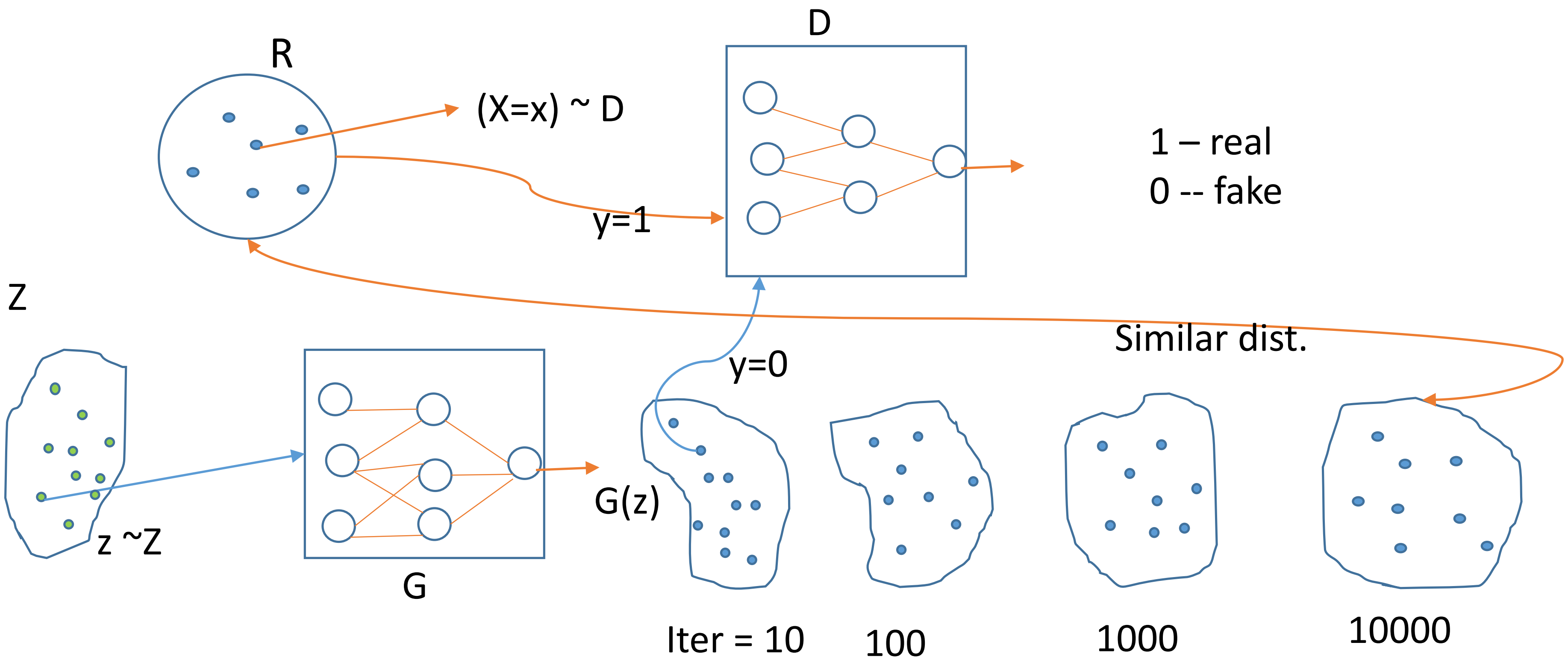
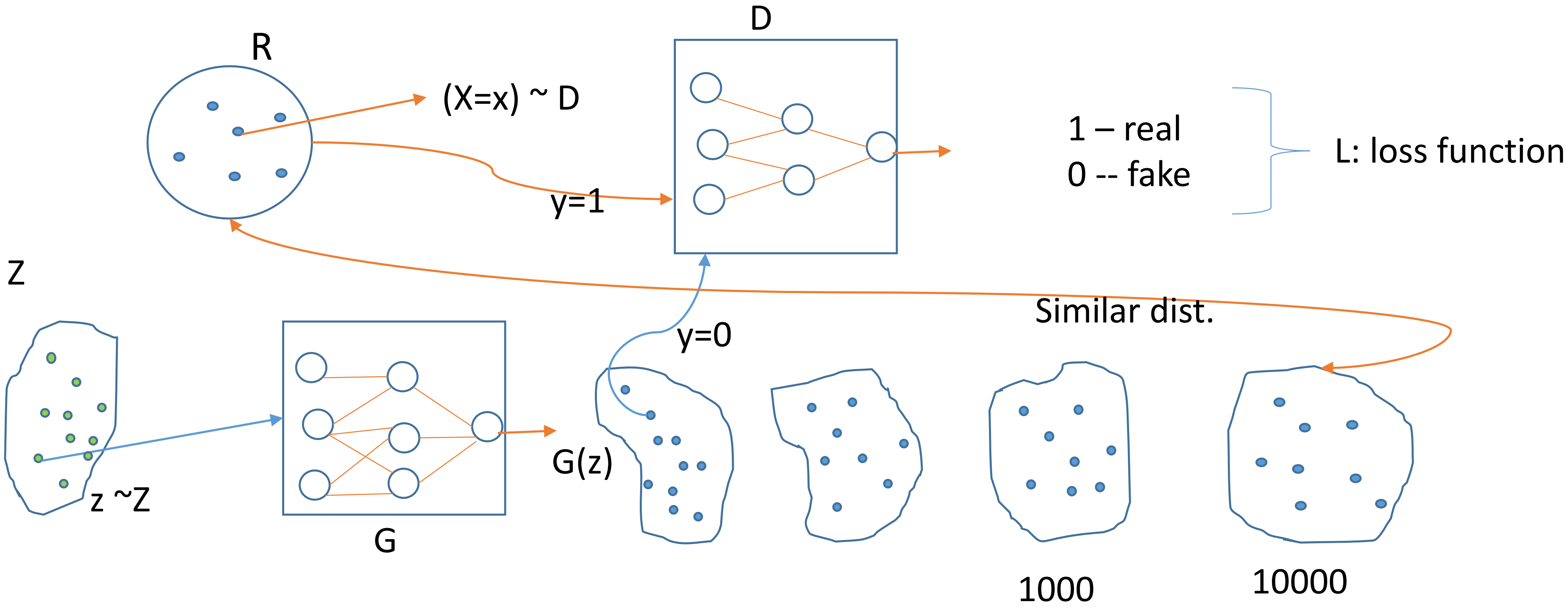
