My Notes

Ibrahim Habib

Table of contents

Notes

Welcome to my notes website!

Part I

Papers

1 From Learning Models of Natural Image Patches to Whole Image Restoration

These are the notes on Zoran & Weiss (2011).

- Background
  - Learning Image Priors is quite valuable
  - Used for multiple tasks like image denoising and inpainting
  - Hard task because of the high-dimensionality of images
  - Earlier work learned only small patches prior to reduce computation
- 3 Questions to Answer in the Paper
  - Do patch priors that give high likelihoods yield better **patch** restoration (Better Patch Priors –> Better **Patch** Restoration?)
  - Do patch priors that give high likelihoods yield better **image** restoration (Better Patch Priors –> Better **Image** Restoration?)
  - Can we learn better patch priors?
- Do better batch Priors lead to better patch restoration?
  - They trained several models on 50,000 8 by 8 patches
  - Calculated the log-likelihood of each model on a set of unseen natural images (Measure of *Better Priors*)
  - Calculated the performance in denoising using MAP estimates (Measure of *Better Patch Restoration*)
  - Found strong correlation
  - **Answer**: Yes
- Do better batch Priors lead to better image restoration?
  - First, How to restore image from patch priors?

* 3 earlier techniques are mentioned that are simple but weak
* The author describes a new framework that maximizes the EPLL (Expected-Patch-Log-Likelihood) while ensuring the restored image is close to the corrupted one
* Minimize $f_p(\mathbf{x}|\mathbf{y}) = \frac{\lambda}{2}\|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2 - EPLL_p(\mathbf{x})$
* $EPLL_p(\mathbf{x}) = \sum_i \log p(\mathbf{P}_i\mathbf{x})$
* Optimization is done using a method called "Half Quadratic Splitting" which the paper describes in detail
* He describes extending this restoration task to denoising, deblurring, and inpainting through changing the matrix A.
* The author describes other techniques
  - A common thing is averaging the clean patches to form final estimate of the image
  - The author doesn't do that

– Used the priors from the previous section

– Used EPLL framework to restore 5 corrupted images

– Measured both patch likelihood and restored image quality (PSNR)

– Found strong correlation

– **Answer**: Yes

- Can we learn better patch priors?

  – The authors use a Gausian Mixture Model (GMM) with unconstrained covariance matrix

  – Learning is done through Expectation Maximization Algorithm

  – Calculating the log likelihood of a patch is done through $\log p(\mathbf{x}) = \log(\sum_{k=1}^{K} \pi_k N(\mathbf{x}|\mu_k, \sigma_k))$

  – π_k are the mixing weights

  – The BLS (mean or expected value of the posterior probability) can be calulated easily through a closed form

  – The MAP (mode or maximum of the posterior) is intractable but the other describe a way to estimate it

  – It outperforms other patch-based models in log likelihood, patch restoration, and image restoration

– It outperforms SOTA generic prior models in image denoising
– It is competitive with image specific SOTA models in image denoising

2 Natural Images, Gaussian Mixtures, and Dead Leaves

These are the notes for the paper Zoran & Weiss (2012).

- Unconstrained Gaussian Mixture Models (GMMs) with a small number of mixture components learned from patches perform exteremly good
  - Valuable because
    - They are simple
    - Many of the current models are GMMs with exponential or infinite number of components with constrained covariance matrix
- Paper Overview: A study of the nature of GMMs
  - Proof GMMs are excellent at modeling natural images patches
  - Analysis of properties of natural images captured by GMMs and relation to number of components K
  - Proof of strong connection between natural images statistics and a simple variant of dead leaves models
- GMMs are Really Goos for Natural Images Patches
  - Compared GMM with 200 components to 8 other models
  - Trained on patches from the Berkley Segementation Database and used its test set
  - 3 Experiments
    - **Log-Likelihood**: GMM outperformed all models and similar performance to Karklin and Lewicki
    - **Denoising**:
      - Added independent white Gaussian noise to the test set
      - Calculated MAP for each model given noisy patch
      - Evaluated performance using Peak Signal to Noise Ratio (PSNR)

