Advanced machine learning and data analysis for the physical sciences

Morten Hjorth-Jensen[1,2]

[1]Department of Physics and Center for Computing in Science Education, University of Oslo, Norway
[2]Department of Physics and Astronomy and Facility for Rare Isotope Beams, Michigan State University, East Lansing, Michigan, USA

May 7, 2024

Plans for the week of May 6-10, 2024

Generative models.

1. Finalizing discussion of Generative Adversarial Networks
2. Mathematics of diffusion models and selected examples

Reminder from last week, what is a GAN?

A GAN is a deep neural network which consists of two networks, a so-called generator network and a discriminating network, or just discriminator. Through several iterations of generation and discrimination, the idea is that these networks will train each other, while also trying to outsmart each other.

In its simplest version, the two networks could be two standard neural networks with a given number of hidden of hidden layers and parameters to train. The generator we have trained can then be used to produce new images.

Labeling the networks

For a GAN we have:

1. a discriminator D estimates the probability of a given sample coming from the real dataset. It attempts at discriminating the trained data by the generator and is optimized to tell the fake samples from the real ones (our data set). We say a discriminator tries to distinguish between real data and those generated by the abovementioned generator.

2. a generator G outputs synthetic samples given a noise variable input z (z brings in potential output diversity). It is trained to capture the real data distribution in order to generate samples that can be as real as possible, or in other words, can trick the discriminator to offer a high probability.

At the end of the training, the generator can be used to generate for example new images. In this sense we have trained a model which can produce new samples. We say that we have implicitely defined a probability.

Which data?

GANs are generally a form of unsupervised machine learning, although they also incorporate aspects of supervised learning. Internally the discriminator sets up a supervised learning problem. Its goal is to learn to distinguish between the two classes of generated data and original data. The generator then considers this classification problem and tries to find adversarial examples, that is samples which will be misclassified by the discriminator.

One can also design GAN architectures which work in a semi-supervised learning setting. A semi-supervised learning environment includes both labeled and unlabeled data. See https://proceedings.neurips.cc/paper_files/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf for a further discussion.

Thus, GANs can be used both on labeled and on unlabeled data and are used in three most commonly used contexts, that is

1. with labeled data (supervised training)
2. with unlabeled data (unsupervised learning)
3. a with a mix labed and unlabeled data

Improving functionalities

These two models compete against each other during the training process: the generator G is trying hard to trick the discriminator, while the critic model D is trying hard not to be cheated. This interesting zero-sum game between two models motivates both to improve their functionalities.

Setup of the GAN

We define a probability $p_{\boldsymbol{h}}$ which is used by the generator. Usually it is given by a uniform distribution over the input input $\boldsymbol{h}$. Thereafter we define the distribution of the generator which we want to train, p_g This is the generator's distribution over the data $\boldsymbol{x}$. Finally, we have the distribution p_r over the real sample $\boldsymbol{x}$

Optimization part

On one hand, we want to make sure the discriminator D's decisions over real data are accurate by maximizing $\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})]$. Meanwhile, given a fake sample $G(\boldsymbol{h}), \boldsymbol{h} \sim p_{\boldsymbol{h}}(\boldsymbol{h})$, the discriminator is expected to output a probability, $D(G(\boldsymbol{h}))$, close to zero by maximizing $\mathbb{E}_{\boldsymbol{h}\sim p_{\boldsymbol{h}}(\boldsymbol{h})}[\log(1 - D(G(\boldsymbol{h})))]$.

On the other hand, the generator is trained to increase the chances of D producing a high probability for a fake example, thus to minimize $\mathbb{E}_{\boldsymbol{h}\sim p_{\boldsymbol{h}}(\boldsymbol{h})}[\log(1 - D(G(\boldsymbol{h})))]$.

Minimax game

When combining both aspects together, D and G are playing a **minimax game** in which we should optimize the following loss function:

$$\begin{aligned}\min_G \max_D L(D, G) &= \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{h}\sim p_{\boldsymbol{h}}(\boldsymbol{h})}[\log(1 - D(G(\boldsymbol{h})))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[\log(1 - D(\boldsymbol{x})]\end{aligned}$$

where $\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})]$ has no impact on G during gradient descent updates.

Optimal value for D

Now we have a well-defined loss function. Let's first examine what is the best value for D.

$$L(G, D) = \int_{\boldsymbol{x}} \Big(p_r(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) \Big) dx$$

Best value of D

Since we are interested in what is the best value of $D(\boldsymbol{x})$ to maximize $L(G, D)$, let us label

$$\tilde{\boldsymbol{x}} = D(\boldsymbol{x}), A = p_r(\boldsymbol{x}), B = p_g(\boldsymbol{x})$$

Ignore integral

And then what is inside the integral (we can safely ignore the integral because $\boldsymbol{x}$ is sampled over all the possible values) is:

$$
\begin{aligned}
f(\tilde{\boldsymbol{x}}) &= A \log \tilde{\boldsymbol{x}} + B \log (1 - \tilde{\boldsymbol{x}}) \\
\frac{df(\tilde{\boldsymbol{x}})}{d\tilde{\boldsymbol{x}}} &= A\frac{1}{\tilde{\boldsymbol{x}}} - B\frac{1}{1-\tilde{\boldsymbol{x}}} \\
&= \frac{A - (A+B)\tilde{\boldsymbol{x}}}{\tilde{\boldsymbol{x}}(1-\tilde{\boldsymbol{x}})}.
\end{aligned}
$$

