6.7900 Fall 2024: Lecture Notes 20

1 Auto-regressive Generative Models

Generative models are all the rage now. Examples in natural sciences:

- Generate molecules with desired properties.
- Predict conformation of a protein and a drug

One kind of generative modeling is **auto-regressive**, where we generate a sequence of predictions one step at a time, using previously predicted parts of the sequence to generate the next part.

A distribution over all sequences of length n can be decomposed as follows:

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

and therefore, a generative model that auto-regressively predicts the next part of the sequence at step i would need to sample from:

$$P(x_i|x_{i-1} \dots x_1).$$

Example: Auto-regressive language modeling might generate a sequence of text in the following way (model inputs $\rightarrow$ model outputs):

() $\rightarrow$ You sampled from P($\cdot$)
You $\rightarrow$ are sampled from P($\cdot$| you)
You are $\rightarrow$ taking sampled from P($\cdot$| You are)
You are taking $\rightarrow$ this sampled from P($\cdot$| you are taking)

We might be interested in first learning and then sampling from $P(x_i|x_{i-1} \dots x_1)$. However, there are two challenges in modeling this distribution as sequences grow long. First, it is challenging to model distributions with arbitrary length histories. Second, it is challenging to model distributions with long histories.

A solution to these problems is to make additional modeling assumptions. For instance, in a bigram model, each part of the sequence only depends on the last part:

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

Similarly, the decomposition for a trigram model is as follows:

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\ldots P(x_n|x_{n-1}x_{n-2}).$$

These assumptions come at a cost to performance. However, neural models are able to handle much longer histories. For instance, state of the art transformers can generate sequences of length 100K or more.

2 Two Kinds of Generative Models

The first class of generative models explicitly learns the likelihood function $p(x|\theta)$. Auto-regressive models fall into this class, which as we saw above, explicitly learn $P(x_i|x_{i-1}\ldots x_1)$.

In a second class of generative models, $p(x|\theta)$ is implicitly learned. For this class of models, we usually first sample from some known distribution to produce a latent z, and then learn and transformation or procedure to generate x from z. For instance, we may learn the transformation $p(x|z,\theta)$.

In the next sections, we will explore the second class in more detail.

3 Kernel Density Estimation

Say I have a dataset of one-dimensional samples $\mathcal{D} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$. How can I use these samples to model the distribution that produced them? I would like to model this distribution in such a way that I can later sample from it.

One simple idea is to expand each data point into a local density "kernel". Define a standard normal kernel to be

$$K(x) = N(x|0,1).$$

A kernel centered at μ with standard deviation h is therefore:

$$\frac{1}{h}K(\frac{x-\mu}{h}) = N(x|\mu, h^2)$$

If each data point $x^{(i)}$ contributes some density to nearby x values, a kernel that captures this density for each $x^{(i)}$ is:

$$\frac{1}{h}K(\frac{x-x^{(i)}}{h}).$$

We can sum these contributions over all data points to get a model of the original distribution:

$$\hat{P}(x|h) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h}K(\frac{x - x^{(i)}}{h})$$

Note: hyper-parameter h controls the "spread" of contribution from each $x^{(i)}$ to nearby x values.

Exercise: Is this a valid probability distribution?

Exercise: How would we sample from the resulting distribution? (keep reading for answer)

By construction see that our model is the average of several gaussians. To sample from it, we can sample one of the data points uniformly at random, and then sample from its gaussian:

$$z \sim Cat(z|\frac{1}{N}, \dots \frac{1}{N}) = P(z)$$
$$x \sim \frac{1}{h}K(\frac{x - x^{(z)}}{h}) = P(x|z).$$

Therefore, our final distribution takes the form of a "mixture model":

$$\hat{P}(x|h) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h}K(\frac{x - x^{(i)}}{h}) = \sum_{z=1}^{N} p(z)p(x|z).$$

Such an approach can also apply to high-dimensional data (eg. $x^{(i)}$ is a vector not scalar). However, as points tend to be "far away" in high dimensions, such a model tends to perform poorly.

4 Gaussian Mixture Models

We have the same setup as in the previous section, but this time we choose to model our data with a fixed number of clusters, k, which might be a more reasonable modeling assumption. Each cluster z will be modeled as a Gaussian distribution:

$$N(x|\mu_z, \Sigma_z)$$

and has some weight π_z in our mixture. Note that $\sum_{z=1}^{k}\pi_z = 1$. Therefore, our model is

$$P(x|\theta) = \sum_{z=1}^{k} \pi_z N(x|\mu_z, \Sigma_z).$$

Exercise: How would we sample from this model?