            - GMMs performed exceptionally well
        * **Sample Quality**: Generated samples from the models. No quantitative analysis, but the GMMs samples looked capturing the structure of natural images like edges and textures
    - GMMs are really good
    - No claims of being the best
- Analysis of the results
    - Adding more components to the GMM increases the performance; although, they seem to be converging at an upper bound (this is shown experimentally)
    - GMM as a Generative Process
        * You can generate a new sample from a GMM by choosing one of the K components and sampling N independent Gaussian variables with mean 0 and variance 1. Put these values in a vector and call it z.
        * To compute the sample, use $\mathbf{x} = \mathbf{V}_k \mathbf{D}_k^{0.5} \mathbf{z}$
        * $\mathbf{V}_k$ is the eigenvector matrix of Σ_k
        * $\mathbf{D}_k$ is the eigenvalues matrix of Σ_k
        * It can be shown that this operation make the random vector $\mathbf{z}$ follow the correaltion structure of Σ_k
        * **Conclusion**: To understand GMM, we need to understand their eigenvalues and eigenvectors
    - Eigenvalues and Eigenvectors of the First Components
        * Have very eigenvectors
        * Eigenvalues spectrum has very similar structure but differs with a multiplicative constant
        * This is equivalent to using a Gaussian Scale Mixture model
        * These components capture the contrast variability of the image patches
    - The Upcoming Components
        * Adding complements shows more specialized components capturing different properties of the natural images
        * They capture textures and boundaries on various scales and orientations
- Mini Dead Leaves

- The mini dead leaves model is a variant of the dead leaves model proposed by the authors that works on batches
- It divides patches into two types (flat and edges)
  - Flat patches are sampled from a texture producing model (They use a GSM trained on natural images)
  - Edge patches randomly select an angle and distance from the center that divide the patch into two parts. Each part is sample separately as a flat patch
- They model contrast and occlusions in images
- They created images using mini dead leaves and trained GMM and the other previously used models on the new images.
- The GMM performed incredibly well proving that its great performance on natural images is probably due to its ability to model occulsions and contrast
- The mini dead leaves is weaker than GMM and expected to perform worse
  - Primiarily due to weakness of GSM in modeling textures and the weak occlusion creation strategy (a random straight line)

3 Deep Image Prior

These are the notes for Ulyanov et al. (2018)

- **Main Point**: The structure of ConvNets is sufficient to capture a great deal of low-level statistics before any training
  - The study focuses on the prior captured by a deep convolutional network, independent of any training
- The Driving Reasoning
  - ConvNets are SOTA for many image-related tasks (super-resolution, image reconstruction, denoising, etc.)
  - They are usually trained on huge datasets.
  - It can be assumed that large training datasets is the reason of the great performance, but learning isn't a sufficient explanation
  - Generalization requires the structure of the network to resonate with the structure of the data
- Their Method
  - Basically, the authors fit a randomly initialized ConvNet on the noisy image and use it for the generation task.
  - The Task
    - They consider inverse tasks such as denoising, super-resolution, and inpainting.
    - Expressed as energy-minimization problem: $x^* = \min_x E(x; x_0) + R(x)$
      - $E(x; x_0)$: Task dependendt data term (e.g. How similiar the reconstructed image is to the noisy one?)
      - $R(x)$ is a requalization term (e.g. the probability x occurs in nature as determined by the prior of a pretrained model)
      - x_0 is the noisy/low-resolution/occulded image
      - x^* is the model's predicted clean/high-resolution/inpainted image
    - Deep networks are applied by mapping a random code z to an image x: $x = f_\theta(z)$
  - Their method
    - Instead of finding the parameters by training on a large dataset, they learn to map z to the given x_0

    - $\theta^* = \arg\min_\theta E(f_\theta(z); x_0)$
    - and they set the regualizer to zero. Thus,
    - $x^* = f_{\theta^*}(z)$
  - Why it works?
    - It is expected that their model learns the noise in x_0
    - This doesn't happen because the ConvNet architecture has high resistance to learning noise and low resistance to learning the signal
    - => the model learns the signal before it learns the noise
    - => They stop training early before the model learns the noise

- Applications
  - They apply their model to multiple tasks including denoising, super-resolution, inpainting, etc.
  - In all tasks, the model outperforms or is very close to the SOTA no-training models and is close to those that train on large datasets

- To **Summarize**, ConvNets are really good image priors reagardless of the training data

