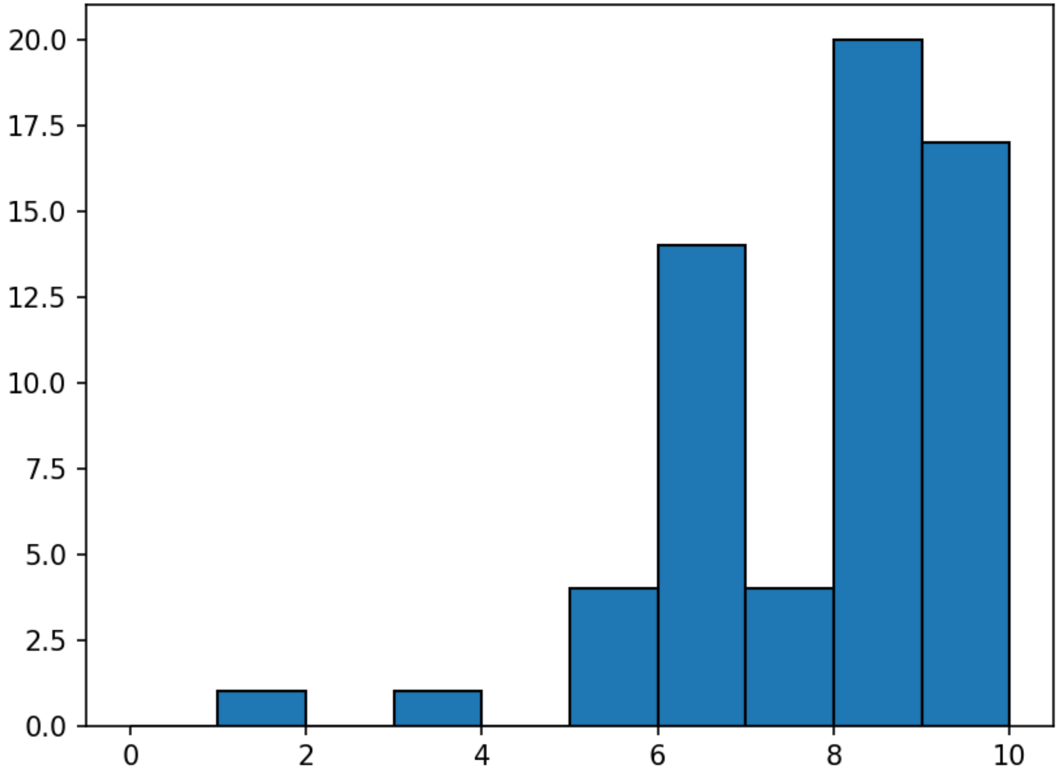
Deep Learning
lecture 5
Normalizing Flows

Yi Wu, IIIS

Spring 2024

Mar-29

Coding Project 1



Logistics

- Coding Project 2 due Tomorrow!
  - You can modify any code (**except evaluation script**) you like for training.
- Always keep track of your training stats!
  - E.g., loss, accuracy, validation/test error
  - Store your model stats as well
    - Check points, hyper-params, gradient norm, parameter norm, samples
    - Use a spreadsheet to manage your experiments

- 2 homework assignments will be released today
- Get prepared for Coding Project 3
  - You will need to implement generative models ☺

Today’s Lecture

- Sampling Methods
  - Draw samples from a distribution

- Normalizing Flow Model
  - A model class that we can easily draw samples from

- Auto-Regressive Flow

Energy-Based Model (Recap)

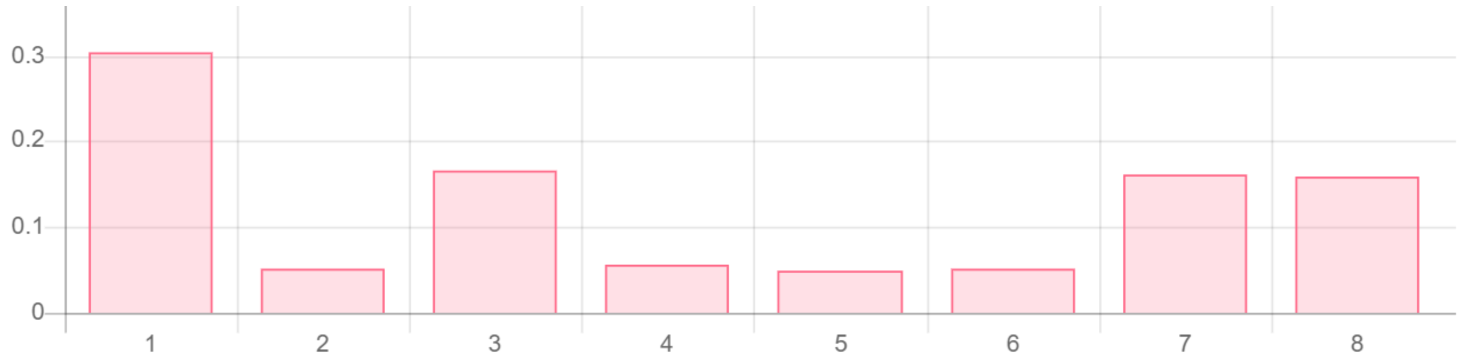
- A particular class of density function

$$P(x) = \frac{1}{Z}\exp(-E(x;\theta))$$

- Maximum Likelihood Training
  - $L(\theta) = \log P(x) = -E(x;\theta) - \log Z(\theta)$
  - Z: partition function
- Contrastive Divergence Algorithm
  - $\nabla_\theta L(\theta) \approx \nabla_\theta\big(-E(x_{train};\theta) + E(x_{sample};\theta)\big)$
- How to sample from an energy-based model?
  - Or in general: sample from $p(x)$

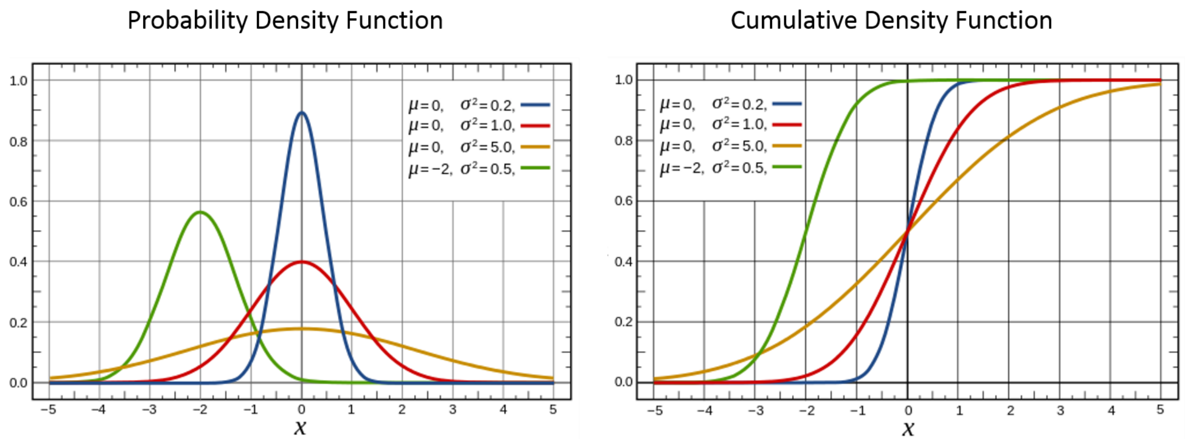
Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let's start from an easy example
  - Categorical distribution?
  - Solution: uniform sampling, find the category with cumulative density
    - *The mapping from CDF to value is called Inverse distribution function (quantile function)*



Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let's start from an easy example
  - Categorical distribution
  - Gaussian distribution?
    - No closed-form CDF!
    - Central-limit theorem
      - Sample $X_i \sim Beroulli(0.5)$
      - $E[X_i] = 0.5; Var[X_i] = 0.5^2$
      - $S_N = \frac{1}{N}\sum_{i=1}^{N} X_i$
      - As $N \to \infty$, $\sqrt{N}(S_N - 0.5) \sim N(0, 0.5^2)$



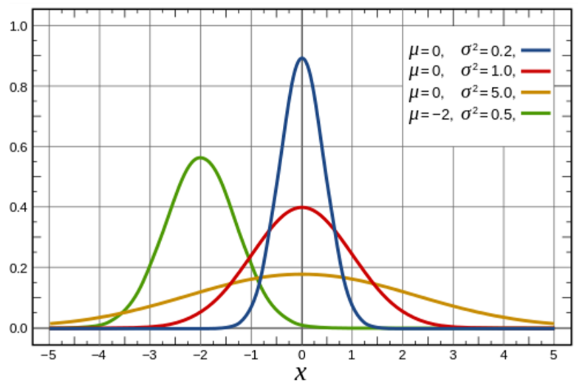
Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let’s start from an easy example
  - Categorical distribution
  - Gaussian distribution?
    - No closed-form CDF!
    - Central-limit theorem
    - Box–Muller transform
      - Most practical method (FYI)
      - Uniform → Normal
      - Polar form transformation

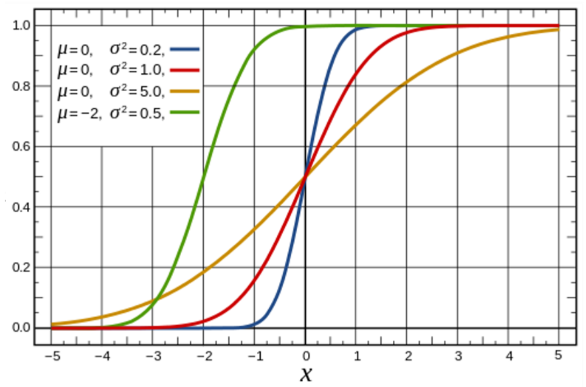
```
def box_muller():
    # Avoid getting u == 0.0
    u1, u2 = 0.0, 0.0
    while u1 < epsilon or u2 < epsilon:
        u1 = random.random()
        u2 = random.random()

    n1 = math.sqrt(-2 * math.log(u1)) * math.cos(2 * math.pi * u2)
    n2 = math.sqrt(-2 * math.log(u1)) * math.sin(2 * math.pi * u2)
    return n1, n2
```

