**10-707: Advanced Deep Learning** **Spring 2020**

# Lecture 8: February 10

*Lecturer: Andrej Risteski* *Scribes: Ini Oguntola*

**Note**: *LaTeX template courtesy of UC Berkeley EECS dept.*



## 8.1 Unsupervised Learning

For supervised learning our goal is clear: learn a function that maps inputs to outputs. For unsupervised learning our goal is much less clear. If there are no labels, what does it mean to learn?

In general, there are three types of things we can hope to do:

- **Structure learning:** Fit a parameterized structure (e.g. clustering, low-dimensional subspace, manifold) to data to reveal something meaningful about data
- **Distribution learning:** Learn a (parameterized) distribution close to data generating distribution
- **Representation / feature learning:** Learn a (parametrized) distribution that implicitly reveals an "embedding"/"representation" of data for downstream tasks

### 8.1.1 Structure Learning

**Goal:** to fit a parametrized structure (e.g. clustering, low-dimensional subspace) to data to reveal something meaningful about data

One of the most classical ways to do structure learning is with what is called *principal component analysis (PCA)*. PCA looks to explain away the data by looking for the directions of highest variance. By only using a few of these directions, we can project to a low-dimensional subspace that explains most of the variance in our data.

Another classical type of structure learning is *clustering*, which looks to the group data into clusters in a way such that samples in the same cluster are similar to each other, and samples in different clusters are dissimilar.

Today, these techniques are often used for visualization, projecting high-dimensional data to 2D or 3D space.

### 8.1.2 Distribution Learning

**Goal:** Learn a (parameterized) distribution close to data generating distribution

Here are some typical choices of parameterized distributions:

**Classical Choices:**

- *Fully-observed* graphical models (both undirected and directed). The most common are Markov random fields, which are models that encode some sparse independence structure (e.g. "A is independent of other variables given B and D").
- *Latent-variable* graphical models (e.g. mixture models, sparse coding, topic models), where the data is "simple" when conditioned on some unobserved latent variables.

**Semi-Modern Choices:**

- Deep Boltzmann machines and deep belief networks (graphical model analogues of deep neural networks)
- (Variational) autoencoders: model data by enforcing a latent space "bottleneck"
- Energy models (capture dependencies between variables with some measure of "energy" for each configuration of the variables)

**Modern Choices:**

- Generative adversarial networks
- Autoregressive models (e.g. pixelRNN, pixelCNN)
- Flow models

#### 8.1.2.1 GANs

One popular application of GANs is for generating images of faces. GANs trained on the CelebA faces dataset have made significant progress throughout the past few years in producing realistic looking facial images, as shown in Figures 8.1 and 8.2. BigGAN is another GAN model trained on ImageNet [AB19], and is more or less the state of the art in generating photorealistic images (Figure 8.3).

CycleGAN is a conditional generative model that can be used for style transfer [JYZ17] (e.g. adding zebra stripes to an image/video of a horse, or transforming a natural image into the style of a Monet or Van Gogh painting as shown in Figure 8.4). A similar idea has even been applied to transfer facial expressions from one person to another, allowing for real-time reenactment [JT16](Figure 8.4).

### 8.1.3 Representation Learning & Self-Supervised Learning

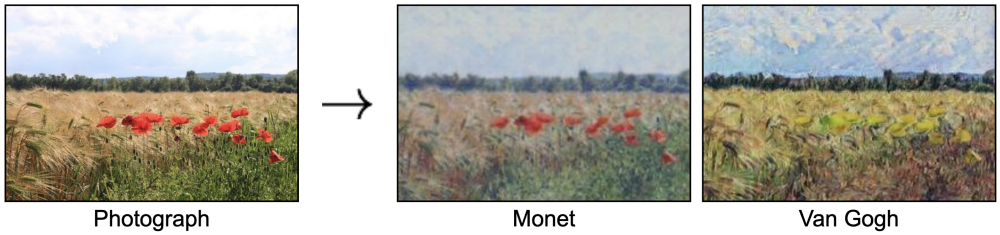
**Goal:** Given unlabeled data, design supervised tasks that induce a good representation for downstream tasks

There is no good mathematical formalization for this type of learning, but the intuition is to "force" the predictor used in the task to learn something "semantically meaningful" about the data.