Part II

Stanford Deep Generative Models Course

4 Introduction

- Generative models view the world through the lens of probabilities
- Given a finite set of samples S generated from an underlying distribution p_{data}, the goal of a generative model is to approximate p_{data} from S
- Parametric vs non-parameteric models
  - The course will focus on parametric models.
  - Parametric models can scale more efficiently with large datasets
  - Parametric models are limited in the family of distributions they can represent
- Learning
  - Given a dataset $data$, the goal is to find the parameters of a generative model θ that closes the distance between p_{data} and p_θ (The model's learnt distribution and the real distribution)
  - Stated mathematically
    - $\min_{\theta \in M} d(p\_data, p_\theta)$
    - Where d is a measure of distance between the two distributions and M is the model's family
  - The problem: Current datasets are way too small in size compared with the possible set of values covered by the true distribution
- Course Focus (Answer the following questions)
  - What is the representation for the model family M ?
  - What is the objective function d( )?
  - What is the optimization procedure for minimizing d( )?
- 3 Fundamental Inference Queries for generative models
  - **Density estimation**: Given a datapoint x what is the probability assigned by the model, i.e., $p_\theta(x)$?
  - **Sampling**: How can we generate novel data from the model distribution, i.e., $x_{new} \sim p_\theta(x)$?

  - **Unsupervised representation learning**: How can we learn meaningful feature representations for a datapoint x?

- Current Challenges
  - Quantitative evaluation of generative tasks is not easy since current metrics fail to reflect desirable qualitative metrics
  - Not all model families permit efficient and accurate inference on all these tasks

5 Background

5.1 Overview of Genarative Model

The main task in generative models is to build a model that approximates a probability P_θ from a set of data points sampled from the true distribution P_{data} (P_θ should be as close as possible to P_{data} according to some distance measure)

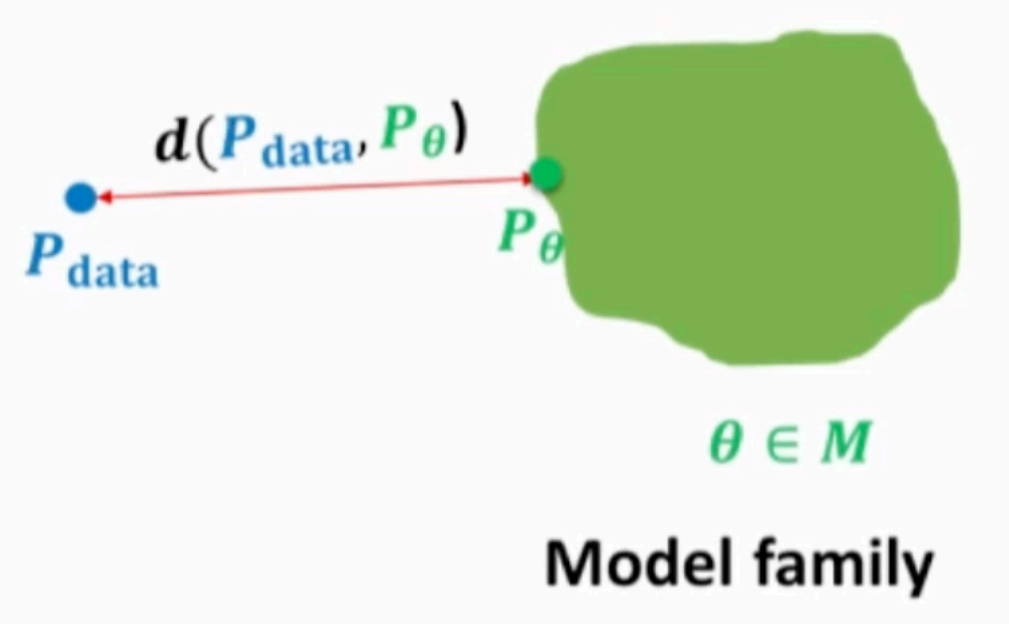

Figure 5.1: High-Level View of Generative Models Tasks

We want to learn a probability distribution $p(x)$ over data x (e.g., Images, Text, Protein Sequence). These distributions could be used for

- **Generation**: Sample $x_{new} \sim p(x)$
- **Density Estimation**: $p(x)$ should be high if x is part of the data (*anomaly detection*)
- **Unsupervised Representation Learning:** Feature extraction

5.2 The Curse of Dimensionality

Representing $p(x)$ is difficult for high-dimensional data (such as images and text). This is easier, however, for simpler data. For example, a Categorical Distribution (n-sided cube) can be parameterized using $n-1$ parameters (one for the probability of each outcome and the last is calculated from the first $n-1$).

This isn't the case for high-dimensional data. A single pixel in an image, for example, needs $256 * 256 * 256 - 1$ parameters. For an image, the number of required parameters grows exponentially with the number of pixels.

