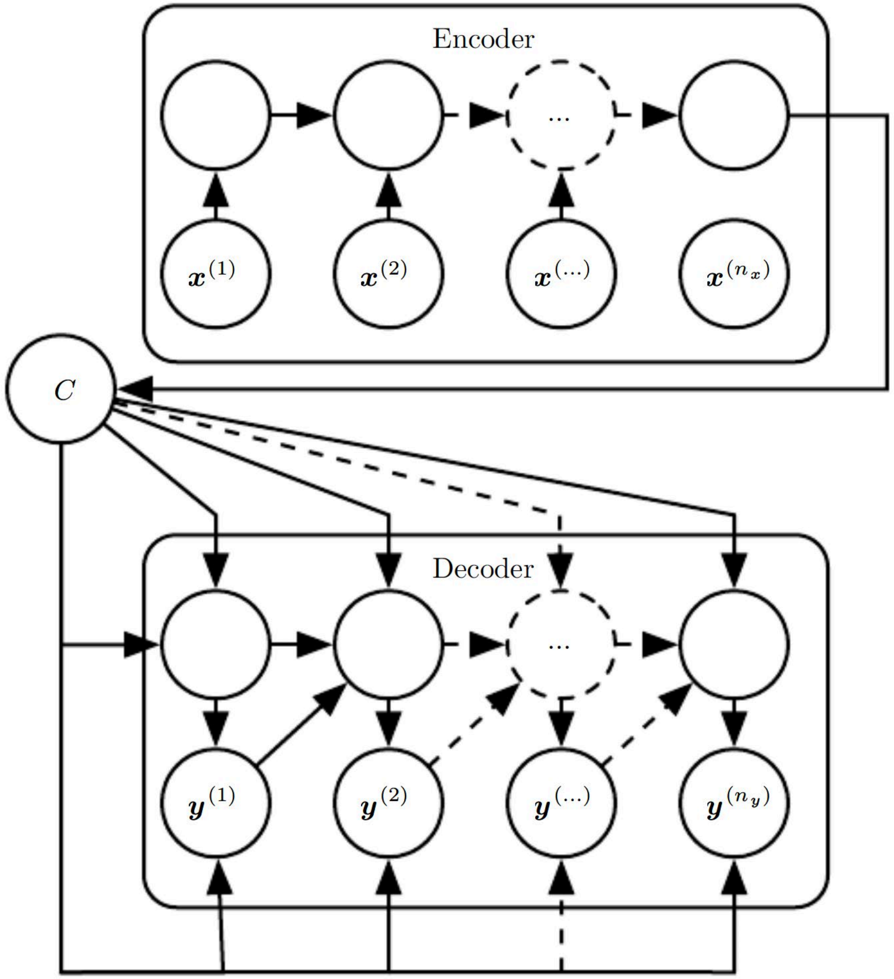
Once trained to minimize error, it can then be used to remove noise in the inputs

$$\boldsymbol{x} \approx g\left(f\left(\widetilde{\boldsymbol{x}}\right)\right)$$

Seq-to-Seq (Auto-)encoders

Note that autoencoders can also be applied to sequence models.

In this case, both the encoder f and the decoder g are recurrent networks. This can even handle variable input lengths!



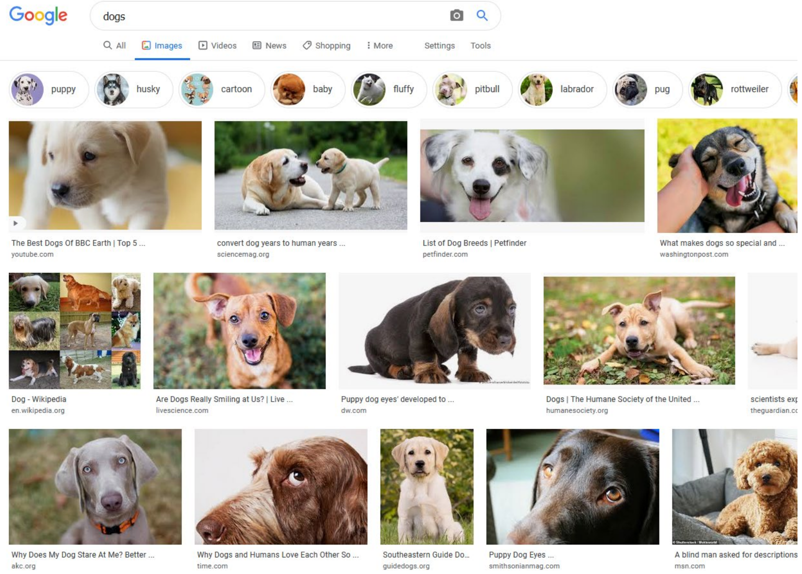
Demo: Autoencoders for Compression and Denoising

