10-423/10-623 Generative AI

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Generative Adversarial Networks (GANs)

Matt Gormley
Lecture 6
Feb. 5, 2024

Reminders

- Homework 1: Generative Models of Text
  - Out: Thu, Jan 25
  - Due: Wed, Feb 7 at 11:59pm
- Matt's office hours on GCal

MODEL: GENERATIVE ADVERSARIAL NETWORK (GAN)

Stable Diffusion still can’t explain GANs

Prompt: slide explaining Generative Adversarial Networks (GANs) for Intro to Machine Learning course, carefully designed, easy to follow

Negative Prompt: boring, unclear, nontechnical

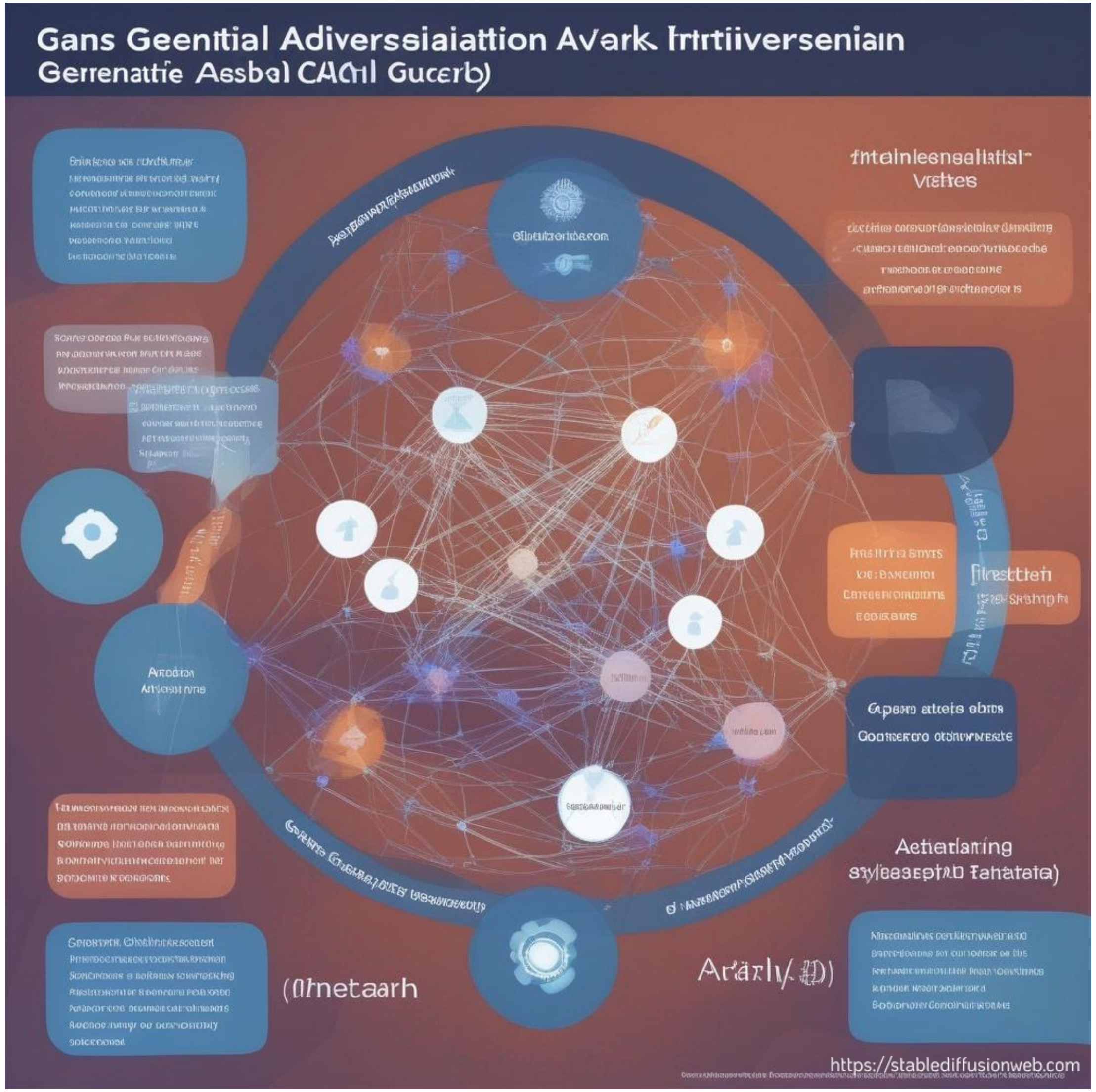


Figure from https://stablediffusionweb.com/

Generative Adversarial Networks (GANs)

A GAN consists of two deterministic neural network models:

1) the Generator

takes a vector of random noise as input, and generates an image

2) the Discriminator

takes in an image classifies whether it is real (label 1) or fake (label 0)

Generator Model

1) the Generator

takes a vector of random noise as input, and generates an image

Example Generator: DCGAN

- An inverted CNN with four **fractionally-strided** convolution layers (not deconvolution)
- These fractional strides grow the size of the image from layer to layer
- The final layer has three channels for red/green/blue

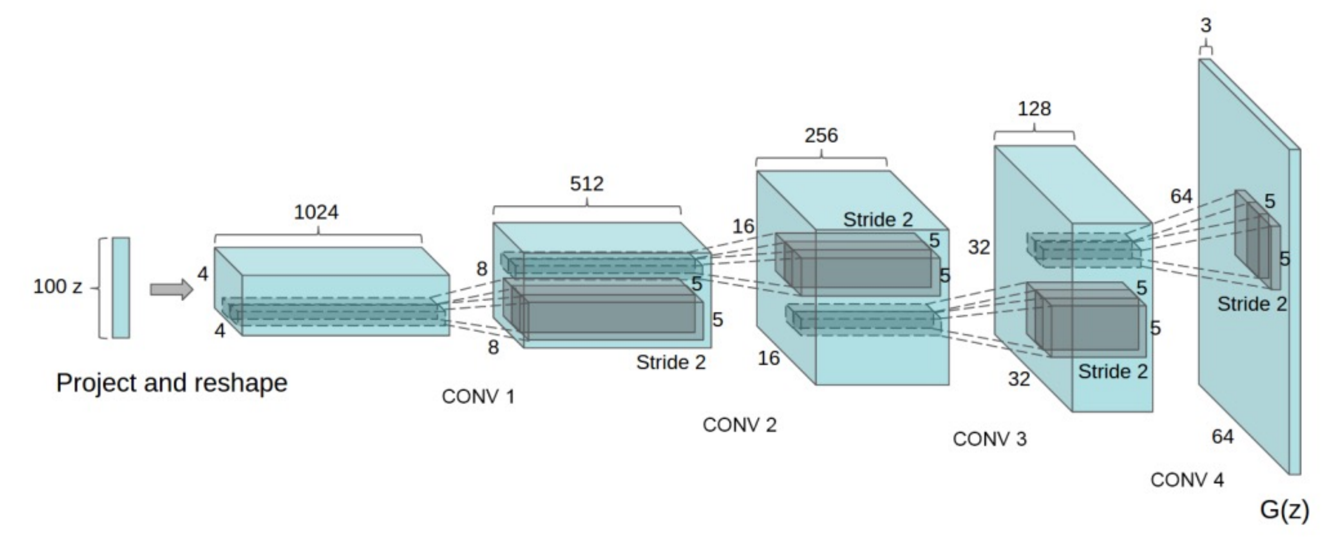


Figure from Radford et al. (2016)

Generative Adversarial Networks (GANs)

A GAN consists of two deterministic neural network models:

1) the Generator

takes a vector of random noise as input, and generates an image

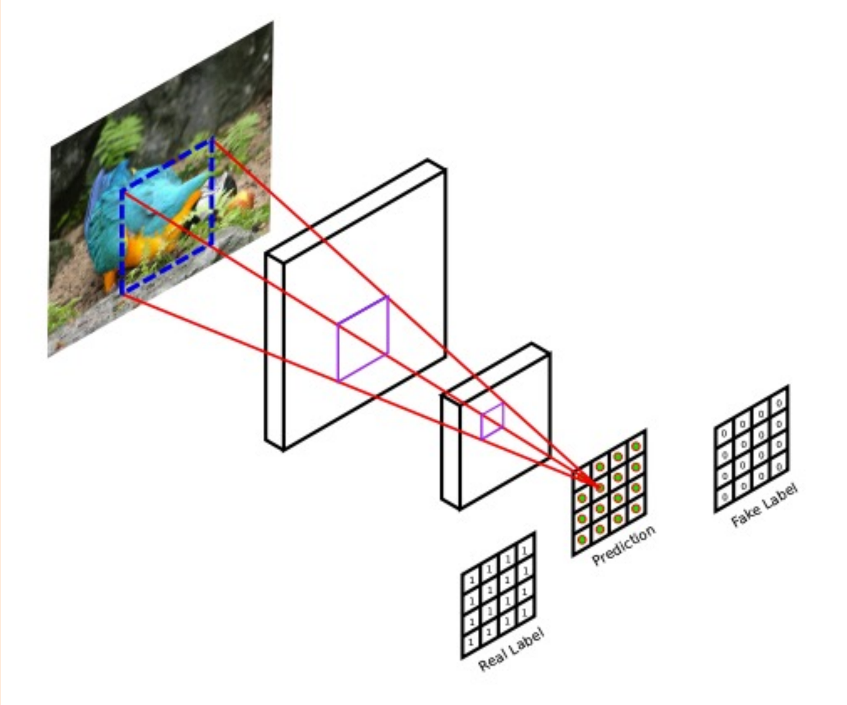
2) the Discriminator

takes in an image classifies whether it is real (label 1) or fake (label 0)

Discriminator Model

Example Discriminator: PatchGAN

- Convolutional neural network
- Looks at each patch of the image and tries to predict whether it is real or fake
- Helps avoid producing blurry images



2) the Discriminator

takes in an image classifies whether it is real (label 1) or fake (label 0)

Figure from Demir & Unal (2018)

Generative Adversarial Networks (GANs)

A GAN consists of two deterministic neural network models:

1) the Generator

takes a vector of random noise as input, and generates an image

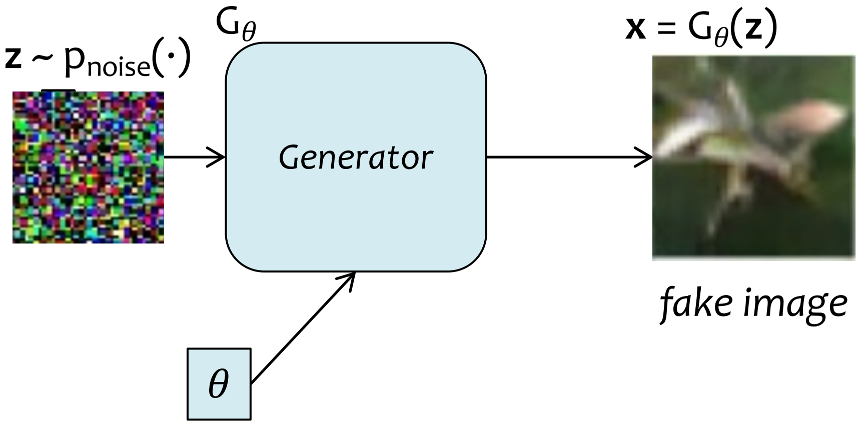
2) the Discriminator

takes in an image classifies whether it is real (label 1) or fake (label 0)

In training, the GAN plays a two player minimax game:

1. the Generator tries to create realistic images to fool the Discriminator into thinking they are real
2. the Discriminator tries to identify the real images from the fake