5.2.1 Independence Assumption

One way to simplify the structure is to assume independence

$$p(x_1, ..., x_n) = p(x_1)p(x_2)...p(x_n)$$

This would require only n parameters (because $p(x_i)$ needs only one parameter) to model the 2^n images.

The problem is that this assumption is way too strong. For example, if you are generating images, you are not allowed to see any other pixel value when extracting the value of a pixel.

5.2.2 Conditional Independence Assumptions

Using the chain rule, you can simplify a multivariate distribution to

$$p(x_1, ..., x_n) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)...p(x_n|x_1, ..., x_{n-1})$$

[1]

The number of parameters, however, is still exponential

$$1 + 2 + ... + 2^{n-1} = 2^n - 1$$

[2]

This is calculated using the following logic

- $p(x_1)$ needs 1 parameter

[1] This is the simplification used by autoregressive models, which are very popular with LLMs. That is predict a token from the previously seen set of tokens.

[2] Assuming the total number of values x could take is 2

- $p(x_2|x_1 = 0)$ needs 1 parameter and $p(x_2|x_1 = 1)$ needs 1 parameter for a total of 2 parameters
- ...

Assuming conditional independence $(X_{i+1} \perp \{X_1, ..., X_{i-1}\}|X_i)$ can simplify the math

$$p(x_1, ..., x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2)...p(x_n|x_{n-1})$$

Making the number of required parameters linear ($2n - 1$).

However, these models are still weak. We need more context to make better estimates. Using only a single word isn't enough to predict the next word.

5.3 Bayesian Network

Bayesian network adds more flexibility relative to the last discussed model by making variables rely on a set of other variables (not strictly only one variable). That is, it specifies $p(x_i)|x_{A_i}$ where x_{A_i} is a set of random variables. In this case, the joint parameterization is

$$p(x_1, ..., x_n) = \prod_i p(x_i|x_{A_i})$$

More formally, a **Bayesian Network** is a DAG where the nodes are random variables and the edges are the dependence relationships between these variables. Each node has one conditional probability distribution specifying the variable's probability conditioned on its parents' values.

Now the number of parameters is exponential in the number of parents, not the number of variables.

5.4 Discriminative vs Generative Models

To show the differences, we will use an example of using Naive Bayes for single-label prediction: classify emails as spam ($Y = 1$) or not ($Y = 0$). In this task, we are asked to model $p(y, x_1, ..., x_n)$ where y is the label, $1 : n$ are the indices of the words in the vocabulary, X_i is a random value that is 1 if word i appears in the email and 0 otherwise.

What Naive Bayes does is assume that words are conditionally independent given Y, making the joint distribution

$$p(y, X_1, ..., x_n) = p(y) \prod_{i=1}^{n} p(x_i|y)$$

After estimating the parameters from the training data, prediction is done using the Bayes rule

$$P(Y = 1|x_1, ..., x_n) = \frac{p(Y = 1) \prod_{i=1}^{n} p(x_i|Y = 1)}{\sum_{y=\{0,1\}} p(Y = y) \prod_{i=1}^{n} p(x_i|Y = y)}$$

The independence assumption made here is that one word's appearance in an email is independent from another's appearance.

This model is a generative one. It attempts to calculate $p(Y, \mathbf{X})$ using $p(Y, \mathbf{X}) = p(Y)p(\mathbf{X}|Y)$. A discriminative model, on the other hand, uses $p(Y, \mathbf{X}) = p(\mathbf{X})p(Y|\mathbf{X})$.

In the generative case, we need to learn both $P(Y)$ and $p(\mathbf{X}|Y)$ and then compute $p(Y|\mathbf{X})$ using Bayes rule. In the discriminative, you just need to estimate the conditional probability $p(Y|\mathbf{X})$. In the discriminative, you don't have to model $p(\mathbf{X})$, while the generative model does learn the input features vector.

Discriminative models usually assume a functional form for the probability of the target condition on the features. That is,

$$p(Y = 1|\mathbf{x}, \ ) = f(\mathbf{x}, \ )$$

For example, logistic regression assumes a linear relationship. Neural networks make it more flexible by adding non-linearities.

To sum it all up, the generative model learns the full joint distribution, making it able to predict anything from anything. Discriminative models learn only the relationship between inputs and the target. This limits the uses of the discriminative model but makes the task much simpler.

6 Autoregressive Models

Autoregressive models are one way of representing $p(x)$ for a generative model.