Semi-supervised Learning

Semi-supervised Learning

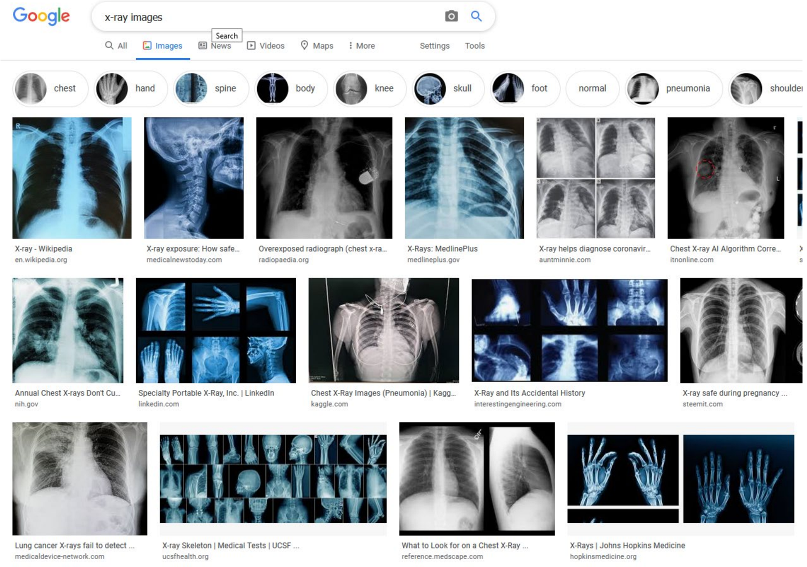
In practice, we often have way more unlabeled data than labelled data

- Unlabeled data can be collected without any domain knowledge
- Labeled data often requires human labeling



dogs

Google
All
Images
Videos
News
Shopping
More
Settings
Tools
puppy
husky
cartoon
baby
fluffy
pitbull
labrador
pug
rottweiler
The Best Dogs Of BBC Earth | Top 5 ...
youtube.com
convert dog years to human years ...
sciencemag.org
List of Dog Breeds | Petfinder
petfinder.com
What makes dogs so special and ...
washingtonpost.com
Dog - Wikipedia
en.wikipedia.org
Are Dogs Really Smiling at Us? | Live ...
livescience.com
Puppy dog eyes' developed to ...
dw.com
Dogs | The Humane Society of the United ...
humanesociety.org
scientists exp
theguardian.co
Why Does My Dog Stare At Me? Better ...
akc.org
Why Dogs and Humans Love Each Other So ...
time.com
Southeastern Guide Do...
guidedogs.org
Puppy Dog Eyes ...
smithsonianmag.com
A blind man asked for descriptions
msn.com



x-ray images

Google
Search
All
Images
News
Videos
Maps
More
Settings
Tools
chest
hand
spine
body
knee
skull
foot
normal
pneumonia
shoulder
X-ray - Wikipedia
en.wikipedia.org
X-ray exposure: How safe...
medicalnewstoday.com
Overexposed radiograph (chest x-ra...
radiopaedia.org
X-Rays: MedlinePlus
medlineplus.gov
X-ray helps diagnose coronavir...
auntminnie.com
Chest X-ray AI Algorithm Corre...
itnonline.com
Annual Chest X-rays Don't Cu...
nih.gov
Specialty Portable X-Ray, Inc. | LinkedIn
linkedin.com
Chest X-Ray Images (Pneumonia) | Kagg...
kaggle.com
X-Ray and Its Accidental History
interestingengineering.com
X-ray safe during pregnancy ...
steemit.com
Lung cancer X-rays fail to detect ...
medicaldevice-network.com
X-ray Skeleton | Medical Tests | UCSF ...
ucsfhealth.org
What to Look for on a Chest X-Ray ...
reference.medscape.com
X-Rays | Johns Hopkins Medicine
hopkinsmedicine.org

Data for Semi-supervised Learning

In supervised learning, we consider a dataset consisting of inputs and labels

$$\mathcal{D} = \left\{\boldsymbol{x}^{(i)}, y^{(i)} : i = 1, \dots, N\right\}$$

In semi-supervised learning, we consider two datasets

$$\mathcal{D}_L = \left\{\boldsymbol{x}^{(i)}, y^{(i)} : i = 1, \dots, N_L\right\}$$

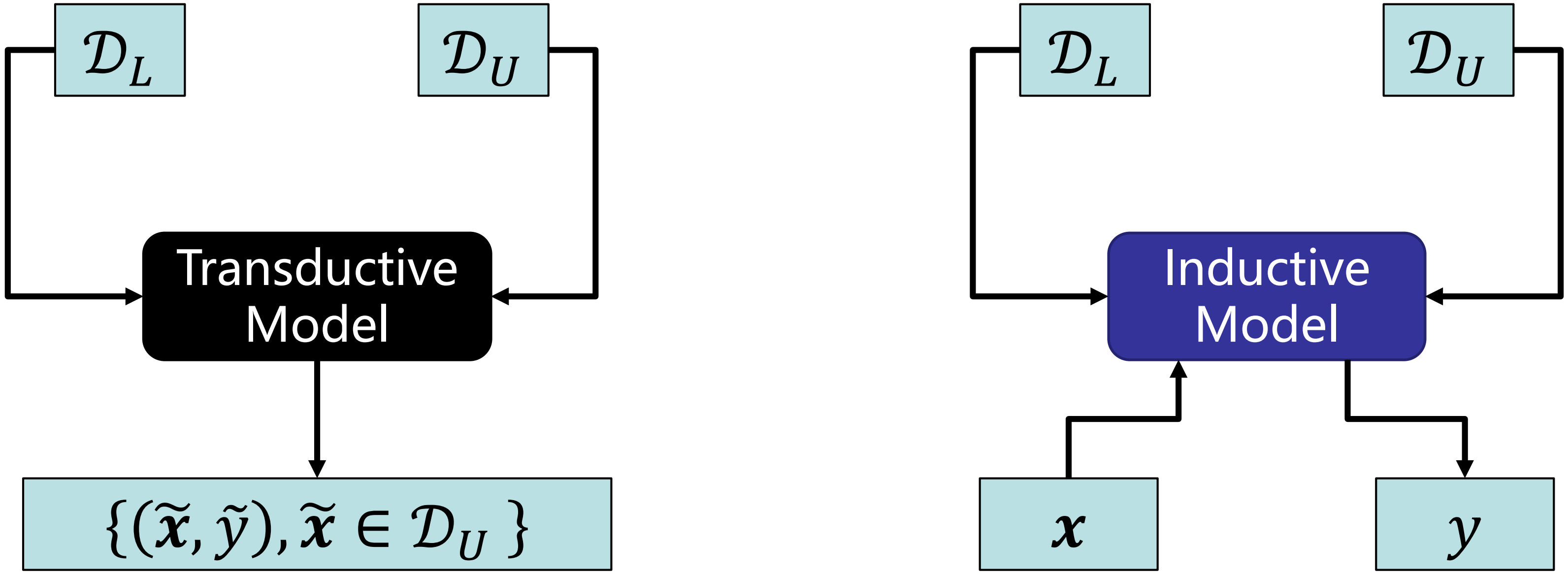
$$\mathcal{D}_U = \left\{\widetilde{\boldsymbol{x}}^{(i)} : i = 1, \dots, N_U\right\}$$