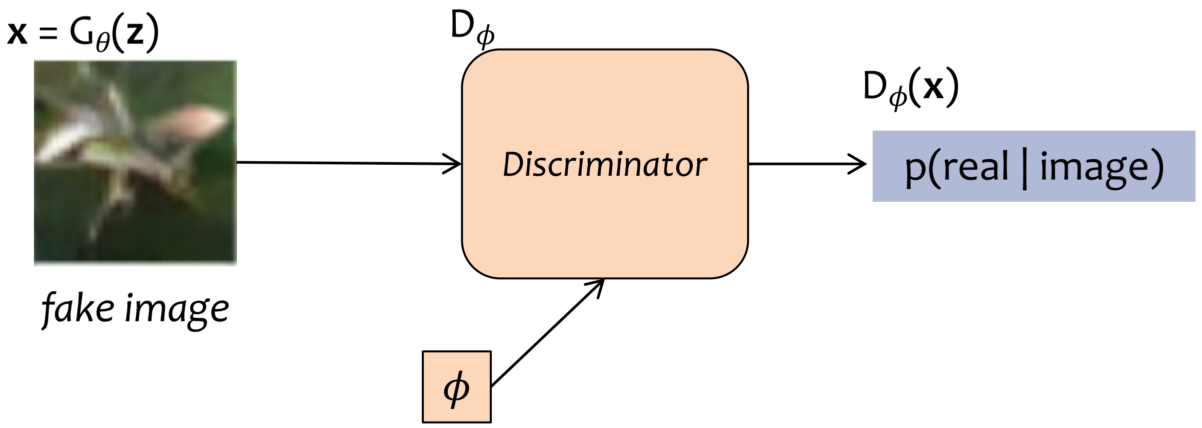
Generative Adversarial Networks (GANs)



Real/fake images from Huang et al. (2017)

Gaussian noise from https://physbam.stanford.edu/cs448x/old/Noise_Review.html

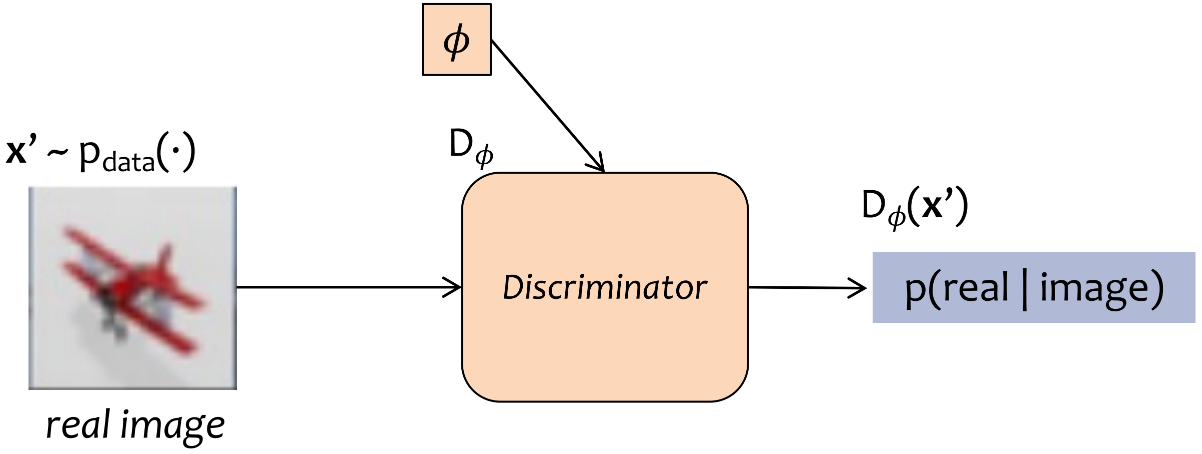
Generative Adversarial Networks (GANs)



Real/fake images from Huang et al. (2017)

Gaussian noise from https://physbam.stanford.edu/cs448x/old/Noise_Review.html

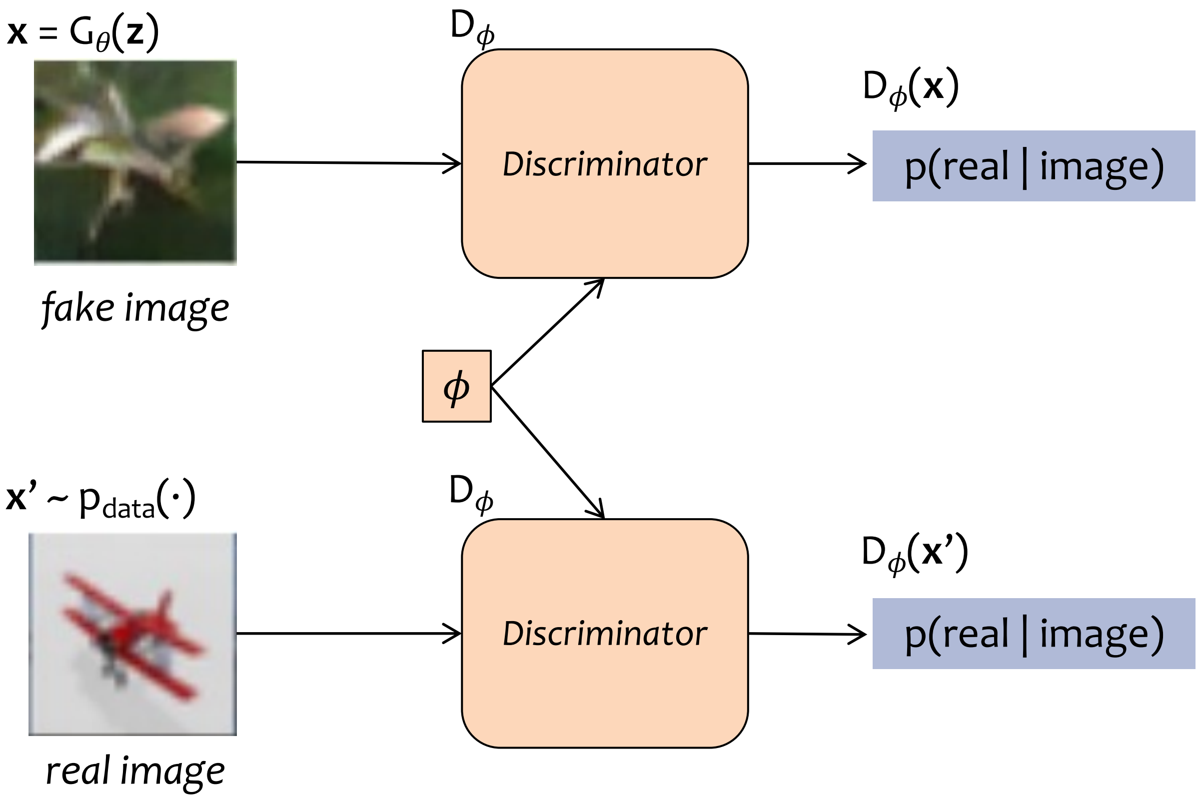
Generative Adversarial Networks (GANs)



Real/fake images from Huang et al. (2017)

Gaussian noise from https://physbam.stanford.edu/cs448x/old/Noise_Review.html

Generative Adversarial Networks (GANs)



Real/fake images from Huang et al. (2017)

Gaussian noise from https://physbam.stanford.edu/cs448x/old/Noise_Review.html

LEARNING FOR GANS

Generative Adversarial Networks (GANs)

A GAN consists of two deterministic neural network models:

1) the Generator

takes a vector of random noise as input, and generates an image

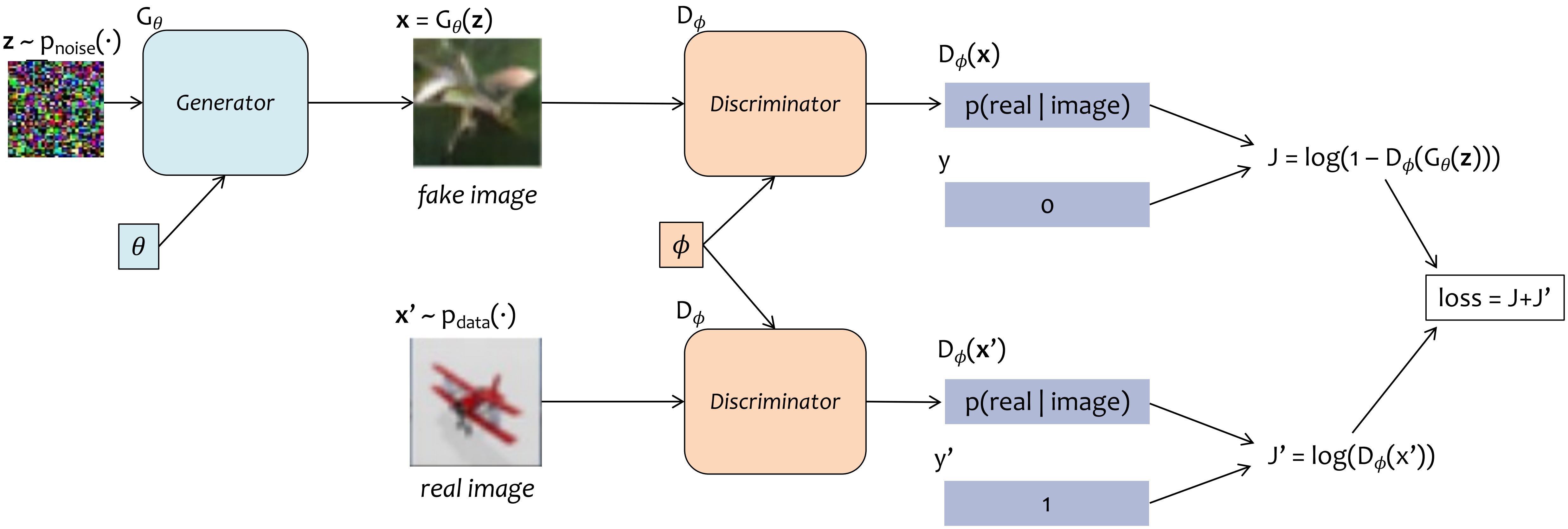
2) the Discriminator

takes in an image classifies whether it is real (label 1) or fake (label 0)

In training, the GAN plays a two player minimax game:

1. the Generator tries to create realistic images to fool the Discriminator into thinking they are real
2. the Discriminator tries to identify the real images from the fake

Generative Adversarial Networks (GANs)



Real/fake images from Huang et al. (2017)

Gaussian noise from https://physbam.stanford.edu/cs448x/old/Noise_Review.html

Generative Adversarial Networks (GANs)

$$\max_{\phi} \log \left(D_\phi(\mathbf{x}^{(i)}) \right) + \log \left(1 - D_\phi(G_\theta(\mathbf{z}^{(i)})) \right)$$

The discriminator is trying to maximize the likelihood of a binary classifier with labels {real = 1, fake = 0}, on the fixed output of the generator

$$\min_{\theta} \log \left(1 - D_\phi(G_\theta(\mathbf{z}^{(i)})) \right)$$

The generator is trying to minimize the likelihood of its generated (fake) image being classified as fake, according to a fixed discriminator

In training, the GAN plays a two player minimax game:

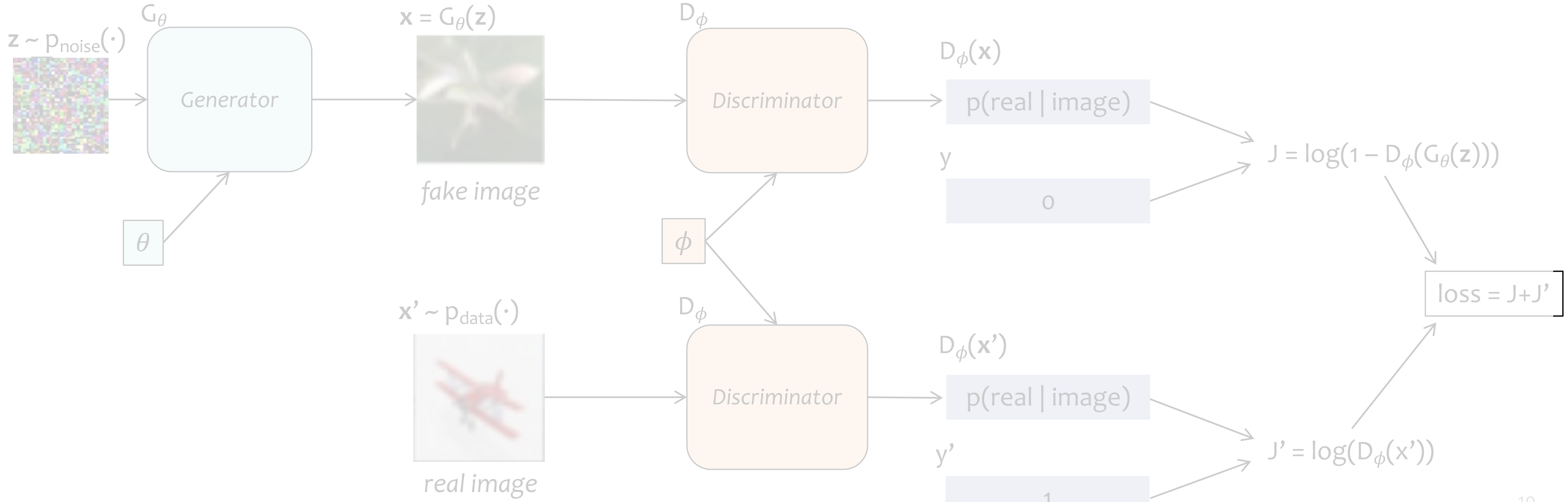
1. the Generator tries to create realistic images to fool the Discriminator into thinking they are real
2. the Discriminator tries to identify the real images from the fake

Learning a GAN

- Objective function is a simple differentiable function
- We chose G and D to be differentiable neural networks

Training alternates between:

- Keep G_θ fixed and backprop through D_ϕ
- Keep D_ϕ fixed and backprop through G_θ



Real/fake images from Huang et al. (2017)

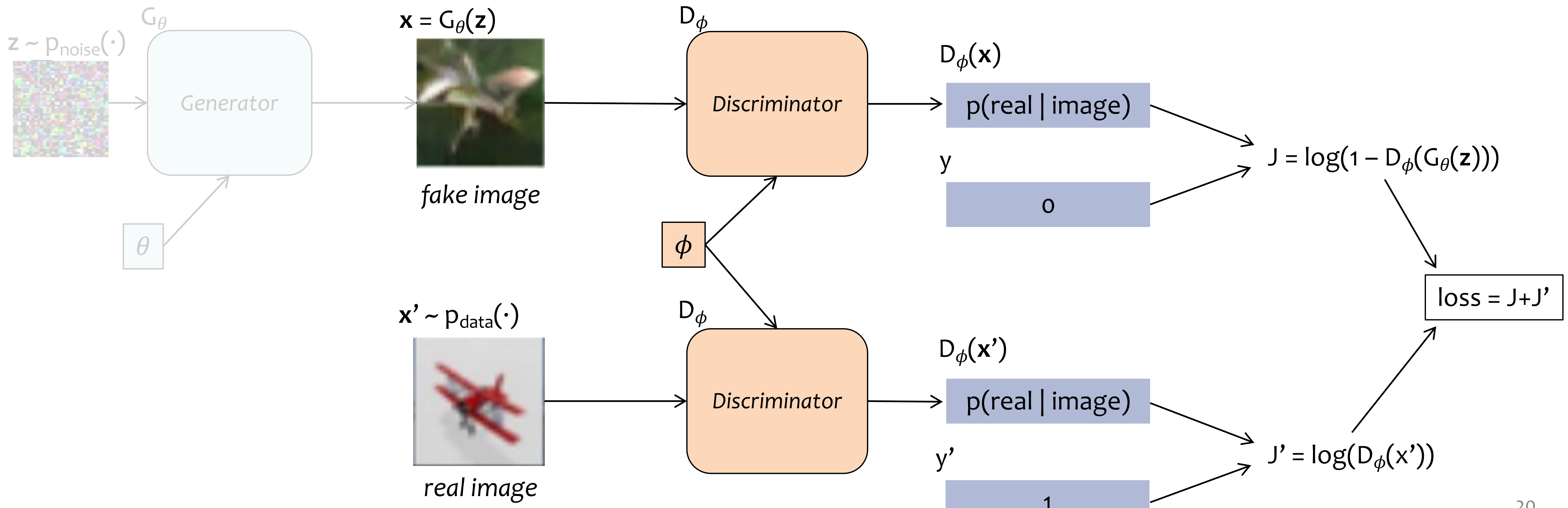
Gaussian noise from https://physbam.stanford.edu/cs448x/old/Noise_Review.html

Learning a GAN

- Objective function is a simple differentiable function
- We chose G and D to be differentiable neural networks

Training alternates between:

- **Keep G_θ fixed and backprop through D_ϕ**
- Keep D_ϕ fixed and backprop through G_θ



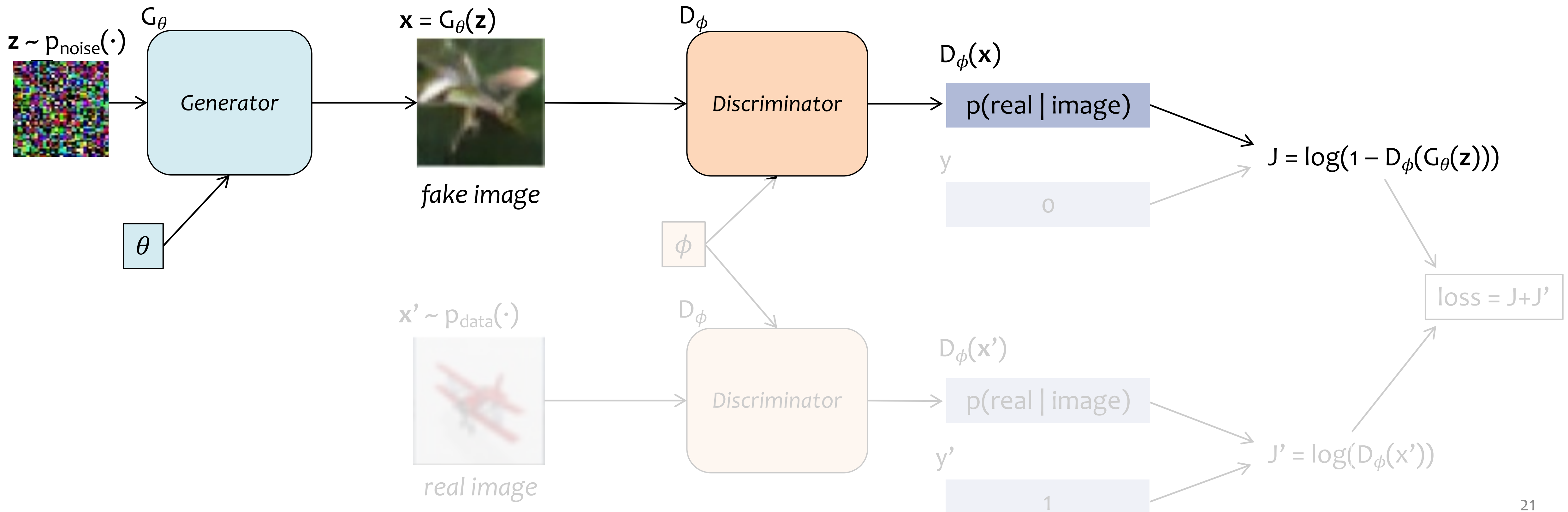
Real/fake images from Huang et al. (2017)

Learning a GAN

- Objective function is a simple differentiable function
- We chose G and D to be differentiable neural networks

Training alternates between:

- Keep G_θ fixed and backprop through D_ϕ
- **Keep D_ϕ fixed and backprop through G_θ**



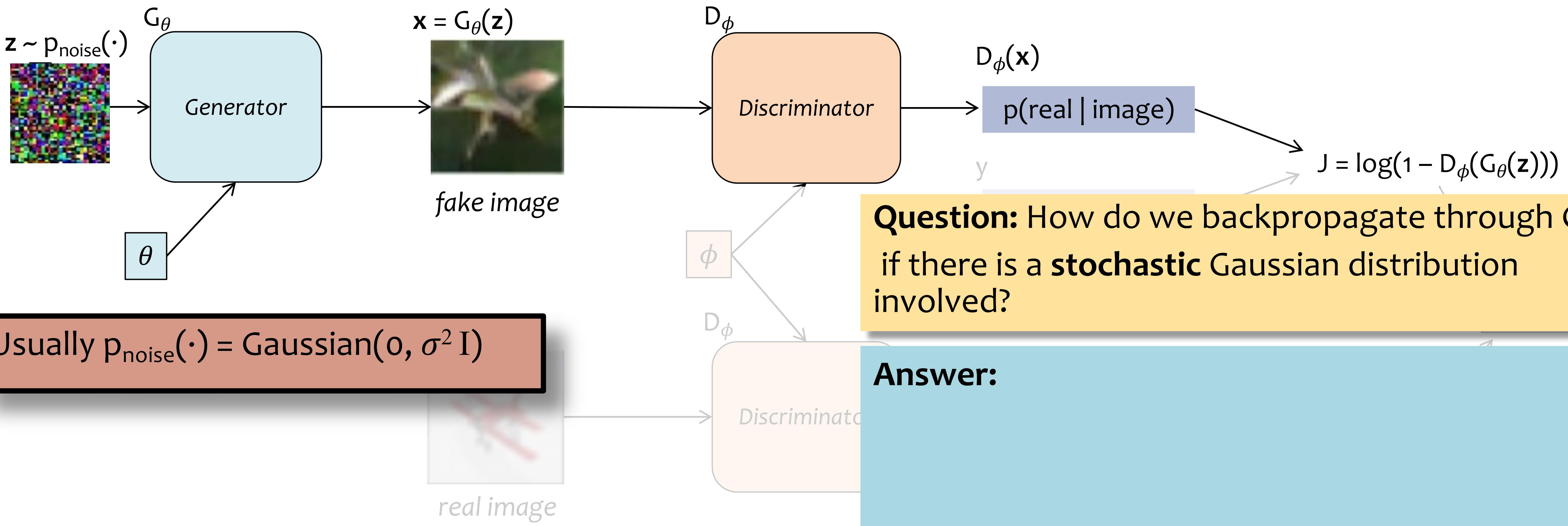
Real/fake images from Huang et al. (2017)

Learning a GAN

- Objective function is a simple differentiable function
- We chose G and D to be differentiable neural networks

Training alternates between:

- Keep G_θ fixed and backprop through D_ϕ
- **Keep D_ϕ fixed and backprop through G_θ**



Usually $p_{noise}(\cdot)$ = Gaussian$(0, \sigma^2 I)$

Question: How do we backpropagate through G_θ if there is a **stochastic** Gaussian distribution involved?