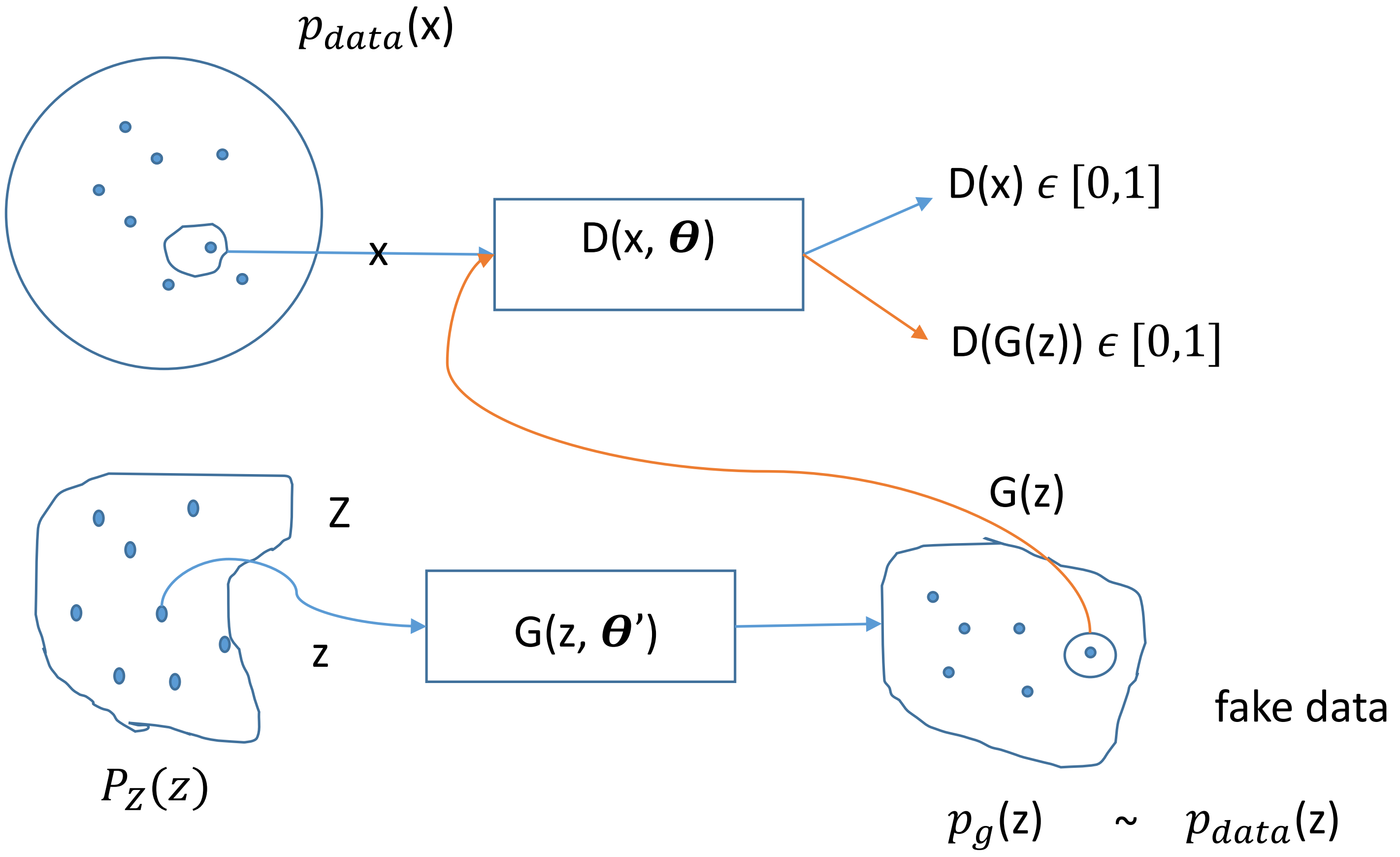
Loss Function of GAN