$\mathcal{D}_L$ **is a labeled dataset and** $\mathcal{D}_U$ **is an unlabeled dataset.**

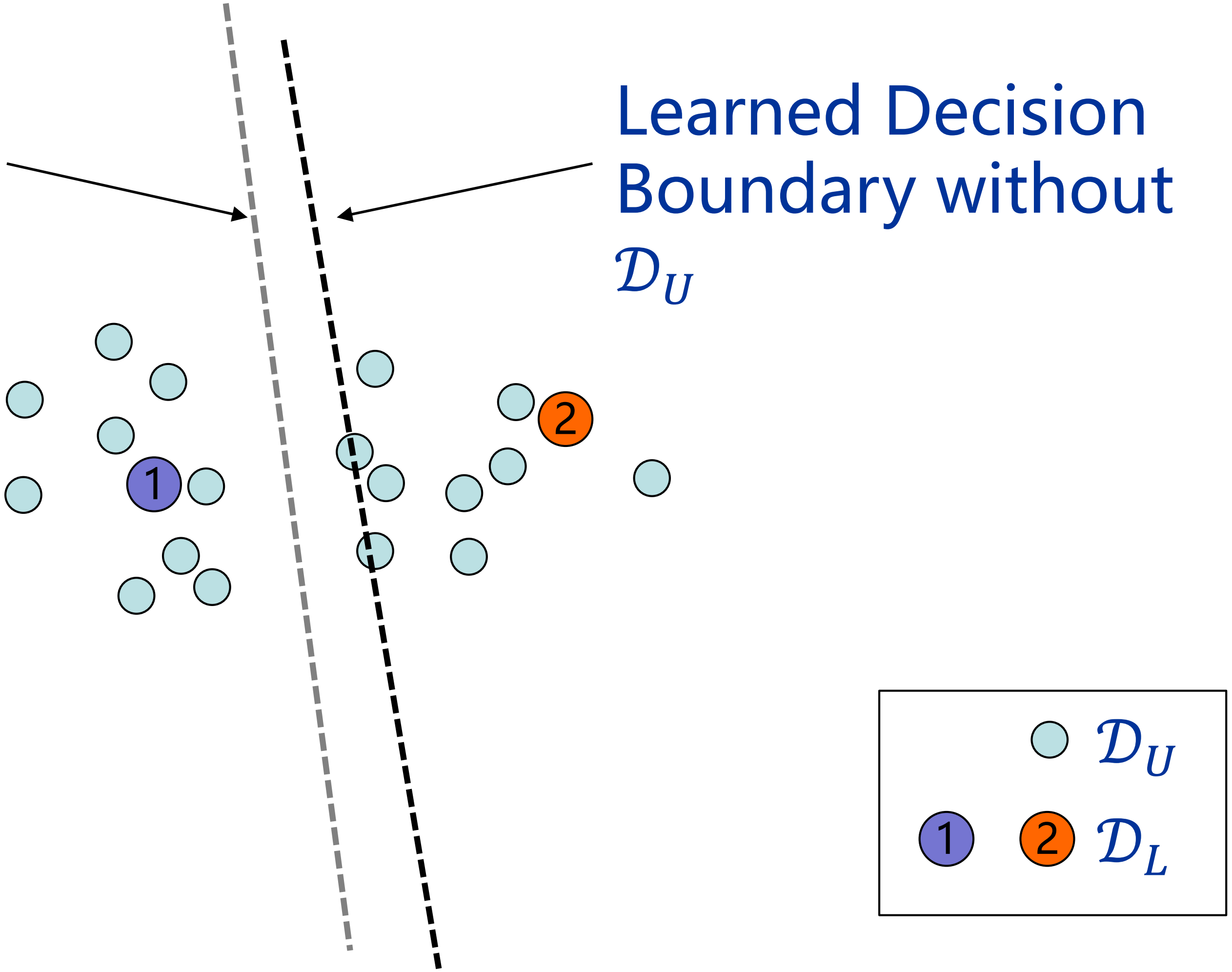
We often have $N_U \gg N_L$

Two Types of Semi-Supervised Problems

- **Transductive: the goal is to label the unlabeled dataset D_U**
- **Inductive: the goal is to learn f^* so as to predict on *unseen* input samples**



How can unlabeled data help?