To sample from this model, we can first think about sampling $z \sim \pi$ and then sampling x from $N(x|\mu_z, \Sigma_z)$. Our model therefore has the following interpretation:

$$P(x|\theta) = \sum_{z=1}^{k} \pi_z N(x|\mu_z, \Sigma_z) = \sum_{z=1}^{k} P(z|\theta)P(x|z,\theta).$$

Our goal is to find the parameters that maximize the log-likelihood of the data:

$$\max_\theta L(D;\theta) = \max_\theta \sum_{i=1}^{N} \log P(x^{(i)}|\theta) = \max_\theta \sum_{i=1}^{N} \log \left[\sum_{z=1}^{k} \pi_z N(x|\mu_z, \Sigma_z) \right].$$

4.1 Optimize via Gradient Ascent

One simple way to try to find the optimum is by using gradient ascent.

Note: the following derivation and analysis is for a *single data point* (ie. we find $\nabla_\theta \log P(x^{(i)}|\theta)$ which we can then sum over data points to get the gradient over the dataset).

We start by writing our expression and taking the gradient through the log:

$$\nabla_\theta \log P(x|\theta) = \frac{1}{P(x|\theta)} \nabla_\theta P(x|\theta).$$

Next, we apply the law of total probability:

$$= \frac{1}{P(x|\theta)} \nabla_\theta \sum_{z=1}^{k} P(x,z|\theta) = \frac{1}{P(x|\theta)} \sum_{z=1}^{k} \nabla_\theta P(x,z|\theta).$$

To simplify $\nabla_\theta P(x,z|\theta)$, we observe that $P\nabla_\theta \log P = \nabla_\theta P$. Note that this simplification is helpful because $P(x,z|\theta)$ is the product of two distributions one of which contains an exponential term.

$$= \frac{1}{P(x|\theta)} \sum_{z=1}^{k} P(x,z|\theta) \nabla_\theta \log P(x,z|\theta).$$

Finally, we observe that $P(x,z|\theta) = P(z|x,\theta)P(x|\theta)$ so our expression is

$$= \frac{1}{P(x|\theta)} \sum_{z=1}^{k} P(z|x,\theta)P(x|\theta)\nabla_\theta \log P(x,z|\theta) = \sum_{z=1}^{k} P(z|x,\theta)\nabla_\theta \log P(x,z|\theta).$$

In summary, we've determined that

$$\nabla_\theta \log P(x|\theta) = \sum_{z=1}^{k} P(z|x,\theta)\nabla_\theta \log P(x,z|\theta)$$

but can we interpret intuitively what this gradient update is doing? Let's look at the terms in the sum individually:

$$\nabla_\theta \log P(x,z|\theta) = \nabla_\theta \log \left[P(x,|z,\theta)P(z|\theta)\right].$$

This expression is almost exactly what our update rule would be if we only had a single Gaussian. It's moving the mean of each Gaussian $N(x|\mu_z, \Sigma_z)$ to the data point $x^{(i)}$ that we're computing the update rule for.

Now for the other term. We see that each update of the Gaussians is weighted by

$$P(z|x,\theta)$$

or how much each Gaussian explains the data point in question.

Finally, consider what behavior we'd see if we summed the gradients over the entire dataset. Over many updates, each Gaussian is pulled to the data points that it best explains.

4.2 Interpretation as Expectation-Maximization

The Expectation-Maximization (EM) algorithm is another formulation for optimizing the likelihood: $\max_\theta L(D;\theta)$. It works in two steps:

- E-Step: posterior assignments. Given the current estimate of the cluster parameters, for each data point $x^{(i)}$, compute weights $Q(z|x) = P(z|x,\theta)$ encoding the probability that $x^{(i)}$ is in cluster z.

- M-Step: Given fixed weights $Q(z|x)$, update the cluster parameters:

$$\theta \leftarrow \theta + \eta \sum_{z=1}^{k} Q(z|x)\nabla_\theta \log P(x,z|\theta)$$

Note: In the "true" EM algorithm, we run the M step to convergence before running the E step again. However, the nice connection we want you to see is that if we run the gradient step only once in the M-step, the algorithm in this section is exactly equivalent to the algorithm in the previous section!