Probability Density Function



Cumulative Density Function



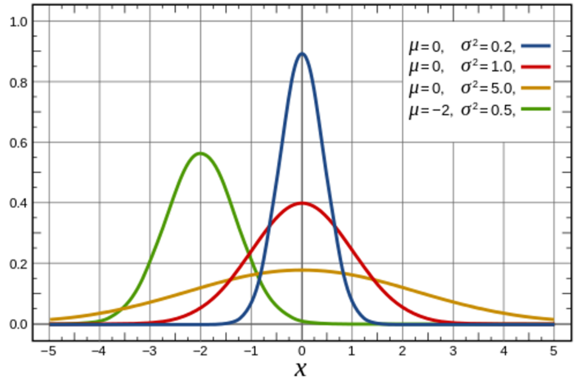
Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let’s start from an easy example
  - Categorical distribution
  - Gaussian distribution?
    - No closed-form CDF!
    - Central-limit theorem
    - Box–Muller transform
    - General case $x \sim N(\mu, \sigma^2)$
    - High-dimensional case $x \sim N(\mu, \Sigma)$
      - $z \sim N(0, I)$
      - $x = \Sigma z + \mu$

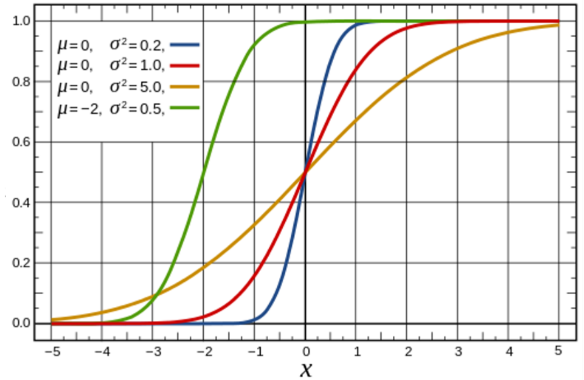
```python
def box_muller():
    # Avoid getting u == 0.0
    u1, u2 = 0.0, 0.0
    while u1 < epsilon or u2 < epsilon:
        u1 = random.random()
        u2 = random.random()

    n1 = math.sqrt(-2 * math.log(u1)) * math.cos(2 * math.pi * u2)
    n2 = math.sqrt(-2 * math.log(u1)) * math.sin(2 * math.pi * u2)
    return n1, n2
```

Probability Density Function

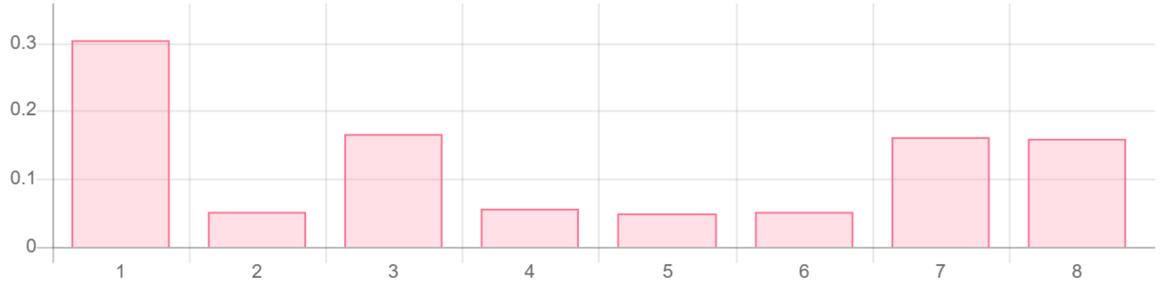


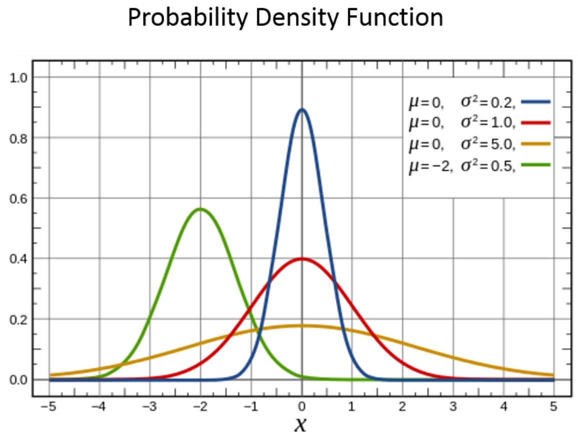
Cumulative Density Function

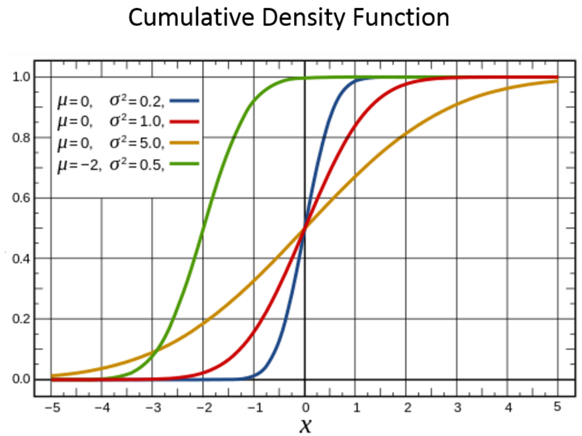


Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let's start from an easy example
  - Categorical distribution
  - Gaussian distribution
    - Idea: (1) use "easy" distributions to draw sample & (2) apply mathematical transform
  - More complex distribution $p(x)$?

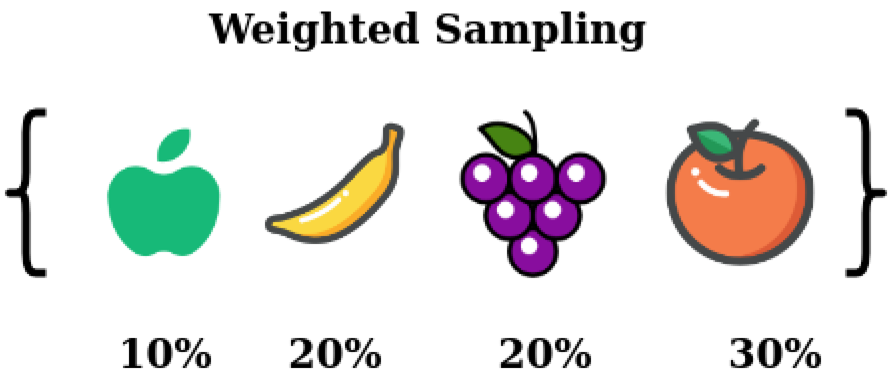


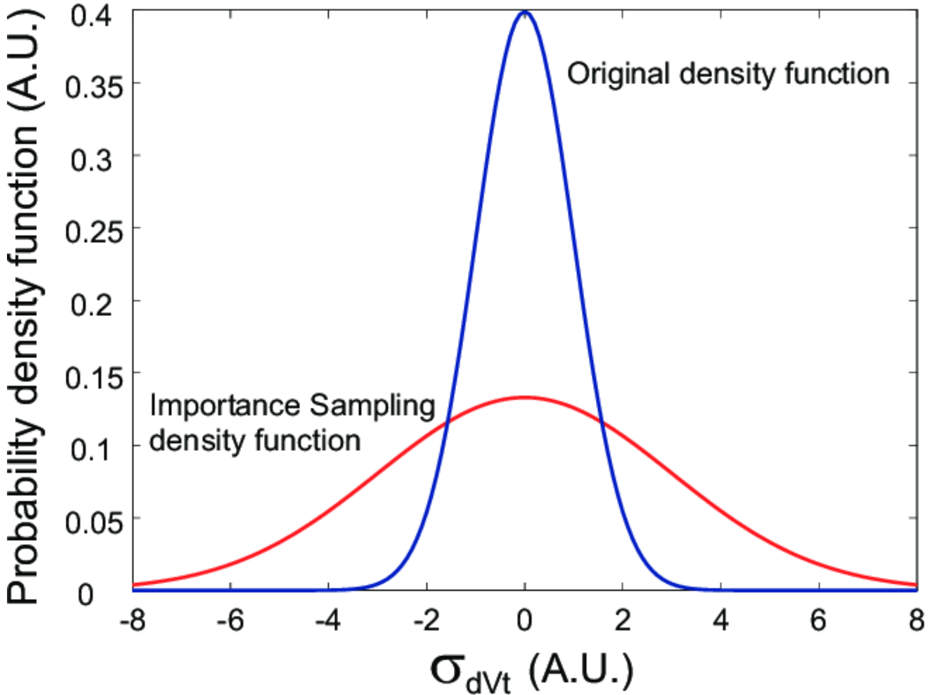




Sampling Methods

- Goal: sampling from $p(x)$
  - No CDF or nice mathematical property available
- Idea: weighted samples
  - sample from "easy" distribution $q(x)$ (e.g. uniform)
  - Use $p(x)/q(x)$ as the weight for the sample
- Importance Sampling
  - $q(x)$ proposal distribution
  - $\frac{p(x)}{q(x)}$ importance weight
  - $\mathrm{E}_{x\sim p}[f(x)] = \mathrm{E}_{x\sim q}\left[\frac{p(x)}{q(x)} f(x)\right]$
  - ***Quiz: what if partition function Z is unknown?***

