Deep Learning

CS256 - Topics in Artificial Intelligence

April 16, 2018

Neural Networks

- We have studied a lot of neural network
- In general, they all realize a vector function

$$\mathbf{x} \in \mathbb{R}^n \to \mathbf{y} \in \begin{cases} [0:1]^n & \text{classification} \\ \mathbb{R}^n & \text{regression} \end{cases}$$

Neural Network in context

We will discuss in this class two important application that make use of NN

- Creating realistic, but fake new data (GAN)
  (*today and following lectures*)
- Using neural network for interactive systems (RL)
  (*later*)

Overview

Generative Adversarial Network

Discriminative vs. Generative

- In a classification task, a neural network takes an input x and computes class probabilities $P(C_1|x), P(C_2|x), \ldots$
- The alternative way to model the relation between input data x and classes C_i, is to have a separate model for each class
  - Such a model describes the probability of input data given a class

Discriminative vs. Generative

- Example for a generative model: characters in a word

Letter	*English*	*French*	*German*	*Spanish*	*Portuguese*
a	8.17%	7.64%	6.52%	11.53%	14.63%
b	1.49%	0.90%	1.89%	2.22%	1.04%
c	2.78%	3.26%	2.73%	4.02%	3.88%
d	4.25%	3.67%	5.08%	5.01%	4.99%
e	12.70%	14.72%	16.40%	12.18%	12.57%
f	2.23%	1.07%	1.66%	0.69%	1.02%
g	2.02%	0.87%	3.01%	1.77%	1.30%
h	6.09%	0.74%	4.58%	0.70%	0.78%
i	6.97%	7.53%	6.55%	6.25%	6.19%
j	0.15%	0.61%	0.27%	0.49%	0.40%
k	0.77%	0.05%	1.42%	0.01%	0.02%
l	4.03%	5.46%	3.44%	4.97%	2.78%
m	2.41%	2.97%	2.53%	3.16%	4.74%
			...		

Why are generative systems useful?

With probabilities $P(\text{input}|\text{class})$, we can

- Do classification $P(\text{class}|\text{input}) = \frac{P(\text{input}|\text{class})P(\text{class})}{P(\text{input})}$
  - A big and topic, but not for this lecture series

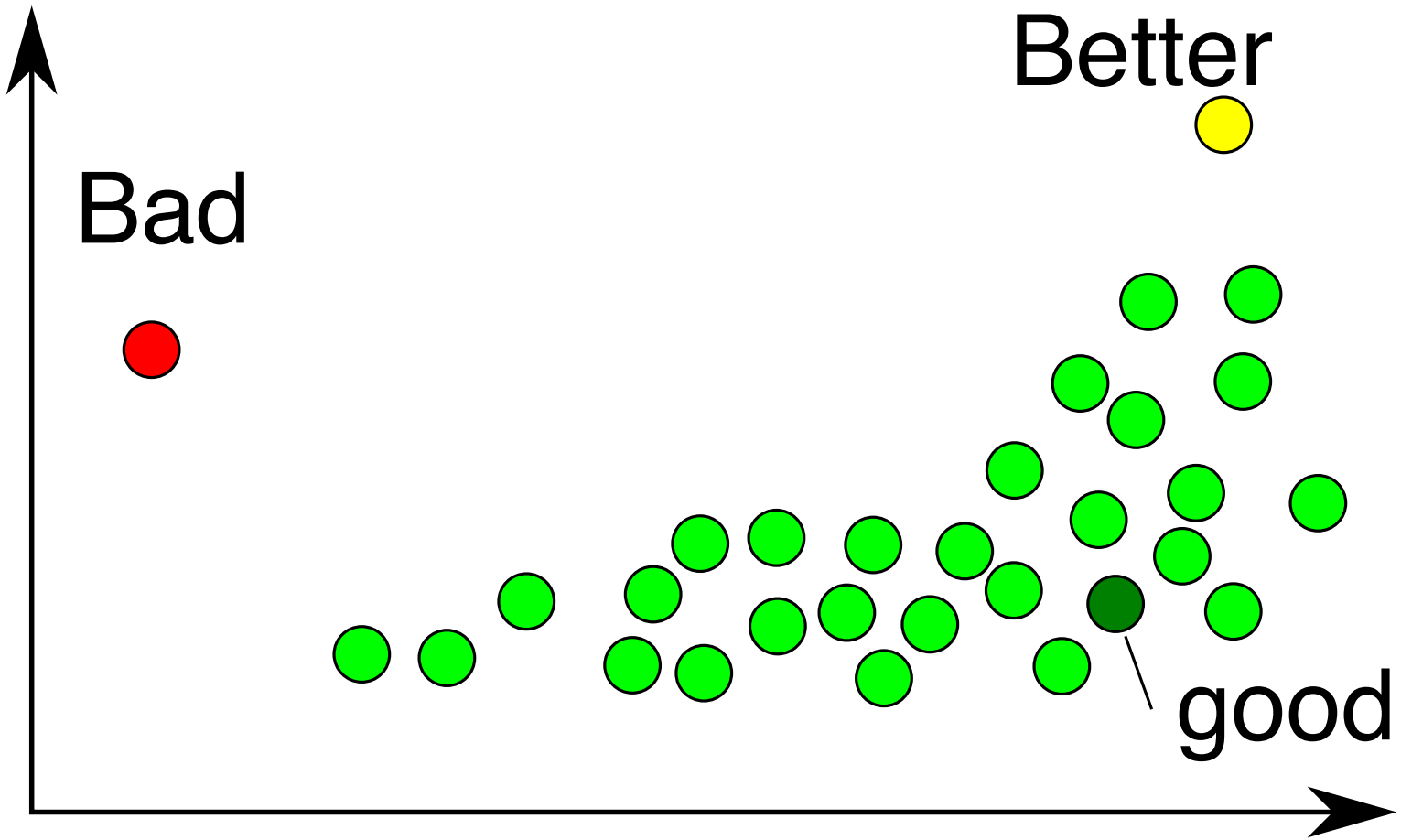
Why are generative systems useful?