Figure 8.1: 4 years of progression on faces [MB17]

Figure 8.2: whichfaceisreal.com

Figure 8.3: BigGAN [AB19]




(a) CycleGAN [JYZ17])


(b) Face2Face [JT16]

Figure 8.4: Style Transfer with Conditional GANs

Many modern ideas about representation learning have come out of NLP. Typically one would perform a task such as predicting the next word given previous 5 words, or predicting a middle word given surrounding 5 words, etc.

There are many examples of representation learning in vision as well, which are often quite different in nature. One example is rotation prediction (i.e. predicting the 2D rotation applied to some input image), which has been shown to provide a powerful supervisory signal for semantic feature learning in images. Another example is a "jigsaw puzzle", which takes various patches of an image and tries to predict the relative position of one piece with respect to another.

#### 8.1.3.1 What makes a "good" representation?

It is not obvious what makes a representation "good". In many ways this can be philosophical, rather than mathematical.

A phrase often used in representation learning is "semantically meaningful". The idea is that with semantically meaningful features, training a linear classifier on top of them should work well for many (if not most) reasonable tasks.

Another common word is "disentangled". The intuition here is that in a generative setting, we should be able to "vary" features independently, and generate an output that is reasonable. However, this is controversial and not well understood. Are the features disentangled in priors? Posteriors? It is not clear what "disentangled" even means. It has even been shown that models can be reparameterized to produce arbitrarily bad entangling [FL19].

Something else to look for are indirect benefits, such as improvement in sample complexity. If we wish to use learn features to train a supervised model on top of, a good representation could save us on sample complexity. For example, consider a recommender setting (like Netflix) where we essentially want to infer user ratings for movies they have not rated yet. A common approach is to use nearest neighbors, which works well if we have many ratings per user, but a good representation could allow us to do it well with only few ratings per user.

## 8.2 Relationships Between the Tasks

**Structure learning and feature learning often blend**. For example, low dimensional features in PCA or the cluster a point belongs to in clustering can be viewed as features.

**Structure learning / feature learning is in general weaker than distribution learning**. For example, methods like PCA or clustering cannot be used to generate new samples.

**Feature, not a bug:** It has been argued that self-supervised learning works because the task we are solving is easier (both computationally and statistically).

**Distribution learning often implies representation learning.** Many distributions we fit are latent-variable models. For instance, say we are modeling the joint distribution between some latent variables h and the observed data x:

$$P_\theta(x, h) = P_\theta(h)P_\theta(x \mid h) \tag{8.1}$$

The latent variables are often viewed as a "representation".

The posterior distribution $P_\theta(h \mid x)$ captures a distribution over representations, given some values of the observed data. However, a-priori, this distribution is not an easy distribution to approximate / sample from!

$$P_\theta(h \mid x) = \frac{P_\theta(h)P_\theta(x \mid h)}{\int_{h'} P_\theta(h')P_\theta(x \mid h')} \tag{8.2}$$

$\int_{h'} P_\theta(h')P_\theta(x \mid h')$ is often a hard high-dimensional sum / integral.

### 8.2.1 Representation Learning and Distribution Learning

Common ways to get a handle of the posterior distribution $P_\theta(h \mid x)$ are variational methods and MCMC methods.

**Question:** How accurately do we need to approximate $P_\theta(h \mid x)$ if we want a good representation?

**Answer:** Let $Q_\theta(h \mid x)$ be an approximation of $P_\theta(h \mid x)$ such that:

$$KL(Q_\theta(h \mid x) \,\|\, P_\theta(h \mid x)) \leq \epsilon \tag{8.3}$$

Recall the definition of KL divergence:

$$KL(q \,\|\, p) = \mathbb{E}_q \log\left(\frac{q}{p}\right) \tag{8.4}$$

Suppose the way we will use the representation $h$ is to solve classification tasks. Namely, say the labels for task are:

$$(x_t, c(h_t)) \quad \text{where } (x_t, h_t) \sim P_\theta(x, h) \text{ and } c(h_t) \in \{-1, +1\} \tag{8.5}$$

In other words, the labels are a function of the *latent* variables, and the observables are the data.

