Deep Unsupervised Learning

6 March 2019

Unsupervised Learning

- Capture rich patterns in raw data with deep networks in a label-free way
  - Generative Models: recreate raw data distribution
  - Self-supervised learning: tasks that require semantic understanding
- Applications
  - Generate novel data
  - Compression
  - Improve downstream tasks
  - Flexible building blocks

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x → y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

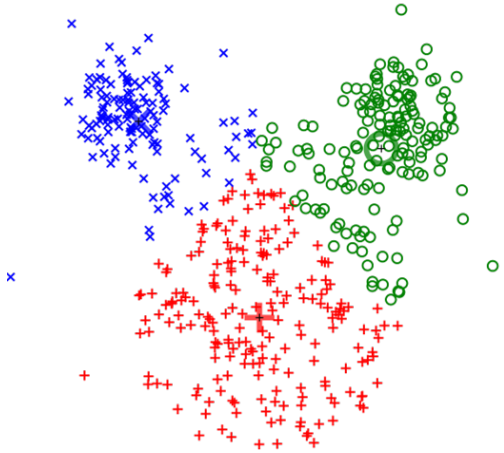
Data: x

Just data, no labels!

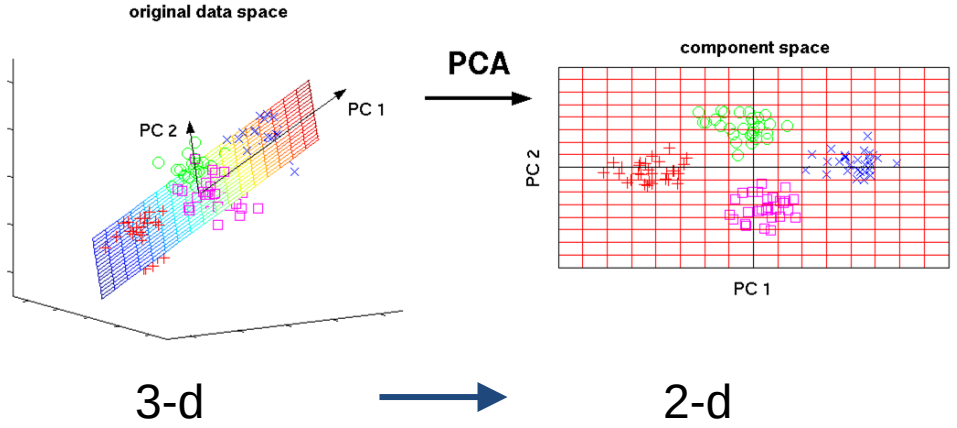
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Unsupervised Learning



K-means clustering

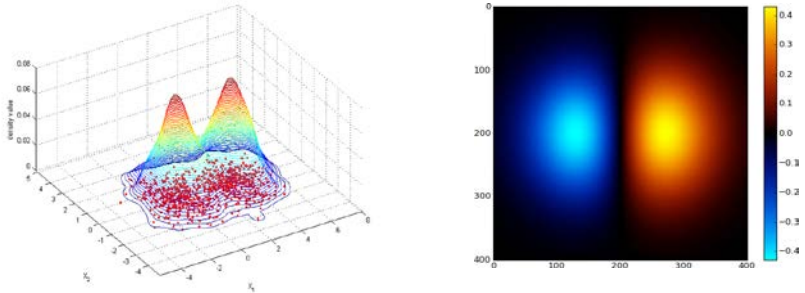


Principal Component Analysis
(Dimensionality reduction)

Unsupervised Learning

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

- Density estimation: estimating the underlying distribution of the data.
- Example 1: have points in 1-d, and we can try to fit a gaussian to the density
- Example 2: 2d data. model the density to be higher where more points are concentrated

2-d density images left and right are CC0 public domain

Generative Models

Given training data, generate new samples from same distribution

Training data ~ $p_{data}(x)$ Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Addresses density estimation.

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

Likelihood Models I: Autoregressive Models

Likelihood-based models

- How do we solve these problems?
  - Generating data: synthesizing images, videos, speech, text
  - Compressing data: constructing efficient codes
  - Anomaly detection
- **Likelihood-based models: estimate p_{data} from samples**
  $x^{(1)}, \ldots, x^{(n)} \backsim p_{data}(x)$
  Learns a distribution p that allows:
  - Computing p(x) for arbitrary x
  - Sampling x $\backsim$ p(x)
- We first assume: **discrete** data

- We want to estimate distributions of **complex, high-dimensional data**
  - A 128x128x3 image lies in a ~50,000-dimensional space
- We also want computational and statistical efficiency
  - Efficient training and model representation
  - Expressiveness and generalization
  - Sampling quality and speed
  - Compression rate and speed

Estimating frequencies by counting

- Goal: **Estimate p_{data} from samples $x^{(1)}, \ldots, x^{(n)} \backsim p_{data}(x)$**
- Suppose the samples take on values in a finite set {1, …, k}
- The model: a **histogram**
  - (Redundantly) described by k nonnegative numbers: p1, …, pk
  - To train this model: count frequencies
- p_i = (# times i appears in the dataset) / (# points in the dataset)