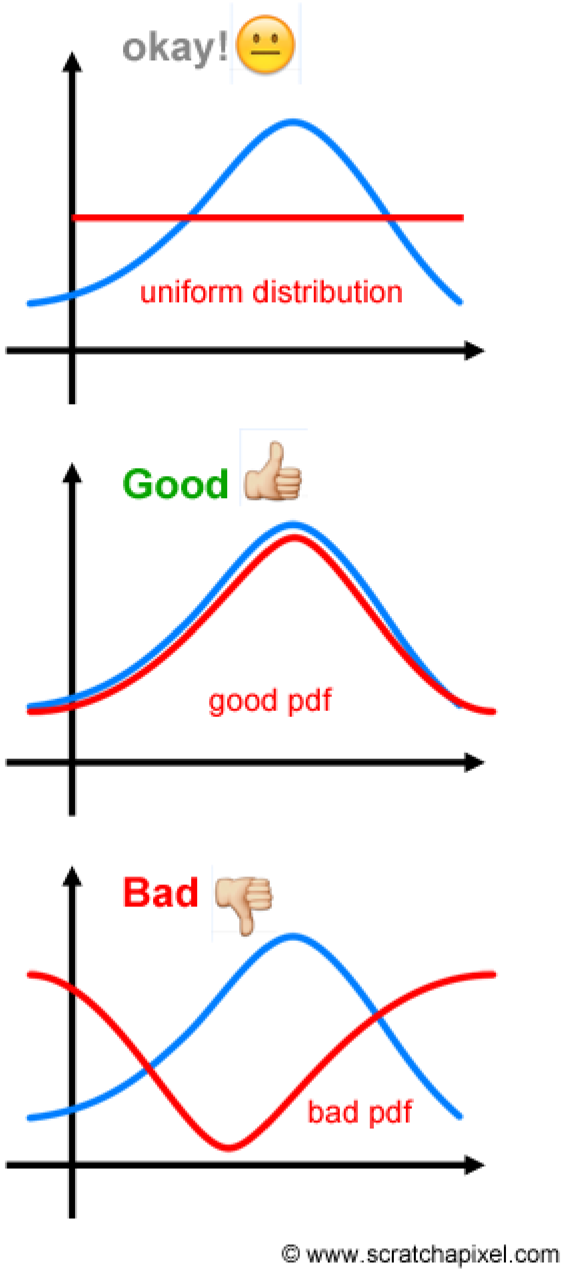




Sampling Methods

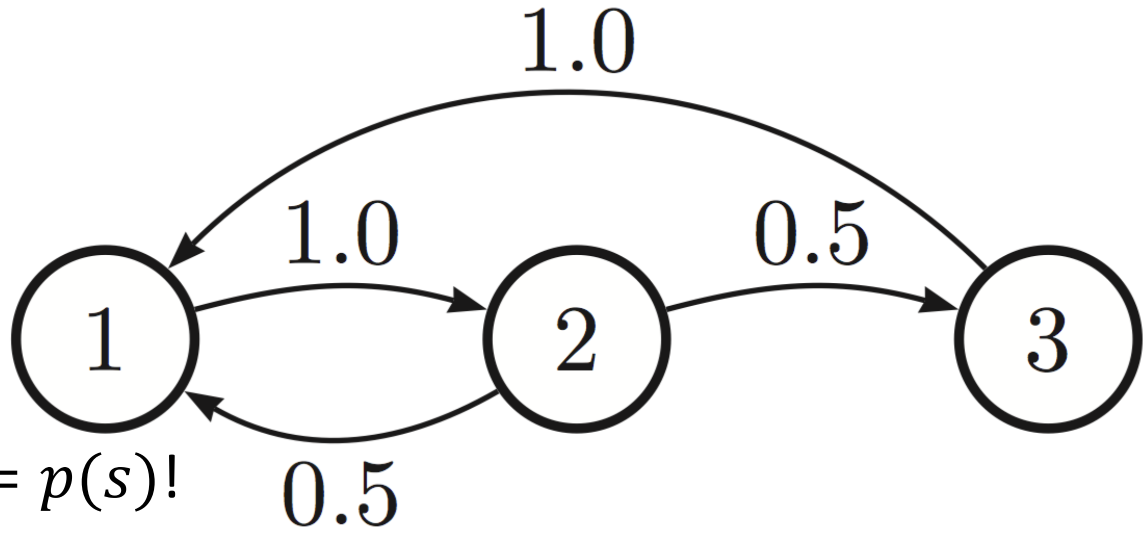
- Goal: sampling from $p(x)$
  - No CDF or nice mathematical property available
- Idea: weighted samples
  - sample from "easy" distribution $q(x)$ (e.g., uniform)
  - Use $p(x)/q(x)$ as the weight
- Importance Sampling
  - $q(x)$ proposal distribution
  - **How to choose $q(x)$???**
  - $q(x)$ needs to similar to $p(x)$
    - Your homework ☺

What if we don't have a good proposal?



Markov Chain Monte-Carlo

- Markov Chain
  - A state space S, a transition probability $P(s_j|s_i) = T_{ij}$
  - T is the transition matrix
  - We also use $T(s_i \to s_j)$ to denote T_{ij}
- A Markov Chain has a stationary distribution with a proper T
  - Current distribution over states π_t
  - Single step transition $\pi_{t+1} = T\pi_t$
  - Stationary distribution $\pi = T^{\infty}\pi_0$
- Sampling is easy!
- Goal: construct a Markov Chain
  - With a desired stationary distribution $\pi = p(s)$!



Markov Chain Monte-Carlo

- How to ensure π is a stationary distribution of a Markov Chain?
  - Detailed Balance (sufficient condition)

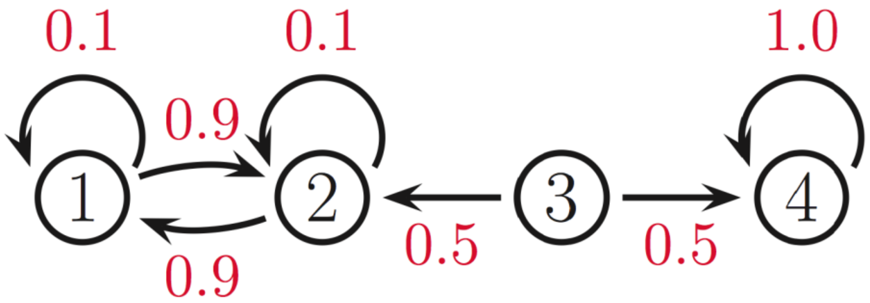
$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$

Markov Chain Monte-Carlo

- How to ensure π is a stationary distribution of a Markov Chain?
  - Detailed Balance (sufficient condition)
$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$
  - Design a Markov chain satisfying detailed balance for desired density $p(s)$!

Markov Chain Monte-Carlo

- How to ensure π is a stationary distribution of a Markov Chain?
  - Detailed Balance (sufficient condition)
$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$
  - Design a Markov chain satisfying detailed balance for desired density $p(s)$!
- How to ensure a unique stationary distribution exist?
  - The Markov chain is ergodic（遍历性）！
$$\min_{z} \min_{z':\pi(z')>0} \frac{T(z \to z')}{\pi(z')} = \delta > 0$$
Intuitively: you can visit any desired state with positive probability from any state
  - Examples:



$$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Metropolis Hastings Algorithm

- Construct a valid Markov Chain for $p(s)$
  - Detailed balance: $p(s)T(s \to s') = p(s')T(s' \to s)$
  - Ergodicity
- Metropolis Hastings Algorithm
  - A proposal distribution $q(s'|s)$ to produce next state s' based on s
  - Draw $s' \sim q(s'|s)$
  - $\alpha = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right)$ ($q(s \to s')$ to denotes $q(s'|s)$ for simplicity)
  - Transition to s' (accept) with probability α (acceptance ratio);
  - O.w., stays at s (reject)
- MH constructs a valid Markov chain with a proper proposal q!
  - Homework ☺

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow q(s \to s')$
  - Transition to s' with probability $\alpha(s \to s')$;
  - O.w., stays at s
  - Repeat

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \mathrm{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow s + \mathrm{noise}$
  - Transition to s' with probability $\alpha(s \to s')$;
  - O.w., stays at s
  - Repeat

Metropolis Hastings Algorithm: Example

- Choice of $q(s \rightarrow s')$
  - Random proposal $q(s \rightarrow s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \rightarrow s'$
  - $\alpha(s \rightarrow s') = \min\left(1, \frac{p(s')q(s' \rightarrow s)}{p(s)q(s \rightarrow s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow s + \text{noise}$
  - Transition to s' with probability $\min\left(1, \frac{p(s')}{p(s)}\right)$;
  - O.w., stays at s
  - Repeat

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow s^t + \text{noise}$
  - If $E(s') < E(s^t)$; then accept $s^{t+1} \leftarrow s'$
  - Else accept s' with probability $\exp\left(E(s^t) - E(s')\right)$
  - Repeat

Presented last week ☺

Metropolis Hastings Algorithm

- The simplest way to construct a valid Markov chain
  - Flexible, simple and general
  - **Quiz: proposal $\boldsymbol{q}$ in MH v.s. Importance Sampling**
    - A: $q(s'|s)$ v.s. $q(s)$; in MH, q generates local samples; in IS, q outputs “blind” guesses

- Issue
  - Curse of dimensionality: samples a completely new state
  - Acceptance ratio: what if acceptance rate is low?

Metropolis Hastings Algorithm

- The simplest way to construct a valid Markov chain
  - Flexible, simple and general
  - **Quiz: proposal q in MH v.s. Importance Sampling**
    - A: $q(s'|s)$ v.s. $q(s)$; in MH, q generates local samples; in IS, q output "blind" guesses

- Issue
  - Curse of dimensionality: samples a completely new state
  - Acceptance ratio: what if acceptance rate is low?