With probabilities $P(\text{input}|\text{class})$, we can

- Do classification $P(\text{class}|\text{input}) = \frac{P(\text{input}|\text{class})P(\text{class})}{P(\text{input})}$
  - A big and topic, but not for this lecture series
- Generate new data:
  - Pick a class, i.e. *German*, and sample characters accordingly
  - Lot's of fun (sometimes also useful)
    - E.g. Ensesrinneatdhue (could be German?!?)

Ideas?

- Given a set of data given, how to generate fake, new data?
- Unlike in your homework where you had seed character given to continue with text production



An idea

- In a generative system we have a distribution and sample from this
  - In your homwork, the output of the neural network was a probability distribution of characters

character	a	b	c	d	e	...	$\rightarrow d$
P(character)	0.1	0.02	0.04	0.2	0.1	...	

- Now, we want a system that directly produces an element

An idea

- In a generative system we have a distribution and sample from this
  - In your homwork, the output of the neural network was a probability distribution of characters

$$\begin{array}{lccccccc} \text{character} & a & b & c & d & e & \dots & \\ P(\text{character}) & 0.1 & 0.02 & 0.04 & 0.2 & 0.1 & \dots & \end{array} \rightarrow d$$

- Now, we want a system that directly produces an element
  - Randomness needs to be introduced at the beginning
  - The input to the system is a random number (or a random vector)

An idea

- We need some randomness, if we want to create a new output
- Use a random number generator RNG $\in [0; 1]$
- Train a neural network that maps RNG $\to \mathbb{R}^n$
  - Possible architecture: RNG $\to x \to Wx \to h(Wx) \to \ldots$
  - works with feed forward, convolution, pooling, etc.

An idea

- We need some randomness, if we want to create a new output
  - Use a random number generator RNG $\in [0; 1]$
- Train a neural network that maps RNG $\to \mathbb{R}^n$
  - Possible architecture: RNG $\to x \to Wx \to h(Wx) \to \ldots$
  - works with feed forward, convolution, pooling, etc.
- Loss function?

An idea

- We need some randomness, if we want to create a new output
  - Use a random number generator RNG $\in [0; 1]$
- Train a neural network that maps RNG $\to \mathbb{R}^n$
  - Possible architecture: RNG $\to x \to Wx \to h(Wx) \to \ldots$
  - works with feed forward, convolution, pooling, etc.
- Loss function:
  - How similar is the network output to the target distribution?

An idea

- We need some randomness, if we want to create a new output
  - Use a random number generator RNG $\in [0; 1]$
- Train a neural network that maps RNG $\to \mathbb{R}^n$
  - Possible architecture: RNG $\to x \to Wx \to h(Wx) \to \ldots$
  - works with feed forward, convolution, pooling, etc.
- Loss function:
  - How similar is the network output to the target distribution?
  - Instead of defining a fixed similarity function, we can get some inspiration from nature

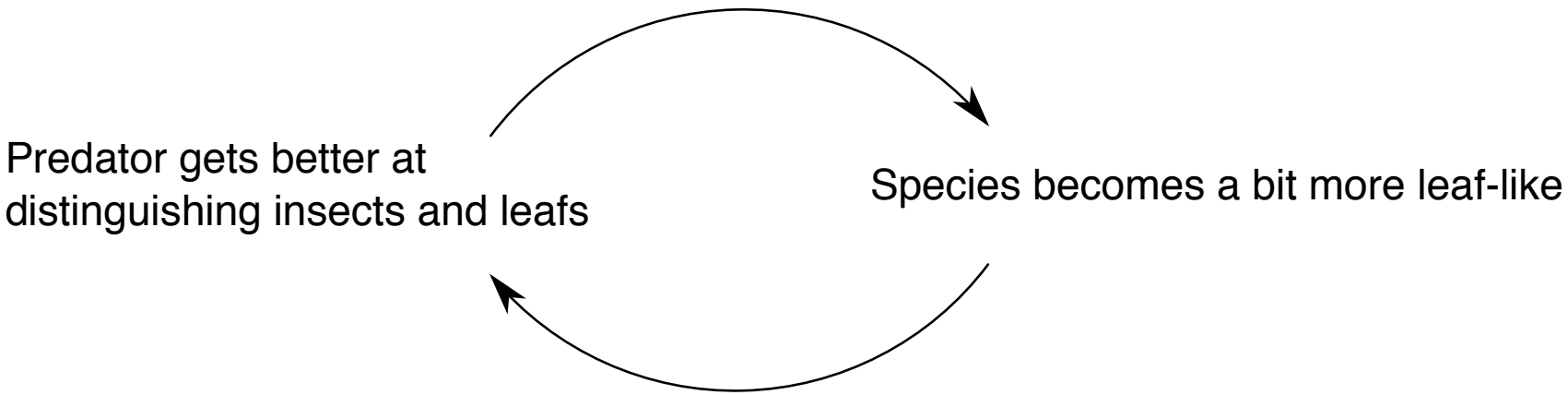
Examples from nature

Examples from nature

- What happened here?
- How did this insect evolve?