Learned Decision Boundary with $\mathcal{D}_U$

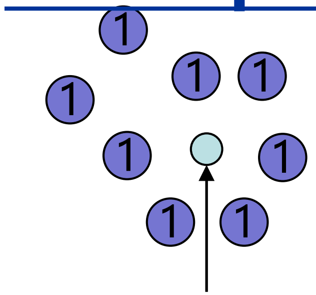
Learned Decision Boundary without $\mathcal{D}_U$

The Underlying Assumptions for Semi-supervised Learning

The reason unlabeled data can help is based on at least one of the following assumptions

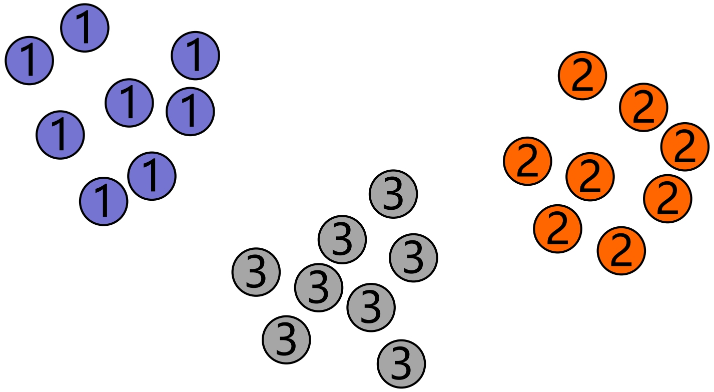
- Continuity assumption: *Input data close to one another are likely to have the same/similar label*
- Cluster assumption: *Input data form discrete clusters, and nearby clusters share the same/similar label*
- Manifold assumption: *Input data lie approximately on a low-dimensional manifold*

Continuity Assumption

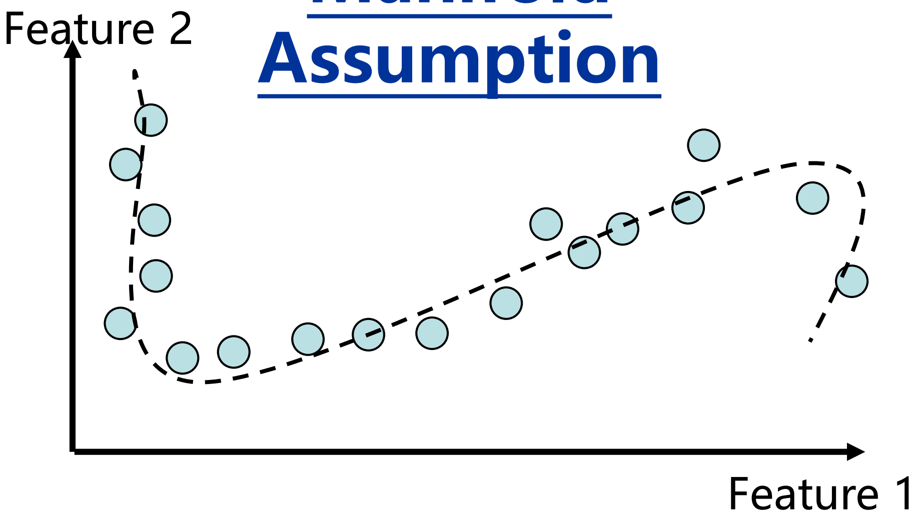


Likely to be class 1

Cluster Assumption



Manifold Assumption



Simplest Heuristic: Self Training

Main idea: use partially trained models and predict labels, which are then used to train the model recursively

Algorithm (simplest form):

1. Train $\hat{f}$ using $\mathcal{D}_L$
2. Take $\tilde{\boldsymbol{x}} \in \mathcal{D}_U$, predict $y = \hat{f}(\tilde{\boldsymbol{x}})$
3. Add $(\tilde{\boldsymbol{x}}, y)$ to $\mathcal{D}_L$ and go back to 1

Variations in Self Training

We can vary steps 2-3. Instead of adding one data-point at a time, we can

- Add a few most confident $\left\{\left(\tilde{\boldsymbol{x}}^{(i)}, \hat{f}\left(\tilde{\boldsymbol{x}}^{(i)}\right)\right)\right\}$ to $\mathcal{D}_L$
- Add all $\left\{\left(\tilde{\boldsymbol{x}}^{(i)}, \hat{f}\left(\tilde{\boldsymbol{x}}^{(i)}\right)\right)\right\}$ to $\mathcal{D}_L$
- Add all $\left\{\left(\tilde{\boldsymbol{x}}^{(i)}, \hat{f}\left(\tilde{\boldsymbol{x}}^{(i)}\right)\right)\right\}$ to $\mathcal{D}_L$ but keep a weightage of the sample points by confidence

Why would self-training work at all?

Example: Self Training for a Simple Classifier <Lecture Notebook>

Let $d = 1$, $f^*(x) = \mathbb{I}_{x>0}$

We are given training data

$$\mathcal{D}_L = \{(-1,0), (2,1)\}$$
$$\mathcal{D}_U = \{-1, 1\}$$

Suppose we fit a SVM model (linear NN with hinge loss), then we obtain $\hat{f}(x) = \mathbb{I}_{x>\frac{1}{2}}$ (maximum margin solution)

If we then use self training (all data in $\mathcal{D}_U$), we obtain augmented dataset

$$\mathcal{D}_L = \{(-1,0), (2,1), (-1,0), (1,1)\}$$

for which the maximum margin solution is $\hat{f}(x) = \mathbb{I}_{x>0} = f^*(x)$

The Pros and Cons of Self-Training

Advantages:

- Simple to implement
- A general, meta-algorithm

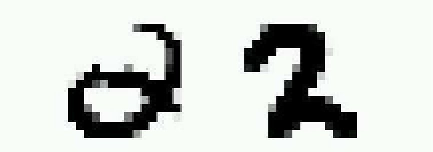
Disadvantages:

- Early mistakes could mislead the classifier and reinforce itself
- Little can be said theoretically (so far!)