The natural way to measure the distance with an eye towards classification tasks is total variation distance:

$$TV(Q_\theta(* \mid x) \parallel P_\theta(* \mid x)) = \sup_\Omega \left|\Pr_{h\sim Q_\theta(*|x)}[\Omega] - \Pr_{h\sim P_\theta(*|x)}[\Omega]\right| \tag{8.6}$$

Why is this useful? Take $\Omega = \{\text{“}C(h) \text{ is the correct label”}\}$.

$\Pr_{h\sim Q_\theta(*|x)}[\Omega]$ is the probability of having the correct answer using $Q_\theta(* \mid x)$. Then $\left|\Pr_{h\sim Q_\theta(*|x)}[\Omega] - \Pr_{h\sim P_\theta(*|x)}[\Omega]\right|$ is the difference in probability of having the correct answer between $Q_\theta(* \mid x)$!

Hence, $TV(Q_\theta(h \mid x) \parallel P_\theta(h \mid x)) \le \epsilon'$ implies that the difference in performance between using $P_\theta(* \mid x)$ and $Q_\theta(* \mid x)$ is at most $\epsilon'$ (on the input sample $x$).

Hence, $\mathbb{E}[TV(Q_\theta(h \mid x) \parallel P_\theta(h \mid x))] \le \epsilon'$ implies that the difference in performance on the classification task is at most $\epsilon'$.

To conclude, by Pinsker's inequality, we have:

$$TV(Q_\theta(h \mid x) \parallel P_\theta(h \mid x)) \le \sqrt{\frac{1}{2} KL(Q_\theta(h \mid x) \parallel P_\theta(h \mid x))} \le \sqrt{\frac{1}{2}\epsilon} \tag{8.7}$$

So, if we want a good performance on classification task, we need an extremely close approximation to the posterior!

## 8.3 Training Techniques

There are two typical families of training algorithms: likelihood-based and likelihood-free.

**Likelihood-based:** maximize the likelihood of the data under the model (possibly with some approximations).

$$\max_\theta \sum_{\text{samples } x_i} \log p_\theta(x_i) \tag{8.8}$$

Typical used approximations are variational inference (optimize tractable deterministic approximation of posteriors) and MCMC methods (idea is to approximate difficult quantities like posteriors with sampling).

**Likelihood-free:** use a surrogate loss. In GANs, we train a discriminator to tell real and generated samples apart. In noise-contrastive training, we encourage the model to put probability mass away from “fake” samples.

## 8.4 The Classics

### 8.4.1 PCA

**Goal:** find a k-dimensional (linear) subspace explaining most of the variance in the data.

For simplicity we will assume that the data is centered; that is $\mathbb{E}[x] = 0$ (we can just subtract the mean).

As a warmup, let $k = 1$. Then we have:

$$\max_{v:\|v\|=1} \frac{1}{m} \sum_{\text{samples } x_i} \langle v, x_i \rangle^2 \tag{8.9}$$

where $\mathbb{E}[\langle v, x \rangle^2]$ is the variance.

Now for the general case, we have:

$$\max_{\{v_1 \dots v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} (\text{length of } x_i \text{ on } span(v_1 \dots v_k))^2 \tag{8.10}$$

$$= \max_{\text{orthonormal } \{v_1 \dots v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^{k} \langle x_i, x_j \rangle^2 \tag{8.11}$$

where $\mathbb{E}[\sum_j \langle v_j, x \rangle^2]$ is the variance.