Mimicry Evolution

- Predator eats prey
- More leaf-like than non-leaf-like insects survive



Mimicry Evolution as NN setup

- How can we formulate this as a NN problem?
- What is the loss function?

Mimicry Evolution as NN setup

- Predator and Prey are given by NN

$$\mathrm{NN}_{Prey} \text{ and } \mathrm{NN}_{Predator}$$

- $\mathrm{NN}_{Predator}$ simulates the decision making process of the predator

$$\text{image} \rightarrow \begin{cases} \text{is leaf} \\ \text{is not leaf} \end{cases}$$

- Obviously a binary classification process, trained by minimizing cross-entropy loss

Mimicry Evolution as NN setup

- Predator and Prey are given by NN

$$\text{NN}_{Prey} \text{ and } \text{NN}_{Predator}$$

- NN_{Prey} simulates the appearance of the Prey

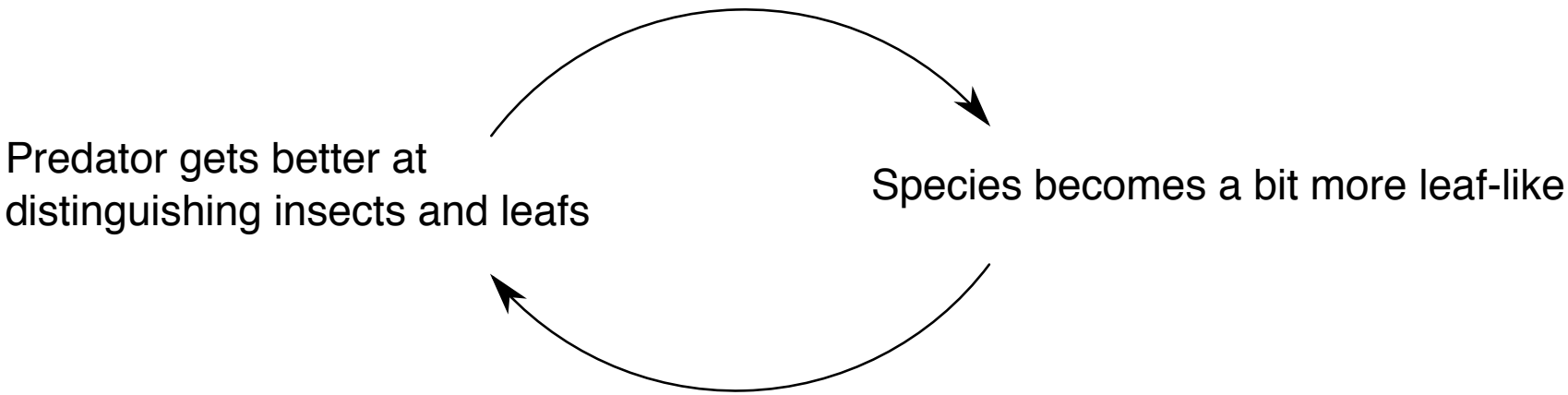
$$\text{Random vector} \rightarrow \text{image}$$

- Loss function also depends on the $\text{NN}_{Predator}$ binary classification process
  - Output of the NN_{Prey} is input to the $\text{NN}_{Predator}$
  - goal is to **maximize** it's cross entropy loss
    - The larger the cross entropy loss, the high the probability of mis-classification by the $\text{NN}_{Predator}$
  - Error gradient is passed through the $\text{NN}_{Predator}$ to the NN_{Prey}