Answer:

Real/fake images from Huang et al. (2017)

Learning a GAN

- Training data consists of a collection of m unlabeled images $x^{(1)}, \ldots, x^{(m)}$
- Optimization is similar to block coordinate descent
- But instead of exactly solving the min/max problem, we take a step of mini-batch SGD

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.

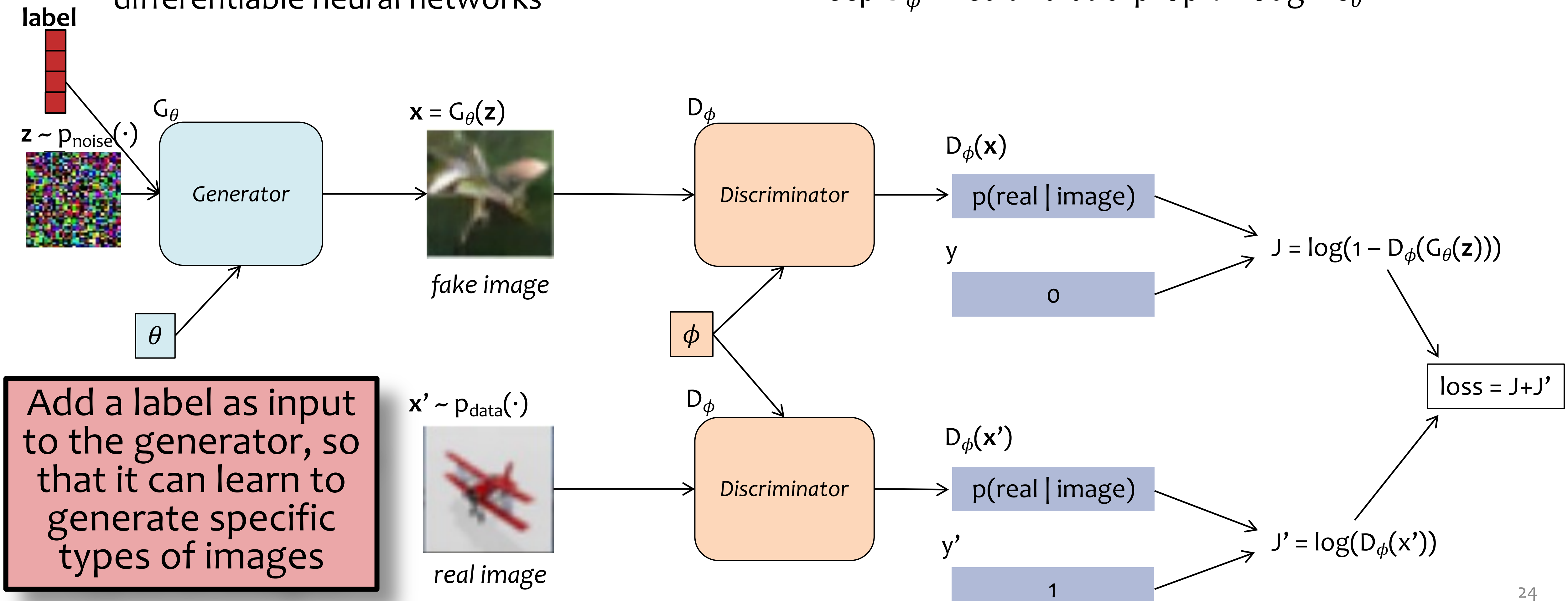
Figure from https://arxiv.org/pdf/1406.2661.pdf

Class-conditional GANs

- Objective function is a simple differentiable function
- We chose G and D to be differentiable neural networks

Training alternates between:

- Keep G_θ fixed and backprop through D_ϕ
- Keep D_ϕ fixed and backprop through G_θ



Real/fake images from Huang et al. (2017)

SCALING UP THE MODEL SIZE

Scaling Up the Model Size

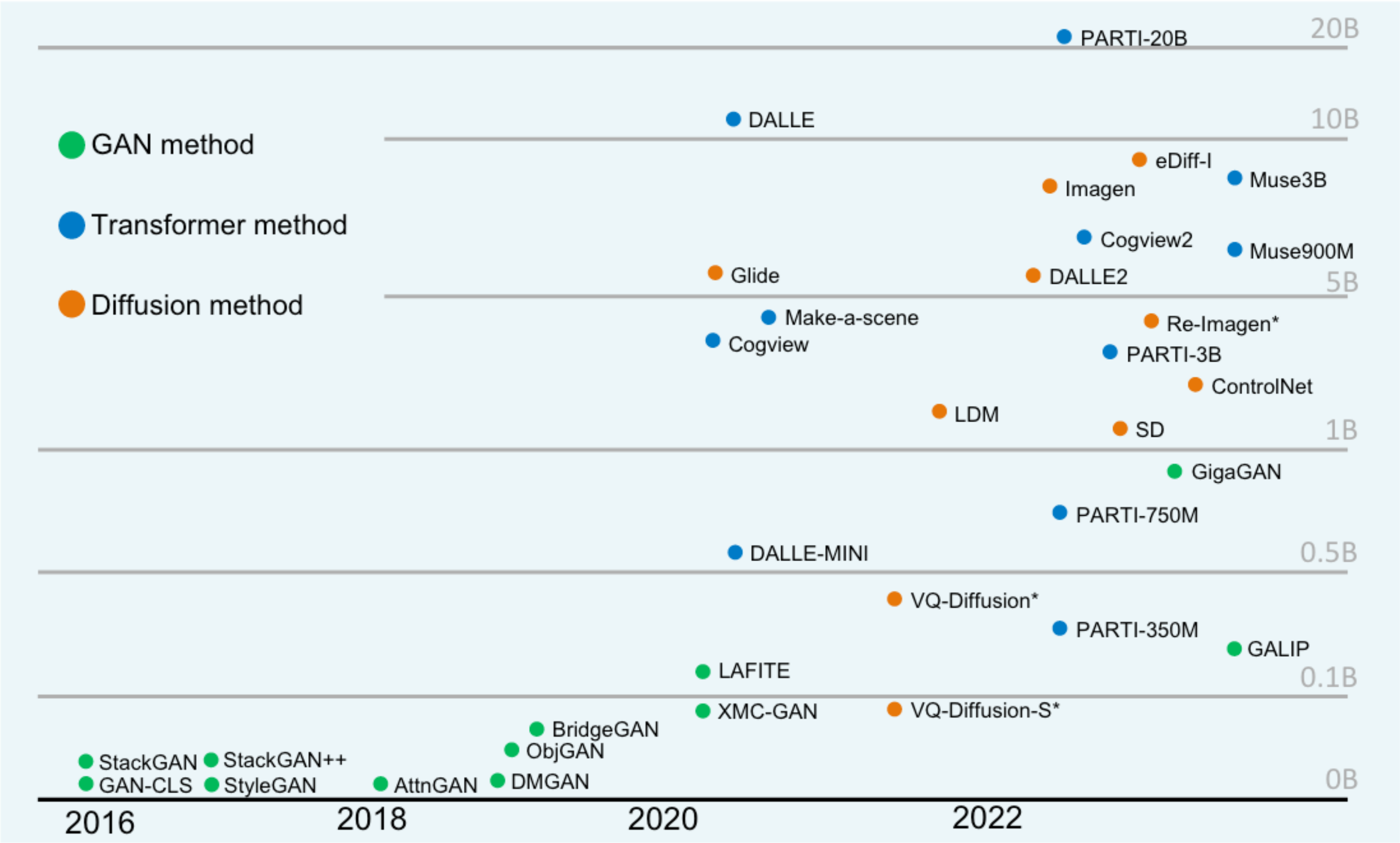


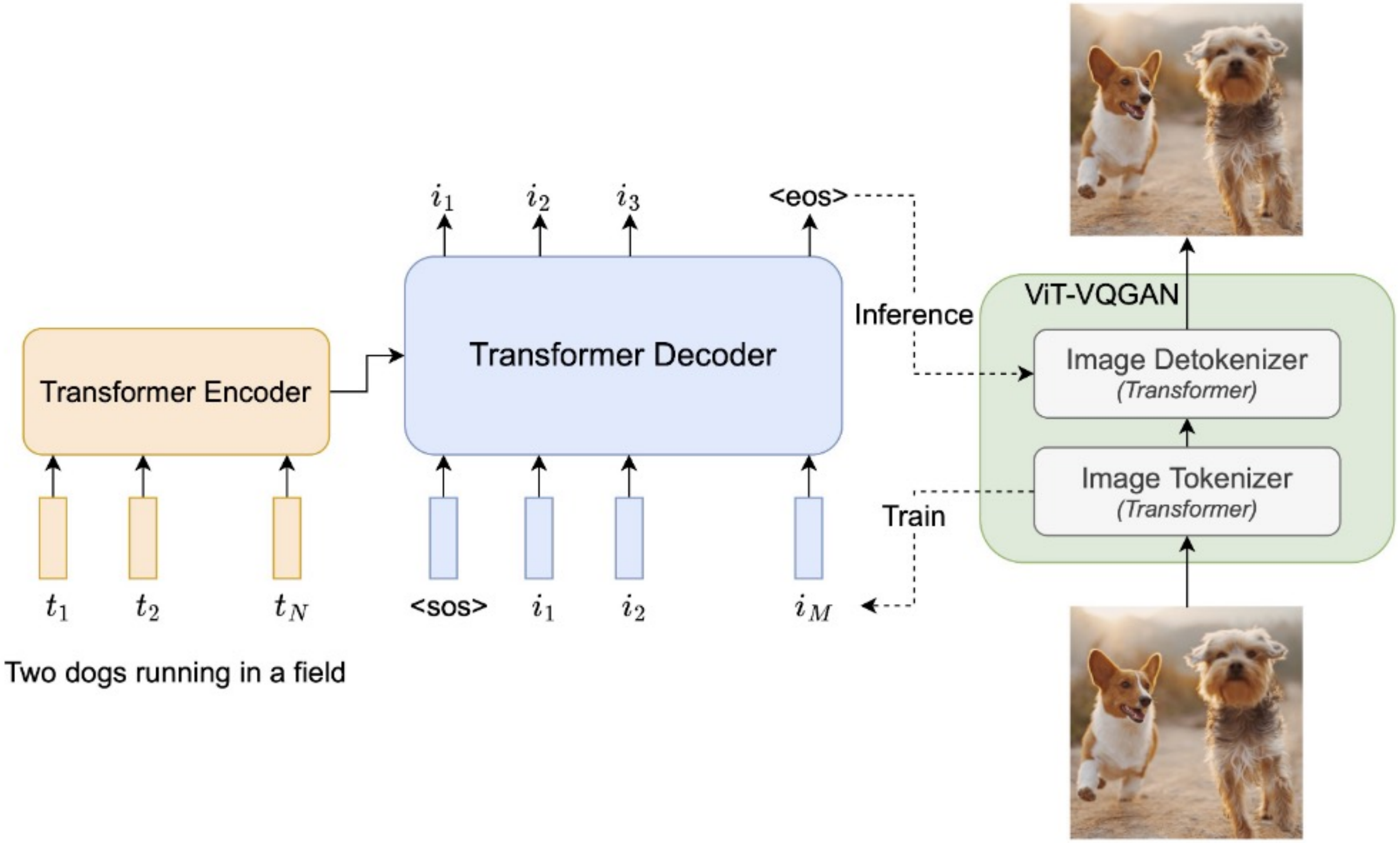
Fig. 5. Timeline of TTI model development, where green dots are GAN TTI models, blue dots are autoregressive Transformers and orange dots are Diffusion TTI models. Models are separated by their parameter, which are in general counted for all their components. Models with asterisk are calculated without the involvement of their text encoders.

