6.883 Science of Deep Learning – Spring 2018 February 28, 2018

Lecture 7: Generative Models and Expectation-Maximization

Lecturer: Constantinos Daskalakis

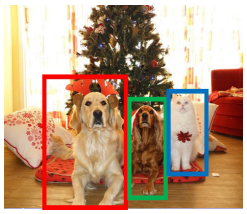
Scribes: Mucong Ding, Sirui Lu, Wei-Ning Hsu
(Revised by Andrew Ilyas and Manolis Zampetakis)

1 Introduction

1.1 Supervised vs Unsupervised Learning

	Supervised Learning	Unsupervised Learning
Data:	(x, y) x is data, y is label	raw data x, no labels!
Goal:	Learn a function to map $x \to y$	Learn some underlying structure
Examples:	Classification, regression, object detection, semantic segmentation, image captioning, etc.	Clustering, feature learning, dimensionality reduction, density estimation, etc.

(a) Classification: Cat



(b) Object Detection: {DOG, DOG, CAT}

(c) Semantic Segmentation: {GRASS, CAT, TREE, SKY}

(d) Image captioning: A cat sitting on a suitcase on the floor

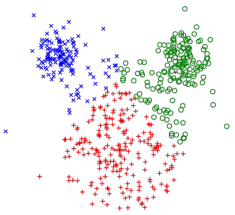
Figure 1: Supervised Learning

So far in this course, we've been talking about supervised learning, where the data is labeled. Specifically, we have a collection of pairs (x, y) where x is data, and y is the corresponding label. The goal is to learn the mapping from data to labels. Some examples of supervised learning are given in Figure 1. When training set consists of images, our goals can be: determining what do images contain (Figure 1a), identifying objects from images (Figure 1b), image captioning (Figure 1d), or finding segmentations (Figure 1c). All these examples are instances of supervised learning.

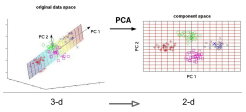
In contrast, in unsupervised learning, we only have raw data. Some examples of supervised learning are given in Figure 2, and include clustering (Figure 2a) and principal component analysis (Figure 2b).

Unsupervised learning is an essential topic in AI for the following two reasons:

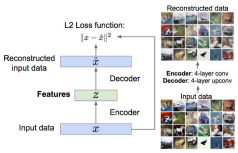
1. Training data is cheap, while labeling is an expensive task.
2. By solving unsupervised learning, we gain better understanding the structure of the data.



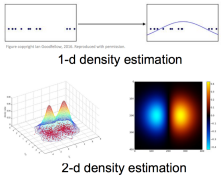
(a) K-means clustering



(b) Principal Component Analysis (Dimensionality reduction)



(c) Autoencoders (Feature learning)



(d) 1-d & 2-d density estimation

Figure 2: Unsupervised learning: (a) Clustering problems involve partitioning unlabeled data into groups based on similarity between the datapoints themselves. (b) Principal component analysis (PCA) aims to find the most important components of high-dimensional input data, and as such is often used for dimensionality reduction. (c) In deep learning, autoencoders (Figure 2c) aim to learn a representation (encoding) for a set of data, typically for dimensionality reduction and extracting crucial features. Autoencoders attempt to accomplish this by jointly optimizing an encoder, which reduces the dimensionality of the data, and a decoder, which tries to reconstruct the original data from the encoding (thus ensuring that the encoding does not lose information). (d) Finally, density estimation (Figure 2d) is another vital problem, which is the construction of an estimate, based on observed data, of an unobservable underlying probability density function.

1.2 Generative Models

One of the most important goals of unsupervised learning is understanding the inherent structure of the given unlabeled data. One popular approach to achieving this goal is the use of generative models. In this approach, we assume that our data has been generated by some unknown probability distribution, and our goal is to construct a model which captures this distribution as accurately as possible.

There are mainly two categories of generative model: explicit density estimators and implicit density estimators. Explicit density estimation attempts to learn the distribution $p_{\text{model}}(x)$ directly—the end result is a model which, for any input x, can estimate $p_{\text{data}}(x)$. Examples of explicit density estimators are image classifiers (which, as we recall, can be viewed as models of $p(\text{label}|\text{image})$, sequence models in natural language processing, and other likelihood-estimating models. In contrast, implicit density estimation models do not directly yield estimates of $p_{\text{data}}(x)$, but instead produce samples from $p_{\text{model}}(x)$ directly. Such models are sometimes the more natural approach to modeling probability distributions—for example in climate, weather, and ecology, our physical understanding of systems can be used to create implicit generative models via simulation (while by contrast assigning a likelihood to specific chains of events would be rather difficult).

1.3 Taxonomy of Generative Models

Despite the unified goal of understanding the underlying distribution of a dataset, different types of generative models are best suited for different tasks. In particular, exact problem circumstances often dictate uses of different solutions within generative modeling. These circumstances lead to the so-called "taxonomy" of generative models given in Figure 3. In this section, we briefly elaborate on this taxonomy in the context of the main models we will look at in this lecture.

