ML and Optimization Lecture 24

- Generative Modeling: computing probabilities, and sampling
- Latent space models
- GANs

Generative Modeling

Given samples from a population

- faces of celebrities
- handwritten digits
- English short stories

Learn the density function of this population, $p_\theta(\cdot) = p_{data}(\cdot)$

Two fundamental tasks:

- computing probabilities $p_\theta(x_{new})$

  useful for e.g. anomaly detection

- sampling $x_{new} \sim p_\theta(\cdot)$

Some approaches to generative modeling can do one, the other, or both of these tasks

Naive approach : MLE

We can fit a distribution to (complex high-dimensional) data by using MLE:

$$\hat{\theta} = \underset{\theta}{\arg\min} -\frac{1}{n} \sum_{i=1}^{n} \log p_\theta(x_i)$$

This can then be used directly for computing probabilities

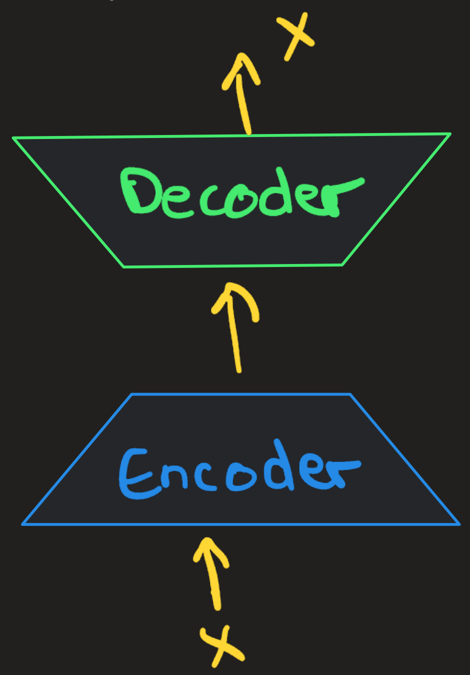
Sampling can also be done, but expensively, e.g. via Markov Chain Monte Carlo sampling (MCMC) with the Metropolis-Hastings algorithm.

- construct a conditional distribution $P(\cdot|x)$ that is easily exactly sampled and has the property that if $x_{t+1} \sim P(\cdot|x_t)$ then as $t \to \infty$, $x_t \sim p_\theta(\cdot)$
- expensive, slow, and good $P(\cdot|x)$ are highly specific to the problem at hand

Latent space approach

Many complex high-dimensional populations are effectively low-dimensional, and this implicit structure can be extracted via techniques like PCA or autoencoders.

Consider autoencoders: $x \approx D(E(x))$ where D, the decoder, and E, the encoder, are learned from data.



One can then potentially sample from the same distribution used to train the autoencoder if we sample z in the "latent space" from the appropriate distribution and take

$$x_{new} = D(z).$$

Q: what is the appropriate distribution? and how do we sample from it?

Generative Adversarial Networks (GANs) (2014)

A latent space approach where $z \sim p_z(\cdot) = \mathcal{N}(0, \sigma^2 I) \in \mathbb{R}^p$

Idea: Learn a generator $G(\cdot\,; \theta_g)$, a neural network, to make $x_{new} \sim G(z; \theta_g)$ "look like" a sample from p_{data}

To determine whether x_{new} looks like it came from p_{data}, assume access to a discriminator $D(\cdot\,; \theta_d)$ that models the probability that its input is from p_{data} rather than from the generator

Generative — we can sample from p_{data} approximately

Adversarial — the generator attempts to fool the discriminator

Network — the generator and discriminator are networks

A two-player game:

- train D to discriminate:
  - i) maximize $\log(D(x))$ on samples from p_{data}
  - ii) maximize $\log(1-D(x))$ on generator samples
- train G to fool D by minimizing $\log(1-D(x))$ on generator samples

$$\hat{\theta}_g, \hat{\theta}_d = \underset{\theta_g}{\operatorname{argmin}}\ \underset{\theta_d}{\operatorname{argmax}}\ V(\theta_g, \theta_d), \quad \text{where}$$

$$\boxed{V(\theta_g, \theta_d) = \mathbb{E}_{x \sim p_{data}}\left[\log D(x)\right] + \mathbb{E}_{z \sim p_z}\left[\log(1 - D(G(z)))\right]}$$

In practice, we can learn the parameters with variants of gradient ascent-descent

$$\theta_{g,t+1} = \theta_{g,t} - \lambda \nabla_{\theta_g} V(\theta_{g,t}, \theta_{d,t})$$

$$\theta_{d,t+1} = \theta_{d,t} + \lambda \nabla_{\theta_d} V(\theta_{g,t}, \theta_{d,t})$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