We can compute this efficiently using *singular value decomposition*. Rewrite above equation as:

$$\max_{\text{orthonormal } \{v_1 \dots v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^{k} \langle x_i, x_j \rangle^2 \tag{8.12}$$

$$= \max_{\text{orthonormal } \{v_1 \dots v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^{k} (v_j^T x_i)(x_i^T v_j) \tag{8.13}$$

$$= \max_{\text{orthonormal } \{v_1 \dots v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^{k} v_j^T (x_i x_i^T) v_j \tag{8.14}$$

$$= \max_{\text{orthonormal } \{v_1 \dots v_k\}} \sum_{j=1}^{k} v_j^T \left( \frac{1}{m} \sum_{\text{samples } x_i} x_i x_i^T \right) v_j \tag{8.15}$$

$$= \max_{\text{orthonormal } \{v_1 \dots v_k\}} \sum_{j=1}^{k} v_j^T D v_j \tag{8.16}$$

If D is diagonal (with positive entries), then if we take into account that $(v_j)_i^2 = 1$, it easy to see that the max is $\sum_{j=1}^{k} D_{jj}$:

$$\sum_{j=1}^{k} v_j^T D v_j = \sum_j \sum_i (v_j)_i^2 D_{ii} \leq \sum_{j=1}^{k} D_{jj} \tag{8.17}$$

In general we can reduce to diagonal case by taking the SVD of $D = U \tilde{D} U^T$, where it is easy to see the max is $\sum_{j=1}^{k} \lambda_j(D)$.

### 8.4.2 Clustering

**Goal:** group the data into clusters of nearby points

There are at least three things needed for clustering:

1. A *proximity measure*: either
   - similarity measure $s(x_i, x_j)$, large if $x_i, x_j$ are similar
   - dissimilarity (distance) measure $d(x_i, x_j)$, small if $x_i, x_j$ are similar
2. A *criterion function* to evaluate a clustering
3. An *algorithm* to compute clustering

#### 8.4.2.1 K-Means Clustering

If the distance metric is the Euclidean distance, and the measure of cohesion is the average distance from the centroid: we get the k-means objective:

$$\text{argmin}_{\{r_{nk},\boldsymbol{\mu}_k\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \tag{8.18}$$

We can compute clusters with a natural iterative algorithm, which as we will see later is a variant of the EM (expectation-maximization) algorithm:

**Input**: Data set $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(m)} | x^{(i)} \in \mathbb{R}^n\}$
**Output**: Cluster centroids $\mu_{i=1,\cdots,k} \in \mathbb{R}^n$;Cluster assignments $c \in \mathbb{Z}$
Initialize $k$ cluster centroids $\mu_1, \cdots, \mu_k \in \mathbb{R}^n$ randomly from $X$;
**repeat**
  **for** $i = 1, \cdots, m$ **do** `// Update cluster assignments`
    set $c^{(i)} = arg\ \underset{j}{min} \|x^{(i)} - \mu_j\|^2$;
  **end**
  **for** $j = 1, \cdots, k$ **do** `// Update cluster centroids`
    set $\mu_j = \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}}$;
  **end**
**until** *Convergence*;
**return** $\mu$ and $c$;

**Algorithm 1:** Algorithm of batch-version for K-means

Two weakness of k-means clustering are:

1. It is very sensitive to outliers
2. It is not suitable for non-spherical clusters

## References

[MB17] M. Brundage, S. Avin, J. Clark, H. Toner, P. Eckersley, B. Garfinkel, A. Dafoe, P. Scharre, T. Zeitzoff and B. Filar, and others, "The malicious use of artificial intelligence: Forecasting, prevention, and mitigation," *arXiv preprint arXiv:1802.07228*, 2017.

[AB19] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *7th International Conference on Learning Representations*, 2019.

[JYZ17] J.Y. Zhu, T. Park, P. Isola, and A.A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[JT16] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Niessner, "Face2face: Real-time face capture and reenactment of rgb videos," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[FL19] F. Locatello, M. Lucic, R. Gunnar, S. Gelly, B. Schölkopf, and O. Bachem, "Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations," *Proceedings of the 36th International Conference on Machine Learning*, 2019.