Approaches to explicit density estimation. Explicit density estimation is the most common way to define a generative model. The most trivial example of explicit density estimation is to postulate your data comes from Gaussian distribution and try to find the mean and variance of this Gaussian. Under this hypothesis, the log-likelihood is well-behaved and the parameters of the model even have a closed form (namely, the empirical mean and variance). More broadly, there exists a class of models/probems

[1]Figure copyright Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.[2]

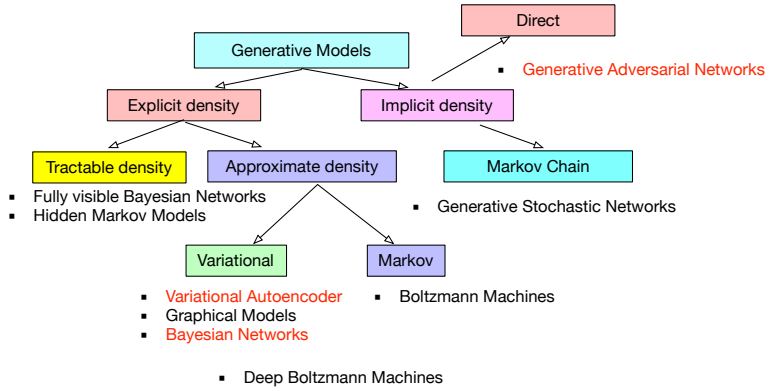


Figure 3: **Taxonomy of generative models**[1]: Generative models can differ by whether estimate density explicitly (by providing probability/likelihood) or implicitly (by providing samples). Within explicit models, learning the underlying distribution is sometimes tractable, otherwise an approximation method is needed (VAEs, which we discuss further later on, correspond to the case where approximation is chosen to be a variational one). Within implicit models, some models can transform an initial state into a sample, while others (such as GANs, which we also discuss later) generate samples from scratch.

for which we can provably find optimal parameters, such as (fully-visible) Bayesian Networks, and Hidden Markov Models.

On the other hand, there are many examples where the computation of a maximum likelihood estimate is a very hard or even simply intractable problem. In these cases, to get an explicit density estimate, we have to use some sort of heuristic or approximation in order to find optimal model parameters. In this lecture, we will discuss the use of a *variational* approximation to solve the explicit density estimation, starting with the classical Expectation-Maximization algorithm, and leading up to a discussion of Variational Autoencoders.

Approaches to implicit density estimation As we previously noted, sometimes constructing a model that provides densities explicitly is difficult or unrealistic. These cases are better suited to implicit generative models, which abandon the goal of explicitly modeling the system and instead attempt to construct a model which provides samples from a distribution approximating that of the natural data. Here, we'll look at implicit density estimation in the context of Generative Adversarial Networks (GANs).

2 Variational Inference

2.1 Pearson's crabs

The paradigmatic use of variational inference comes from a problem motivated by Karl Pearson in the late 19th century in his paper [5]. In this paper, Pearson set out to analyze data provided by the zoologist Weldon and his wife, regarding the forehead widths and heights of a population of crabs. When conducting his analysis, however, Pearson found something rather peculiar—contrary to the rest of the zoological data he had analyzed, the forehead widths and heights did not seem to follow a normal distribution. Instead, Pearson notes:

> *In the case of certain biological, sociological, and economic measurements there is, however, a well-marked deviation from this normal shape, and it becomes important to determine the direction and amount of such deviation. The asymmetry may arise from the fact that the units grouped together in the measured material are not really homogeneous. It may happen that we have a mixture of* $2, 3, \cdots, n$ *homogeneous groups, each of which deviates about its*

own mean symmetrically and in a manner represented with sufficient accuracy by the normal curve.

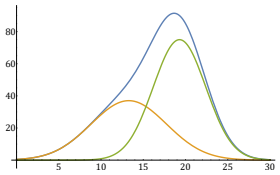


Figure 4: The analysis reproduces Pearson's original fit with two normal components. The presence of two components was interpreted by Pearson as evidence that there were two species of crabs.

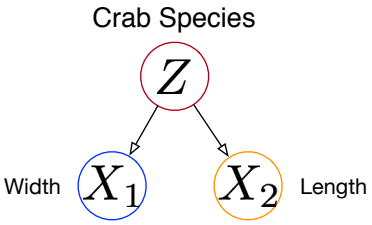


Figure 5: The probabilistic graphical model representation of Pearson's crabs. Z(species of a crab) is the only latent variable, while both X_1(width of a crab) and X_2(length of a crab) are observable variables.

This observation and the analysis that followed was (a) the first example of a so-called *mixture model* used to characterize a natural phenomenon, and (b) the very first example of a Bayesian network with a hidden variable, which Pearson used in order to explain the data. In particular, he interpreted the presence of two components as evidence that there were two species of crabs. The label of species was a hidden variable that Pearson couldn't observe, and based on this label, the measurement was drawn from a separate (Gaussian) distribution. Pearson's assumption about the presence of an unobserved latent variable allowed him to solve for the parameters of the model, and thus accurately model the measurement in question.