- Can we design a proposal distribution $q(s \to s')$ such that it always gets accepted?

Gibbs Sampling

- Gibbs sampling
  - $s = (s_0, s_1, \dots, s_N)$, we construct a coordinate-wise $q(s_i \to s_i')$
  - $q(s_i \to s_i') = p(s_i' | s_{j \neq i})$ (conditional distribution)
- Dimensionality
  - Sample a single coordinate per step.
- Gibbs sampling always get accepted!
  - Acceptance ratio is always 1, $\alpha(s_i \to s_i') = 1$ Prove it in your homework ☺
- Assumption
  - An easy to sample conditional distribution
    - Conjugate Prior and Exponential Family (https://en.wikipedia.org/wiki/Conjugate_prior)
  - What if no closed-form posterior?
    - Learn a neural proposal to approximate the true posterior! ☺
      (meta-learning MCMC proposals, Wang, Wu, et al NIPS2018)

Sampling Methods

- Story so far
  - Importance Sampling
    - Simplest solution
  - Metropolis-Hastings algorithm
    - Good local proposal → high acceptance ratio
  - Gibbs sampling
    - Posterior is easy-to-sample
    - The “default” method for machine learning among 2002~2012
- General Issues for MCMC
  - Slow convergence due to sampling (recap: SGD v.s. GD)
  - Can we use gradient information for MCMC?
    - Energy function is differentiable!

Stochastic Gradient MCMC

- MCMC with Langevin dynamics
  - "Bayesian learning via stochastic gradient langevin dynamics"
    - ICML 2011, Max Welling& Yee Whye The (ICML 2021 test-of-time award)
  - Given N data $X_1, \ldots, X_N$, define $p(\theta \rightarrow \theta')$ by

$$\theta' \leftarrow \theta + \frac{\epsilon_t}{2}\left(\log p(\theta) + \sum_i \nabla_\theta \log P(x_i|\theta)\right) + N(0, \epsilon_t I)$$

  - Condition for a valid Markov Chain
    - $\sum_t \epsilon_t = \infty$ and $\sum_t \epsilon_t^2 < \infty$
    - Interpretation
      - (stochastic) gradient descent first (∇_θ is large); MCMC around local minimum ($\nabla_\theta \approx 0$)
    - No need of MH acceptance rule
- Additional Reading:
  - Hamiltonian Monte Carlo (SGD with momentum): https://arxiv.org/pdf/1701.02434.pdf
  - https://arogozhnikov.github.io/2016/12/19/markov_chain_monte_carlo.html

Non-Sampling Methods

- Approximate Bayesian Inference
  - Mean-field methods
  - Message Passing
  - Variational Inference (next lecture ☺)

- Scoring Matching
  - Match the first order component of model and data (lecture 9)

- **Design a model from which we can easily draw sample!**

Energy-Based Models (Recap)

- Pros: powerful & flexible
  - An arbitrarily complex density function $p(x) = \frac{1}{Z}\exp\big(-E(x)\big)$
- Cons: hard to sample/train
  - Hard to sample
    - MCMC sampling
  - Partition function
    - No closed-form calculation for likelihood
    - Cannot optimize MLE loss exactly
    - MCMC sampling
- Easy to sample & tractable likelihood?

Intuition

- Goal: design $p(x)$ s.t.
  - Easy to sample
  - Tractable likelihood (density function)

Intuition

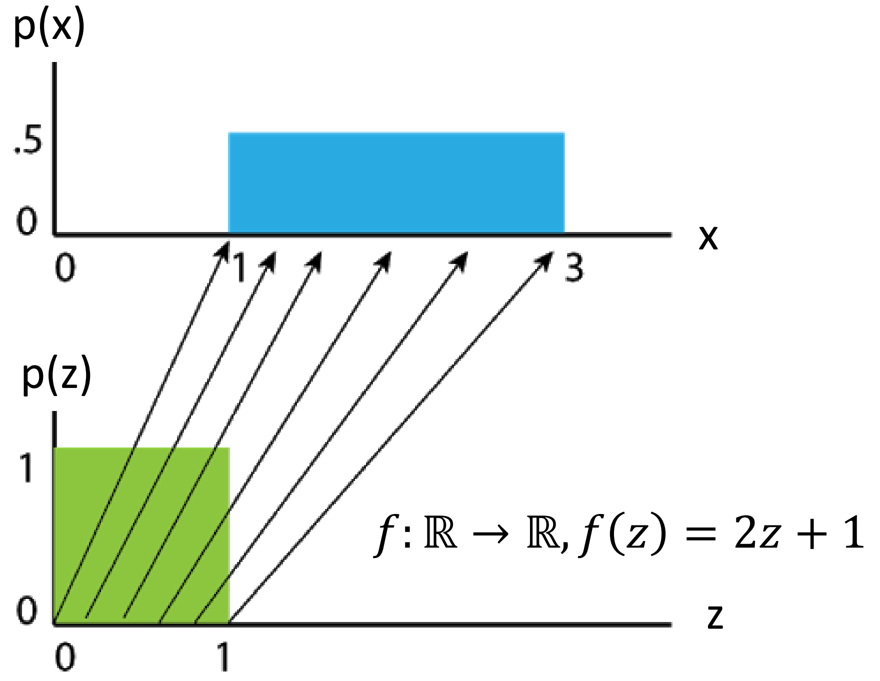
- Goal: design $p(x)$ s.t.
  - Easy to sample
  - Tractable likelihood (density function)
- Easy to sample?
  - Assume a continuous variable z
  - Uniform & Gaussian! $z \sim \mathrm{Unif}(0,1)$ or $z \sim N(0,1)$
    - Also closed-form density function
  - $x = f(z)$, x is also easy to sample

Intuition

- Goal: design $p(x)$ s.t.
  - Easy to sample
  - Tractable likelihood (density function)
- Easy to sample?
  - Assume a continuous variable z
  - Uniform & Gaussian! $z \sim \mathrm{Unif}(0,1)$ or $z \sim N(0,1)$
    - Also closed-form density function
  - $x = f(z)$, x is also easy to sample
- Tractable Density?

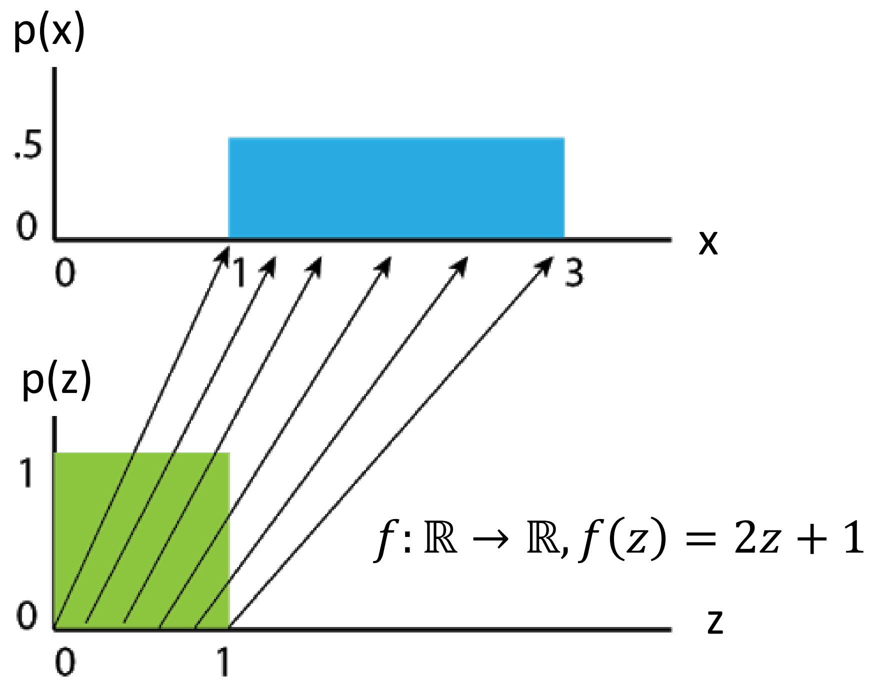
1-D Example

- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an “easy” distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z \sim \text{unif}\,(0,1)$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = ?$



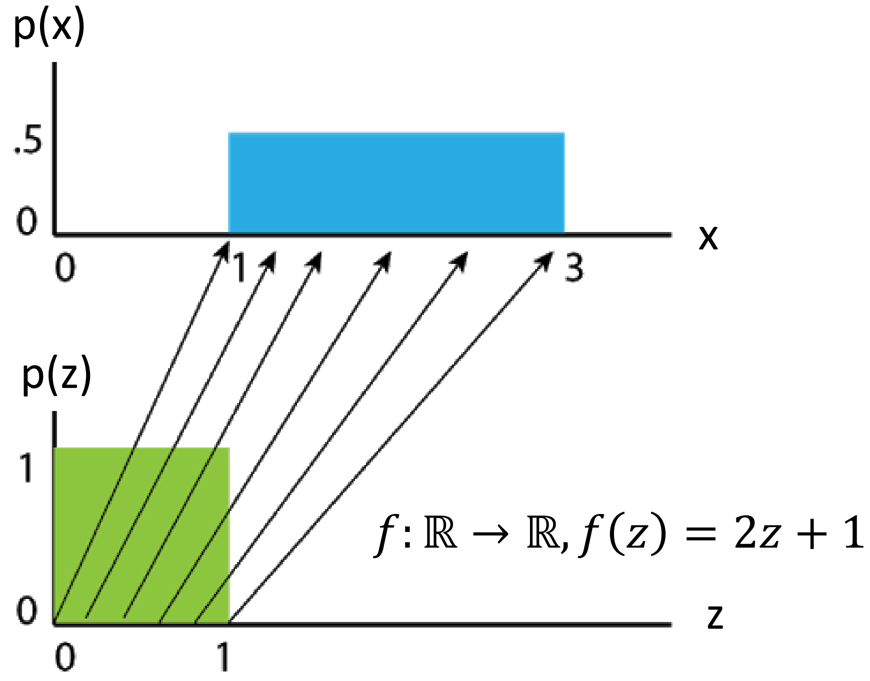
1-D Example

- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z \sim \text{unif}\,(0,1)$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = \frac{1}{2}$
    - $x = a \cdot z + b$, then $p(x) = 1/|a|$ (for $a \neq 0$)
  - $x = f(z), p(x) = ?$
    - Assume $f(z)$ is a bijection