Best values

Thus, if we set $\frac{df(\tilde{\boldsymbol{x}})}{d\tilde{\boldsymbol{x}}} = 0$, we get the best value of the discriminator: $D^*(\boldsymbol{x}) = \tilde{\boldsymbol{x}}^* = \frac{A}{A+B} = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})+p_g(\boldsymbol{x})} \in [0, 1]$. Once the generator is trained to its optimal, p_g gets very close to p_r. When $p_g = p_r$, $D^*(\boldsymbol{x})$ becomes $1/2$. We will observe this when running the code below here.

When both G and D are at their optimal values, we have $p_g = p_r$ and $D^*(\boldsymbol{x}) = 1/2$ and the loss function becomes:

$$
\begin{aligned}
L(G, D^*) &= \int_{\boldsymbol{x}} \Big(p_r(\boldsymbol{x}) \log(D^*(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D^*(\boldsymbol{x})) \Big) d\boldsymbol{x} \\
&= \log \frac{1}{2} \int_{\boldsymbol{h}} p_r(\boldsymbol{x}) d\boldsymbol{x} + \log \frac{1}{2} \int_{\boldsymbol{x}} p_g(\boldsymbol{x}) d\boldsymbol{x} \\
&= -2 \log 2
\end{aligned}
$$

What does the Loss Function Represent?

The JS divergence between p_r and p_g can be computed as:

$$
\begin{aligned}
D_{JS}(p_r \| p_g) =& \frac{1}{2} D_{KL}(p_r || \frac{p_r + p_g}{2}) + \frac{1}{2} D_{KL}(p_g || \frac{p_r + p_g}{2}) \\
=& \frac{1}{2} \Big(\log 2 + \int_x p_r(\boldsymbol{x}) \log \frac{p_r(\boldsymbol{x})}{p_r + p_g(\boldsymbol{x})} d\boldsymbol{x} \Big) + \\
& \frac{1}{2} \Big(\log 2 + \int_x p_g(\boldsymbol{x}) \log \frac{p_g(\boldsymbol{x})}{p_r + p_g(\boldsymbol{x})} d\boldsymbol{x} \Big) \\
=& \frac{1}{2} \Big(\log 4 + L(G, D^*) \Big)
\end{aligned}
$$

What does the loss function quantify?

We have

$$
L(G, D^*) = 2 D_{JS}(p_r \| p_g) - 2 \log 2.
$$

Essentially the loss function of GAN quantifies the similarity between the generative data distribution p_g and the real sample distribution p_r by JS divergence when the discriminator is optimal. The best G^* that replicates the real data distribution leads to the minimum $L(G^*, D^*) = -2\log 2$ which is aligned with equations above.

Modification of GANs

Pros and Cons of GANs

Diffusion models, basics

Diffusion models are inspired by non-equilibrium thermodynamics. They define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise. Unlike VAE or flow models, diffusion models are learned with a fixed procedure and the latent variable has high dimensionality (same as the original data).

Problems with probabilistic models

Historically, probabilistic models suffer from a tradeoff between two conflicting objectives: *tractability* and *flexibility*. Models that are *tractable* can be analytically evaluated and easily fit to data (e.g. a Gaussian or Laplace). However, these models are unable to aptly describe structure in rich datasets. On the other hand, models that are *flexible* can be molded to fit structure in arbitrary data. For example, we can define models in terms of any (non-negative) function $\phi(\boldsymbol{x})$ yielding the flexible distribution $p(\boldsymbol{x}) = \frac{\phi(\boldsymbol{x})}{Z}$, where Z is a normalization constant. However, computing this normalization constant is generally intractable. Evaluating, training, or drawing samples from such flexible models typically requires a very expensive Monte Carlo process.

Diffusion models

Diffusion models have several interesting features

- extreme flexibility in model structure,
- exact sampling,
- easy multiplication with other distributions, e.g. in order to compute a posterior, and
- the model log likelihood, and the probability of individual states, to be cheaply evaluated.

Original idea

In the original formulation, one uses a Markov chain to gradually convert one distribution into another, an idea used in non-equilibrium statistical physics and sequential Monte Carlo. Diffusion models build a generative Markov chain which converts a simple known distribution (e.g. a Gaussian) into a target (data) distribution using a diffusion process. Rather than use this Markov chain to approximately evaluate a model which has been otherwise defined, one can explicitly define the probabilistic model as the endpoint of the Markov chain. Since each step in the diffusion chain has an analytically evaluable probability, the full chain can also be analytically evaluated.

Diffusion learning

Learning in this framework involves estimating small perturbations to a diffusion process. Estimating small, analytically tractable, perturbations is more tractable than explicitly describing the full distribution with a single, non-analytically-normalizable, potential function. Furthermore, since a diffusion process exists for any smooth target distribution, this method can capture data distributions of arbitrary form.