It turns out that the this "latent variable" observation extends well beyond this simple setting, and is essential to modeling a wide variety of natural and synthetic data. Unfortunately, while the presence of latent variables allows for better modeling of the world around us, it also makes the optimizing the parameters of these models significantly harder.

As an example, consider once again our model for the crab population, where an unobserved latent variable z (species) influences an observed variable x (forehead size). We have a good model of our observed variable given the latent one, i.e. $p_\theta(x|z)\ \forall z$, and we can also parameterize a distribution over the latent variable $p_\theta(z)$. Now, when we are given i.i.d. samples from the distribution, $\mathcal{S} = \{x_i\}$, our log-likelihood is given by:

$$\ell(S) = \sum_{\boldsymbol{x}\in\mathcal{S}} \log\left(\sum_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})\right).$$

Crucially, note that the summation inside the log in the above is necessitated by the fact that we have no access to the latent variable $\boldsymbol{z}$ (and thus need to marginalize it out of the likelihood). If z was observable, we could simply compute the log-likelihood of both variables as the sum of $\log(p_\theta(x|z)p_\theta(z))$. Unfortunately, the log of the summation above also makes the likelihood highly non-convex and difficult to optimize directly.

Variational inference is a heuristic used to circumvent this intractability, in which we opt to maximize a lower bound of the likelihood function, rather than optimizing the likelihood directly. The most prominent example of a variational inference algorithm is the *expectation-maximization* (EM) algorithm.

2.2 The Expectation-Maximization Algorithm

The expectation-maximization (EM) algorithm is one of the most popular ways to deal with the non-convexity of the likelihood function caused to the presence of latent variables. Although it is a single framework, the EM algorithm can be derived and interpreted in a variety of ways—here, we view EM as a way of constructing progressively better lower bounds for the likelihood and maximizing these lower bounds, thus ensuring that the likelihood continues to increase. EM can also be interpreted as the construction of artificial latent variables, or as an alternating maximization algorithm [**?**].

Consider our equation for the log-likelihoods of observable variables $\boldsymbol{x}$ given latent variables $\boldsymbol{z}$:

$$\ell_S(\theta) = \sum_{\boldsymbol{x}\in\mathcal{S}} \log\left(\sum_{\boldsymbol{z}} p_\theta(\boldsymbol{x},\boldsymbol{z})\right) \tag{1}$$

$$= \sum_{\boldsymbol{x}\in\mathcal{S}} \log\left(\sum_{\boldsymbol{z}} \frac{q(\boldsymbol{z})}{q(\boldsymbol{z})} p_\theta(\boldsymbol{x},\boldsymbol{z})\right) \qquad \text{for any distribution } q(\cdot) \tag{2}$$

$$= \sum_{\boldsymbol{x}\in\mathcal{S}} \log\left(\mathbb{E}_{\boldsymbol{z}\sim q}\left[\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}\right]\right). \tag{3}$$

Now, since $\log(x)$ is a concave function, we can apply Jensen's inequality, which says that for any concave function f and random variable X, $f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]$:

$$\ell_S(\theta) \geq \sum_{\boldsymbol{x}\in\mathcal{S}} \mathbb{E}_{\boldsymbol{z}\sim q}\left[\log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}\right)\right] \tag{4}$$

$$= \sum_{\boldsymbol{x}\in\mathcal{S}} \sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}\right) =: Q_S(\theta) \tag{5}$$

Thus, introducing the distribution $q(z)$ solves our problem and takes the summation outside of the logarithm. The question becomes: what should we choose $q(z)$ to be? Observe that our new function $Q_S(\theta)$ differs from the likelihood $\ell_S(\theta)$ only due to Jensen's inequality. As such, a natural way to proceed is to set $q(z)$ such that Jensen's inequality is *tight* at the current guess for the best parameter, $\theta^{(t)}$. Noting that Jensen's inequality is tight when X is a constant random variable, we find that we want:

$$p_{\theta^{(t)}}(\boldsymbol{x},\boldsymbol{z}) \;\alpha\; q(\boldsymbol{z}) \;\forall\; \boldsymbol{z}.$$

Combining this constraint with the fact that q must be a probability distribution gives that:

$$q(\boldsymbol{z}) = \frac{p_{\theta^{(t)}}(\boldsymbol{x},\boldsymbol{z})}{\sum_{\boldsymbol{z}} p_{\theta^{(t)}}(\boldsymbol{x},\boldsymbol{z})} = p_{\theta^{(t)}}(\boldsymbol{z}|\boldsymbol{x})$$

This is all we need to introduce the EM algorithm: at some timestep t when our current guess for the parameter θ is $\theta^{(t)}$, we alternate between:

1. **E Step:** For each sample $\boldsymbol{x} \in \mathcal{S}$, compute the distribution q as follows:

$$q(\boldsymbol{z};\boldsymbol{x},\theta^{(t)}) = p_{\theta^{(t)}}(\boldsymbol{z}|\boldsymbol{x})$$