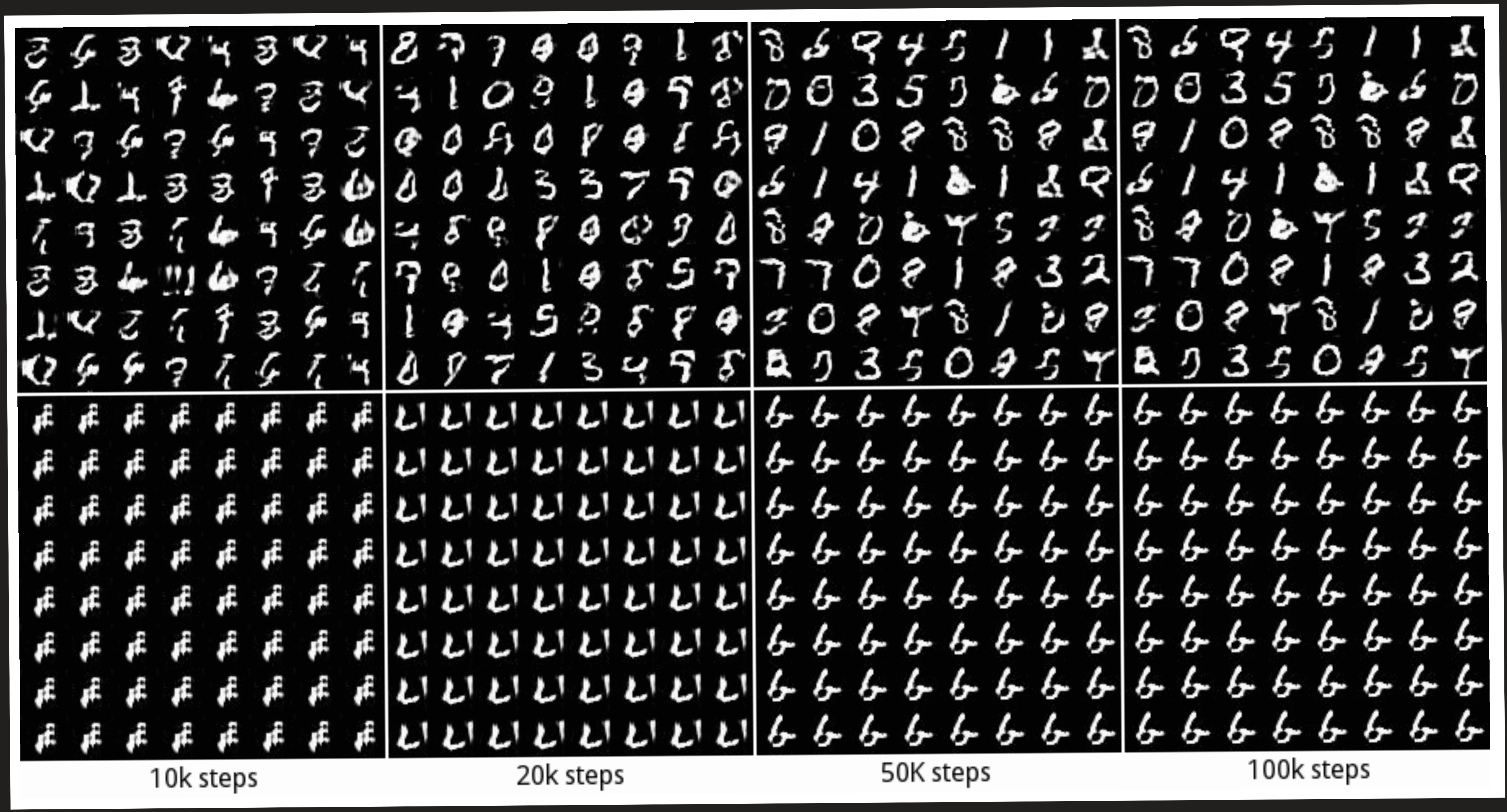
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

← algorithm of the original GAN paper (p_g is our p_z)

NB: it can perform better for the generator to maximize $\log(D(G(z)))$

Some notes

- If the architectures for G and D have high enough capacity, and the min-max optimization problem is solved, then $G(z;\theta_g) \sim p_{data}$, so we can exactly sample from p_{data}
- In practice, solving the min-max optimization problem is challenging. A common failure mode is "mode collapse", where the generator learns to sample from only one mode of a multimodal p_{data}
- The features created in the discriminator can be used for discriminative learning, e.g. classification



Example of mode collapse in learning a GAN on MNIST:

- top: well-trained variant of a GAN
- bottom: standard GAN