RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

METEO
FRANCE
À VOS CÔTÉS, DANS UN
CLIMAT QUI CHANGE

Machine Learning – Recap' n°6

Pierre Lepetit
ENM, le 20/12/2024

GANs

- Intuition: « We train [the discriminator] D to maximize the probability of assigning the correct label to both training examples and samples from [the generator G]. We simultaneously train the G to minimize log(1 – D(G(z)) »

  Generative Adversarial Networks, Goodfellow et al.

- Formalisme mathématique → « two-player minimax game » :

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

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

METEO FRANCE
À VOS CÔTÉS, DANS UN CLIMAT QUI CHANGE

GANs

- Intuition: « We train... »
- Formalisme mathématique → « two-player minimax game »

- Algorithme :

for number of training iterations **do**

  for k steps **do**

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

GANs

- Codage avec pytorch (exemple):

```
# STEP 1: Discriminator optimization

        netD.zero_grad()

        D_real = netD(x).view(-1)
        label = torch.ones((batch_size,)).cuda()
        errD_real = bce(D_real, label)
        errD_real.backward()

        # Generated images
        fake = netG(z)
        D_fake = netD(fake.detach()).view(-1)


        label.fill_(0.)
        errD_fake = bce(D_fake, label)
        errD_fake.backward()

        optimizerD.step()
```

GANs

- Codage avec pytorch (exemple):

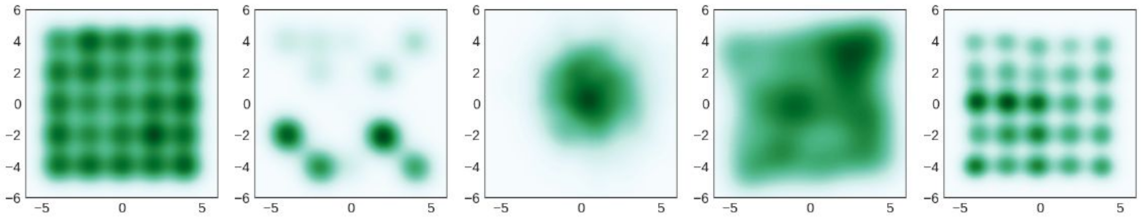
```
# STEP 2: Generator optimization

        netG.zero_grad()

        D_fake2 = netD(fake).view(-1)

        label.fill_(1.)
        errG = bce(D_fake2, label)
        errG.backward()

        optimizerG.step()
```

GANs

- Pourquoi est-ce que ça « peut marcher » ?

→ https://github.com/nanopiero/ML_S5/blob/main/exercices/sheet2_corr.ipynb

- En pratique, la « convergence » vers la loi des images réelles est difficile à obtenir.
  eg : mode collapses.



(f) True Data (g) GAN (h) ALI (i) Unrolled (j) VEEGAN

→ Wasserstein GAN / Spectral Normalization & Cie

Learning to rank

- Fonction de coût : Hinge Loss / Ranknet Loss / Listnet Loss
  - Hinge Loss :
    $L(f_w(x_0), f_w(x_1) ; y_{01}) = \max(0, 1 - y_{01} (f_w(x_1) - f_w(x_0)))$ où $y_{01} = +1$ si $x_1 \blacktriangleright x_0$
    -1 sinon
  - Ranknet Loss :
    $L_\sigma(f_w(x_0), f_w(x_1) ; y_{01}) = \delta_{01}\, \sigma(f_w(x_1) - f_w(x_0)) + \ln(1 + \exp(\sigma(f_w(x_1) - f_w(x_0))))$

    où $\delta_{01} = +1$ si $x_1 \blacktriangleright x_0$ et σ est une constante
    -1 sinon
- Applications : moteurs de recherche / proxy pour une quantité scalaire.
  eg : ELO score, Visibilité