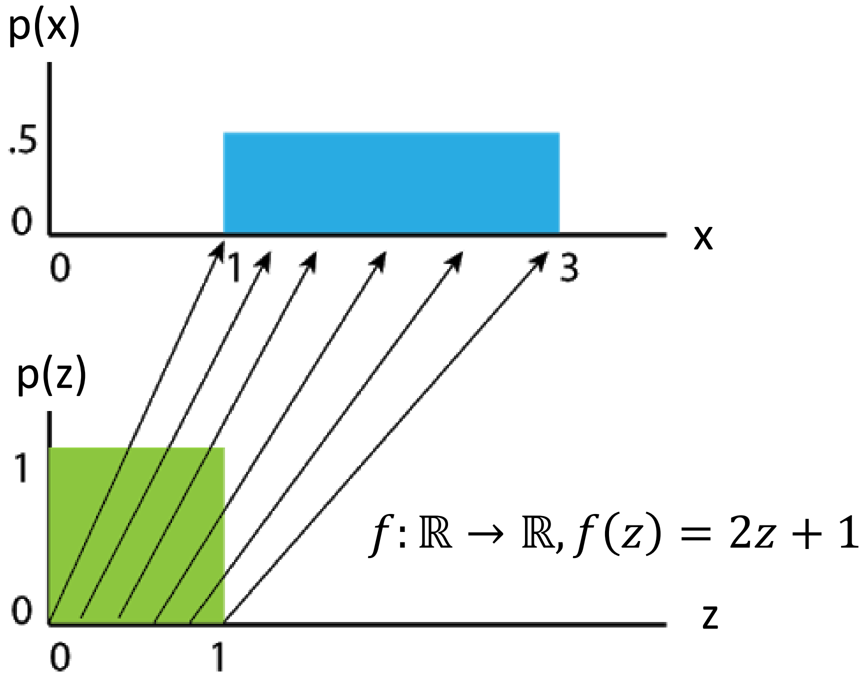
1-D Example

- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z \sim \text{unif}\ (0,1)$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = \frac{1}{2}$
    - $x = a \cdot z + b$, then $p(x) = 1/|a|$ (for $a \neq 0$)
  - $x = f(z)$, then $p(x) = p(z) \left|\frac{dz}{dx}\right|$
    - Assume $f(z)$ is a bijection
    - $p(x)dx = p(z)dz$



1-D Example

- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z \sim \text{unif}\,(0{,}1)$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = \frac{1}{2}$
    - $x = a \cdot z + b$, then $p(x) = 1/|a|$ (for $a \neq 0$)
  - $x = f(z)$, then $p(x) = p(z)\left|\frac{dz}{dx}\right| = |f'(z)|^{-1}p(z)$
    - Assume $f(z)$ is a bijection
    - $p(x)dx = p(z)dz$

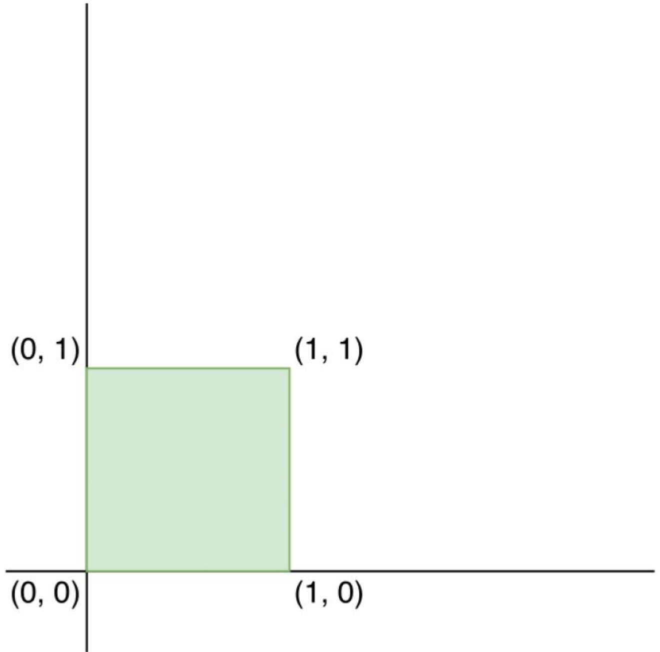


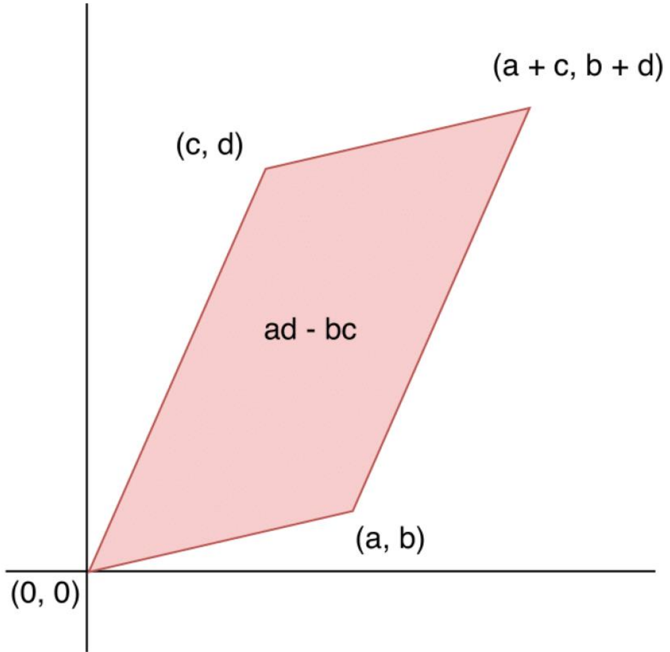
2-D Example

- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z = [z_1, z_2] \sim \text{unif}\,([0,1] \times [0,1])$
  - Density $p(z) = 1$
  - $x = Az$, then $p(x) = ?$
    - $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

2-D Example

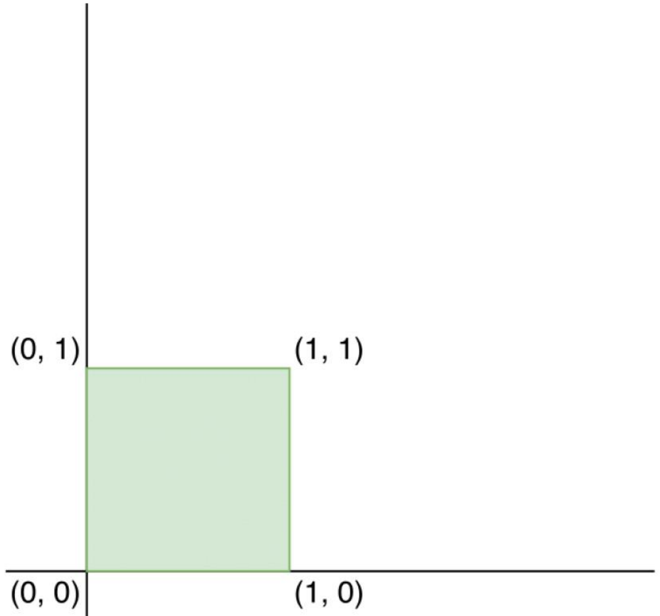
- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z = [z_1, z_2] \sim \text{unif}\left([0,1] \times [0,1]\right)$
  - Density $p(z) = 1$
  - $x = Az$, then $p(x) = ?$
    - $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$
    - z is mapped to a parallelogram

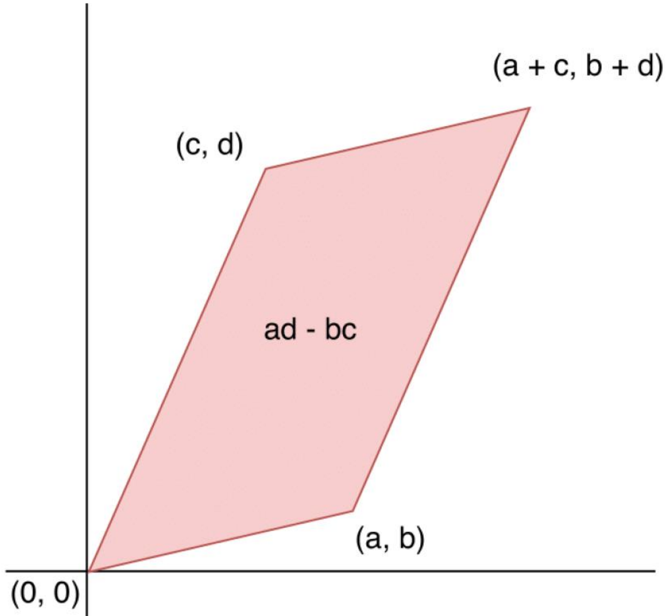




2-D Example

- Goal: design $f(z;\theta)$ s.t.
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood
- Uniform: $z = [z_1, z_2] \sim \mathrm{unif}\,([0,1] \times [0,1])$
  - Density $p(z) = 1$
  - $x = Az$, then $p(x) = 1/S$
    - $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$
    - z is mapped to a parallelogram
    - $S = |ad - bc|$, the area





