Reading Notes on *Generative Adversarial Nets*

Florian Hartmann

Abstract

Generative Adversarial Nets (GANs) are a new kind of generative model that is trained by letting two neural networks compete with each other. The first network is supposed to generate data that is similar to the training data, even though it does not have direct access to this data. A second network then has the task to differentiate between real and fake data. These two models provide a feedback signal to each other and are jointly trained. At some point, the generator learns to imitate the data distribution well enough and can generate realistic examples.

1 Introduction

- Machine learning tries to learn probability distributions over data
- *Discriminative* models have been quite successful thanks to advances in deep learning
- GANs try to make use of these techniques to train a *generative* model
- They consist of two models: A generative one and a discriminative one
- Those networks are made to compete. The generative one tries to generate data that is similar to some training data, while the discriminative one has to decide if a given piece of data is from the generator or from the training set
- At some point the generator becomes so good that the discriminator cannot distinguish between real and fake anymore. The authors give an analogy to counterfeiters that are producing fake banknotes
- In the case of GANs, both models are neural networks. The generator receives noise as input

2 Related work

- (*Note: Fully understanding this section seemed to require much more prior knowledge than the rest of the paper*)
- Directed graphical models, undirected graphical models with Deep Boltzmann machines, and deep belief networks are mentioned

3 Adversarial nets

- The generator G learns a distribution p_g over the data x
- It receives as input noise z sampled from a distribution $p_z(z)$ and produces an output $G(z)$
- (*Note: This noise seems to be used to make it much harder for the generator to just memorize data*)
- The discriminator D takes a data point x and assigns a probability representing how likely x was sampled from p_g as opposed to the actual data distribution
- D is optimized to assign the correct labels. G is optimized to make the discriminator think that the data is actually real
- This leads to a minimax game. We want to choose the best possible generator (min) given that the discriminator works as well as possible for this generator (max)

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

- The first expectation means that the discriminator should recognize the actual data as such. The second expectation means that it should consider the generated data as fake
- The discriminator wants to maximize this, while the generator wants to minimize it
- Given neural nets with enough capacity, we can learn any distribution with this scheme
- To make this optimization work in practice:
  - It is important not to train the discriminator too much in the individual steps. Otherwise it overfits quickly and learns to memorize the training data perfectly, giving the generator no chance to compete. This is why the number of gradient descent steps for the discriminator in each iteration is a hyperparameter
  - One has to ensure that the gradients behave nicely. The paper mentions that maximizing $\log(D(G(z)))$ instead of minimizing $\log(1 - D(G(z)))$ can help in the beginning

4 Theoretical Results

- For the theoretical analysis, we assume that both models have infinite capacity. This means that the generator could represent p_{data} perfectly
- The analysis shows that there is a global optimum at $p_g = p_{data}$ that is reached by the proposed gradient descent algorithm
- By reformulating the loss a bit, we can see that we are trying to minimize something similar to the Kullback-Leibler divergence between p_g and p_{data}
- (*Note: Maybe I'm missing some required math knowledge to fully follow the explanations but it seems like the proof is skipping some parts. For example, I don't see why $U(p_g, D)$ is always convex in p_g*)

5 Experiments

- Standard building blocks are used for both networks
- Evaluated on MNIST, the Toronto Face Dataset and CIFAR10
- Mechanism for evaluating generative models: Estimate probabilities for test data using a kernel density estimator (Gaussian Parzen window) and report the average logarithm. This should be maximized
- GANs perform well

6 Advantages and disadvantages

- Disadvantages:
  - No explicit representation of $p_g(x)$
  - D and G need to be synchronized well for training. The *Helvetica* scenario is that G maps too many z-values to the same x, leading to little diversity
- Advantages:
  - No Markov chains are needed
  - Training with backpropagation means powerful models can be learned
  - These are computational advantages
  - Maybe there are statistical advantages because the generator never directly sees any real data

7 Conclusions and future work

- Extensions for GANs related to other kinds of generative models are suggested
- One could try to predict z from x
- GANs could produce more training data in cases where too little is available for supervised learning
- The noise distributions p_z needs to be chosen well. By tuning this, better results might be possible