Figure from Bie et al. (2023)

Scaling Up the Model Size

The Pathways Autoregressive Text-to-Image (Parti) model:

- treat image generation as a sequence-to-sequence problem
- text prompt is input to encoder
- sequence of image tokens is output of decoder
- ViT-VQGAN takes in the image tokens and generates a high-quality image



Scaling Up the Model Size

Prompt: A portrait photo of a kangaroo wearing an orange hoodie and blue sunglasses standing on the grass in front of the Sydney Opera House holding a sign on the chest that says Welcome Friends!

Parti with different model sizes

Figure from https://sites.research.google/parti/

Watermarking & Attribution

- **Watermarking**
  - A digital watermark allows one to identify when an image has been created by a model
  - Most methods for image generation (GANs, VAEs, stable diffusion) can be augmented with watermarking
- **Fake-image Detection**
  - Goal: identify fakes even without a watermark
- **Model Attribution**
  - Identify which generative model created an image (e.g. Dalle-2 vs. SDXL)
  - Very successful (natural watermarks)
- **Image Attribution**
  - Goal: identify the source images that led to the generation of a new image
  - Extremely challenging

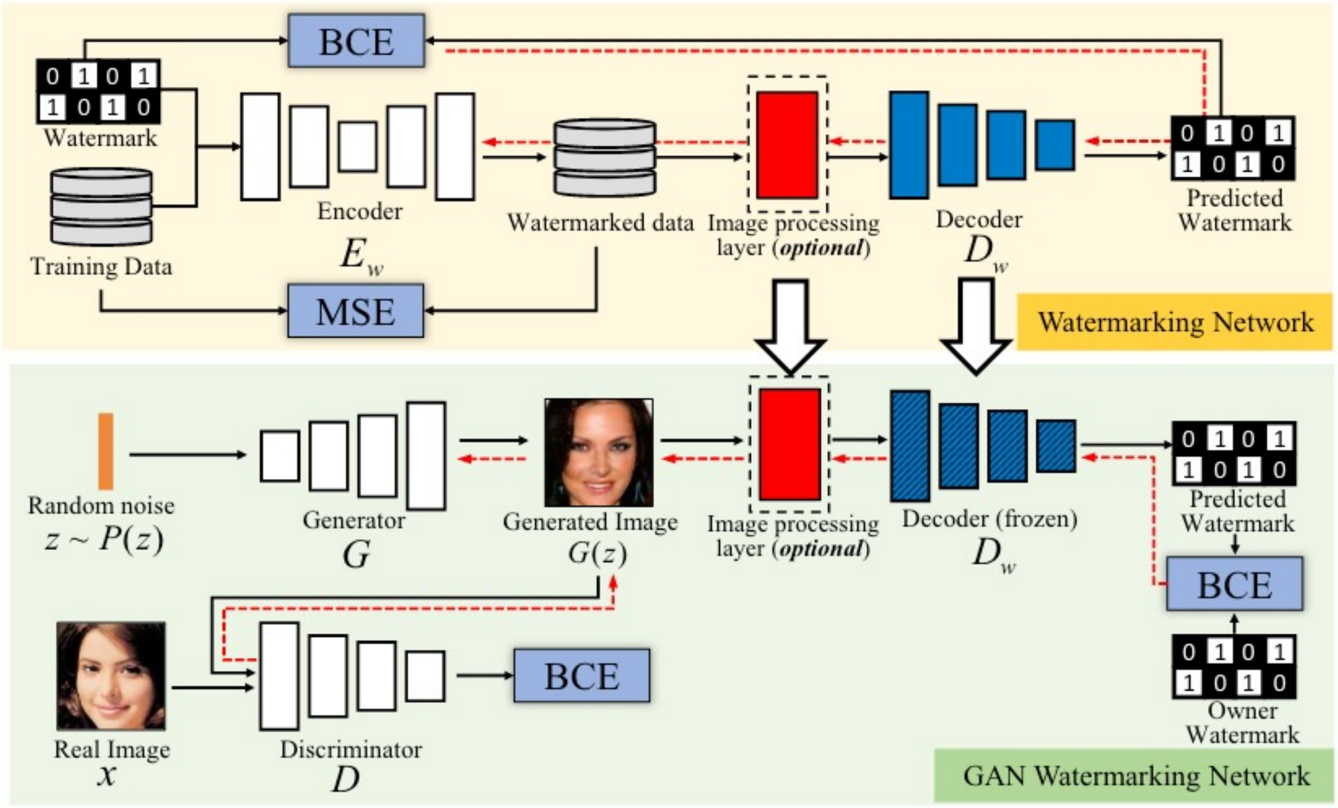


Figure from Fei et al. (2022)

SOCIETAL IMPACTS OF IMAGE GENERATION

Societal Impacts of Image Generation

Pros

- New tools for artists
- Faster creation of memes

Cons

- Copyright infringement / loss of work for artists
- Societal decrease in creativity
- Potential to create dehumanizing content
- Fake news / false realities / increased difficulty of fact checking
- Not rooted in reality
- Video generation is around the corner

DIFFUSION MODELS AND VARITIONAL AUTOENCODERS (VAES)

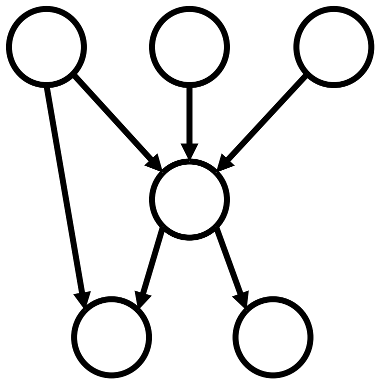
Diffusion Models

- Next we will consider (1) **diffusion models** and (2) **variational autoencoders (VAEs)**
  - Although VAEs came first, we're going to dive into diffusion models since they will receive more of our attention
- The steps in defining these models is roughly:
  - Define a probability distribution involving Gaussian noise
  - Use a variational lower bound as an objective function
  - Learn the parameters of the probability distribution by optimizing the objective function
- So what is a variational lower bound?

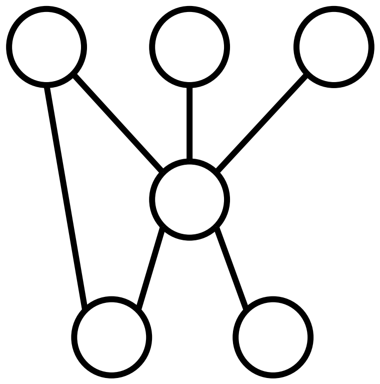
DIRECTED GRAPHICAL MODEL

Three Types of Graphical Models

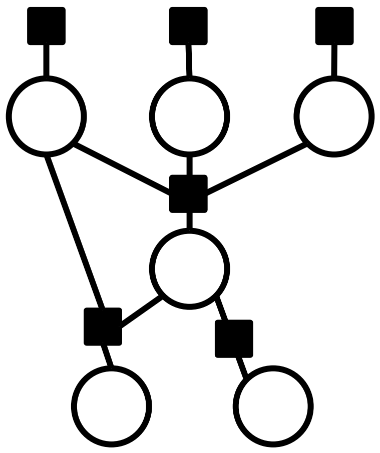
Directed Graphical Model



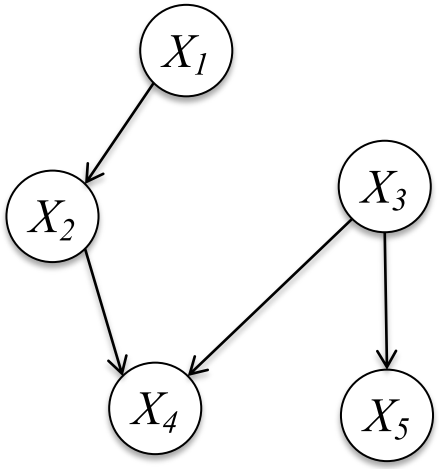
Undirected Graphical Model



Factor Graph



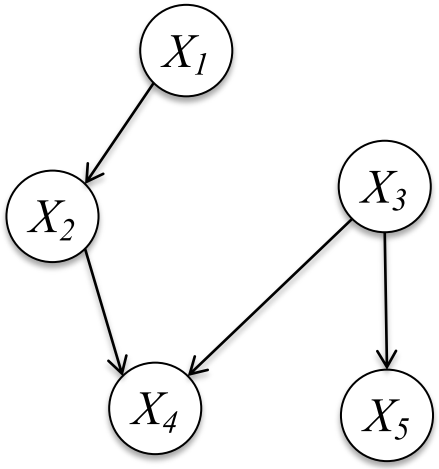
Bayesian Network



$$p(X_1, X_2, X_3, X_4, X_5) =$$
$$p(X_5|X_3)p(X_4|X_2, X_3)$$
$$p(X_3)p(X_2|X_1)p(X_1)$$

Bayesian Network

Definition:



$$P(X_1, \ldots, X_T) = \prod_{t=1}^{T} P(X_t \mid \text{parents}(X_t))$$

- A Bayesian Network is a **directed graphical model**
- It consists of a graph **G** and the conditional probabilities **P**
- These two parts full specify the distribution:
  - Qualitative Specification: **G**
  - Quantitative Specification: **P**

Qualitative Specification

- Where does the qualitative specification come from?

  – Prior knowledge of causal relationships
  – Prior knowledge of modular relationships
  – Assessment from experts
  – Learning from data (i.e. structure learning)
  – We simply prefer a certain architecture (e.g. a layered graph)
  – …

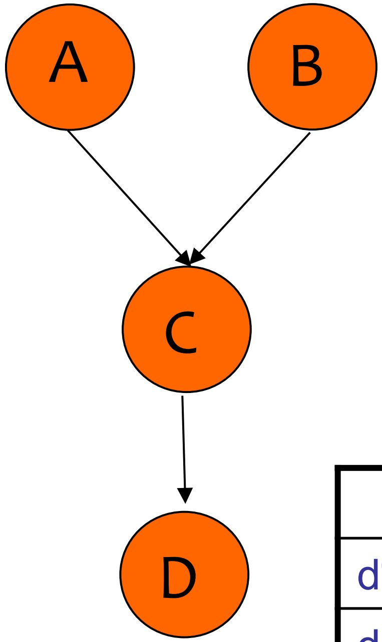
Quantitative Specification

Example: Conditional probability tables (CPTs)
for discrete random variables

a^0	0.75
a^1	0.25

b^0	0.33
b^1	0.67

$$P(a,b,c.d) = P(a)P(b)P(c|a,b)P(d|c)$$



	a^0b^0	a^0b^1	a^1b^0	a^1b^1
c^0	0.45	1	0.9	0.7
c^1	0.55	0	0.1	0.3

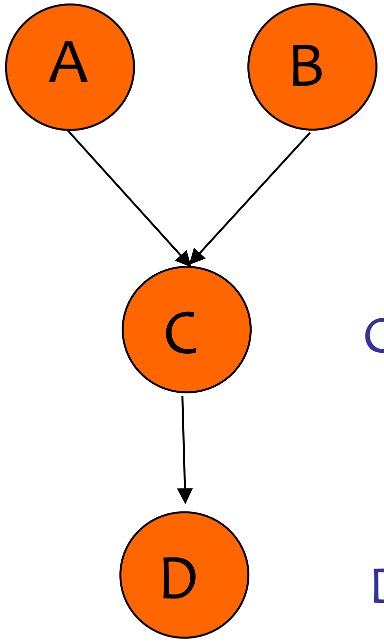
	c^0	c^1
d^0	0.3	0.5
d^1	07	0.5

Quantitative Specification

Example: Conditional probability density functions (CPDs) for continuous random variables

$$P(a,b,c.d) = P(a)P(b)P(c|a,b)P(d|c)$$

$A \sim N(\mu_a, \Sigma_a)$ $B \sim N(\mu_b, \Sigma_b)$



$C \sim N(A+B, \Sigma_c)$

$D \sim N(\mu_d + C, \Sigma_d)$

P(D| C)