- Discriminator: to distinguish actual data from fake data
- Generator: to create fake data to fool the discriminator




R
(X=x) ~ D
D
y=1
1 – real
0 -- fake
L: loss function
Z
z ~Z
G
G(z)
y=0
Similar dist.
1000
10000

Conventions (notations) to understand Loss function



Cross entropy

$$H = \sum_{c=1}^{C} \sum_{i=1}^{n} -y_{c,i} log_2(p_{c,i})$$

P_{ci} is the probability of the predicted i^{th} class, it will never be 0 (softmax output) , otherwise a big problem.

$Y_{c,i}$ is the true class probability (usually, one-hot encoding)

Cross entropy is a measure of how similar two distributions are.

Example

Model 1							
		Predicted prob.			Actual class (one hot encoding)		
		boy	girl	other	boy	girl	other
data1	boy	0.4	0.3	0.3	1	0	0
data2	girl	0.3	0.4	0.3	0	1	0
data3	boy	0.5	0.2	0.3	1	0	0
data4	other	0.8	0.1	0.1	0	0	1
		Error rate= ¼=25% CROSS ENTROPY =6.966					

Boy's cross entropy=-(1*log(0.4)+1*log(0.5)) = 2.322 (taking $\log_2$)
Girl's cross entropy = -(1*log(0.4))=1.322
Other's cross entropy=-(1*log(0.1))=3.322
Overall cross entropy=6.966

Model 2							
		Predicted prob.			Actual class (one hot encoding)		
		boy	girl	other	boy	girl	other
data1	boy	0.7	0.1	0.2	1	0	0
data2	girl	0.1	0.8	0.1	0	1	0
data3	boy	0.9	0.1	0.0	1	0	0
data4	other	0.4	0.3	0.3	0	0	1
		Error rate= ¼=25% CROSS ENTROPY =2.725					

$=-(1 \times \log(0.7) +1 \times \log(0.8)+ 1 \times \log(0.9) + 1 \times \log(0.3))= 2.725$

Total = 2.725

Model 2 is better in terms of cross entropy

- For the binary case here: We have
- real – 1
- fake – 0
- Loss function for Discriminator:

$$L(\hat{y}, y) = -[y\, log\hat{y} + (1 - y)\log(1 - \hat{y})]$$

Where $\hat{y}$ is the predicted value; y is the label

Discriminator loss func.