The previous lecture discussed three ways to model a joint distribution:

1. Chain Rule:
   - $p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_1, x_2, x_3)$
   - Fully General
   - No Assumptions
   - Exponential Size
2. Bayes Network:
   - $p(x_1, x_2, x_3, x_4) \approx p_{CPT}(x_1)p_{CPT}(x_2|x_1)p_{CPT}(x_3|x_2)p_{CPT}(x_4|x_1)$
   - Assumes conditional independence
   - Uses tabular representation via conditional probability table (CPT)
3. Neural Models
   - $p(x_1, x_2, x_3, x_4) \approx p(x_1)p_{Neural}(x_2|x_1)p_{Neural}(x_3|x_1, x_2)p_{Neural}(x_4|x_1, x_2, x_3)$
   - Assumes a specific form for the conditionals

We will use an example of training a model on MNIST, assuming each pixel can only be black or white, and a total of 784 pixels.

To use an autoregressive model, we start by picking an order of all the random variables. It uses the same format of Neural models, where each conditional is approximated with a function on all the previous variables in the ordering. The function could be something simple (logistic regression, for example) or a much more complex deep neural network.

6.1 Examples of Autoregressive Model

6.1.1 Fully Visible Sigmoid Belief Network (FVSBN)

The conditional variables $X_i|X_1, ..., X_{i-1}$ are Bernoulli with parameters

$$\hat{x}_i = p(X_i = 1 | x_1, ..., x_{i-1}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j-1}^{i-1} \alpha_j^i x_j)$$

You evaluate the multivariable distribution by multiplying all the conditionals.

You can sample from the distribution by sampling one value at a time

- Sample $\bar{x}_1 \sim p(x_1)$
- Sample $\bar{x}_2 \sim p(x_2 | x_1 = \bar{x}_1)$

Sampling here is relatively easy.

Conditional sampling isn't easy. One example is image inpainting.

The number of parameters is

$$1 + 2 + 3 + ... + n \approx n^2$$

This is a weak model due to the weakness of logistic regression.

6.1.2 Neural Autoregressive Density Estimation (NADE)

Use a neural network instead of logistic regression.

This may result in a lot of parameters. One way to simplify the number of parameters is to tie the weights. That is repeat the weights for the same parameter.

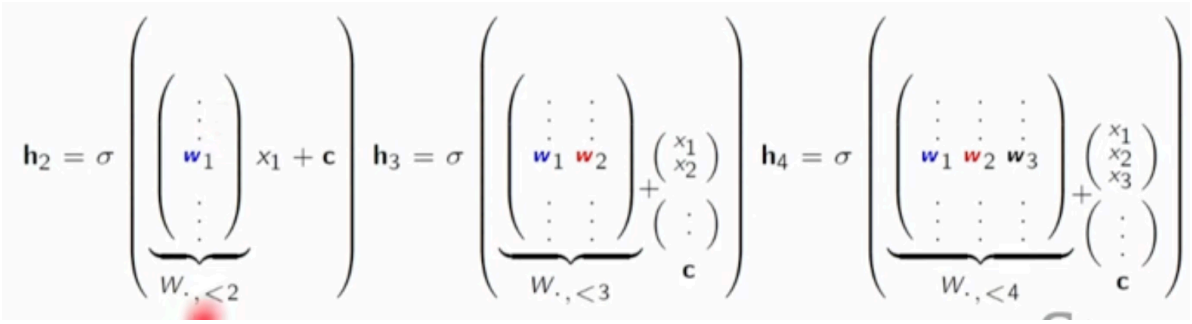

Figure 6.1: Repeated Weights to Minimize the Parameter Count

Using a single hidden layer with dimension d, the number of parameters is $O(nd)$.

6.1.3 RNADE

This is to model continuous data without discretizing it. This is done by making $\hat{x}_i$ parameterize a continuous distribution. One example is using a mixture of K Gaussians.

6.2 Autoencoders vs Autoregressive Models

A vanilla autoencoder isn't a generative model because it can't be used to generate new data points.

An autoencoder doesn't enforce an ordering on the input variables. If it does, it becomes an autoregressive model. To make the autoencoder a generative model, you have to make it correspond to a valid Bayesian Network (DAG).

Using masks (a masked autoencoder), you can enforce the DAG structure.

6.3 Recurrent Neural Networks (RNNs)

RNNs keep a summary of the previously seen history (the past random variables) and keep updating it as new information becomes available. The number of parameters is constant in terms of n.

They are very slow in training because the calculation is sequential. Another issue is using only a single vector to summarize all the history. This eases computation but is a huge assumption.