D

C

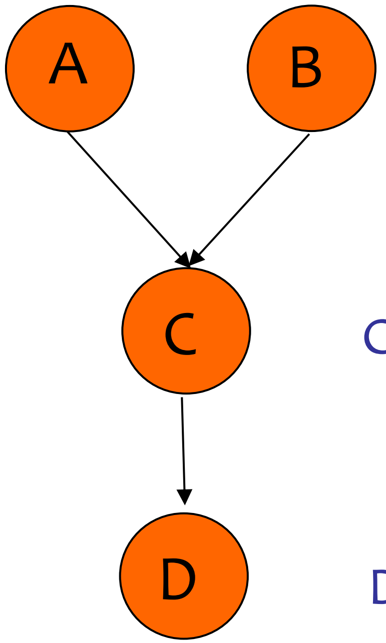
Quantitative Specification

Example: Combination of CPTs and CPDs for a mix of discrete and continuous variables

$$P(a,b,c.d) = P(a)P(b)P(c|a,b)P(d|c)$$

a^0	0.75
a^1	0.25

b^0	0.33
b^1	0.67



$C \sim N(A+B, \Sigma_c)$

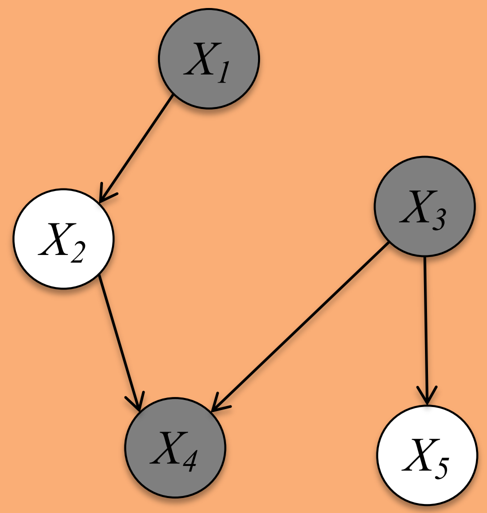
$D \sim N(\mu_d + C, \Sigma_d)$

Observed Variables

- In a graphical model, **shaded nodes** are "**observed**", i.e. their values are given

Example:

$$P(X_2, X_5 \mid X_1 = 0, X_3 = 1, X_4 = 1)$$



MARKOV MODEL

Markov Model

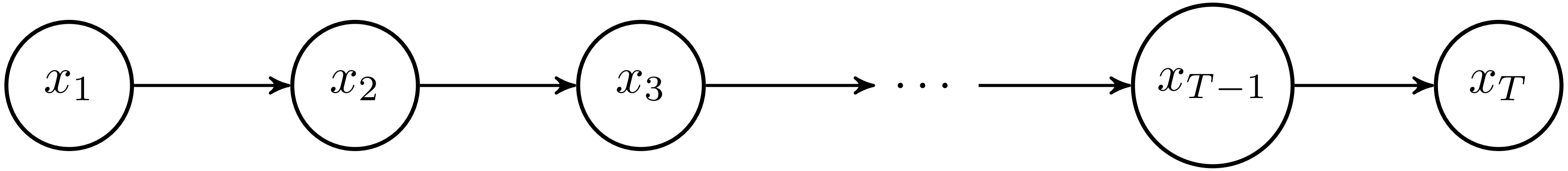
- **Markov assumption:** for a sequence of random variables, the probability distribution over x_t random variables is conditionally independent of x_1 ,… , x_{t-2} given x_{t-1}

$$p(x_t \mid x_1, \ldots, x_{t-1}) = p(x_t \mid x_{t-1})$$

- **Markov model:** defines a joint distribution over a sequence of variables using a Markov assumption

$$p(x_1, \ldots, x_T) = p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

- We can represent the Markov model as a **directed graphical model**

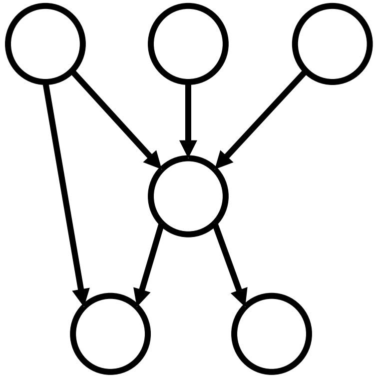


RNN as a DGM

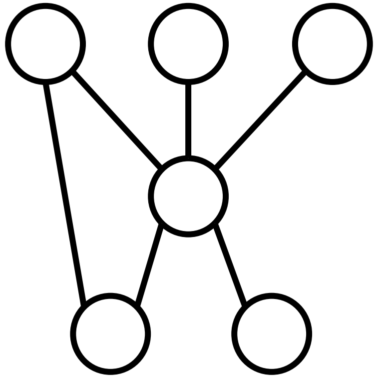
UNDIRECTED GRAPHICAL MODELS

Three Types of Graphical Models

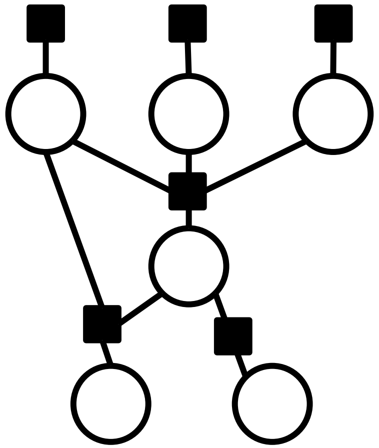
Directed Graphical Model



Undirected Graphical Model



Factor Graph



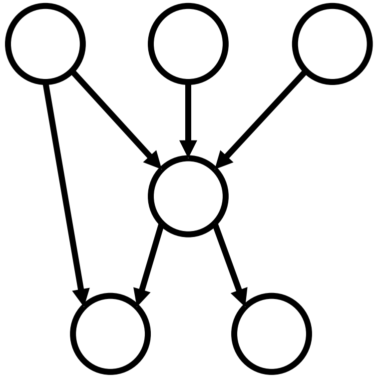
Undirected Graphical Models

Representation of both directed and undirected graphical models

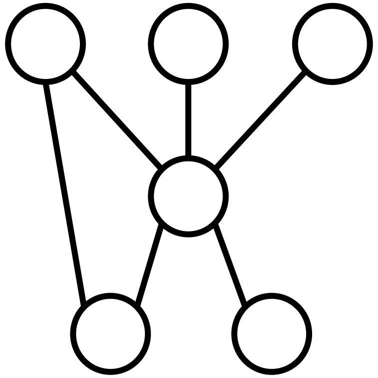
FACTOR GRAPHS

Three Types of Graphical Models

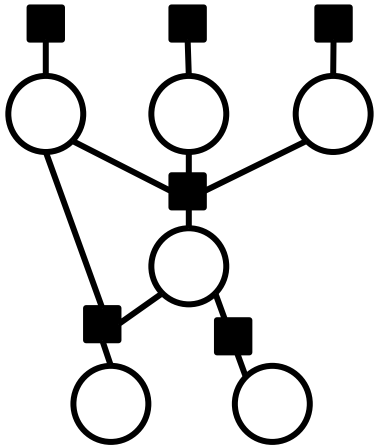
Directed Graphical Model



Undirected Graphical Model



Factor Graph



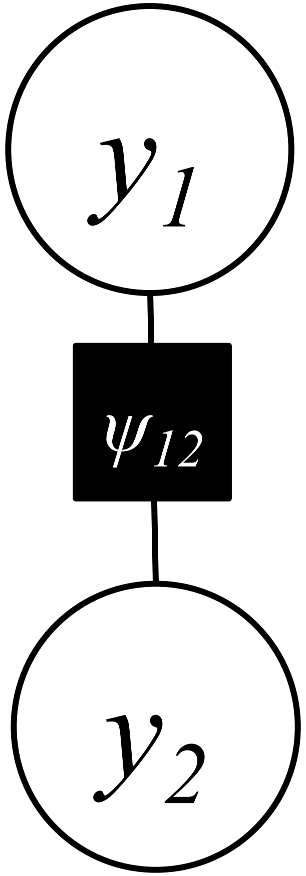
Mathematical Modeling

Factor Graphs

Factor Graph

(bipartite graph)

- variables (circles)
- factors (squares)



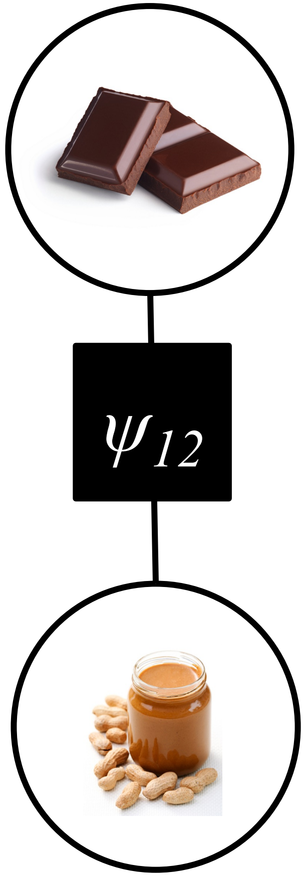
Mathematical Modeling

Factor Graphs

Factor Graph

(bipartite graph)

- variables (circles)
- factors (squares)



Each **random variable** can be assigned a **value**

The collection of values for all the random variables is called an **assignment.**

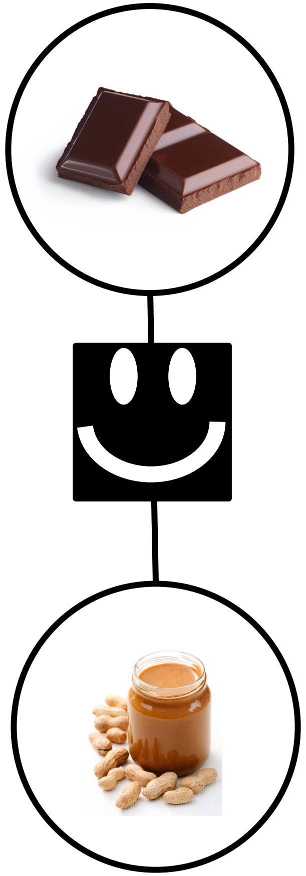
Mathematical Modeling

Factor Graphs

Factor Graph

(bipartite graph)

- variables (circles)
- factors (squares)



Factors have local opinions about the assignments of their neighboring variables

Mathematical Modeling

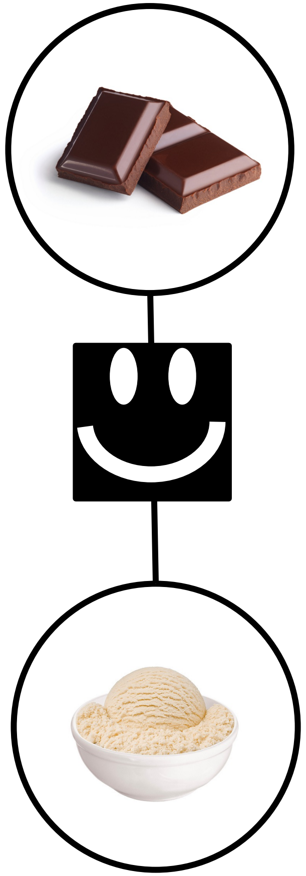
Factor Graphs

Factor Graph

(bipartite graph)