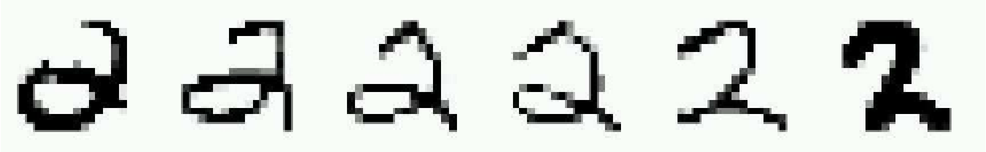
Label Propagation

Another commonly used method for semi-supervised learning is label propagation

The primary idea is the clustering and manifold assumption



not similar



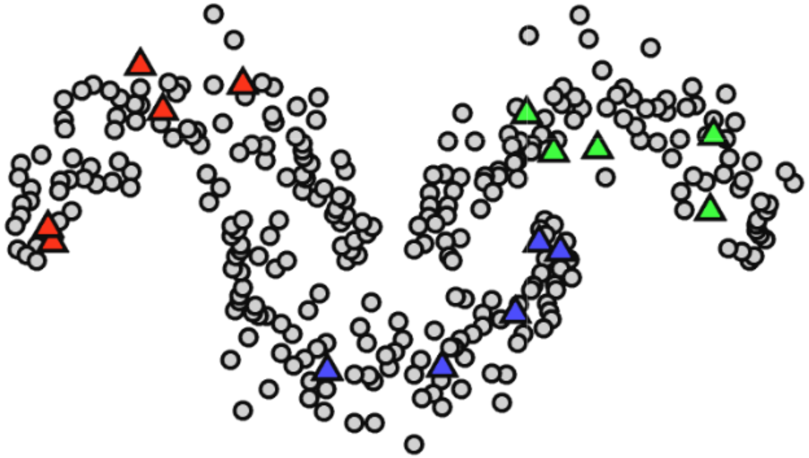
‘indirectly’ similar
with stepping stones

The Basic Label Propagation Algorithm

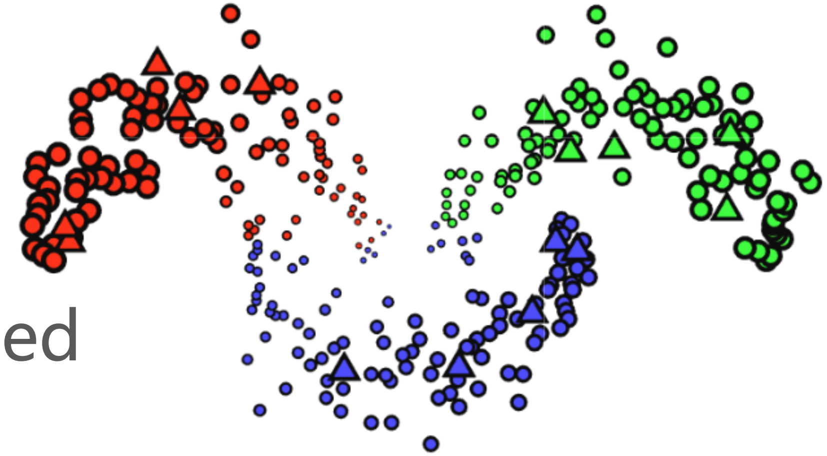
There are many variations of the label propagation algorithm, but the most basic can be described as follows:

1. Represent the samples on a graph (edges represent closeness)
2. In a random order, assign to each unlabeled sample the majority label of its neighbors
3. If every node has a label that is equal to the majority label of its neighbors, stop. Otherwise, repeat 2 with a different random order

Visualization of Label Propagation



Triangles are labeled data
Colors are classes
Circles are unlabeled data



Labels are propagated to unlabeled samples
Size is the confidence

https://arxiv.org/pdf/1904.04717.pdf

Learning Across Tasks

Multi-Task Learning

Multi-task learning is concerned with learning to perform more than one task at a time, or learning common representations from different tasks.

In particular, the tasks can be supervised or unsupervised, or mixed.

We will discuss two very useful approaches:

