Unsupervised Learning Notes

November 28th, 2017
Chloe Kaubisch, Justin Chen

Topics: Unsupervised Learning, Autoencoders, GANs

Paper: https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf

https://arxiv.org/abs/1701.00160

NOTES: Overview

1. Recap

- Reinforcement learning, what it is and how it relates to GANs
  - Model-based learning
- Supervised learning--contrast with unsupervised learning

2. Types of Unsupervised Learning Algorithms

2.1 Auto-encoders

- High-level introduction
- Types and purposes
  - Denoising auto encoder
  - Sparse auto encoder
  - Mention the variational auto encoder (VAE), Contractive auto encoder (CAE)

2.2 Generative Models

- Restricted Boltzmann machines (RBM)
- Generative Adversarial Networks (GAN)
  - Deep Convolutional Generative Adversarial Networks (DCGAN)
  - Wasserstein Generative Adversarial Networks (WGAN)
  - Conditional Generative Adversarial Networks (CGAN)
- Problems with GANs
- Improvements to be made
- Open questions

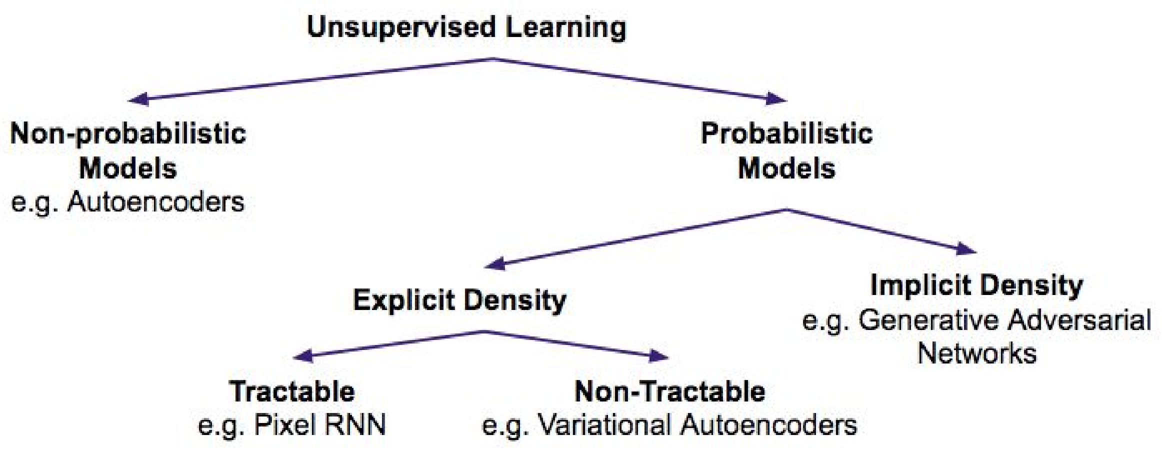
References

1. Recap

- Supervised Learning vs Unsupervised Learning:
    - Supervised learning requires large amounts of *labeled* data
- Semi-Supervised Learning
    - Missing labels

2. Types of Unsupervised Learning

- Generative Models:
    - What are they?
        - A model that takes a training set of samples of a distribution P*data* and learns to represent an approximation of that distribution
    - Why use them?
        - Model-based reinforcement learning
            - A model could be used to output possible futures, given a current state and possible actions of an agent
        - Semi-supervised learning
            - Modern deep learning networks require a huge amount of labeled data
            - Semi-supervised learning uses a training set with a large amount of missing data--GANs can be trained with missing data to provide predictions on missing outputs
        - Multi-modal outputs:
            - For many tasks a single input may correspond to many different correct outputs
            - E.g. predicting next frame of a video
        - Some tasks intrinsically require the generation of good samples
            - E.g. synthesizing high-resolution version of a low-quality image, creating art, neural style transfer
    - Types:



