Generative Adversarial Networks

Team Toutpourlagan

Lucas Henneçon
Ilian Benaissa-Lejay
Simon Liétar

Wasserstein GAN

- Use Wasserstein distance instead of Jensen-Shannon divergence:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\| x - y \| \right]$$

- Using the Kantorovich-Rubinstein duality [1]:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

- Loss function of the WGAN:

$$\min_G \max_{\|D\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}\left[D(x)\right] - \mathbb{E}_{x\sim\mathbb{P}_G}\left[D(x)\right]$$

⇨ Discriminator 1 Lipschitz continuous

[1] Arjovsky, M., Chintala, S., & Bottou, L. (2017, July). Wasserstein generative adversarial networks. In *International conference on machine learning* (pp. 214-223). PMLR.

Wasserstein GAN

- Don't use a sigmoid at the output of D : output not in [0,1]
- To ensure Lipschitz-continuity of D, clip the weights between [-c, c]

Algorithm:

Update of D (nd times):

- Sample (Xi) batch of real images,
- Generate batch (Zi) with G
- Backprop and RMSProp as optimizer
- Clip weights between [-c, c]

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} D(Z_i) - \frac{1}{N}\sum_{i=1}^{N} D(X_i)$$

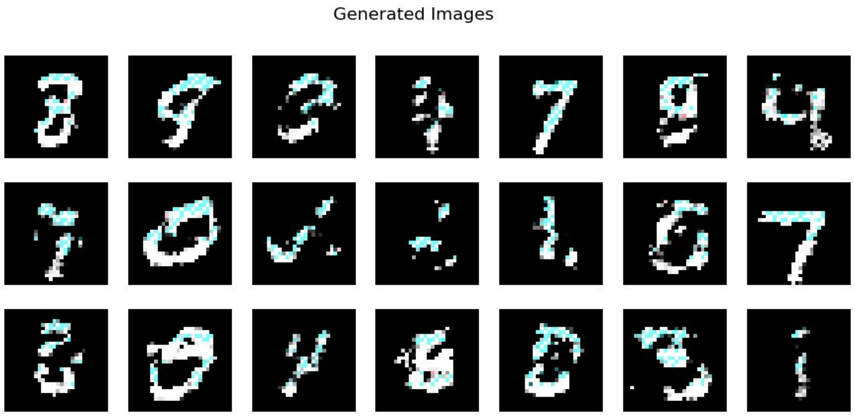
Update of G:

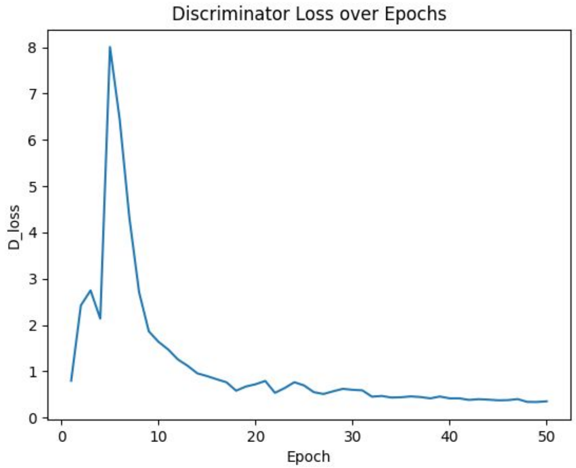
- Generate batch (Zi) with G
- Backprop and RMSProp as optimizer

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[D(Z_i)]$$

Wasserstein GAN

- Hyperparameters from the paper
- FID = 70





Hyperparameters = 5e-5, batch_size=64, critic_iterations=5, weight_clip =0.01

Wasserstein GAN - Gradient Penalty

- New way to ensure 1-Lipschitz continuity of D [2]:

$$L = \underbrace{\mathbb{E}_{\tilde{x}\sim\mathbb{P}_G}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]}_{\text{Original WGAN Discriminator loss}} + \lambda \underbrace{\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]}_{\text{Gradient penalty}}.$$

Algorithm:

Update of D (nd times):

- Sample (Xi) batch of real images, Generate batch (Zi) with G
- Qi = εXi + (1-ε)Zi with ε following U[0,1]
- $\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} D(Z_i) - \frac{1}{N}\sum_{i=1}^{N} D(X_i) + \lambda \sum_{i=1}^{N} \left(\|\nabla_{Q_i} D(Q_i)\|_2 - 1\right)^2$
- Backprop and Adam as optimizer

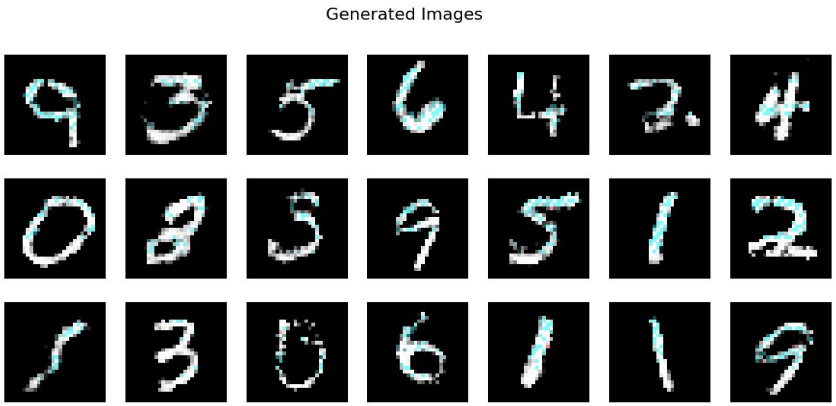
Update of G:

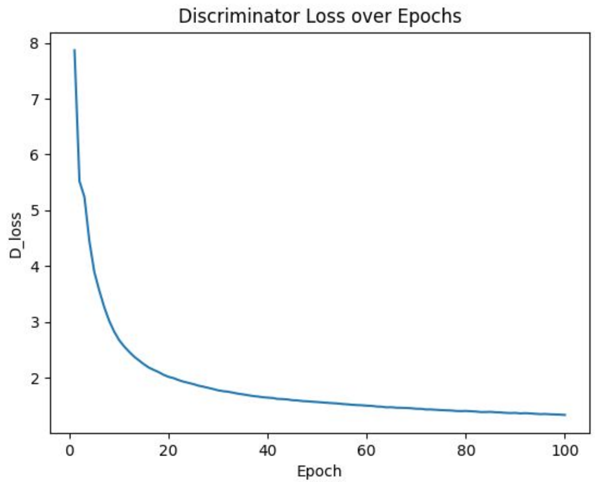
- Same as WGAN with Adam as optimizer

[2] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*, *30*.

Wasserstein GAN-GP

- More stable
- WGAN-GP : FID = 35.5, Precision = 0.45, Recall = 0.27
- VanillaGAN : FID = 28.3 Precision = 0.52, Recall = 0.22





What we’ll try next

- Training on more epochs
- Fine tuning
- Different optimizers for D and G
- Rejection Sampling (find a criterion)
- fGANS

Computing precision and recall

- Based on nearest-neighbor search as introduced by Kynkäänniemi et al. (2019) and implemented using a k-d tree
- The paper computes precision and recall using an intermediate layer of VGG-16, which may not be suitable here