6.4 Attention-Based Models

Attention-based models can access the hidden vectors for all the previous tokens when predicting a new one. This is in contrast to the single vector used in RNNs.

This is done by comparing the set of query vectors with the new key vector. These values are then merged together based on some method that decides which query vectors to focus on more.

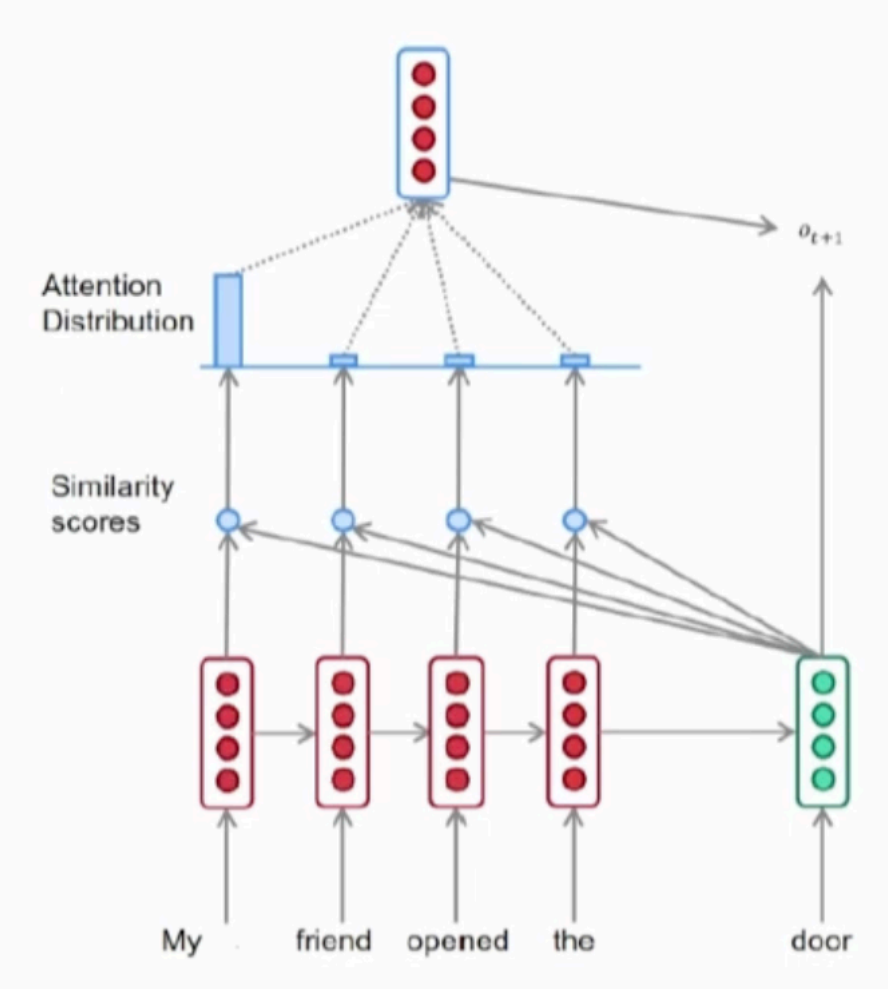

Figure 6.2: Attention-based Models

The merged vector is then used to predict the next token.

6.5 CNNs

To use them in autoregressive models, you have to make sure the convolutions respect your selected ordering and the model doesn't peek into upcoming variables (pixels in the case of working on images). This can be done using masked convolutions.

6.6 Disadvantages of Autoregressive Models

They are very hard to use for representation learning and unsupervised learning.

7 Maximum Likelihood Learning

We are given a dataset $\mathcal{D}$ of m samples from P_{data}. We assume the samples are IID. We are also given a family of models $\mathcal{M}$. The task is to learn a model P_θ. That captures P_{data} from the sampled data.

This task is hard because m is way, way too small compared to the number of possible states of the variables.

To be able to extract the best model, we need to define the word *best*.

We need a measure of similarity between the two distributions P_θ and P_{data}.

7.1 Kullback-Leibler Divergence (KL-Divergence)

$$D(p||q) = \sum_x p(x) log \frac{p(x)}{q(x)}$$

This quantity is non-negative, equal to zero iff $p = q$, and asymmetric ($D(p||q) \neq D(q||p)$).

It measures the expected number of extra bits required to describe samples $p(x)$ using a compression code based on q instead of p.