        - Variational Autoencoders
        - GANs
    - Probabilistic Models
    - Implicit vs Explicit Density
        - Generative models address density estimations, a core problem in unsupervised learning
            - Variation in input space
            - How much individual features vary
        - Estimating underlying distribution of your training data
        - Explicit: explicitly define and solve for *Pmodel*
        - Implicit: learn a model that can produce samples from *Pmodel* without having to explicitly define it
    - Maximum Likelihood & KL-Divergence
- Autoencoders
    - Simply, a three-layer neural network, just an input layer, one hidden layer, and an output layer
    - Often trained with a single layer encoder and a single layer decoder. Doesn't necessarily have to be, but is generally symmetrical, e.g. a 4-layer encoder and a 4-layer decoder
    - Should output a reconstruction of its input
    - Traditionally used for dimensionality reduction or feature learning, but more recently have come to be used in generative modeling
    - Definitions:

        - Undercomplete: dimension of hidden layer is less than dimension of input layer. This forces the autoencoder to learn the most valuable features
        - Overcomplete: dimension of hidden layer is greater than the dimension of the input layer. The autoencoder can learn to simply copy its input to its output, and learns nothing useful about the data.
            - This issue is addressed by the denoising autoencoder
        - Recirculation: alternate method of training autoencoders by comparing activations on original input to activations on reconstructed input--seen as more biologically plausible than backpropagation
        - It has been seen that with too much leeway this occurs, and the network learns nothing.
    - Encoder:
        - Take input from high-dimensional to lower-dimensional space
    - Decoder:
        - Reconstructs high-dimensional input
    - Training:
        - Loss function computes loss between input and output
    - Types:
        - Variational
            - “Probabilistic spin”
            - Assume sample x is generated from some unobserved underlying distribution z (latent features)
            - Sample first from z (P(z)), then generate x from sampling from a conditional (P(x|z))
            - Issues:
                - intractable
        - Sparse
    - Usage:
        - Encoders often used to initialize supervised models
            - Tack on a classifier to your encoder
            - input-->encoder-->classifier
        - This is valuable because you have now used unlabeled data to learn a good basic feature representation, useful when you don’t have a lot of (labeled) data to work with
            - You can sidestep the problems that come with having small training sets, like overfitting

- Generative Adversarial Networks
    - Relatively new machine learning architecture introduced by Ian Goodfellow and his colleagues in 2014 at the University of Montreal
    - **Universal approximators** for probability distributions
    - Issues addressed by GANs:
        - The structure of the generator is not strict
        - Doesn’t need Markov chains
            - Markov chains often fail to scale to high dimensional spaces
        - Can generate samples in parallel -- e.g. FVBNs need to run every time you want to produce a sample
    - Can be equated to a “game” between two components: the generator and the discriminator. The generator creates samples that mimic the training data, and the discriminator must determine whether a given sample is real or generated.
        - Generator: typically a deep neural network
        - Discriminator:
        - They’re trying to reach a Nash equilibrium, essentially a saddle point
    - Training:
        - Given a distribution *Pdata* you want to learn a model *Pmodel* that will generate samples from the same distribution
            - You want *Pmodel* to learn to be similar to *Pdata*
            - Minimax function:
                - You alternate gradient ascent on the discriminator (because you’re trying to maximize) and gradient descent on the generator

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

        - On each step, you take two mini-batches: one from the dataset and another from
    - Semi-supervised learning with GANs
    - DCGAN: deep convolutional generative adversarial networks

    - iGAN:
        - https://www.youtube.com/watch?v=5jfViPdYLic
    - Problems:
        - Non-convergence
        - Helvetica Scenario
            - Partial collapse
            - Complete collapse

3. References

https://arxiv.org/pdf/1701.00160.pdf

Unsupervised Learning

What it is:

A type of machine learning which relies on unlabeled data to draw conclusions about said data.

It's more subjective than supervised learning, because you have no concrete objective, like trying to predict a value, you're simply looking for structure in data.