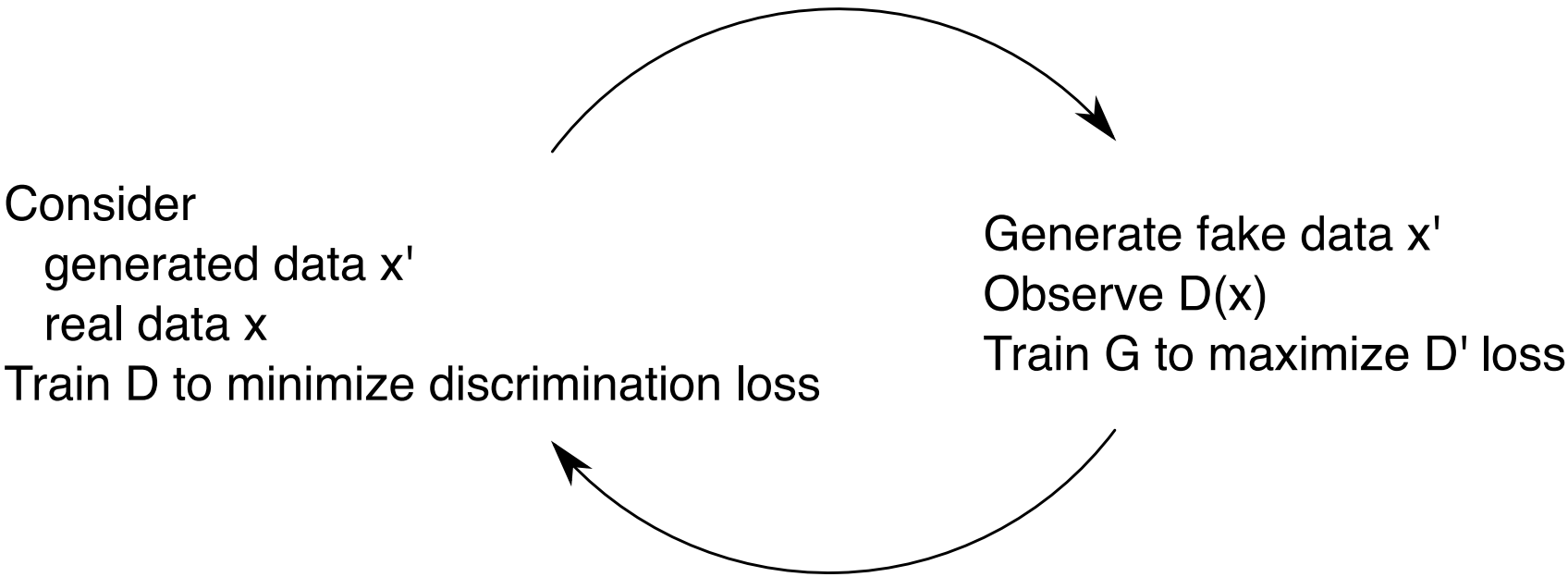
Mimicry Evolution as a NN setup

- The $\text{NN}_{Predator}$ makes a decision, hence it is a **d**iscriminator
  - Commonly referred to as D
- The NN_{Prey} network generates a shape, hence it is a **g**enerator
  - Commonly referred to as G

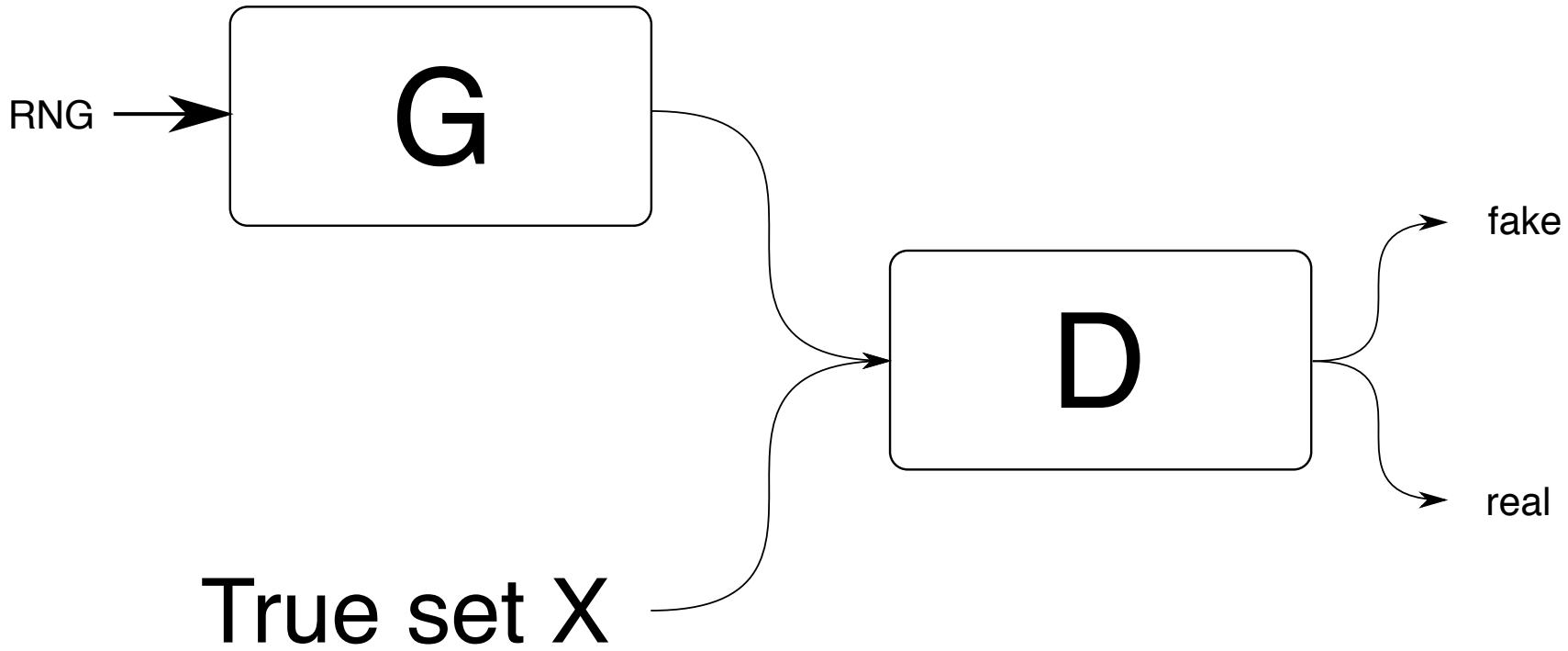
Nature


Predator gets better at
distinguishing insects and leafs
Species becomes a bit more leaf-like