2. **M Step:** Maximize the following lower bound on the likelihood, which is tight at $\theta^{(t)}$:

$$Q_S(\theta;\;\theta^{(t)}) = \sum_{\boldsymbol{x}\in\mathcal{S}} \sum_{\boldsymbol{z}} q(\boldsymbol{z};\;\boldsymbol{x},\theta^{(t)}) \log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z};\;\boldsymbol{x},\theta^{(t)})}\right)$$

$$\theta^{(t+1)} = \arg\max_{\theta}\; Q_S(\theta;\;\theta^{(t)})$$

We can show that using the above algorithm, we actually increase the likelihood $\ell_S(\theta)$ at each iteration. In fact, this turns out to be a straightforward argument. First, we know that $Q(\theta^{(t+1)};\theta^{(t)}) \leq \ell_S(\theta^{(t+1)})$ (since we proved via Jensen's inequality that Q lower bounds ℓ). Then, by construction we know that $Q(\theta^{(t+1)};\theta^{(t)}) \geq Q(\theta^{(t)};\theta^{(t)})$, and since we designed $Q(\cdot;\theta^{(t)})$ to be equal to $\ell(\theta)$ at $\theta^{(t)}$, we can combine these inequalities to yield $\ell(\theta^{(t+1)}) \geq \ell(\theta^{(t)})$.

An illustration of the EM algorithm is given in Figure 6.

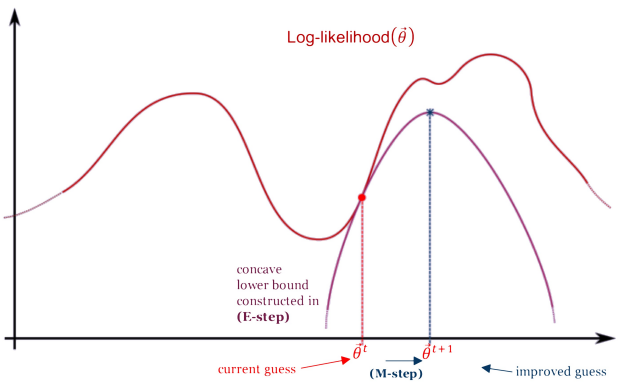


Figure 6: An illustration of the approach taken by the expectation-maximization (EM) algorithm in maximizing likelihood. The x-axis represents the parameter θ. The top (red) line is the log-likelihood $\ell_S(\theta)$. In the E step, we compute a lower bound of $\ell(\theta)$ using the current value of θ, $\theta^{(t)}$. This lower bound, which we denote $Q(\theta|\theta^{(t)})$, is shown as purple curve in the figure. In the M step, we maximize Q with respect to θ, and let $\theta^{(t+1)}$ be the next value of θ.

2.2.1 EM algorithm in the case of mixture of Gaussian distributions

To better illustrate the EM algorithm, we now apply it the aforementioned problem: learning a mixture of k Gaussians. Here, our parameters θ are the means and covariance matrices of the Gaussians in the mixture, $\{\mu_i\}$ and $\{\Sigma_i\}$. Now, suppose we are given a set of samples $\mathcal{S} = \{\boldsymbol{x_i}\}$ drawn from the mixture of Gaussians. (We'll assume the Gaussians are mixed uniformly; in reality we can also optimize over the mixing ratios α.)

Starting at some initial estimate $\theta^{(0)}$, the EM algorithm takes the following form:

1. **E Step:** We let the indicator variable z_{ij} indicate whether sample $\boldsymbol{x_i}$ was drawn from the j^{th} Gaussian of the mixture. Then,

$$q_{ij} := q(z_{ij}; \boldsymbol{x_i}, \mu_j^{(t)}, \Sigma_j^{(t)}) = p_{\mu_j^{(t)}, \Sigma_j^{(t)}}(z_{ij}|x_i) = \frac{\mathcal{N}(x_i; \mu_j^{(t)}, \Sigma_j^{(t)})}{\sum_{l=1}^{k} \mathcal{N}(x_i; \mu_l^{(t)}, \Sigma_l^{(t)})}$$

2. **M Step:** Plugging in the above into $Q(\theta)$ as derived above, and setting $\nabla_{\mu_j} Q(\theta) = 0$ and $\nabla_{\Sigma_j} Q(\theta) = 0$ gives the update:

$$\mu_j = \frac{\sum_i q_{ij} x_i}{\sum_i q_{ij}}$$

$$\Sigma_j = \frac{\sum_i q_{ij} (x_i - \mu_j)(x_i - \mu_j)^\top}{\sum_i q_{ij}}$$

Although EM algorithm is used widely and local convergence nearly follows by construction, there are few known global convergence guarantees for this method. In fact, global convergence for mixtures of two Gaussians with known covariance matrices was only established by relatively recent work (2016) [17]. In particular, the authors show that in the population model, where the algorithm is given access to infinitely many samples from the mixture, converges geometrically to the correct mean vectors. A simple, closed-form expressions for the convergence rate is also given. Using the convergence rate from their result gives that, in one dimension, ten steps of the EM algorithm initialized at infinity result in less than 1% error estimation of the means.