- variables (circles)
- factors (squares)

Factors have local opinions about the assignments of their neighboring variables



Mathematical Modeling

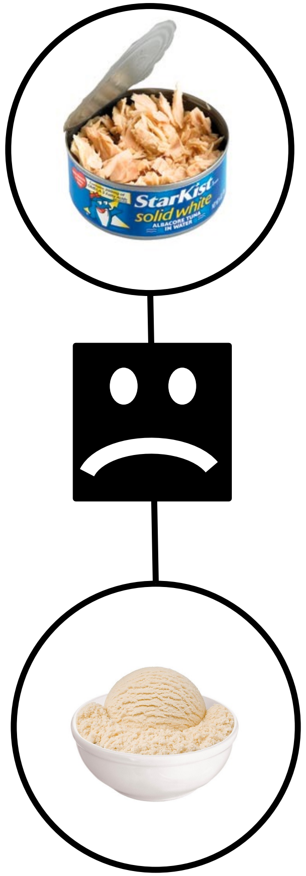
Factor Graphs

Factor Graph

(bipartite graph)

- variables (circles)
- factors (squares)

Factors have local opinions about the assignments of their neighboring variables



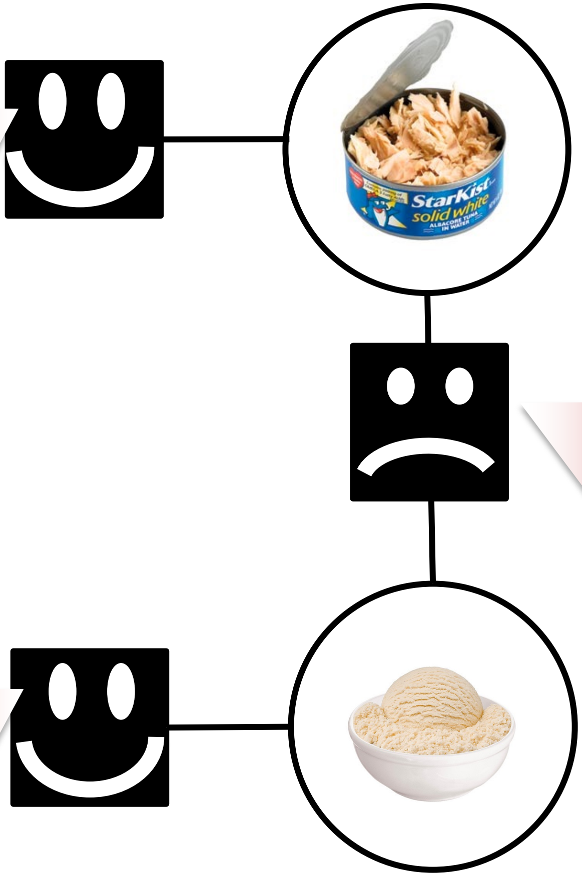
Mathematical Modeling

Factor Graphs

$P(\text{tuna, ice cream}) \;=\; ?$

Those opinions are expressed through potential tables

chocolate	0.1
peanut butter	5
ice cream	1
tuna	6
…	



	chocolate	peanut butter	ice cream	tuna	…
chocolate	2	9	7	0.1	
peanut butter	4	2	3	0.2	
ice cream	7	3	2	0.1	
tuna	0.1	0.2	0.1	2	
…					

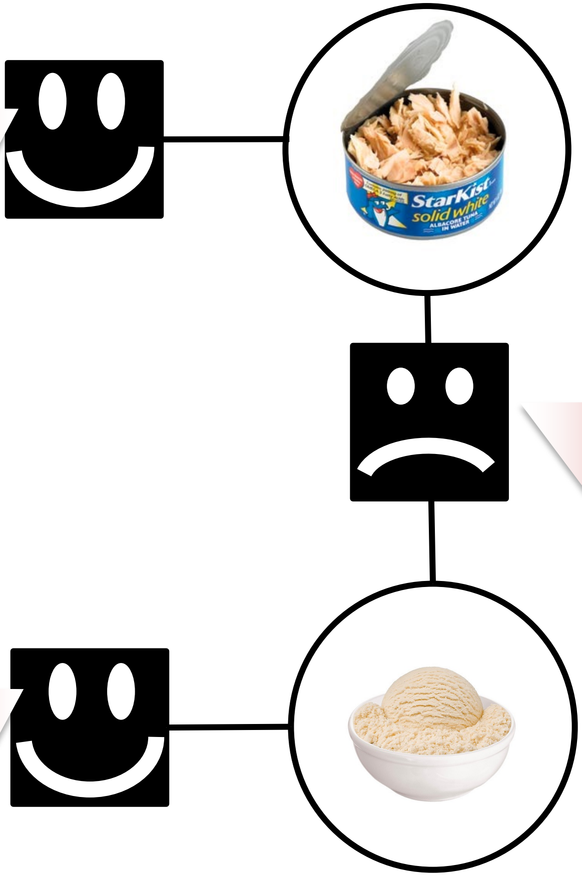
chocolate	4
peanut butter	8
ice cream	7
tuna	3
…	

Mathematical Modeling

Factor Graphs

$$P(\text{tuna, ice cream}) = \frac{1}{Z}(6 * 7 * 0.1)$$

chocolate	0.1
peanut butter	5
ice cream	1
tuna	6
…	



Uh-oh! The probabilities of the various assignments sum up to Z > 1.

So divide them all by Z.

	cho	peanu	Ice	tuna	⋮
chocolate	2	9	7	0.1	
peanut butter	4	2	3	0.2	
ice cream	7	3	2	0.1	
tuna	0.1	0.2	0.1	2	
…					

chocolate	4
peanut butter	8
ice cream	7
tuna	3
…	

The combined potential tables of all factors defines the probability of an assignment

How General Are Factor Graphs?

- Factor graphs can be used to describe
  - Markov Random Fields (undirected graphical models)
  - Conditional Random Fields
  - Bayesian Networks (directed graphical models)

Factor Graph Notation

- Variables:

$$\mathcal{X} = \{X_1, \ldots, X_i, \ldots, X_n\}$$

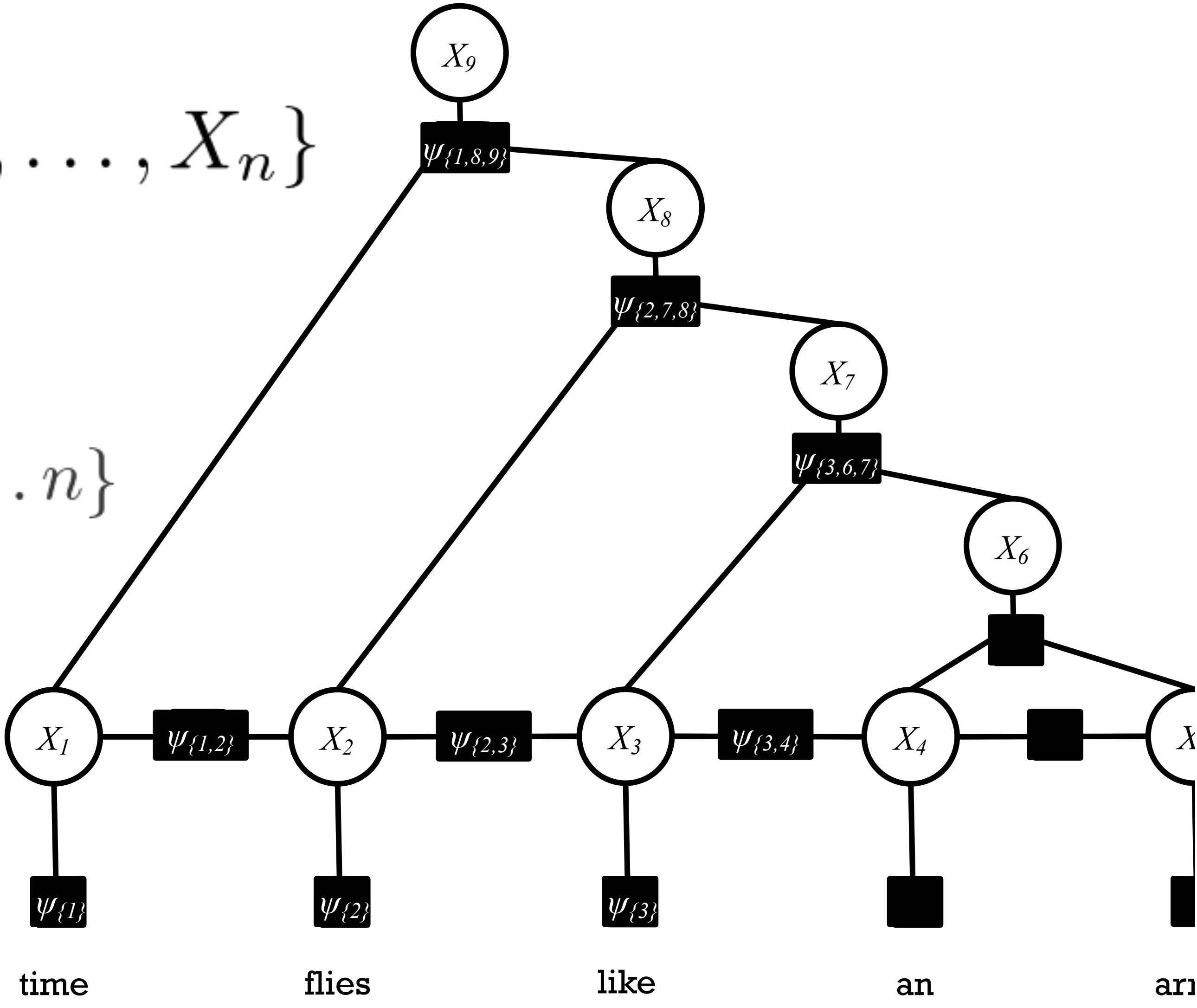
- Factors:

$$\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$$

$$\text{where } \alpha, \beta, \gamma, \ldots \subseteq \{1, \ldots n\}$$

Joint Distribution

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{x}_\alpha)$$



Factors are Tensors

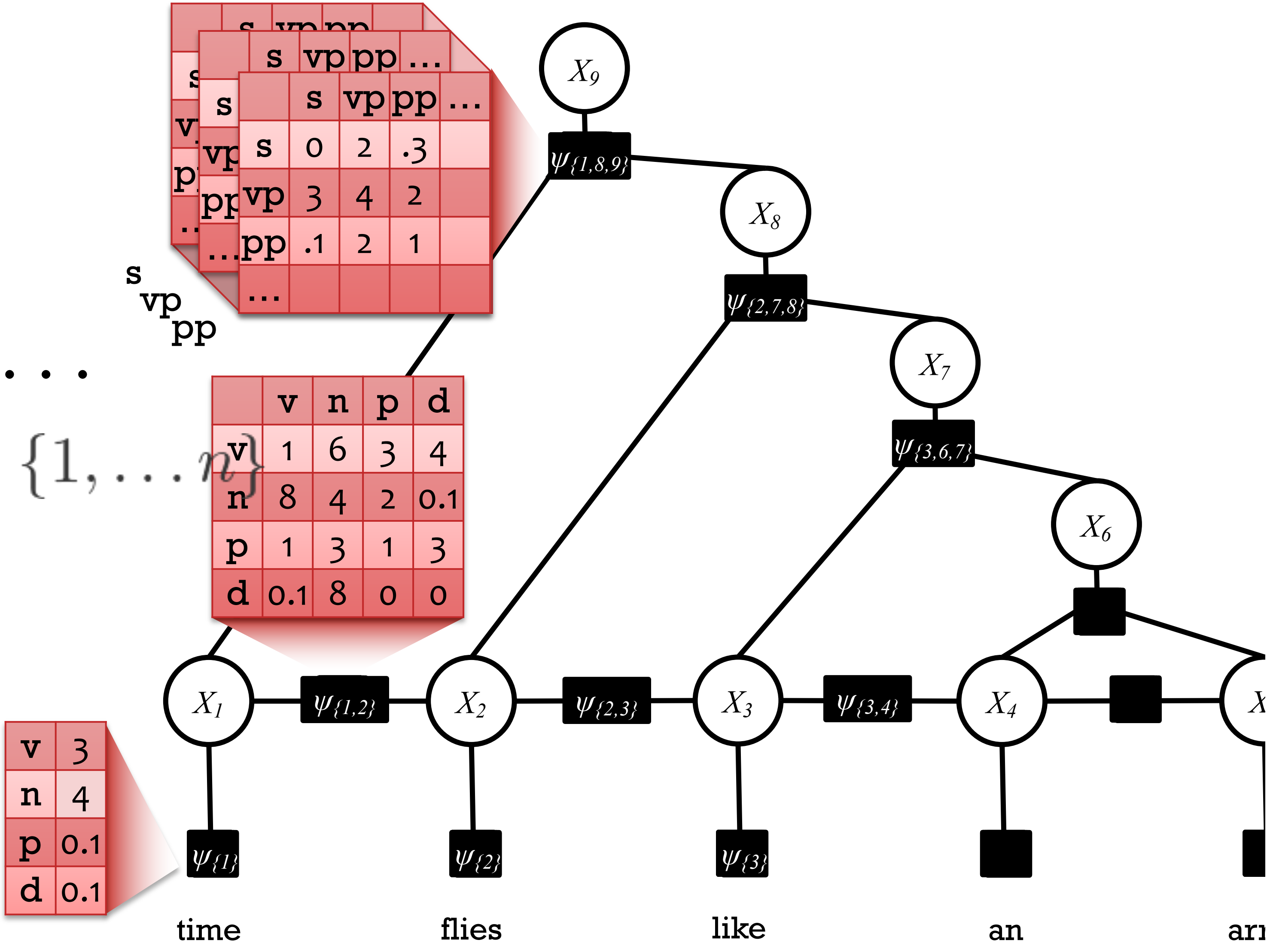
- *Def*: the **arity** of a factor is the number of neighbors (variables) it has

- Factors:

$$\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$$

where $\alpha, \beta, \gamma, \ldots \subseteq \{1, \ldots n\}$

- *Def*: a **unary factor** touches one variables
- *Def*: a **binary factor** touches two variables
- *Def*: a **ternary factor** touches three variables



	s	vp	pp	…
s	0	2	.3	
vp	3	4	2	
pp	.1	2	1	
…				

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

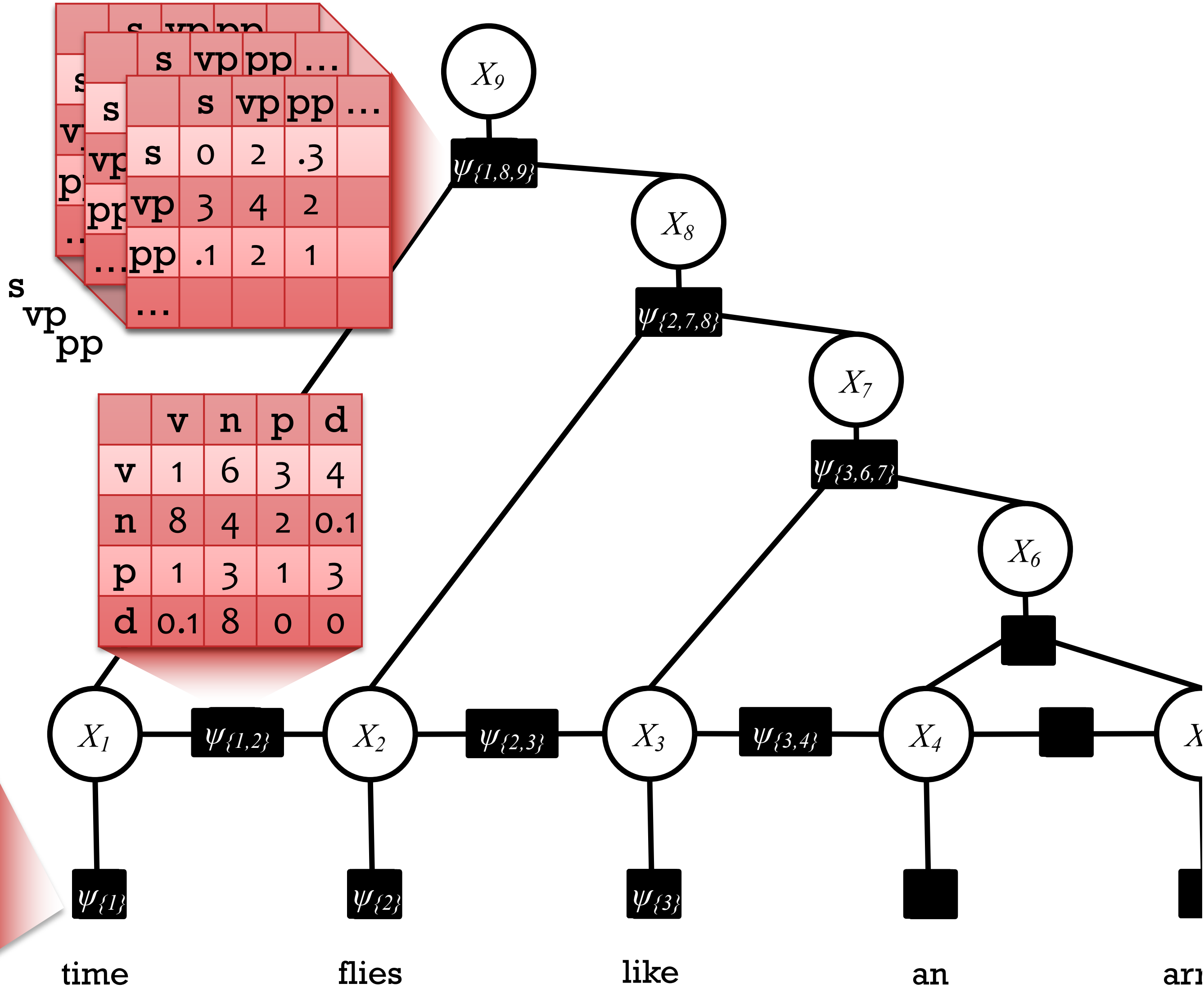
v	3
n	4
p	0.1
d	0.1

Factors are Tensors

- Factors must contain **non-negative** values -- this ensures we have a valid probability distribution
- We also sometimes refer to factors as **potential functions** or **potentials** (like UGMs)

Joint Distribution

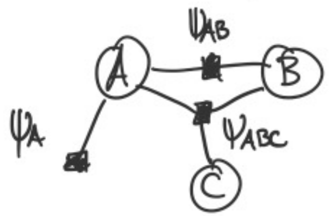
$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_{\alpha}(\boldsymbol{x}_{\alpha})$$



	s	vp	pp	...
s	0	2	.3	
vp	3	4	2	
pp	.1	2	1	
...				

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

Ex: Factor Graph over Binary Variables



a	$\psi_A(a)$
0	2
1	7

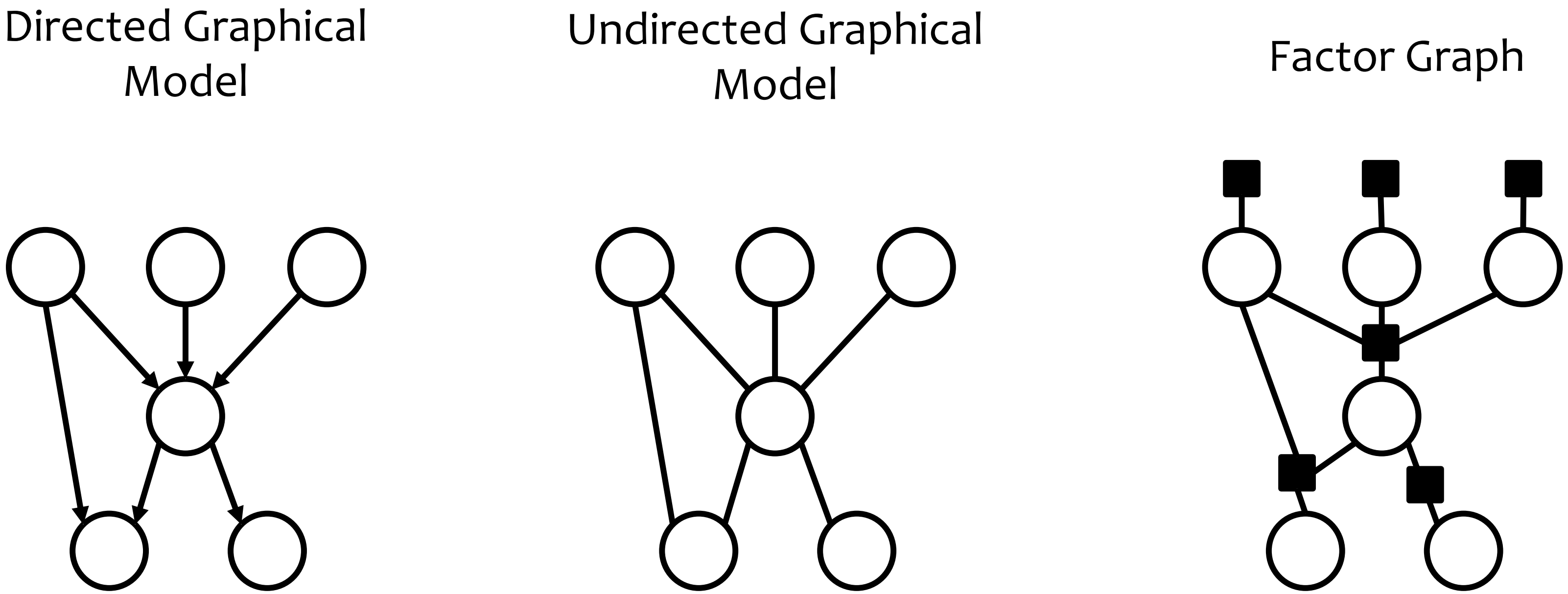
a	b	$\psi_{AB}(a,b)$
0	0	4
0	1	3
1	0	6
1	1	2

a	b	c	$\psi_{ABC}(a,b,c)$
0	0	0	6
0	0	1	2
0	1	0	1
	...		...
1	1	1	5

$$P(A=a, B=b, C=c) = p(a,b,c) = \frac{1}{Z} \underbrace{\psi_A(a)\,\psi_{AB}(a,b)\,\psi_{ABC}(a,b,c)}_{s(a,b,c)} \Rightarrow Z = \sum_a \sum_b \sum_c s(a,b,c)$$

a	b	c	ψ_A	ψ_{AB}	ψ_{ABC}	$s(\cdot)$	$p(\cdot)$
0	0	0	2	4	6	48	48/Z
0	0	1	2	4	2	16	16/Z
0	1	0	2	3	1	6	6/Z
	...			...		⋮	
1	1	1	7	2	5	+ 70	70/Z
						Z	

Locally Normalized vs. Globally Normalized



$$P(X_1, \ldots, X_T) = \prod_{t=1}^{T} P(X_t \mid \text{parents}(X_t))$$

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$$