- For a sample coming from R, y=1 and $\hat{y} = D(x)$

  L(D(x), 1) = -log(D(x)) -----(A)

- For a sample coming from the Generator, the label is that y=0, and

$\hat{y} = D(G(z))$, so in this case,

L(D(G(z), 0) = -(1-0)log(1-D(G(z))

= -log(1-D(G(z)) -----(B)

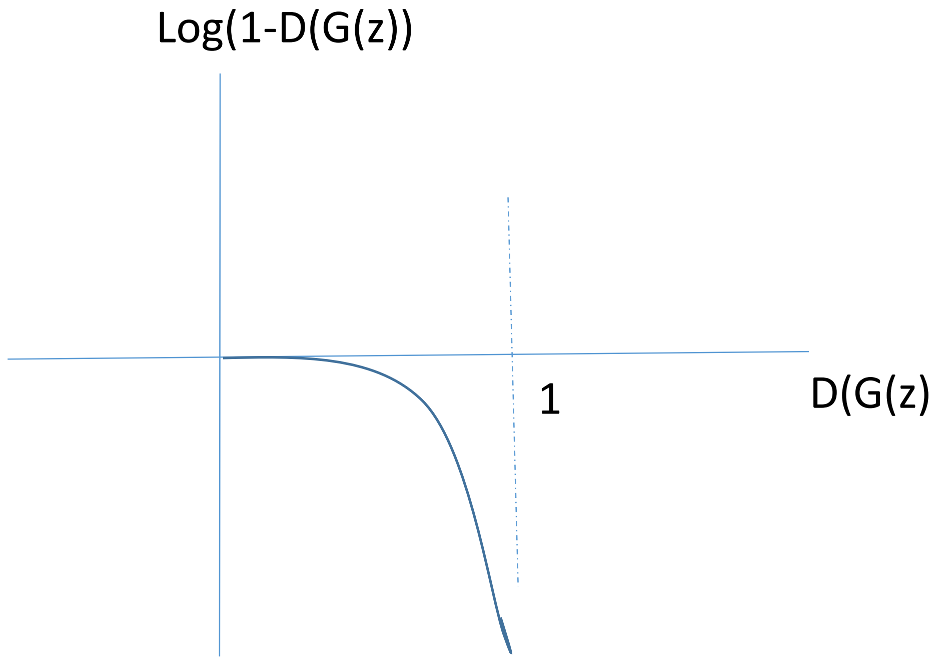
Loss function for D

- The objective of the discriminator is to correctly classify fake data and real data. For this (A) and (B) should be minimized.
- In other words,
- Objective func.

$$\max \quad \log\left(D(x)\right) + \log(1 - D(\mathrm{G}(z)))$$

Loss function for G

- The generator wants to fool the discriminator, so it wants to push D(G(z)) to 1, so it minimizes
- log(1-D(G(z)) so → min [log(D(x)) + log (1-D(G(z))]



Put together

$$\underset{G}{min}\ \underset{D}{\max}\ \{\log\big(D(x)\big) + \log(1 - D\big(G(z)\big)\}$$

Goodfollw's equation:

$$\underset{G}{min}\ \underset{D}{\max}\ V(D,G) = \underset{x \sim P_{data}(x)}{E}\ [\log\big(D(x)\big)] + \underset{z \sim P_Z(z)}{\mathrm{E}}[\log(1 - D\big(G(z)\big)]$$

How to train it?

- Take half of the training samples from the real dataset and half from generated fake dataset, and train D for a while.
- Then fix D, and train G to generate samples that fool the current D.
- Repeat the two steps until D cannot tell real apart from fake.

- But does it always converge to the case that D fails to distinguish?
- A big Proof

The math

- Part 1 (Loss function)
- Part 2 (Optimization)
- Part 3 (The GAN training algorithm)
- Part 0 (Introduction)

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

1:24 / 14:24

GoodFellow 2014