At runtime

- **Inference** (querying pi for arbitrary i): simply a lookup into the array p1, …, pk
- **Sampling** (lookup into the inverse cumulative distribution function)
  1. From the model probabilities p1, …, pk, compute the cumulative distribution
     $F_i = p_1 + \cdots + p_i$ for all $i \in \{1, \ldots, k\}$
  2. Draw a uniform random number $u \sim [0, 1]$
  3. Return the smallest i such that $u \leq F_i$
- **Are we done?**

Failure in high dimensions

- No, because of the **curse of dimensionality**. Counting fails when there are too many bins.
  - (Binary) MNIST: 28x28 images, each pixel in {0, 1}
  - There are $2784 \approx 10236$ probabilities to estimate
  - Any reasonable training set covers only a tiny fraction of this
  - Each image influences only one parameter. No generalization whatsoever!
- **Solution: function approximation**. Instead of storing each probability, store a parameterized function $p_\theta(x)$

Likelihood-based generative models

- Recall: goal: to **estimate p_{data} from $x^{(1)}, \ldots, x^{(n)} \backsim p_{data}(x)$**
- We introduce **function approximation**: learn θ so that $p_\theta(x) \approx p_{data}(x)$.
  - How do we design function approximators to effectively represent complex joint distributions over x, yet remain easy to train?
  - There will be many choices for model design, each with different tradeoffs and different compatibility criteria.
- **Designing the model and the training procedure go hand-in-hand.**

Fitting distributions

- Given data $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$ sampled from a "true" distribution $\boldsymbol{p_{data}}(\boldsymbol{x})$
  - Set up a model class: a set of parameterized distributions p_θ
  - Pose a search problem over parameters

$$\underset{\theta}{argmin}\ loss(\theta, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)})$$

  - Want the loss function + search procedure to:
    - Work with large datasets
    - Yield θ such that p_θ matches p_{data}
    - Note that the training procedure can only see the empirical data distribution, not the true data distribution: we want the model to generalize.

Maximum likelihood

- ML: given a dataset $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$, find θ by solving the optimization problem:

$$\underset{\theta}{argmin}\ loss\left(\theta, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\right) = \frac{1}{n}\sum_{i=1}^{n} -\log p_\theta(x^{(i)})$$

- If the model family is expressive enough and if enough data is given, then solving the ML problem will yield parameters that generate the data
- Equivalent to minimizing KL divergence between the empirical data distribution and the model

$$\hat{p}_{data}(x) = \frac{1}{n}\sum_{i=1}^{n} 1[x = x^{(i)}]$$

$$KL(\hat{p}_{data}||p_\theta) = \mathbb{E}_{x\sim\hat{p}_{data}}[-\log p_\theta(x)] - H(\hat{p}_{data})$$

Stochastic gradient descent

- Maximum likelihood is an optimization problem.
- **Stochastic gradient descent** (SGD).
  - SGD minimizes expectations: for f a differentiable function of θ, it solves

$$\underset{\theta}{argmin}\, \mathbb{E}[f(\theta)]$$

  - With maximum likelihood, the optimization problem is

$$\underset{\theta}{argmin}\, \mathbb{E}_{x\sim\hat{p}_{data}}[-\log p_\theta(x)]$$

  - **Why maximum likelihood + SGD?** It works with large datasets and is compatible with neural networks.

Designing the model

- **Key requirement for maximum likelihood + SGD: efficiently compute log p(x) and its gradient**
- **We will choose models p_θ to be deep neural networks**, which work in the regime of high expressiveness and efficient computation
- **How exactly do we design these networks?**
  - Any setting of θ must define a valid probability distribution over x:

$$\text{For all } \theta, \sum_x p_\theta(x) = 1 \text{ and } p_\theta(x) \geq 0 \text{ for all } x$$

  - log $p_\theta(x)$ should be easy to evaluate and differentiate with respect to θ
  - This can be tricky to set up!

Autoregressive models

- First, given a Bayes net structure, setting the conditional distributions to neural networks will yield a tractable log likelihood and gradient. Great for maximum likelihood training!

$$\log p_\theta(x) = \sum_{i=1}^{d} \log p_\theta(x_i | parents(x_i))$$

- But is it expressive enough? Yes, assuming a fully expressive Bayes net structure: any joint distribution can be written as a product of conditionals

$$\log p(\mathbf{x}) = \sum_{i=1}^{d} \log\ p\ (x_i | \mathbf{x}_{1:i-1})$$

- This is called an **autoregressive model**. So, an expressive Bayes net structure with neural network conditional distributions yields an expressive model for p(x) with tractable maximum likelihood training.

A toy autoregressive model

- Two variables: x1, x2
- Model: p(x1, x2) = p(x1) p(x2|x1)
  - p(x1) is a histogram
  - p(x2|x1) is a multilayer perceptron
    - Input is x1
    - Output is a distribution over x2 (logits, followed by softmax)