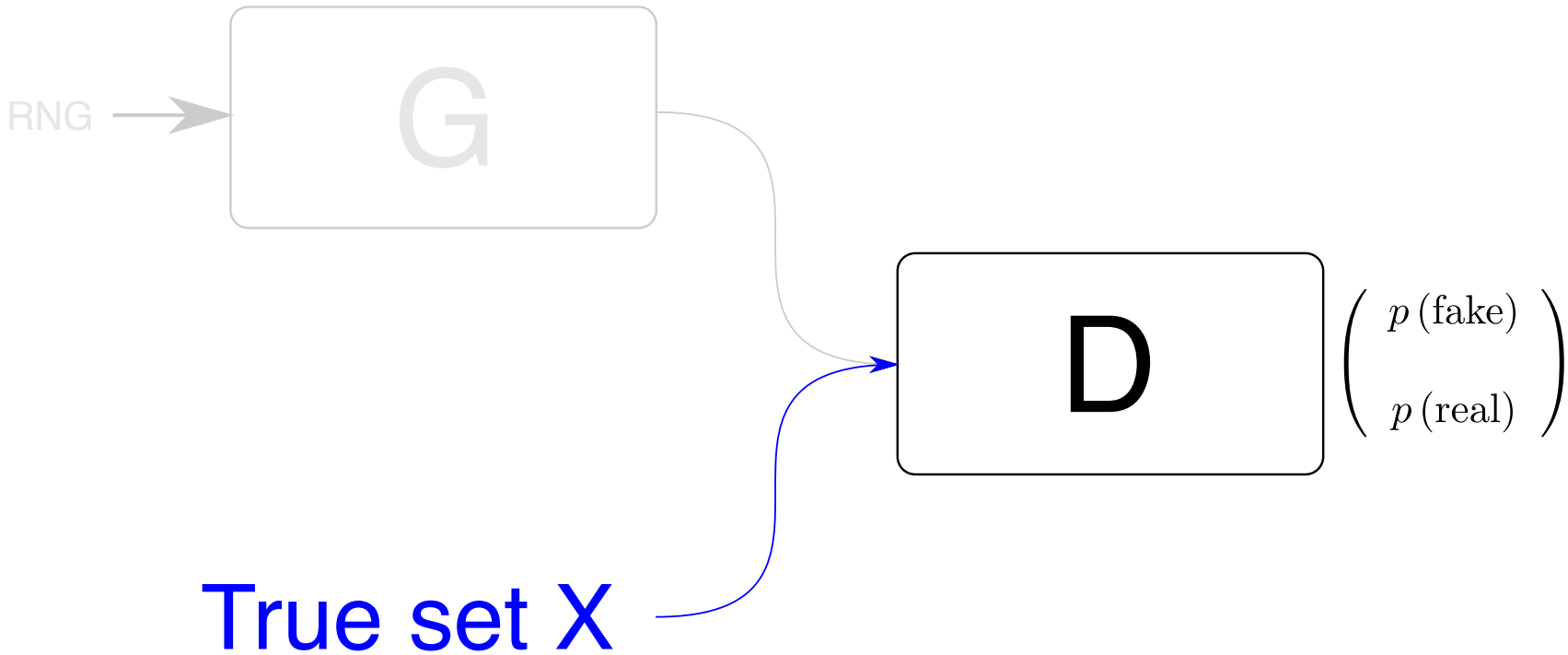
Computer


Consider
generated data x'
real data x
Train D to minimize discrimination loss
Generate fake data x'
Observe D(x)
Train G to maximize D' loss