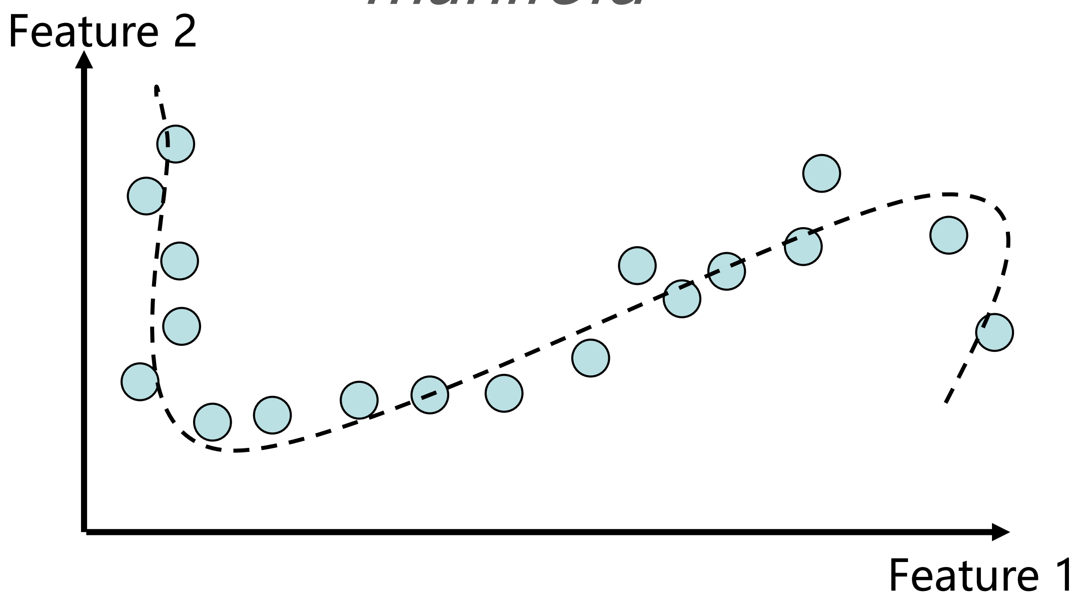
- Transfer learning
- Domain adaptation

The Manifold Assumption

For learning across tasks, the fundamental assumption is some similarity in the features that are relevant for different tasks.

In particular, the manifold assumption as at play:

Input data lie approximately on a low-dimensional manifold



Transfer Learning

Transfer learning concerns the case where data across tasks are similar, but the tasks may be very different.

Key idea: since the data is same/similar, different tasks may depend on similar features extracted from the data!

This leads to many useful practical approaches

- Sharing hidden states
- Warm-start and fine-tuning with pre-trained networks
- Unsupervised pre-training (e.g. using autoencoders)

Sharing Hidden States

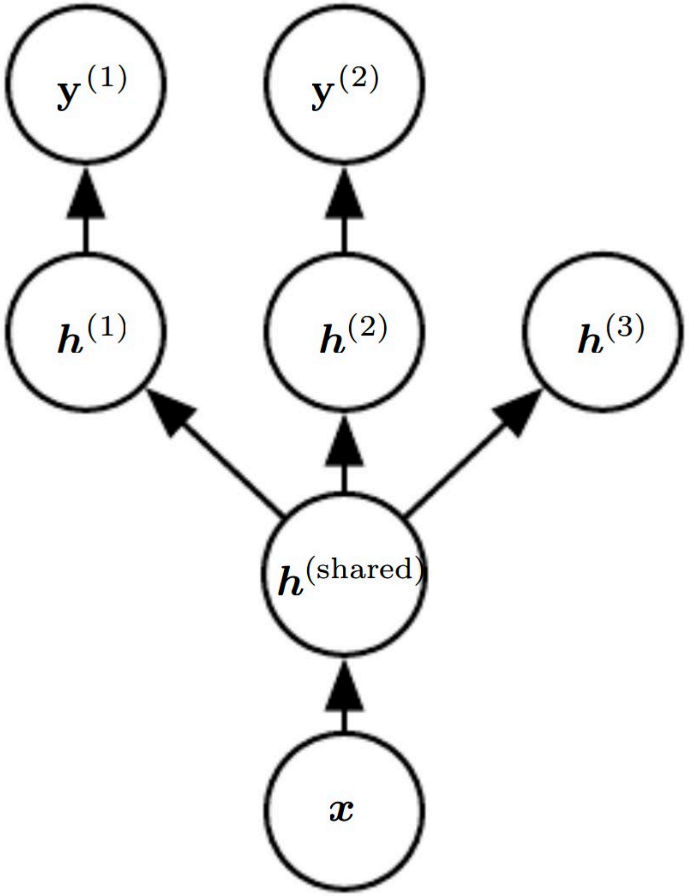
For the two tasks

$y^{(1)}, y^{(2)}$

we can train a common deep network sharing hidden states

$\boldsymbol{h}^{(\text{shared})}$

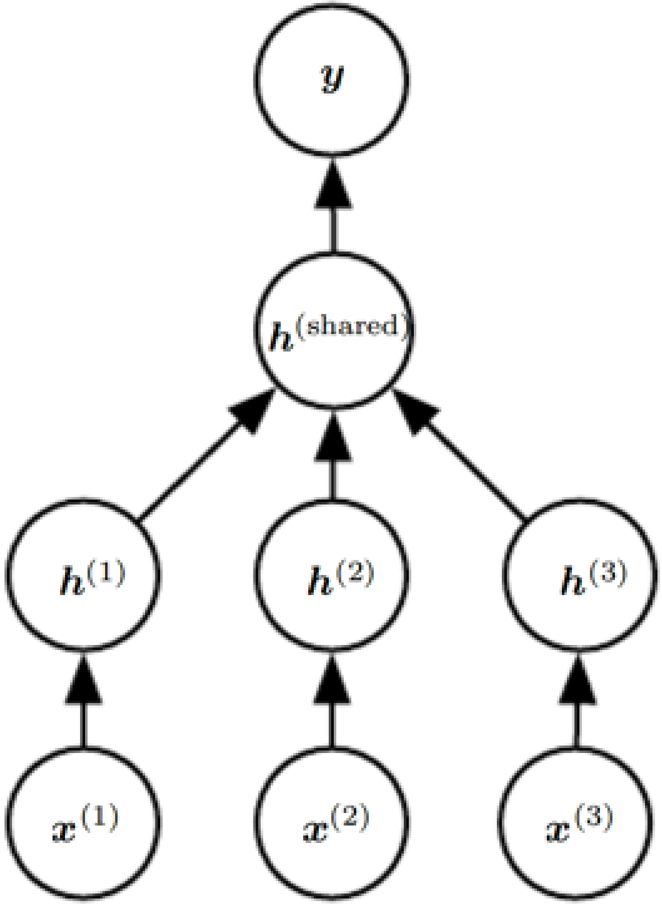
This allows the training of both tasks to affect the representation we learn.