The KL divergence can be used for our task because

$$D(P_{data}||P_\theta) = \mathbf{E}_{\mathbf{x}\sim P_{data}}[log(\frac{P_{data}(\mathbf{x})}{P_\theta})]$$

This can be simplified to

$$D(P_{data}||P_\theta) = \mathbf{E}_{\mathbf{x}\sim P_{data}}[\log P_{data}(\mathbf{x})] - \mathbf{E}_{\mathbf{x}\sim P_{data}}[\log P_\theta(\mathbf{x})]$$

Since the first term is constant in θ, you can just minimize

$$-\mathbf{E}_{\mathbf{x}\sim P_{data}}[\log P_\theta(\mathbf{x})]$$

or equivalently maximize

$$\mathbf{E}_{\mathbf{x}\sim P_{data}}[\log P_\theta(\mathbf{x})]$$

In other words, minimizing the KL-divergence is equivalent to maximizing the log-likelihood.

We can't directly calculate the expectation because we don't have access to P_{data}. What we can compute is the empirical log-likelihood. This is done by approximating the expectation in the following manner.

$$\mathbf{E}_{\mathcal{D}}[\log P_\theta(\mathbf{x})] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

8 Variational Autoencoders (VAE)

8.1 Latent Variable Models

Latent variables $\mathbf{z}$ are hidden variables used to capture sources of variability in the data that aren't explicitly stated (e.g., gender, eye color, pose, hair color in human face images). These variables are initialized randomly and trained through unsupervised learning. You may then build a model that reasons conditioned on the latent variables. This is usually easier because latent variables usually have a smaller dimension. That is $p(\mathbf{x}|\mathbf{z})$ is easier to model than $p(\mathbf{x})$. These variables usually correspond to high-level features.

Neural networks can be used to model the conditionals (deep latent variable models. Examples:

$$\mathbf{z} \sim \mathcal{N}(0, I)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$$

where $\mu_\theta(\mathbf{z})$ and $\Sigma_\theta(\mathbf{z})$ are neural networks.

8.2 Mixture of Gaussians

This is a Bayes net $\mathbf{z} \to \mathbf{x}$ where

$$\mathbf{z} \sim \text{Categorical}(1, ..., K)$$

$$p(\mathbf{x}|\mathbf{z} = k) = \mathcal{N}(\mu_k, \sigma_k)$$

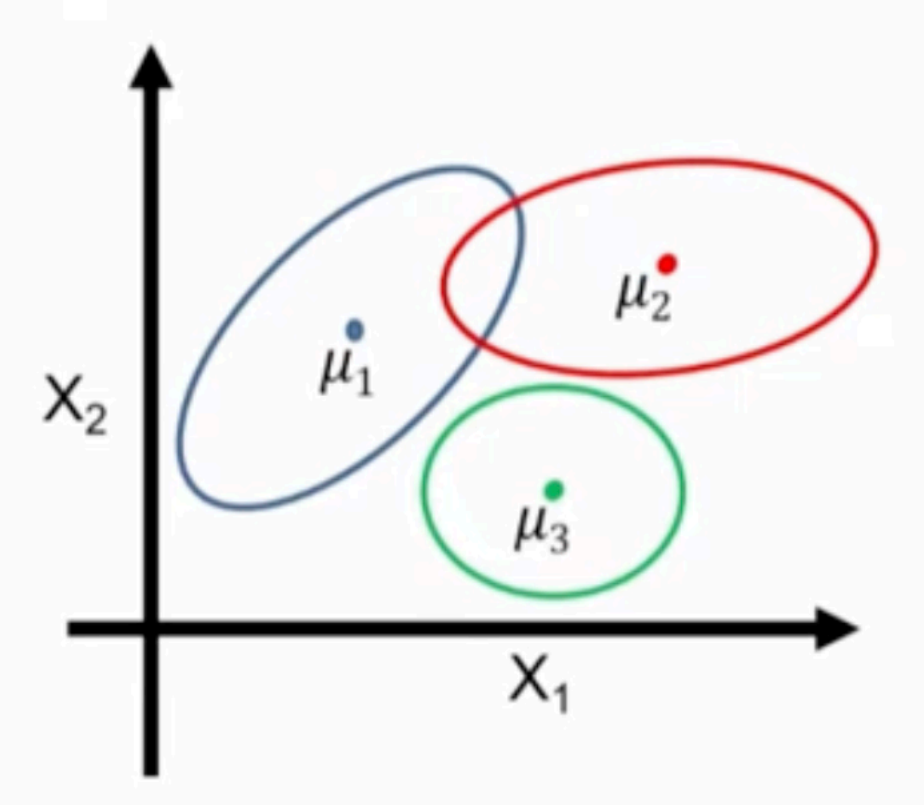

Figure 8.1: Mixture of Gaussians

8.3 Mixture Models

Using latent variables allows us to formulate $p(x)$ as a mixture of simpler models, creating a complex model from simpler ones. This is because