Usage for unsupervised machine learning and associated algorithms:

// optional?? clustering/dimensionality reduction not relevant to topic: maybe bring it up briefly to ensure everyone is familiar but don't go too into depth

Clustering:

- attempts to partition data into subgroups
- To cluster the data you need a measure of similarity or dissimilarity
- Useful for classification and anomaly detection
- K-Means Algorithm, Hierarchical Clustering:
    - K-Means:
    -

Dimensionality Reduction:

- Used to reduce redundant features
- A lot of datasets today, especially those in use where unsupervised machine learning can be applied, are high-dimensional e.g. genetic sequencing data, internet activity data, etc.
- D. R. simply reduces the number of features
- It's useful for data compression (this becomes important regarding autoencoders!), removing noise/redundant features, and avoiding overfitting

- Methods for dimensionality reduction: Principal Component Analysis (PCA), Multidimensional Scaling (MDS)

Pretraining

What it is:

Changing how you initialize the values of the hidden layers in a neural network to avoid getting stuck in local optima. Or essentially, training your neural network to learn deeper and more complex structures, to avoid overfitting the data.

How it works:

The goal of pretraining a neural network is to solve the problem of overfitting by forcing the neural network to represent latent structure rather than just identify superficial features. You're asking the neural net to learn what distinguishes the input data from random data, not just classify it. Essentially, you're training your neural net to make specific generalizations -- to generalize what your input actually is, therefore ideally reducing overfitting.

To pretrain a neural network you iterate through the hidden layers, training each individually and fixing the weights of the previous layer as you do so, so as to capture increasingly more complex representations. Once all your layers are pretrained you perform backpropagation (fine tuning just the output layer).

Autoencoders // Sparse Autoencoders

An autoencoder is a feed-forward neural network trained to reproduce its input at the output layer.

Can be divided into two parts: the encoder, which computes the values of the hidden layer, and the decoder, which attempts to reconstruct the original input given the values of the hidden layer. Essentially, the neural network is trying to learn the identity function, or at least an approximation of the identity function.

This may seem trivial, but you can still learn interesting features by placing restraints on the neural network.

Restraints:

- Restrict the size of the hidden layer: neural net is forced to learn a compressed representation of the input
- Sparsity constraint: number of hidden units can be comparable to or even greater than the number of inputs/outputs
  - Constrain $a_i^{(2)} \cong$ -1 for all i
  - The idea is that in the brains of most mammals at any given time, most of the neurons are not firing, so you want to reproduce that in your neural network → hence the constraint
  - How to implement sparsity:
    - Compute: (equation) a.k.a. The average activation of hidden unit j (as averaged over the training set).

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x^{(i)}) \right]$$

    - We now enforce the constraint $\hat{\rho}_j = \rho$, where p is the sparsity parameter, a small value close to 0 (if you use the sigmoid activation function) or -1 (if you use the tanh activation function). Say you're using the tanh activation function, you might pick p = -0.999. You want the average activation of each hidden neuron (denoted by j) to be close to -0.999.
    - To satisfy this sparsity constraint, apply a penalty term to your optimization objective that will penalize p_j when it deviates too far from p.

$$\sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

      - This penalty term is based on the concept of Kullback-Leibler (KL) divergence:

$$\mathrm{KL}(\rho || \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

      - Hence, our penalty term can be rewritten as:

$$\sum_{j=1}^{s_2} \mathrm{KL}(\rho || \hat{\rho}_j)$$

      - Cost function is now:

$$J_{\text{sparse}}(W, b) = J(W, b) + \beta \sum_{j=1}^{s_2} \mathrm{KL}(\rho || \hat{\rho}_j)$$

-

Autoencoder Usage:

- Data compression

Generative Adversarial Networks

- Generative Adversarial Networks is a training framework designed to circumvent maximum likelihood estimation and previous intractable generative techniques
- Goal of framework: Nash equilibria
    - Networks **play a game** instead of trying **directly model the data-generating distribution via optimization**