Multiple Inputs with Common Outputs

We also encounter the reverse situations: multiple input data types for a common output task.

The same feature extraction effect applies.



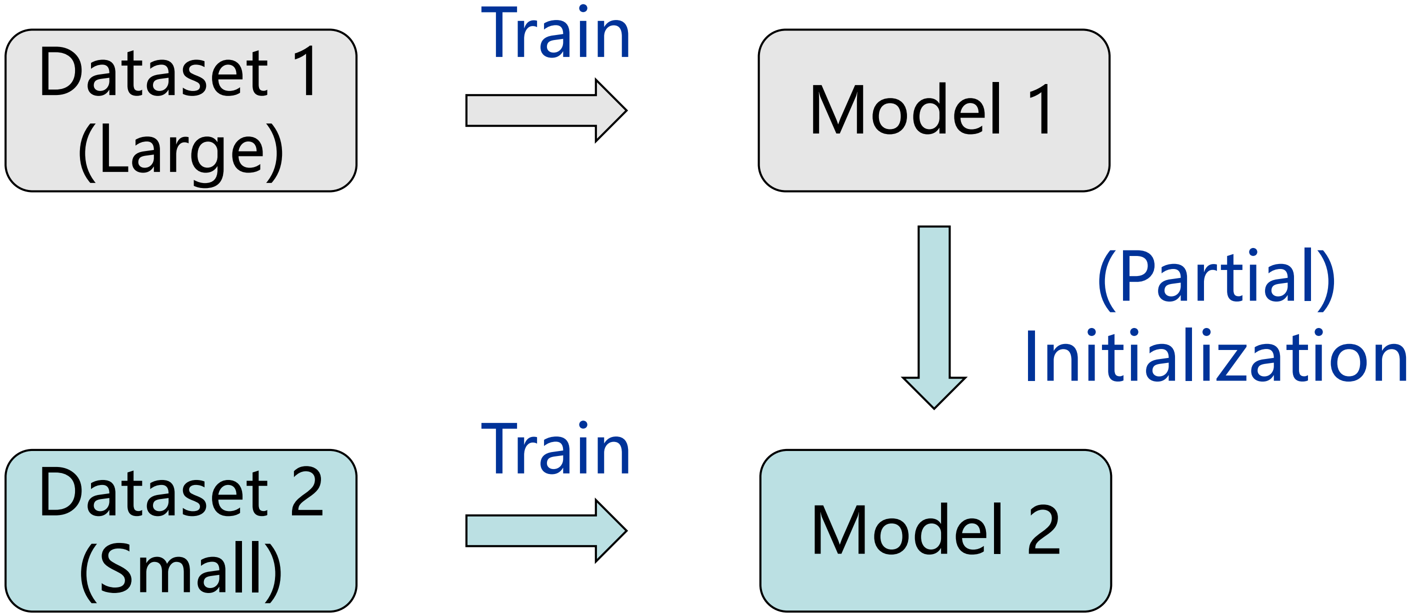
Using Pre-trained Networks

Using pre-trained networks is a practically easy way to perform transfer learning across different tasks/inputs/outputs.

The idea is that we can use a network trained on task 1 to “warm-start” the problem for task 2.

If the inputs/outputs are of different semantics, then some modifications of the networks, e.g. in the input or output layers, are necessary.

Pre-Training



Dataset 1 (Large)

Train

Model 1

(Partial) Initialization

Dataset 2 (Small)