2-D Geometry

- The area of the parallelogram is equivalent to the determinant of A

$$\det A = \det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

- For any linear transformation $x = Az + b$
  - The following holds (for space of any dimensions)

$$p(x) = |\det A|^{-1} \cdot p(z)$$

- More general case: the change of variable

Change of Variable

- Suppose $x = f(z)$ w.r.t. general non-linear $f(\cdot)$
  - the linearized change in volume is determined by the Jacobian of $f(\cdot)$

$$\frac{\partial f(z)}{\partial z} = \begin{bmatrix} \frac{\partial f_1(z)}{\partial z_1} & \cdots & \frac{\partial f_1(z)}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d(z)}{\partial z_1} & \cdots & \frac{\partial f_d(z)}{\partial z_d} \end{bmatrix}$$

- Given a bijection $f(z)$: $\mathbb{R}^d \rightarrow \mathbb{R}^d$
  - $z = f^{-1}(x)$

$$p(x) = p\left(f^{-1}(x)\right)\left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right| = p(z)\left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$$

Change of Variable

- Suppose $x = f(z)$ w.r.t. general non-linear $f(\cdot)$
  - the linearized change in volume is determined by the Jacobian of $f(\cdot)$

$$\frac{\partial f(z)}{\partial z} = \begin{bmatrix} \frac{\partial f_1(z)}{\partial z_1} & \cdots & \frac{\partial f_1(z)}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d(z)}{\partial z_1} & \cdots & \frac{\partial f_d(z)}{\partial z_d} \end{bmatrix}$$

- Given a bijection $f(z)\colon \mathbb{R}^d \rightarrow \mathbb{R}^d$
  - $z = f^{-1}(x)$

$$p(x) = p\left(f^{-1}(x)\right)\left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right| = p(z)\left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$$

  - Since $\frac{\partial f^{-1}}{\partial x} = \left(\frac{\partial f}{\partial z}\right)^{-1}$ (Jacobian of invertible function)

$$p(x) = p(z)\left|\det\left(\frac{\partial f(z)}{\partial z}\right)^{-1}\right| = p(z)\left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|^{-1}$$

Change of Variable

- Suppose $x = f(z)$ w.r.t. general non-linear $f(\cdot)$
  - the linearized change in volume is determined by the Jacobian of $f(\cdot)$

$$\frac{\partial f(z)}{\partial z} = \begin{bmatrix} \frac{\partial f_1(z)}{\partial z_1} & \cdots & \frac{\partial f_1(z)}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d(z)}{\partial z_1} & \cdots & \frac{\partial f_d(z)}{\partial z_d} \end{bmatrix}$$

- Given a bijection $f(z): \mathbb{R}^d \rightarrow \mathbb{R}^d$
  - $z = f^{-1}(x)$

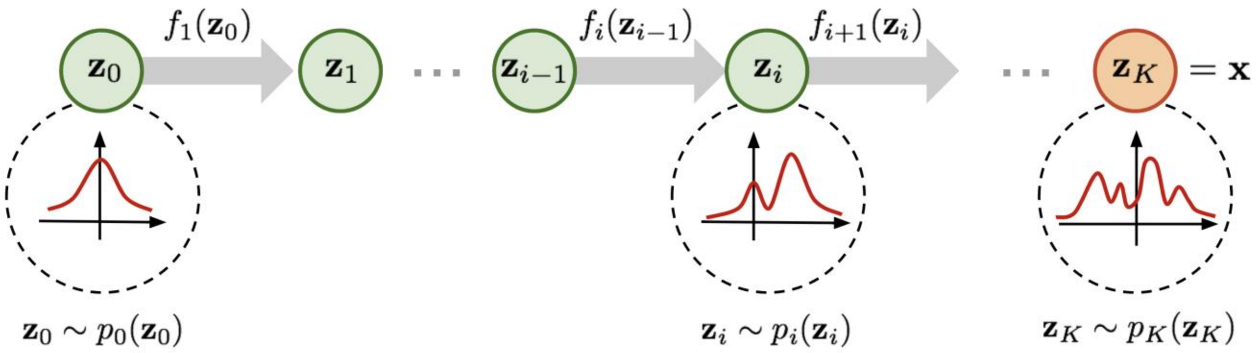
$$p(x) = p\left(f^{-1}(x)\right) \left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right| = p(z) \left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$$

  - Since $\frac{\partial f^{-1}}{\partial x} = \left(\frac{\partial f}{\partial z}\right)^{-1}$ (Jacobian of invertible function)

$$p(x) = p(z) \left|\det\left(\frac{\partial f(z)}{\partial z}\right)^{-1}\right| = p(z) \left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|^{-1}$$

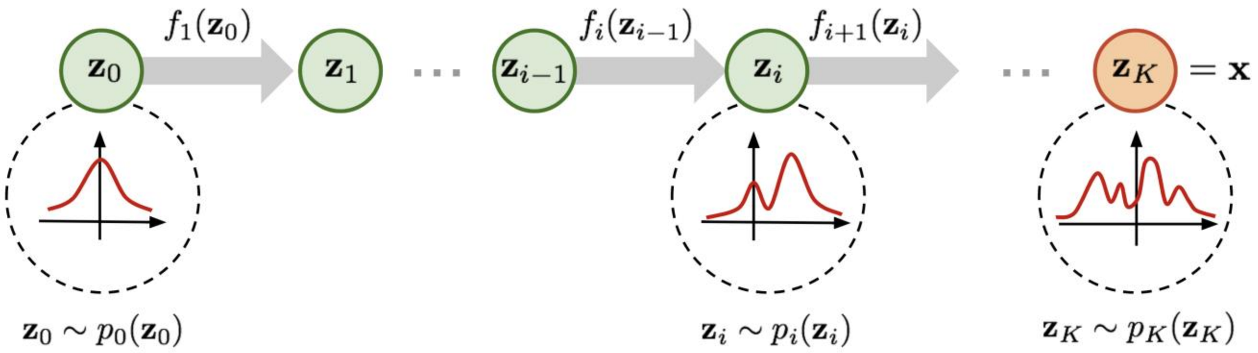
Normalizing Flow

- Idea
  - Sample z_0 from an "easy" distribution, i.e., Standard Gaussian
  - Apply K bijections $z_i = f_i(z_{i-1})$
  - The final sample $x = f_K(z_K)$ has tractable density
- Normalizing Flow
  - $z_0 \sim N(0, I), z_i = f_i(z_{i-1}), x = z_K$ where $x, z_i \in \mathbb{R}^d$ & f_i is invertible
  - Every revertible function produces a normalized density function
  - $p(z_i) = p(z_{i-1}) \left|\det\left(\frac{\partial f_i}{\partial z_{i-1}}\right)\right|^{-1}$



Normalizing Flow

- Idea
  - Sample z_0 from an “easy” distribution, i.e., Standard Gaussian
  - Apply K bijections $z_i = f_i(z_{i-1})$
  - The final sample $x = f_K(z_K)$ has tractable density
- Normalizing Flow
  - $z_0 \sim N(0, I), z_i = f_i(z_{i-1}), x = z_K$ where $x, z_i \in \mathbb{R}^d$ & f_i is invertible
  - Every revertible function produces a normalized density function
  - $p(z_i) = p(z_{i-1}) \left| \det \left(\frac{\partial f_i}{\partial z_{i-1}} \right) \right|^{-1}$



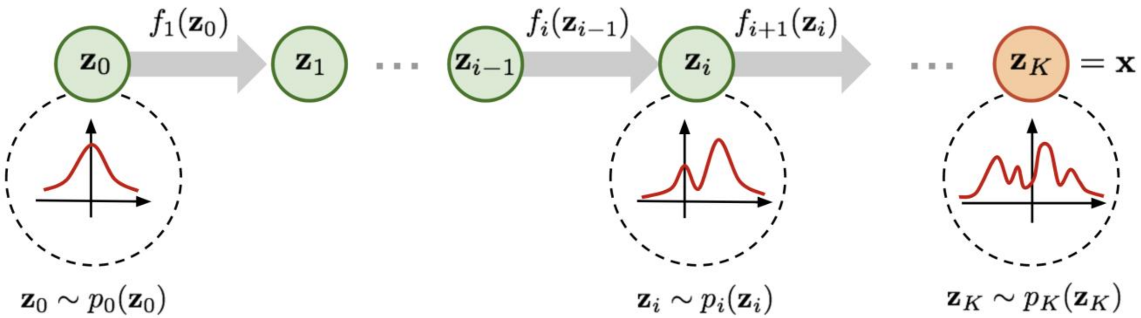
Normalizing Flow

- Generation is trivial
  - Sample z_0, then apply the transformations
- Log-Likelihood

$$\log p(x) = \log p(z_{K-1}) - \log \left| \det \left(\frac{\partial f_K}{\partial z_{K-1}} \right) \right|$$

$$\cdots$$

$$\log p(x) = \log p(z_0) - \sum_i \log \left| \det \left(\frac{\partial f_i}{\partial z_{i-1}} \right) \right|$$



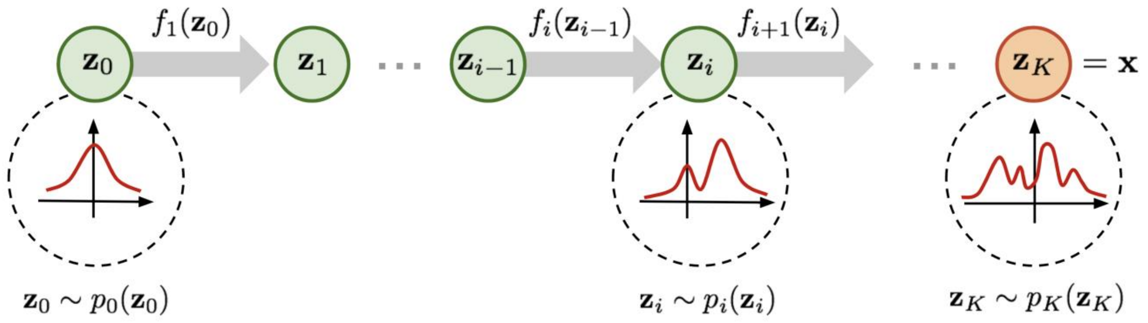
Normalizing Flow

- Generation is trivial
  - Sample z_0, then apply the transformations
- Log-Likelihood

$$\log p(x) = \log p(z_{K-1}) - \log \left| \det \left(\frac{\partial f_K}{\partial z_{K-1}} \right) \right|$$

