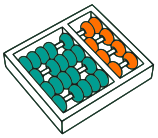

Implicit models: Generative adversarial networks

Lesson No. 06

Gustavo de J. Merli - 262948

1 Introduction

Deep generative models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context. This is a propose to a new generative model estimation procedure that sidesteps these difficulties. In the proposed adversarial nets framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable from the genuine articles.

2 Generative Adversarial Nets (GAN) [1]

To learn the generator's distribution p_g over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $log(1-D(G(z)))$. In other words, D and G play the following two-player minimax game with value function $V(G,D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \tag{1}$$

In practice, we must implement the game using an iterative, numerical approach. Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, we alternate between k steps of optimizing D and one step of optimizing G. This results in D being

maintained near its optimal solution, so long as G changes slowly enough.

for *number of training iterations* **do**
  for *number of training iterations* **do**
    Sim
    Sample minibatch of m noise samples $\{z^{(1)},...,z^{(m)}\}$ from noise prior $p_g(z)$
    Sample minibatch of m examples $\{x^{(1)},...,x^{(m)}\}$ from data generating distribution $p_{data}(x)$
    Update the discriminator by ascending its stochastic gradient:

$$\nabla_\theta \frac{1}{m}\sum_{i=1}^{m}\Big[logD(x^{(i)}) + log(1 - D(G(z^{(i)})))\Big] \tag{2}$$

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

$$\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} log(1 - D(G(z^{(i)}))) \tag{3}$$

end

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter.

3 Deep Convolutional GAN [2]

Deep Convolutional GANs (DCGAN) proposes and evaluates a set of constraints on the architectural topology of Convolutional GANs that make them stable to train in most settings.

There are three changes to CNN architectures. The first is the all convolutional net which replaces deterministic spatial pooling functions (such as maxpooling) with strided convolutions, allowing the network to learn its own spatial downsampling. Using this approach in generator, allowing it to learn its own spatial upsampling, and discriminator.

The second is the trend towards eliminating fully connected layers on top of convolutional features. The strongest example of this is global average pooling which has been utilized in state of the art image classification models. It is found that global average pooling increases model stability but hurt convergence speed. A middle ground of directly connecting the highest convolutional features to the input and output respectively of the generator and discriminator worked well. The first layer of the GAN, which takes a uniform noise distribution Z as input, could be called fully connected as it is just a matrix multiplication, but the result is reshaped into a 4-dimensional tensor and used as the start of the convolution stack. For the discriminator, the last convolution layer is flattened and then fed into a single sigmoid output.

The third is Batch Normalization which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. This helps deal with training problems that arise due to poor initialization and helps gradient flow in deeper models. This proved critical to get deep generators to begin learning, preventing the generator from collapsing all samples to a single point which is a common failure mode observed in GANs. Directly applying batchnorm to all layers however, resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer.

The ReLU activation is used in the generator with the exception of the output layer which uses the Tanh function. We observed that using a bounded activation allowed the model to learn more quickly to saturate and cover the color space of the training distribution. Within the discriminator it was found that the leaky rectified activation work well, especially for higher resolution modeling. This is in contrast to the original GAN paper, which used the maxout activation.

Other guidelines for stable DCGAN:

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.

- Remove fully connected hidden layers for deeper architectures.
- Use ReLu activation in generator for all except for the output, which uses Tanh.
- Use LeakyReLu activation in the discriminator for all layers.

4 Wasserstein GAN [3]

Let $\mathbb{P}_r$ be any distribution. Let P_θ be the distribution of $g_\theta(Z)$ with Z a random variable with density p and g_θ a function. Then, there is a solution $f : \mathscr{X} \to \mathbb{R}$ to the problem

$$\max_{||f||_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{4}$$

having

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z)] \tag{5}$$

Now comes the question of finding the function f that solves the maximization problem. To roughly approximate this, something that can be done is train a neural network parameterized with weights w lying in a compact space W and then backprop through $\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z)]$, as we would do with a typical GAN. In order to have parameters w lie in a compact space, something simple we can do is clamp the weights to a fixed box (say $W = [-0.01, 0.01]^l$) after each gradient update.

Data: α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Data: w_0, the initial critic parameters. θ_0, initial generator's parameters.
while θ *has not converged* **do**
  for $t = 0,...,n_{critic}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow clip(w, -c, c)$
  end
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end

Algorithm 2: WGAN, proposed algorithm.

5 Self-Attention GAN [4]

Most GAN-based models for image generation are built using convolutional layers. Convolution processes the information in a local neighborhood, thus using convolutional layers alone is computationally inefficient for modeling long-range dependencies in images.

The image features from the previous hidden layer $x \in \mathbb{R}^{CxN}$ are first transformed into two feature spaces f, g to calculate the attention, where $f(x) = W_f \cdot x$, $g(x) = W_g \cdot x$

$$\beta_{j,i} = \frac{exp(s_{ij})}{\sum_{i=1}^N exp(s_{ij})}, \text{ where } s_{ij} = f(x_i)^T g(x_j), \tag{6}$$

and $\beta_{j,i}$ indicates the extent to which the model attents to the i^{th} location when synthesizing the j^{th} region. Here, C is the number of channels and N is the number of feature locations of features from the previous hidden layer. The output of the attention layer is $o = (o_1, o_2, ..., o_j, ..., o_N) \in \mathbb{R}^{CxN}$, where,

$$o_j = v\left(\sum_{i=1}^N \beta_{j,i} h(x_i)\right), \; h(x_i) = W_h x_i, \; v(x_i) = W_v x_i \tag{7}$$

In the above formulation, $W_g \in \mathbb{R}^{\bar{C}xC}$, $W_f \in \mathbb{R}^{\bar{C}xC}$, $W_h \in \mathbb{R}^{\bar{C}xC}$, and $W_v \in \mathbb{R}^{C\ddot{O}\bar{C}}$ are the learned weight matrices, which are implemented as 1×1 convolutions.

In addition, we further multiply the output of the attention layer by a scale parameter and add back the input feature map. Therefore, the final output is given by

$$y_i = \gamma o_i + x_i \tag{8}$$

where γ is a learnable scalar and it is initialized as 0. Introducing the learnable γ allows the network to first rely on the cues in the local neighborhood – since this is easier – and then gradually learn to assign more weight to the non-local evidence.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[2] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015. arXiv: 1511.06434 [cs.LG].

[3] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017. arXiv: 1701.07875 [stat.ML].

[4] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, *Self-attention generative adversarial networks*, 2018. arXiv: 1805.08318 [stat.ML].