Deep Learning - Foundations and Concepts

Chapter 17. Generative Adversarial Networks

nonlineark@github

April 18, 2025

Outline

Adversarial training

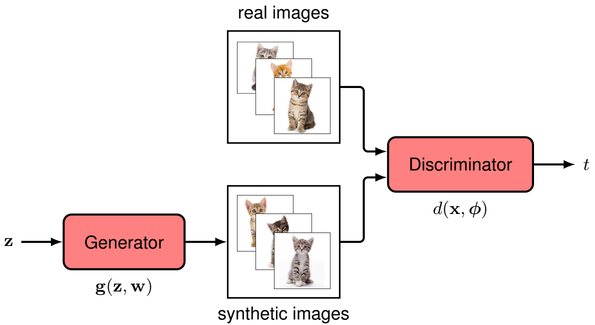
Consider a generative model based on a nonlinear transformation from a latent space z to a data space x:

$$p(z) = \mathcal{N}(z; 0, I)$$
$$x = g(z; w)$$

However, we cannot determine w by optimizing the likelihood function because this cannot, in general, be evaluated in closed form.

Adversarial training

The key idea of generative adversarial networks, or GANs, is to introduce a second discriminator network, which is trained jointly with the generator network and which provides a training signal to update the weights of the generator.



Adversarial training

- The goal of the discriminator network is to distinguish between real examples and synthetic examples, and it is trained by minimizing a conventional classification error function.
- Conversely, the goal of the generator network is to maximize this error by synthesizing examples from the same distribution as the training set.

Loss function

We define a binary target variable:

$$t = \begin{cases} 1, & \text{real data}, \\ 0, & \text{synthetic data}. \end{cases}$$

We train the discriminator network using the standard cross-entropy error function:

$$E(w, \phi) = -\frac{1}{N} \sum_{n=1}^{N} (t_n \log d_n + (1 - t_n) \log(1 - d_n))$$

where $d_n = d(x_n; \phi)$. We can write the error function in the form:

$$\begin{aligned} E_{\text{GAN}}(w, \phi) = &- \frac{1}{N_{\text{real}}} \sum_{n \in \text{real}} \log d(x_n; \phi) \\ &- \frac{1}{N_{\text{synth}}} \sum_{n \in \text{synth}} \log(1 - d(g(z_n; w); \phi)) \end{aligned}$$

Loss function

The unusual aspect is the adversarial training whereby the error is minimized with respect to ϕ but maximized with respect to w:

$$\Delta\phi = -\lambda\nabla_\phi E_n(w,\phi)$$
$$\Delta w = \lambda\nabla_w E_n(w,\phi)$$

where $E_n(w,\phi)$ denotes the error defined for a mini-batch of data points.

Loss function

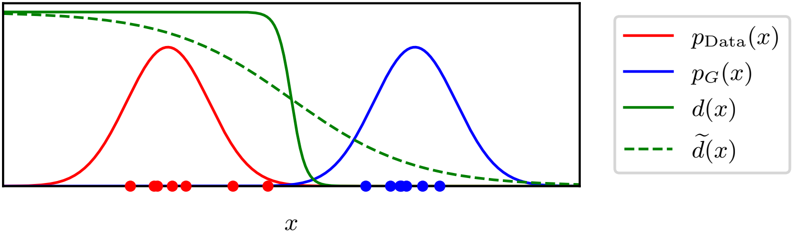
The training process:

1. Calculate error for a mini-batch and update w.
2. Generate a new set of synthetic samples.
3. Calculate error for a mini-batch and update ϕ.
4. Generate a new set of synthetic samples.
5. Go to step 1.

If the generator succeeds in finding a perfect solution, then the discriminator network will be unable to tell the difference between the real and synthetic data and hence will always produce an output of 0.5.

GAN training in practice

Figure: Learning proceeds slowly for the optimal discriminator function $d(x)$



GAN training in practice

This can be addressed by using a smoothed version $\tilde{d}(x)$ of the discriminator function:

- The least-squares GAN modifies the discriminator to produce a real-valued output and replaces the cross-entropy error function with a sum-of-squares error function.
- The instance noise technique adds Gaussian noise to both the real data and the synthetic samples.

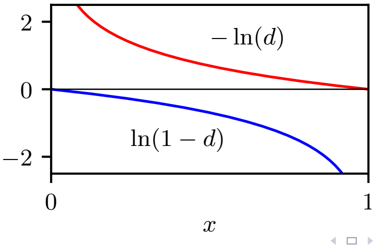
GAN training in practice

For faster training, one change that is often used is to replace the generative network term in the original error function:

$$-\frac{1}{N_{\text{synth}}}\sum_{n\in\text{synth}}\log(1-d(g(z_n;w);\phi))$$

with the modified form:

$$\frac{1}{N_{\text{synth}}}\sum_{n\in\text{synth}}\log d(g(z_n;w);\phi)$$



GAN training in practice

A more direct way to ensure that the generator distribution $p_{\mathrm{G}}(x)$ moves towards the data distribution $p_{\mathrm{Data}}(x)$ is to measure how far apart the two distributions are using the Wasserstein distance:

- Wasserstein GAN.
- Gradient penalty Wasserstein GAN.

Further references.