Machine Learning Bootcamp

August 26, 2019

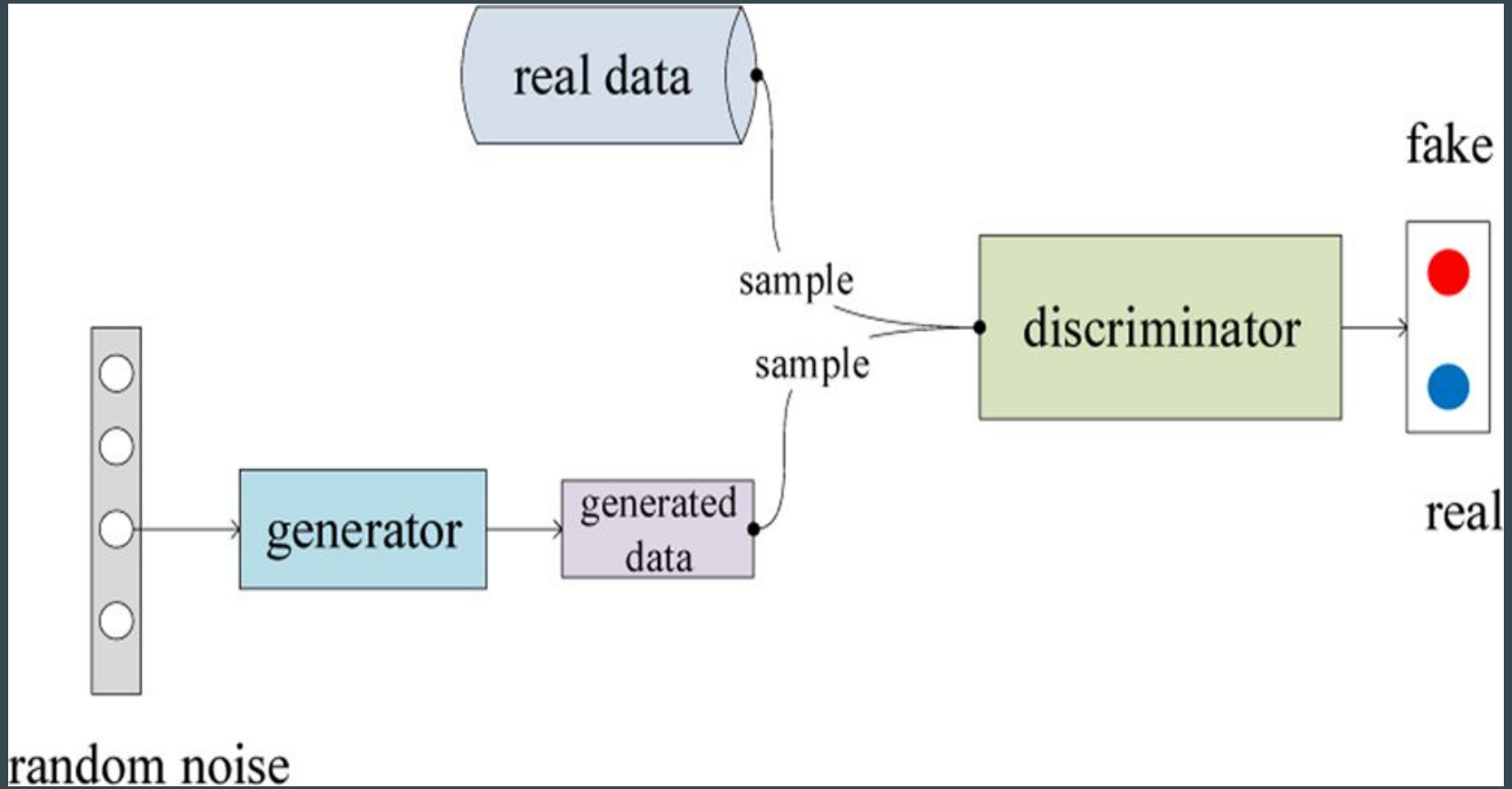
Overview

- Paper name: GENERATIVE ADVERSARIAL NETS.
- Authors: Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozairy, Aaron Courville, Yoshua Bengioz.
- Released: 10 Jun 2014