3 Variational Auto-encoders (VAEs)

3.1 Introduction

EM provides a method for managing low-dimensional, discrete latent variables; however, in many settings this turns out to be insufficient. Variational Auto-encoders (VAEs) provide a canonical way to factor in high-dimensional or continuous latent variables.

In this section, we'll consider the following running example: suppose we have a high-dimensional $\boldsymbol{x} \in \mathbb{R}^n$ depending on a latent $\boldsymbol{z} \in \mathbb{R}^m$ (where $m << n$, but is usually also high-dimensional). Then, the marginal likelihood of $\boldsymbol{x}$ is given by:

$$p_\theta(\boldsymbol{x}) = \int_{\boldsymbol{z}} p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z}, \tag{6}$$

Just as with EM, we can optimize the lower bound of the likelihood function; this time, however, we use a VAE to access this loss function, rather than exploiting Jensen's inequality (as we did in the last section).

3.2 Intuition: Auto-encoders

Recall that one of the crucial goals of unsupervised learning is to discover the hidden structure of data, and to learn a lower-dimensional feature representation from unlabeled training data. An *auto-encoder* provides a simple and intuitive model for solving this problem.

A standard auto-encoder consists of two key parts: an *encoder* and a *decoder*. The encoder is a function $E_\theta : \mathbb{R}^n \to \mathbb{R}^m$ (typically parameterized by a neural network) that maps the high-dimensional input $\boldsymbol{x}$ to the lower-dimensional latent space $\boldsymbol{z}$.

In turn, the decoder is a function $D_\theta : \mathbb{R}^m \to \mathbb{R}^n$ (usually also parameterized by a neural network) that attempts to map vectors from the latent space $\mathbb{R}^m$ *back* to their corresponding inputs in the higher-dimensional input space.

The key intuition behind this model is that if the decoder is able to successfully reconstruct the inputs from the latent space, then the encoder must have successfully "embedded" all the necessary information about the input into the latent space. To induce this sort of behavior, E_θ and D_θ are usually trained in tandem, minimizing some sort of reconstruction error between original inputs and the corresponding output of the decoder. Concretely, we often solve the following optimization problem:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\boldsymbol{x}} \left[d\left(\boldsymbol{x}, D_\theta\left(E_\theta(\boldsymbol{x})\right)\right)\right],$$

under some distance measure d. Auto-encoders are often used in tasks requiring dimension reduction (throwing away the decoder leaves just an encoder which is able to compress the "useful" information contained in $\boldsymbol{x}$ into a smaller latent space), or in visualization.

In the following, we consider how we might be able to generate realistic vectors $\boldsymbol{x} \in \mathbb{R}^n$. It turns out that this exactly the problem that the VAE attempts to solve.

3.3 Decoders as Generative Models

In the previous section, we introduced the decoder of an auto-encoder as a function that attempts to recover an encoded input $\boldsymbol{z} := E_\theta(\boldsymbol{x})$. Note that perhaps the most important role of the decoder was just to ensure that the encoder embedded useful information into the latent space.

In the context of the *variational* auto-encoder, however, the decoder takes on a much larger role in the training process. First, we define a *prior* distribution over the latent space $\mathbb{R}^m$, which we denote $p_\theta(\boldsymbol{z})$. The role of the decoder then, is to provide access to $p_\theta(\boldsymbol{x}|\boldsymbol{z})$.

Assuming that the decoder is able to somewhat accurately capture $p(\boldsymbol{x}|\boldsymbol{z})$, the VAE framework provides a simple way of generating new points in $\mathbb{R}^n$. In particular, we can sample a vector $\boldsymbol{z} \sim p_\theta(\cdot)$, then This raises the question: how can we train a decoder to match this conditional likelihood function?

3.4 Encoders as Inference Models and the Variational Lower Bound

To tackle the intractable likelihood function issue, VAEs pair the generative model provided by the decoder with a complementary *inference* model, $q_\phi(z|x)$ meant to approximate the true posterior $p_\theta(z|x)$. Just as with $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, we typically parameterize q_ϕ with the weights of a neural network—in particular, we reuse the *encoder* from the auto-encoder setup to model this posterior distribution.