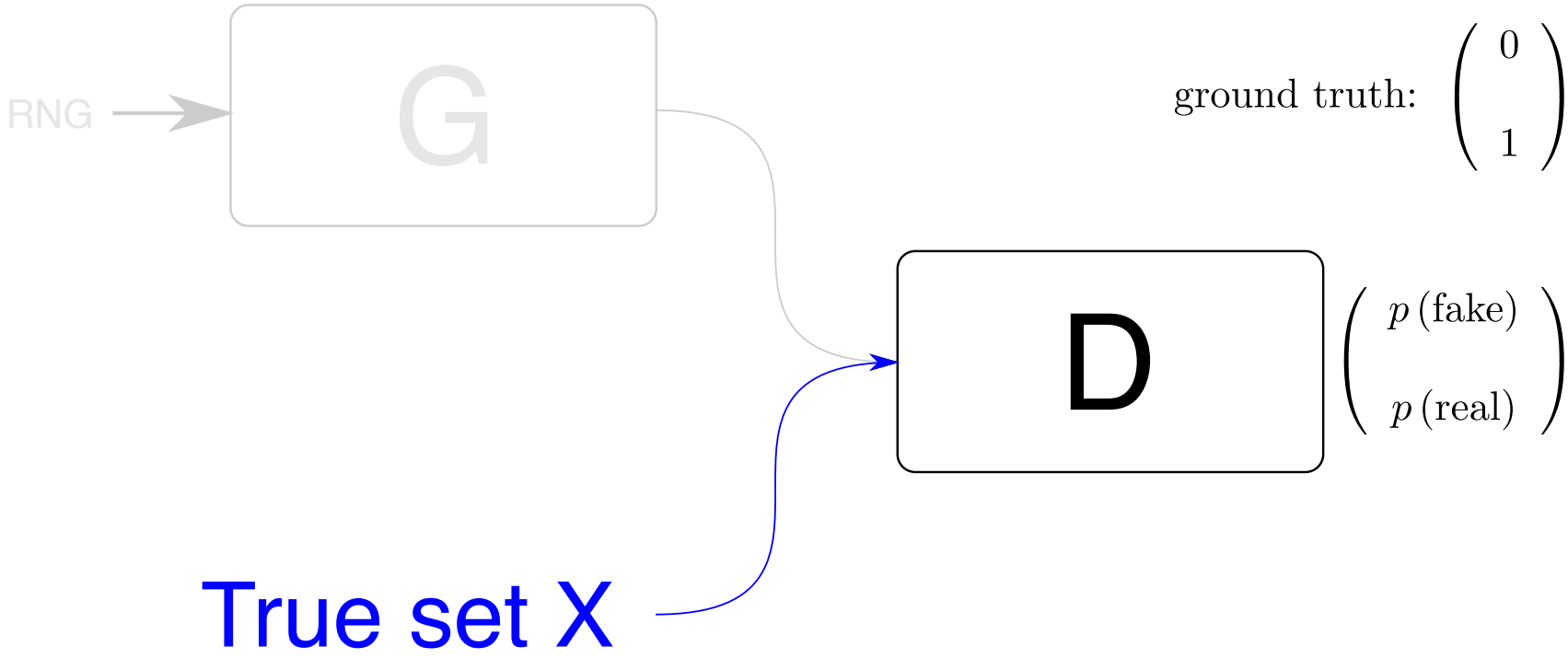
GAN



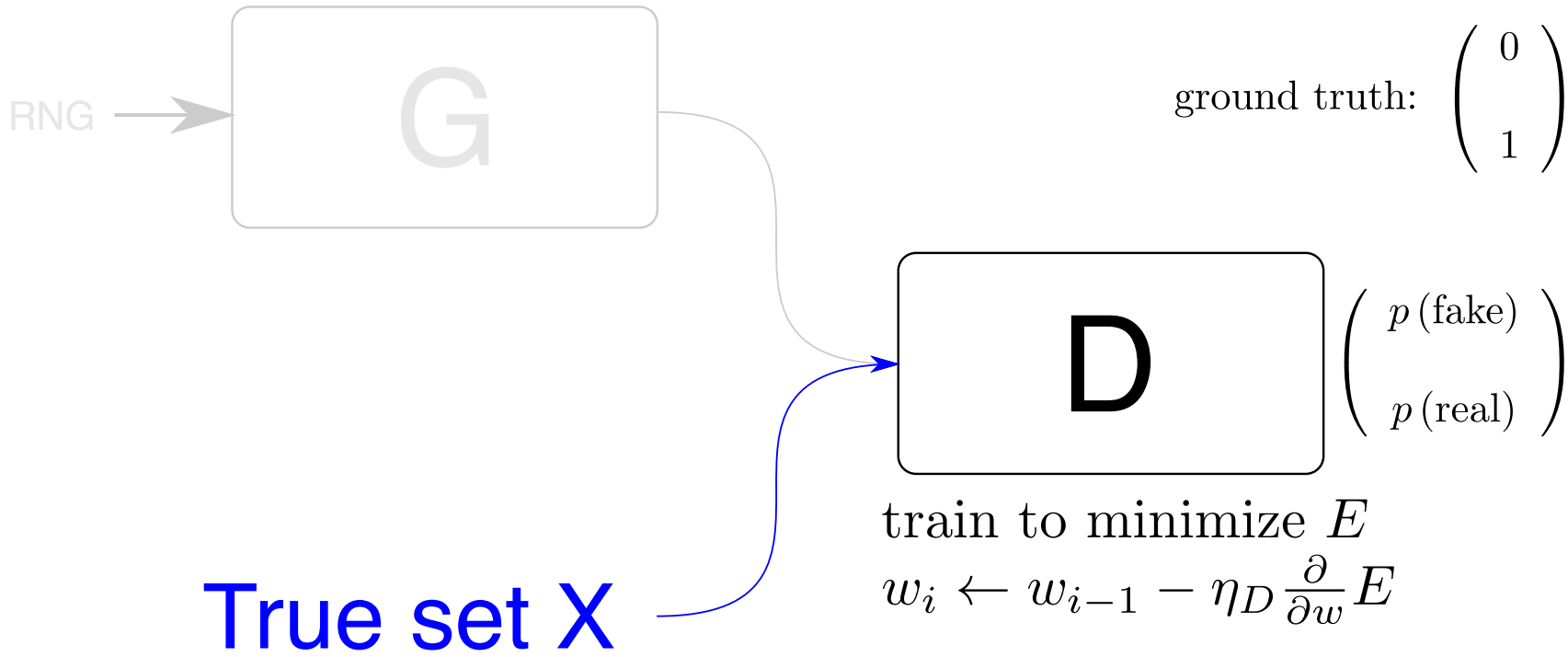
GAN



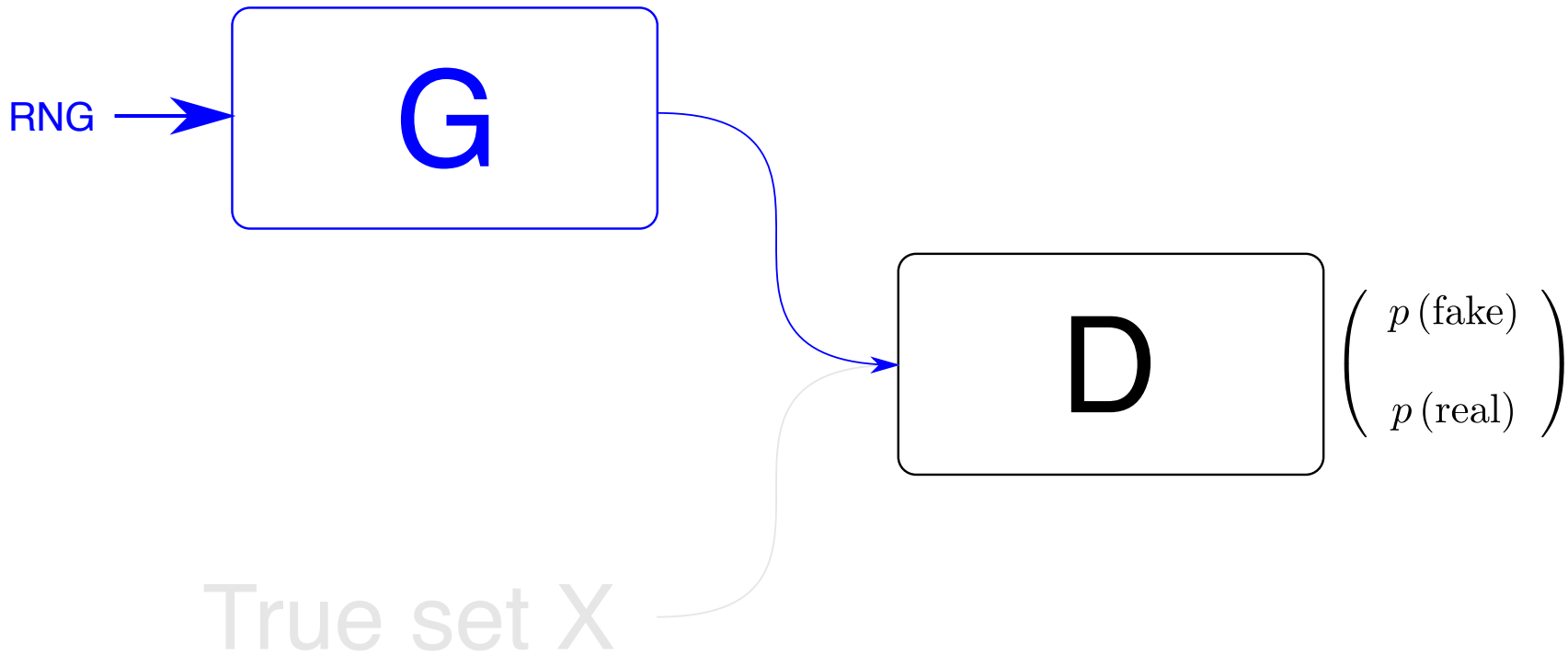
GAN



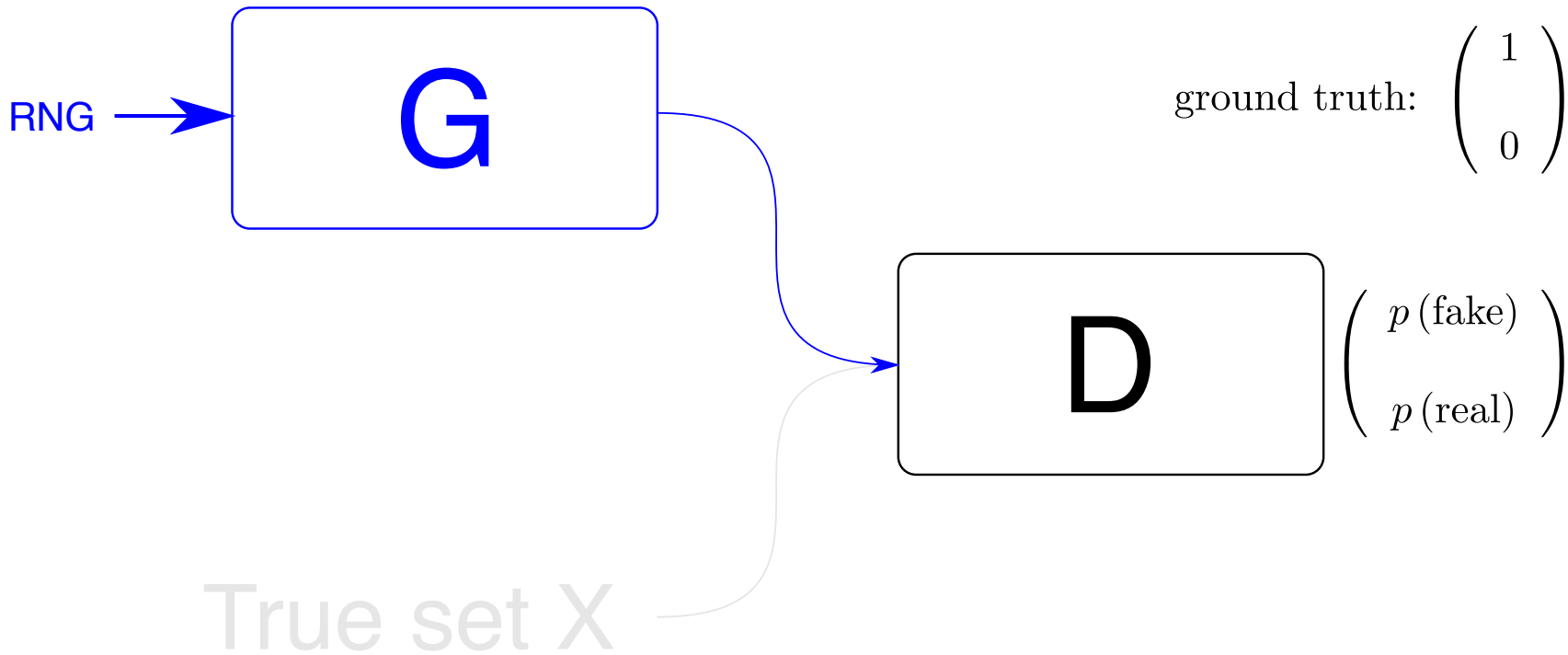
GAN


RNG
G
ground truth:
0
1
D
p (fake)
p (real)
train to minimize E