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, \mathbf{z}) = \sum_z p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$$

8.4 Variational Autoencoder

A mixture of an infinite number of Gaussians.

$$\mathbf{z} \sim \mathcal{N}(0, I)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$$

where $\mu_\theta(\mathbf{z})$ and $\Sigma_\theta(\mathbf{z})$ are neural networks.

There is an infinite number of Gaussians because $\mathbf{z}$ is now continuous.

Once again, $p(\mathbf{x}|\mathbf{z})$ is simple but $p(\mathbf{x})$, the mixture, is complex.

Calculating the probability of observing a certain training data point, $\bar{x}$, because it involves calculating an integral

$$\int_{\mathbf{z}} p(\mathbf{X} = \bar{\mathbf{x}}, \mathbf{Z} = \mathbf{z}, \theta)\, d\mathbf{z}$$

In our setting, we have a dataset $\mathcal{D}$ where for each datapoint $\mathbf{X}$ are observed, but the variables $\mathbf{Z}$ are never observed.

To train the model with maximum likelihood learning, we need to find θ that maximizes

$$\sum_{\mathbf{x}\in\mathcal{D}} \log p(\mathbf{x}, \theta) = \sum_{\mathbf{x}\in\mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}, \theta)$$

Evaluating $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}, \theta)$ is intractable. For example, having 30 binary latent variables will involve the sum of 2^{30} terms. For continuous, the computation is also hard. The gradients with respect to θ are also hard to compute.

You could use Monte Carlo to approximate the hard-to-compute sum.

This is because

$$p_\theta(x) = \sum_{\text{All values of } \mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) = |\mathcal{Z}| \sum_{\mathbf{z}\in\mathcal{Z}} \frac{1}{|\mathcal{Z}|} p_\theta(\mathbf{x}, \mathbf{z}) = |\mathcal{Z}| \mathbb{E}_{\mathbf{z}\sim Uniform(\mathcal{Z})}[p_\theta(\mathbf{x}, \mathbf{z})]$$

Monte Carlo approximates the expected value through sampling k values for $\mathbf{z}$ uniformly at random and approximating using

$$|\mathcal{Z}| \frac{1}{k} \sum_{j=1}^{k} p_\theta(\mathbf{x}, \mathbf{z}^{(j)})$$

The problem here is that uniformly sampling $\mathbf{z}$ won't work in the real world, and we need a smarter way.

A better way is to use importance sampling.

$$p_\theta(x) = \sum_{\text{All values of } \mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z} \in \mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_\theta(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}$$

and once again apply Monte Carlo, but this time sample from $q(\mathbf{z})$.

Now we also need to train $q(\mathbf{z})$ which will also be a neural network. We need to minimize $\log p_\theta(\mathbf{x})$. We can directly minimize this value, but we can minimize a lower bound for it

$$\log p_\theta(\mathbf{x}) = \log \sum_{\mathbf{z}} q(\mathbf{z}) \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}] = \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x}, \mathbf{z}) - \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z}) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta($$

where the inequality follows from Jensen's inequality and the fact that the log function is concave and $H(q)$ is the entropy of q. This inequality holds for any q with it being an equality if $q = p_\theta(\mathbf{z}|\mathbf{x})$. That is the optimal choice of q is $p_\theta(\mathbf{z}|\mathbf{x})$. We should try to choose q as close as possible to $p_\theta(\mathbf{z}|\mathbf{x})$.

In practice, we will train together $q(\mathbf{z})$ (an encoder) and $p(\mathbf{x}|\mathbf{z})$ (a decoder) to minimize the ELBO.

8.5 Training

References

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2018). Deep image prior. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zoran, D., & Weiss, Y. (2011). From learning models of natural image patches to whole image restoration. *2011 International Conference on Computer Vision*, 479–486. https://doi.org/10.1109/ICCV.2011.6126278

Zoran, D., & Weiss, Y. (2012). Natural images, gaussian mixtures and dead leaves. In F. Pereira, C. J. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/e97ee2054defb209c35fe4dc94599061-Paper.pdf