Recall our original problem was dealing with the intractable likelihood function given in Equation 6. Equipped with encoder and decoder networks as described, we can now derive a lower bound for the log-likelihood function that we can optimize instead. We start by writing the log-likelihood as an expectation over the latent variable $\boldsymbol{z}$, for a single sample $x := x^{(i)}$:

$$\log p_\theta(x) = \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x)\right] \tag{7}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\right] \qquad \text{via Bayes': } \frac{P(A|B)P(B)}{P(A)P(B|A)} = 1 \tag{8}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\frac{q_\phi(z|x)}{q_\phi(z|x)}\right] \qquad \text{multiplying by 1} \tag{9}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \tag{10}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right) \tag{11}$$

$$\geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z)) \qquad \text{since } D_{KL}(\cdot||\cdot) \geq 0 \tag{12}$$

$$=: \mathcal{L}(x;\theta,\phi), \tag{13}$$

Examining the loss function. First, we examine each term in Equation 11 separately. The first term can be viewed as a re-derivation of the (auto-encoder) reconstruction loss. In particular, we want to ensure that the generative model $p_\theta(x|z)$ (the decoder), assigns high probability to x given the encoding $q_\phi(z|x)$. The second term represents the distance between the distribution induced by the encoder, and our enforced prior distribution on z. The third term penalizes the distance between the posterior $q_\phi(z|x)$, and the posterior induced by p_θ, $p_\theta(z|x)$. This term is actually intractable to compute, and is thus what leads us to having a lower bound on the log-likelihood, rather than the log-likelihood itself.

We now seek closed-form expressions for the first and second terms. It turns out that these term can actually be greatly simplified if we force the distributions in question to be Gaussian. Setting $p_\theta(z) \sim \mathcal{N}(0,1)$ and $q_\phi(z|x) \sim \mathcal{N}(\mu_{z|x}, \text{diag}(\sigma^2_{z|x}))$ allows us to simplify the second term. Recall that the KL divergence between two multivariate normal distributions with means μ_1 and μ_2 and covariance matrices Σ_1 and Σ_2 is given by:

$$D_{KL}(\mathcal{N}_1||\mathcal{N}_2) = \frac{1}{2}\left[\text{tr}\left[\Sigma_2^{-1}\Sigma_1\right] + (\mu_2-\mu_1)^\top\Sigma_2^{-1}(\mu_2-\mu_1) - d + \log\frac{|\Sigma_2|}{|\Sigma_1|}\right]. \tag{14}$$

Substituting $(0, I)$ for (μ_2, Σ_2) and $(\mu_{z|x}, \text{diag}(\sigma^2_{z|x}))$ for (μ_1, Σ_1) yields:

$$D_{KL}(q_\phi(z|x)||p_\theta(z)) = \frac{1}{2}\left[\left|\left|\sigma_{z|x}\right|\right|^2 + \left|\left|\mu_{z|x}\right|\right|^2 - d - \left|\left|\log\sigma_{z|x}\right|\right|^2\right], \tag{15}$$

where the logarithm in the above is taken element-wise. Similarly, setting $p_\theta(x|z) \sim \mathcal{N}(\mu_{x|z}, \boldsymbol{I})$ allows us to write the first term (which we suggested might correspond to the reconstruction loss) as:

$$\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] = \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\frac{1}{2}\left(\log|\boldsymbol{I}| + (x-\mu_{x|z})^\top\boldsymbol{I}(x-\mu_{x|z}) + d\log(2\pi)\right)\right] \tag{16}$$

$$= -\frac{1}{2}\mathbb{E}_{z\sim q_\phi(z|x)}\left[\left|\left|x-\mu_{x|z}\right|\right|^2\right] - d\log(\sqrt{2\pi}), \tag{17}$$

which is *precisely* the ℓ_2 reconstruction loss if we let the output of the decoder be $\mu_{x|z}$!

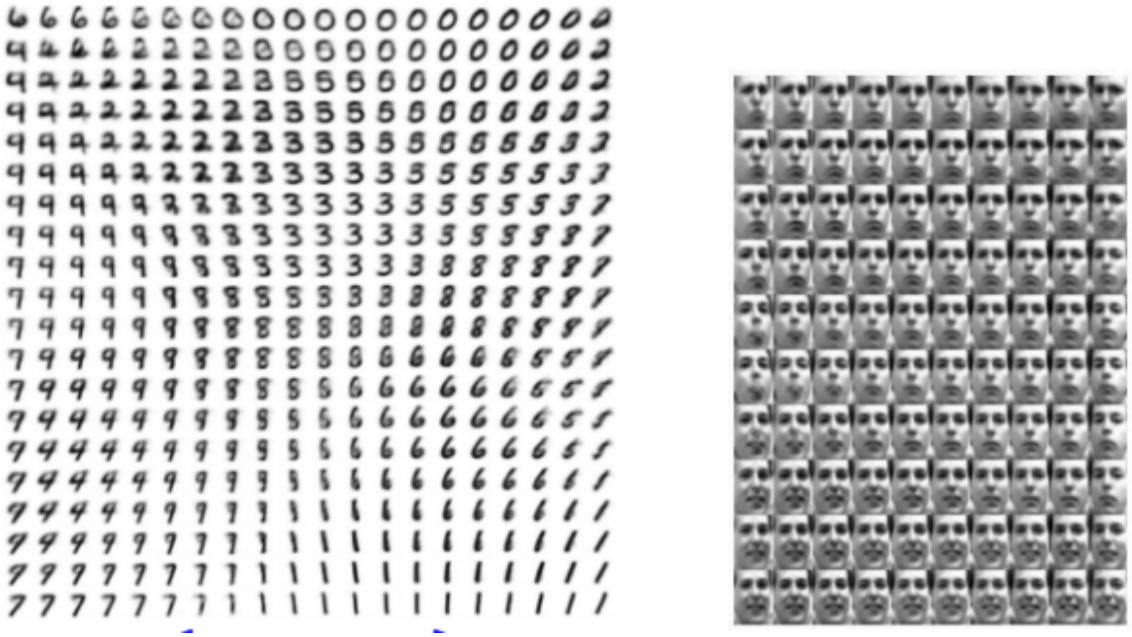
Ensuring backpropagation. A natural choice for maximizing $\mathcal{L}$ where p_θ and q_ϕ are neural networks would be gradient ascent. The barrier that we face with this approach, however, is an inability to differentiate through the random sampling in the expectation, $z \sim q_\phi(\cdot|x)$. Fortunately, the same simplifying assumption we made to make $D_{KL}(q_\phi(z|x)||p_\theta(z))$ tractable (namely that $p_\theta(z)$ and $q_\phi(z|x)$ are normally distributed) also lends itself to circumventing the non-differentiability issue. In order to back-propagate the gradient to the encoder, one can use the following reparameterization trick:

$$\tilde{z} = \boldsymbol{\mu}_{z|x} + \boldsymbol{\sigma}_{z|x} \cdot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 1), \tag{18}$$

which rewrites a sample drawn from $q_\phi(z|x)$ as a deterministic function of its mean and covariance, with an auxiliary variable ϵ. Thus, by making the encoder network output a mean and covariance explicitly and using these to parameterize a normal distribution, we can ensure that gradients can be propagated through the entirety of the expectation.

3.5 Generating Data

After a VAE model is trained, we can traverse the latent space in order to see what factors each latent space dimension captures. Figure 7 shows examples of data generated by two VAEs, trained on the MNIST digit dataset and a face dataset respectively. In the latter example, we can see that varying one dimension changes the degree of the smile while varying the other dimension changes the head pose.



(a) VAE-generated numbers

(b) VAE-generated faces

Figure 7: Data generated by VAEs.

3.6 Summary: VAEs

A variational auto-encoder is a probabilistic version of an auto-encoder, which (in addition to performing dimensionality reduction), enables us to sample from the model to generate data. The VAE accomplishes this task by considering a latent variable model $\boldsymbol{z} \to \boldsymbol{x}$, and deriving a lower bound on an intractable log-likelihood.

Disadvantages and future work. Samples generated by VAEs tend to be blurry or noisy, particularly when compared to other generative models such as generative adversarial networks (GANs).

Furthermore, while the variational lower bound is a tractable objective for optimization, it only relates with the true objective of interest (the marginal likelihood) loosely. The gap between the variational lower bound and the marginal likelihood is determined by how well the inference model can approximate the true posterior, measured in terms of the KL divergence.

To improve the capacity of variational autoencoders, one line of research has focused on introducing more powerful inference models, instead of neural network-parameterized factorial Gaussian distributions. A few examples are normalizing flows [8] and inverse auto-regressive flows [9] that transforming a simple Gaussian into more complex ones. Another line of research takes into account the characteristics of the data being modeled, and introduces graphical models of richer structures, such as hidden Markov models [11], Kalman filters [10], factorized hierarchical prior [12], state space models [13].

4 Generative Adversarial Network (GAN)

What if we give up on explicitly modeling the density, and just want the ability to sample from an underlying distribution? This is precisely the question that motivates the approach of generative adversarial networks (GANs), which do not work with any explicit density function. Instead, using inspiration from game theory, GANs attempt to learn to generate from a training distribution by setting up a two-player game between two concurrently trained agents.

One agent, the *generator* G_θ, is designed to take in a simple, low-dimensional random input (for example, $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$) and transform it into a high-dimensional structured signal. This structured signal is "mixed in" with the training distribution, and is then fed to a *discriminator* D_w, which in turn tries to discern between the inputs generated by G_θ, and those directly from the training set. The generator receives rewards for "fooling" the discriminator (i.e. generating an image that the discriminator labels as being from the training set). Conversely, the discriminator receives rewards for correctly classifying between real and fake inputs.

The method by which this reward is incurred by G_θ and D_w is a subject of intense research, and varies from implementation to implementation. In the original GAN formulation (Goodfellow, 2014), a log-likelihood based objective was used. Here, we give the formulation of Wasserstein-GAN (WGAN), one of the most popular and simple objective functions for training GANs. In the next lecture, we elaborate more on the ties between GAN training and traditional game theory.

4.1 Wasserstein-GAN

Suppose the data we are trying to model resides in Euclidean space $\mathbb{R}^n$. We denote the true high-dimensional distribution of data by F and the modeled distribution (i.e., the distribution of the outputs of the generator network $G_{\theta_g}(z)$) as Q.