...

$$\log p(x) = \underbrace{\log p(z_0)}_{\text{Gaussian density}} - \sum_i \log \left| \det \left(\frac{\partial f_i}{\partial z_{i-1}} \right) \right|$$



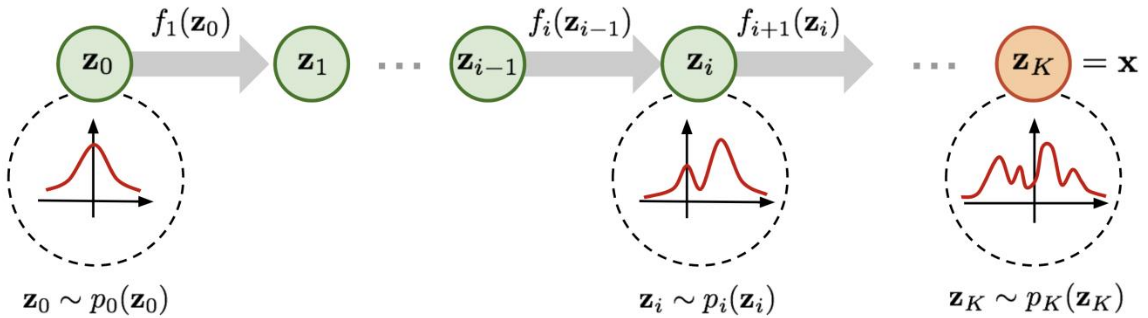
Normalizing Flow

- Generation is trivial
  - Sample z_0, then apply the transformations
- Log-Likelihood

$$\log p(x) = \log p(z_{K-1}) - \log \left|\det\left(\frac{\partial f_K}{\partial z_{K-1}}\right)\right|$$

…

$$\log p(x) = \log p(z_0) - \sum_i \log \left|\det\left(\frac{\partial f_i}{\partial z_{i-1}}\right)\right| \qquad \boldsymbol{O(d^3)}\text{!!!}$$



Normalizing Flow

- Naïve flow model requires extremely expensive computation
  - Determinant of a $d \times d$ matrix
- Idea
  - Design a good bijection $f_i(z)$ such that the determinant is easy to compute
- Technical Keys
  - Bijection
    - Randomly constructed matrices are typically full-rank
  - Structured Jacobian
    - Desired Jacobian structures for fast determinant computation

Planar Flow

- Matrix Determinant Lemma

$$\det(A + uv^T) = (1 + v^T A^{-1} u) \det A$$

- Planar Flow (Rezende & Mohamed, 2016)
  - $f_\theta(z) = z + u \cdot h(w^T z + b)$ (element-wise product)
  - $h(\cdot)$ chosen to be $\tanh(\cdot)$ $(0 < h'(\cdot) \leq 1)$
  - $\theta = [u, w, b], u, w \in \mathbb{R}^d, b \in \mathbb{R}$

$$\det\left(\frac{\partial f}{\partial z}\right) = \det(I + h'(w^T z + b) u w^T) = 1 + h'(w^T z + b) u^T w$$

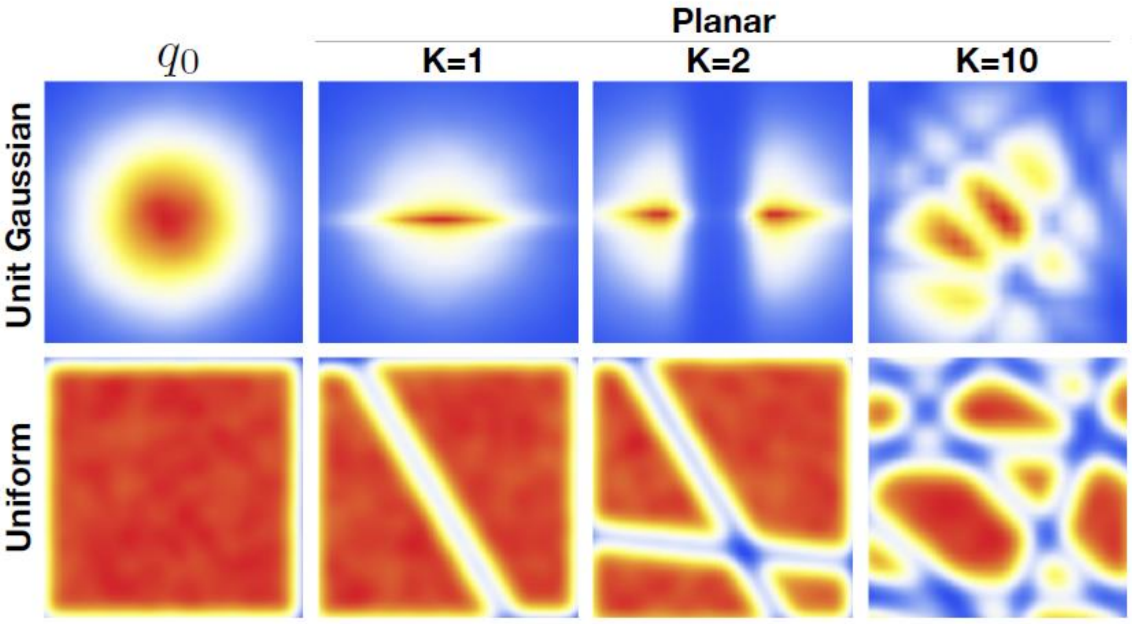
  - Computation is performed in $O(d)$ time

Matrix determinant lemma

  - Remark
    - $u^T w > -1$ to ensure invertibility
    - Normalization on u or w required

Planar Flow

- Planar Flow (Rezende & Mohamed, 2016)
  - $f_\theta(z) = z + uh(w^T z + b)$
  - 10 planar transformations can transform simple distributions into a more complex one



Story So Far

- Normalizing Flow
  - A sequence of invertible functions applied to an easy-to-sample distribution
- Sampling
  - Starting from an “easy” distribution, e.g., isomorphic Gaussian
  - Efficient evaluation of transformations $z \to x$
- Likelihood Estimate
  - Inference over the flow $x \to z$
  - $\log p(x) = \log p(z) - \sum_i \log |\det(\cdot)|$
  - Key idea: efficient determinant evaluation
    - Matrix determinant lemma

Story So Far

- Normalizing Flow
  - A sequence of invertible functions applied to an easy-to-sample distribution
- Sampling
  - Starting from an “easy” distribution, e.g., isomorphic Gaussian
  - Efficient evaluation of transformations $z \rightarrow x$
- Likelihood Estimate
  - Inference over the flow $x \rightarrow z$
  - $\log p(x) = \log p(z) - \sum_i \log |\det(\cdot)|$
  - Key idea: efficient determinant evaluation
    - Matrix determinant lemma
    - **Triangular matrix**

Triangular Jacobian

- Given $x = (x_1, \dots, x_d) = f(z) = (f_1(z), \dots, f_d(z))$

$$J = \frac{\partial f}{\partial z} = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d}{\partial z_1} & \cdots & \frac{\partial f_d}{\partial z_d} \end{bmatrix}$$

- Suppose $x_i = f_i(z)$ only depends on $z_{\leq i}$, then

$$\det J = \det \left| \frac{\partial f}{\partial z} \right| = \det \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d}{\partial z_1} & \cdots & \frac{\partial f_d}{\partial z_d} \end{bmatrix} = \det diag(J) = \prod_i \frac{\partial f_i}{\partial z_i}$$

NICE

- Nonlinear Independent Components Estimation (Dinh et. al, 2014)
  - $z = f(x)$
    - Notational convention for MLE learning
  - we partition x into two disjoint subsets $x_{1:m}$ and $x_{m+1:d}$ for any $1 \leq m \leq d$
  - Forward pass $x \to z$ (inference)
    - $z_{1:m} = x_{1:m}$ (identity)
    - $z_{m+1:d} = x_{m+1:d} + \mu_\theta(x_{1:m})$ (μ_θ is a neural network)
  - Backward pass $z \to x$ (sampling)
    - $x_{1:m} = z_{1:m}$ (identity)
    - $x_{m+1:d} = z_{m+1:d} - \mu_\theta(z_{1:m})$

$$J = \frac{\partial f}{\partial x} = \begin{bmatrix} I_m & 0 \\ \frac{\partial \mu}{\partial x_{1:m}} & I_{d-m} \end{bmatrix}$$