Train

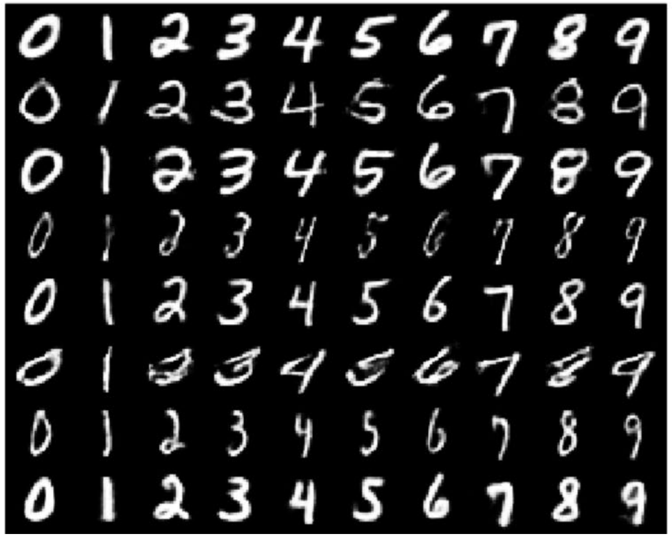
Model 2

Domain Adaptation

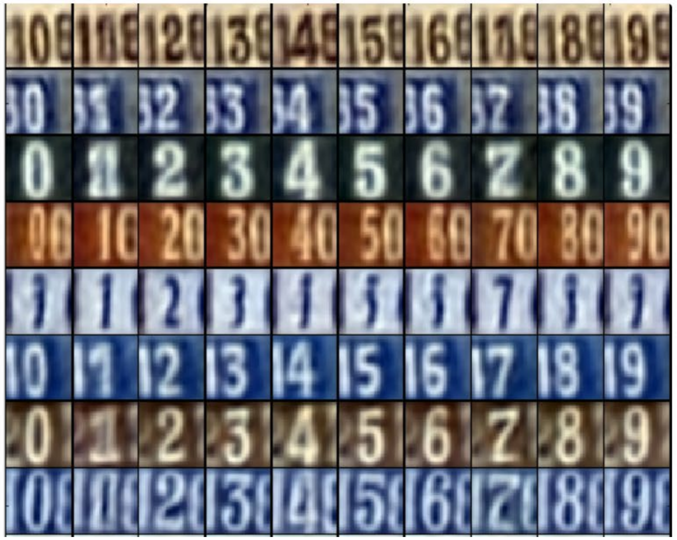
Domain adaptation is slightly different.

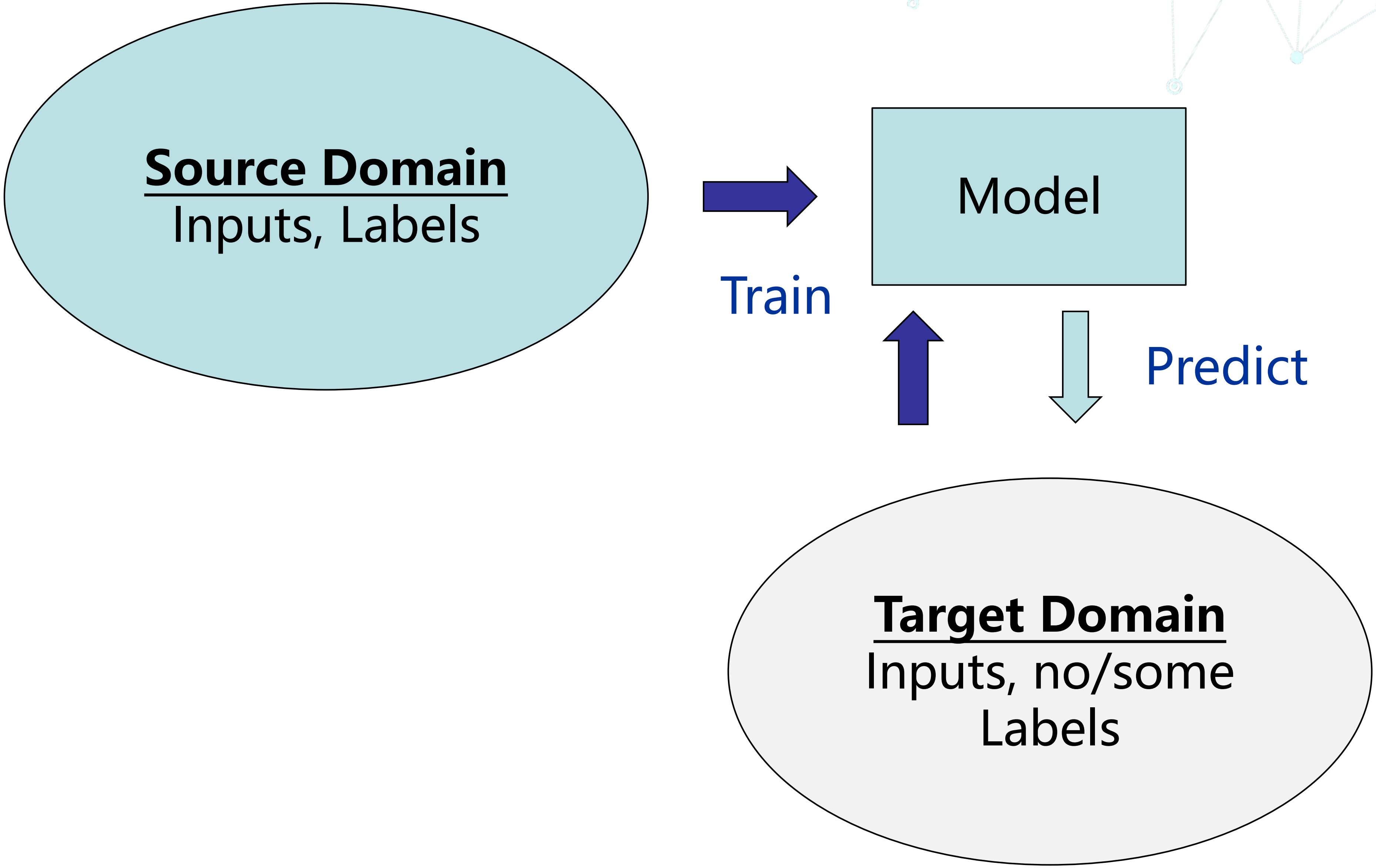
Here, our tasks P_1 and P_2 are identical, but the input data is of different distributions.

MNIST



SVHN





Source Domain

Inputs, Labels

Train

Model

Predict

Target Domain

Inputs, no/some Labels

Domain Adaptation Algorithms

There are many domain adaptation algorithms, some more similar to usual semi-supervised learning than others

Examples

- Label propagation using small set of labelled samples in target domain
- Search for a common representation space for both domains, e.g. through autoencoders, and/or adversarial methods
- Learn transport maps to match distributions

Demo: Transfer Learning

Summary

We have introduced applications of deep learning to cases with little or no labelled data

- Autoencoders for compression and feature learning
- Semi-supervised learning via self-learning and label propagation
- Transfer learning and domain adaptation

The key reason these work is that they perform some form of representation learning!