Since our goal is for the generator G_θ to learn the true distribution, a reasonable approach would simply be to minimize the statistical distance between F and Q. In Wasserstein-GAN (WGAN), the authors find that the *Wasserstein distance* has certain properties that make it well-suited for measuring this distance. Formally, the Wasserstein distance between F and Q is defined as:

$$W(F, Q) = \inf_{\gamma \in \Pi(F,Q)} \Big(\mathbb{E}_{(X,Y)\sim\gamma} \big[\|X - Y\|_1 \big] \Big), \tag{19}$$

where $\Pi(F, Q)$ is the set of all distributions in $\mathbb{R}^n \times \mathbb{R}^n$ whose marginals are F and Q respectively, i.e., the couplings of F and Q. This definition is very intuitive, and essentially represents the ℓ_1 distance under the best possible coupling of the two distributions.

Unfortunately, optimizing this kind objective function is not practical—in particular, calculating an infimum over the set of all couplings is almost always intractable. It turns out, however, that there is a dual representation of the Wasserstein distance. Using *Kantorovich duality*, we can $W(F, Q)$ as the following supremum over the set of all 1-Lipschitz functions from $\mathbb{R}^n$ to $\mathbb{R}$:

$$W(F, Q) = \sup_{D:\mathbb{R}^n \to \mathbb{R}, \text{1-Lipschitz}} \Big(\mathbb{E}_{X\sim F} \big[D(X) \big] - \mathbb{E}_{X\sim Q} \big[D(X) \big] \Big). \tag{20}$$

Intuitively, this supremum represents the performance of the best 1-Lipschitz "separator" of the two distributions. In a perfect world, we could simply train a G_θ that would minimize this Wasserstein distance,

$$\theta = \inf_{\theta_g} \sup_{D:\mathbb{R}^n \to \mathbb{R}, 1\text{-Lipschitz}} \Big(\mathbb{E}_{X \sim F}\big[D(X)\big] - \mathbb{E}_{z \sim \mathcal{N}(0,\mathbb{I})}\big[D(G_{\theta_g}(z))\big] \Big). \tag{21}$$

In practice, we actually run into the same problem as with the primal formulation of the Wasserstein distance: a seemingly intractable supremum over the set of all 1-Lipschitz functions. This intractability, however, turns out to be circumventable (though with a few caveats). In particular, we can instead minimize an *upper bound* on the Wasserstein distance between F and Q, by taking the inner supremum over a smaller model class—the set of 1-Lipschitz deep neural networks. This simplification yields a simple min-max objective that can be solved through first-order methods:

$$\inf_{\theta_g} \sup_{\theta_d} \Big(\mathbb{E}_{X \sim F}\big[D_{\theta_d}(X)\big] - \mathbb{E}_{z \sim \mathcal{N}(0,\mathbb{I})}\big[D_{\theta_d}(G_{\theta_g}(z))\big] \Big). \tag{22}$$

The Lipschitzness of the neural network "discriminator" can be enforced in various ways, such as weight clipping, gradient penalties, or spectral normalization of the weights. To train WGAN, we simply alternate between gradient ascent on w, and gradient descent on θ. While this approach seems to work in practice, research has shown that the simplifying assumptions made when training GANs often reflects back through difficulty training or inability in learning distributions. We revisit these issues in depth in the coming lectures.

References

[1] Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[2] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894.

[6] Pearson's polynomial, http://blog.mrtz.org/2014/04/22/pearsons-polynomial.html, 22-04-2014

[7] Karl Pearson's crab data, http://icarus.math.mcmaster.ca/peter/mix/demex/excrabs.html

[8] Rezende, Danilo Jimenez, and Shakir Mohamed. "Variational inference with normalizing flows." arXiv preprint arXiv:1505.05770 (2015).

[9] Kingma, Diederik P., et al. "Improved variational inference with inverse autoregressive flow." Advances in Neural Information Processing Systems. 2016.

[10] Fraccaro, Marco, et al. "A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning." Advances in Neural Information Processing Systems. 2017.

[11] Ebbers, Janek, et al. "Hidden Markov Model Variational Autoencoder for Acoustic Unit Discovery." Proc. Interspeech 2017 (2017): 488-492.

[12] Hsu, Wei-Ning, Yu Zhang, and James Glass. "Unsupervised Learning of Disentangled and Interpretable Representations from Sequential Data." Advances in neural information processing systems. 2017.

[13] Krishnan, Rahul G., Uri Shalit, and David Sontag. "Structured Inference Networks for Nonlinear State Space Models." AAAI. 2017.

[14] Wikipedia contributors, "Generative model," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Generative_model&oldid=837781569 (accessed April 25, 2018).

[15] Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." Journal of the royal statistical society. Series B (methodological) (1977): 1-38.

[16] Balakrishnan, Sivaraman, Martin J. Wainwright, and Bin Yu. "Statistical guarantees for the EM algorithm: From population to sample-based analysis." The Annals of Statistics 45.1 (2017): 77-120.

[17] Daskalakis, Constantinos, Christos Tzamos, and Manolis Zampetakis. "Ten steps of EM suffice for mixtures of two Gaussians." arXiv preprint arXiv:1609.00368 (2016).