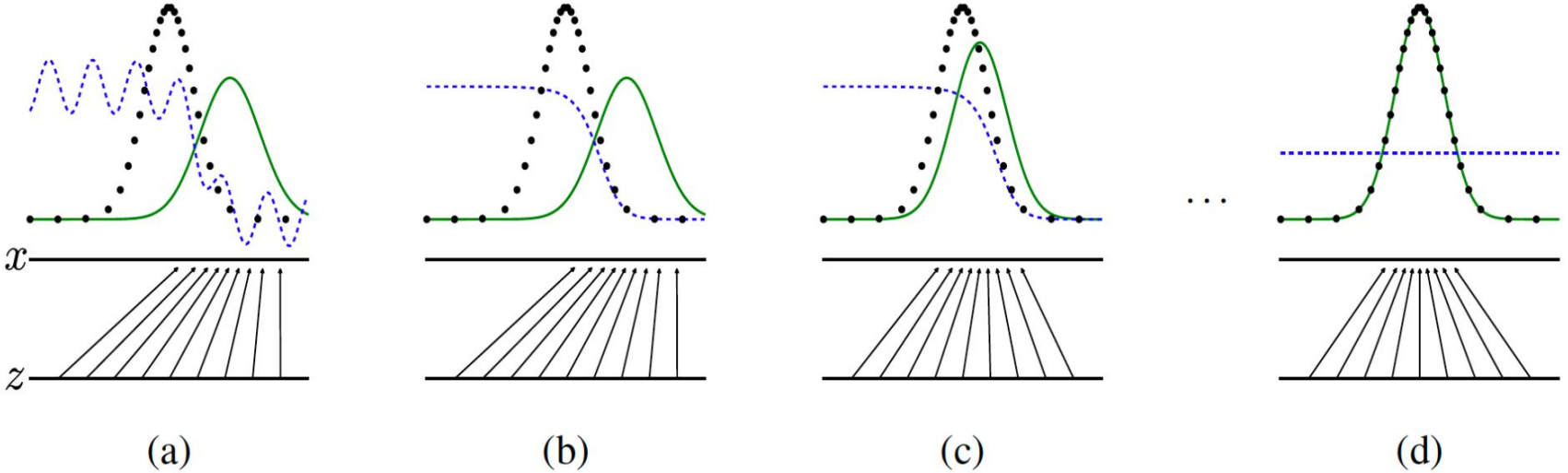
Understanding the problem

Item 1	Item 2	Item 3
Striking successes discriminative models. Backpropagation, dropout piecewise linear activation Difficulty of leveraging the benefits of piecewise linear units in the generative context.	Generative model G to captures the data distribution. Discriminative model D to estimate the probability that a sample is real. Use prior noise distribution as input to G.	Train D to maximize the probability of assigning the correct label to both training examples and samples from the G. Simultaneously train G to minimize. Minimax training.

The generator does not see the data.



The generator goes through the discriminator gradient.



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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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.

Proposed Solution

Generator	• MLP • RuLU, Sigmoid activations
Discriminator	• MLP • Maxout activation
Noise Prior	• Normal distribution
Loss	$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))].$

GAN models suffers badly in the following

Non-convergence

the models do not converge and worse they become unstable.

Mode collapse

the generator produces limited modes, learn the easy samples.

Slow training

the gradient to train the generator vanished.

Techniques I used to fight the above mentioned problems

Feature Matching

Changes the cost function for the generator to minimizing the statistical difference between the features of the real images and the generated images.

One-sided label smoothing

Penalizes the discriminator when the prediction for any real images go beyond 0.9 (D(real image)>0.9).

Used to avoid D overconfidence

Virtual batch normalization

Samples a reference batch before the training and do BN.

BN introduced undesired dependency between samples.

Feature Matching & Wasserstein GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size.
n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **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.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  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 while

```
""" Vanilla GAN """
D = torch.nn.Sequential(
    torch.nn.Linear(X_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, 1),
    torch.nn.Sigmoid()
)

""" WGAN """
D = torch.nn.Sequential(
    torch.nn.Linear(X_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, 1),
)
```

Modifying loss:

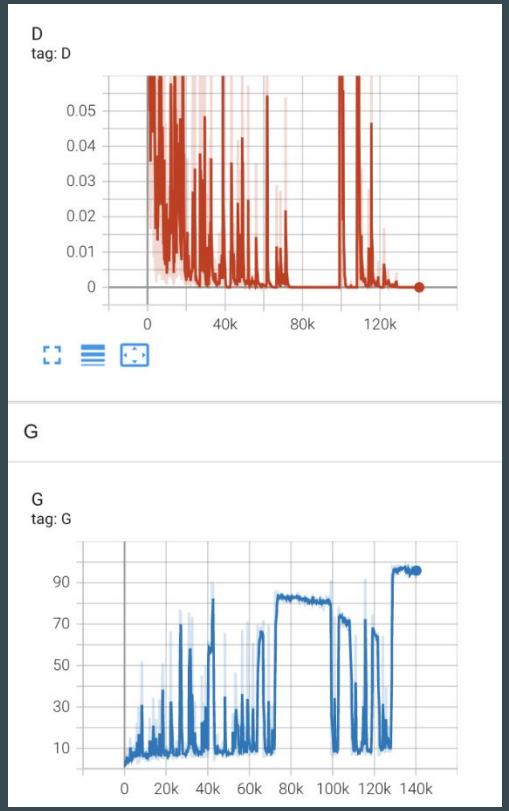
```
""" Vanilla GAN """
# During discriminator forward-backward-update
D_loss = torch.mean(torch.log(D_real) - torch.log(1- D_fake))
# During generator forward-backward-update
G_loss = -torch.mean(torch.log(D_fake))

""" WGAN """
# During discriminator forward-backward-update
D_loss = -(torch.mean(D_real) - torch.mean(D_fake))
# During generator forward-backward-update
G_loss = -torch.mean(D_fake)
```

What I did so far

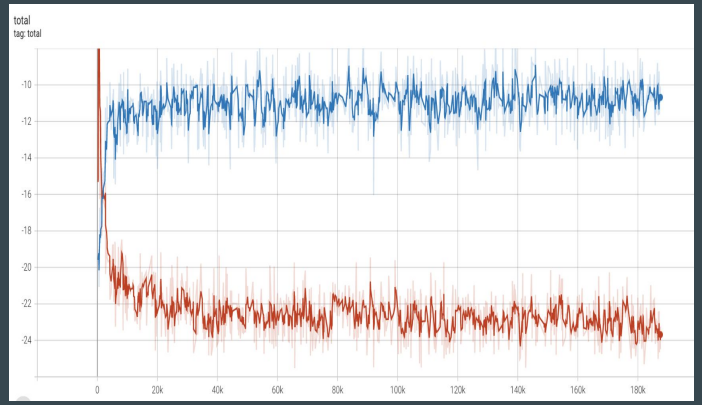
Before mitigating gans problems

- MNIST dataset.
- I sticked to MLP architectures and tried as far as possible to stabilize the training.
- I implemented three different architectures.
- I used the algorithm and loss function proposed by the paper.
- I used Adam optimizer.
- The training was not stable as you can see from the graph.
- The reg graph is D loss.
- The blue graph is G loss.



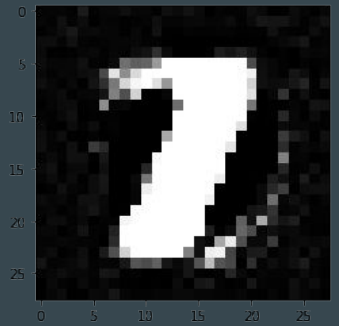
After mitigating gans problems

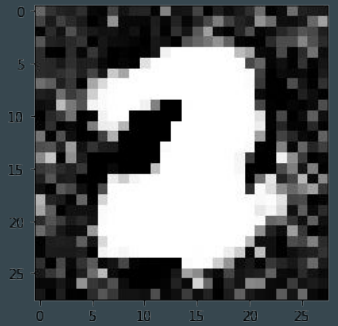
- I changed the discriminator architecture as proposed in WGAN.
- I changed the loss function as proposed in feature matching tech.
- I added VBN layer.
- I used weight clipping for the discriminator wrights.
- I regularized the max probability of the discriminator to be 0.9.
- I used RMSProb optimizer rather than Adam optimizer.
- Stable training, see the training graph.
- Red for the generator, blue for the discriminator.

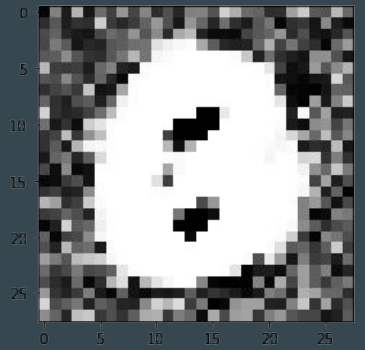


Results form my last model

- It’s not comparable to the paper’s results of course, but at least it’s describable. It show that some sort of training is happening.
- More improvement will follow.







Future Work

- Try other architectures.
- Get the best results.

Thank you for your time !

Questions ?

References

- https://papers.nips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
- https://github.com/ashrafhatim/Generative-Adversarial-Nets.git
- https://wiseodd.github.io/techblog/2017/02/04/wasserstein-gan/
- https://torchgan.readthedocs.io/en/latest/modules/layers.html#virtual-batch-normalization
- https://github.com/Duncanswilson/maxout-pytorch/blob/master/maxout_pytorch.ipynb