$w_i \leftarrow w_{i-1} - \eta_D \frac{\partial}{\partial w} E$
True set X

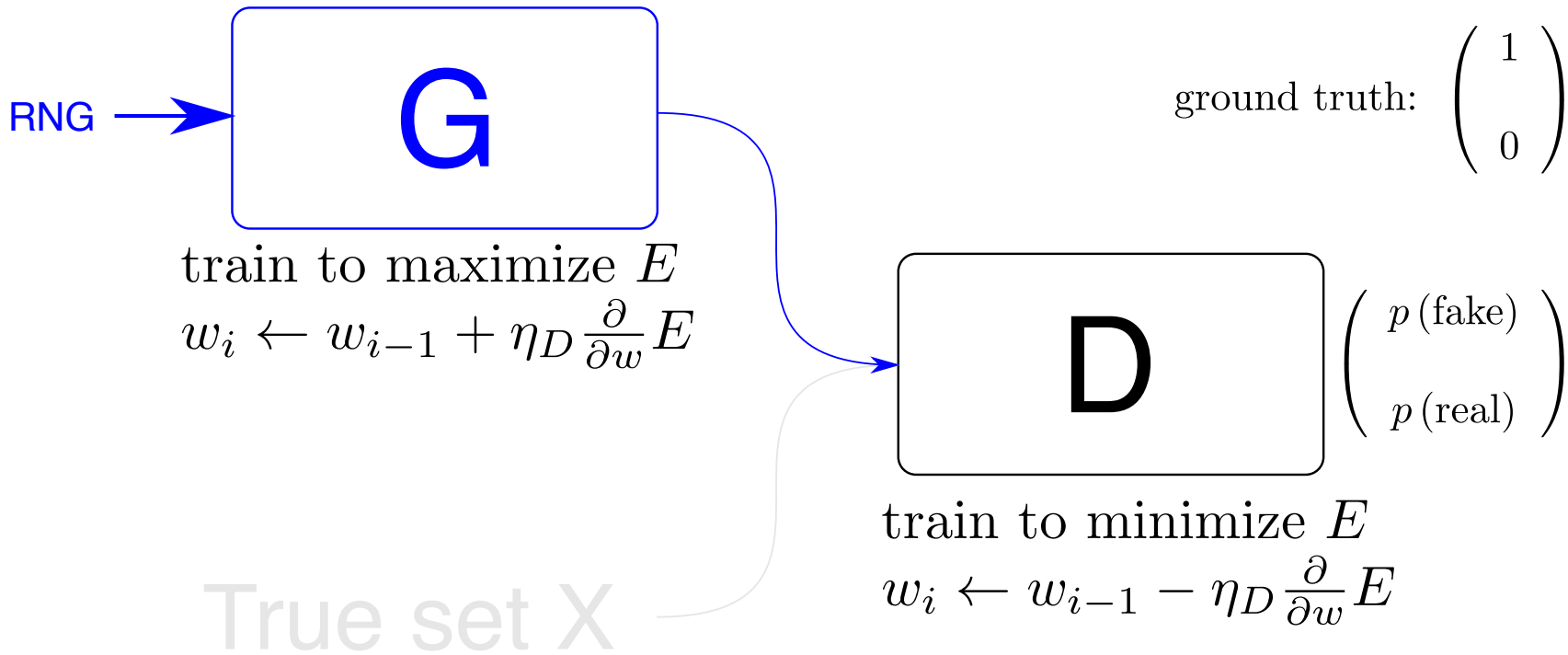
GAN


RNG
G
D
$\begin{pmatrix} p\,(\text{fake}) \\ p\,(\text{real}) \end{pmatrix}$
True set X

GAN



GAN


RNG
G
train to maximize E
$w_i \leftarrow w_{i-1} + \eta_D \frac{\partial}{\partial w} E$
ground truth: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$
D
$\begin{pmatrix} p\,(\text{fake}) \\ p\,(\text{real}) \end{pmatrix}$
train to minimize E
$w_i \leftarrow w_{i-1} - \eta_D \frac{\partial}{\partial w} E$
True set X