- Volume preserving transformation
  - $\det J = 1$

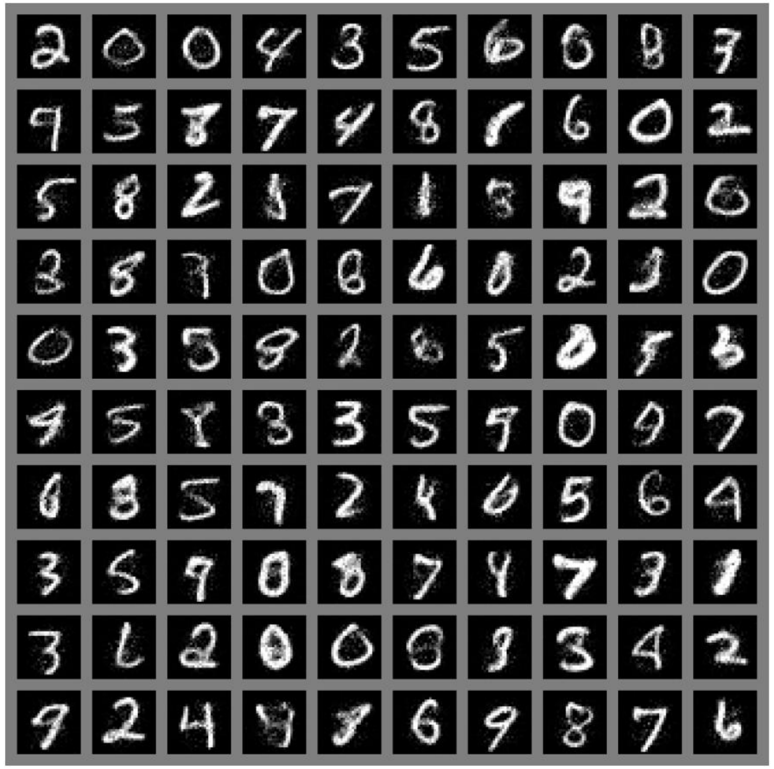
NICE

- Coupling layers are introduced to ensure all dimensions are covered
  - Reverse (or randomly shuffle) the partition before each transformation layer
- First layer of NICE uses a re-scaling layer
  - $z_i = S_{ii}x_i$
  - Ensure non-unit volume transformation
  - Jacobian of forward pass

$$J = \mathrm{diag}(S)$$
$$\det J = \prod_i S_{ii}$$

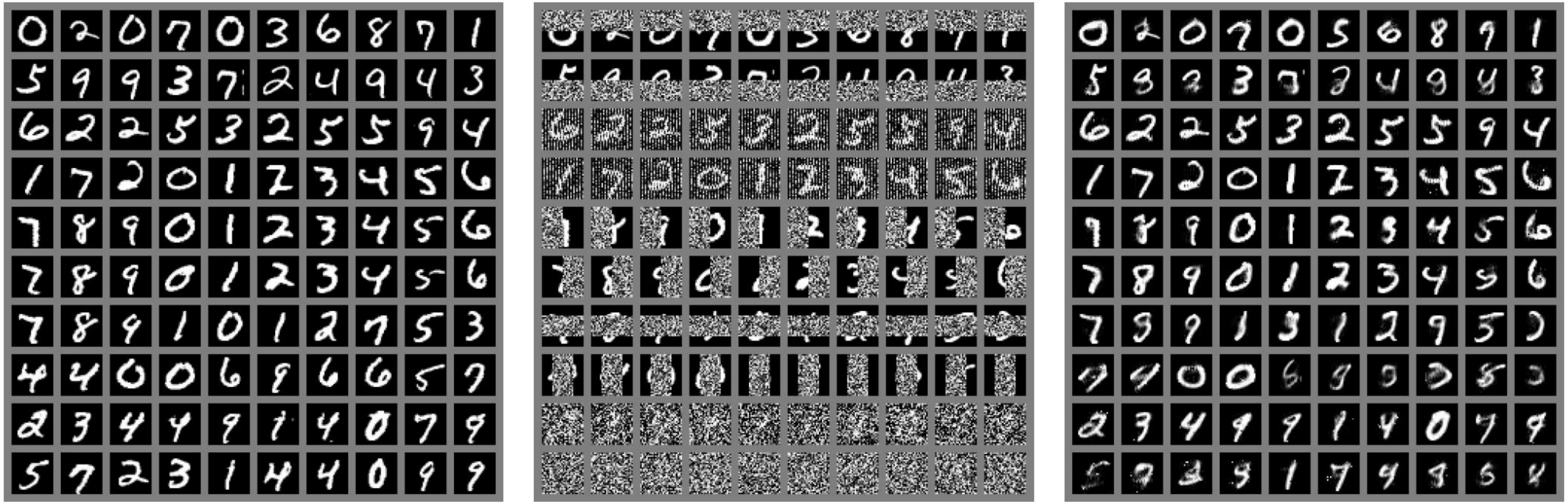
NICE

- Generation Results



NICE

- Inpainting
  - $x = (x_v, x_h)$
  - We have tractable likelihood function $p(x_v, x_h)$!
    - Gradient ascent (stochastic gradient MCMC if you want samples)



Real-NVP

- NICE: most layers maintain an *unchanged* volume
- Non-volume preserving extension of NICE (Dinh et al, 2016)
  - Two partitions over z: $z_{1:m}$ and $z_{m+1:d}$ for any $1 \leq m \leq d$
  - Forward pass $x \to z$ (inference)
    - $z_{1:m} = x_{1:m}$ (identity)
    - $z_{m+1:d} = x_{m+1:d} \cdot \exp(\alpha_\theta(x_{1:m})) + \mu_\theta(x_{1:m})$ (μ_θ & α_θ neural network)
  - Backward pass $z \to x$ (sampling)
    - $x_{1:m} = z_{1:m}$ (identity)
    - $x_{m+1:d} = \left(z_{m+1:d} - \mu_\theta(z_{1:m})\right) \cdot \exp(-\alpha_\theta(x_{1:m}))$
  - Non-volume preserving transformation

$$\det J = \prod_{i=m+1}^{d} \exp(\alpha_\theta(x_{1:m})_i)$$

Real-NVP

- Fun Fact

Accepted as a workshop contribution at ICLR 2015

NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh **David Krueger** **Yoshua Bengio***
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Published as a conference paper at ICLR 2017

DENSITY ESTIMATION USING REAL NVP

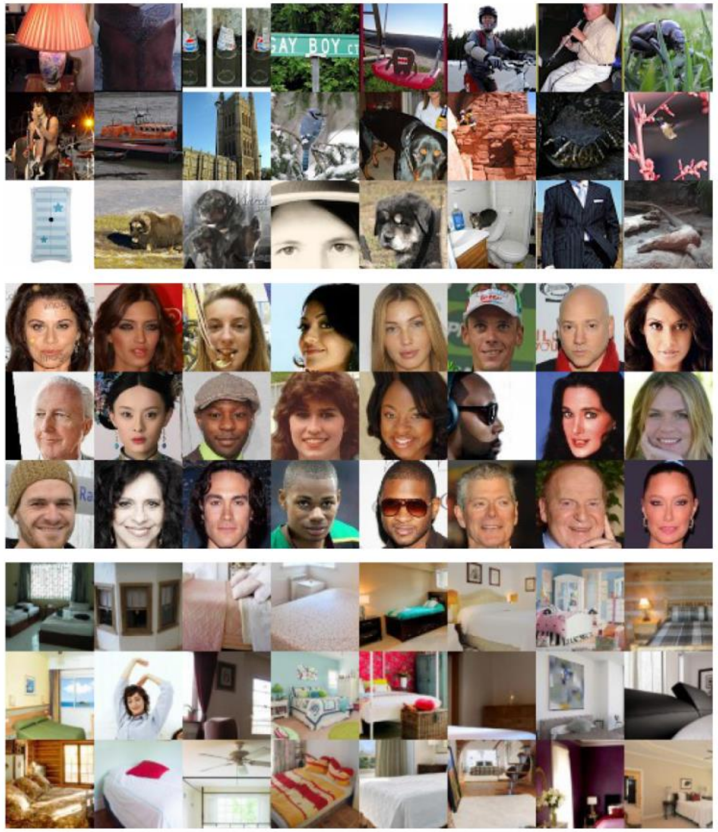
Laurent Dinh*
Montreal Institute for Learning Algorithms
University of Montreal
Montreal, QC H3T1J4

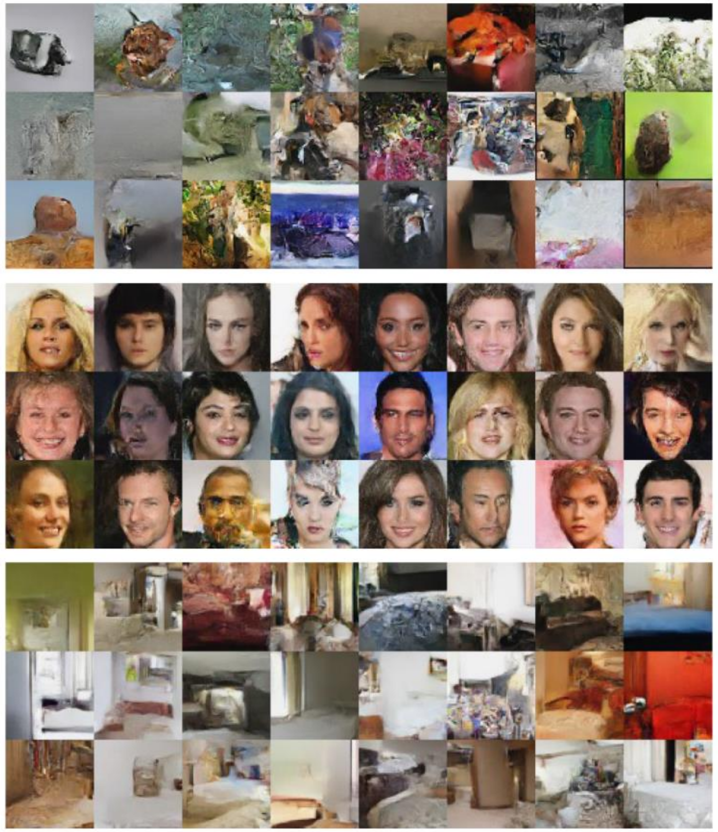
Jascha Sohl-Dickstein
Google Brain

Samy Bengio
Google Brain

Real-NVP

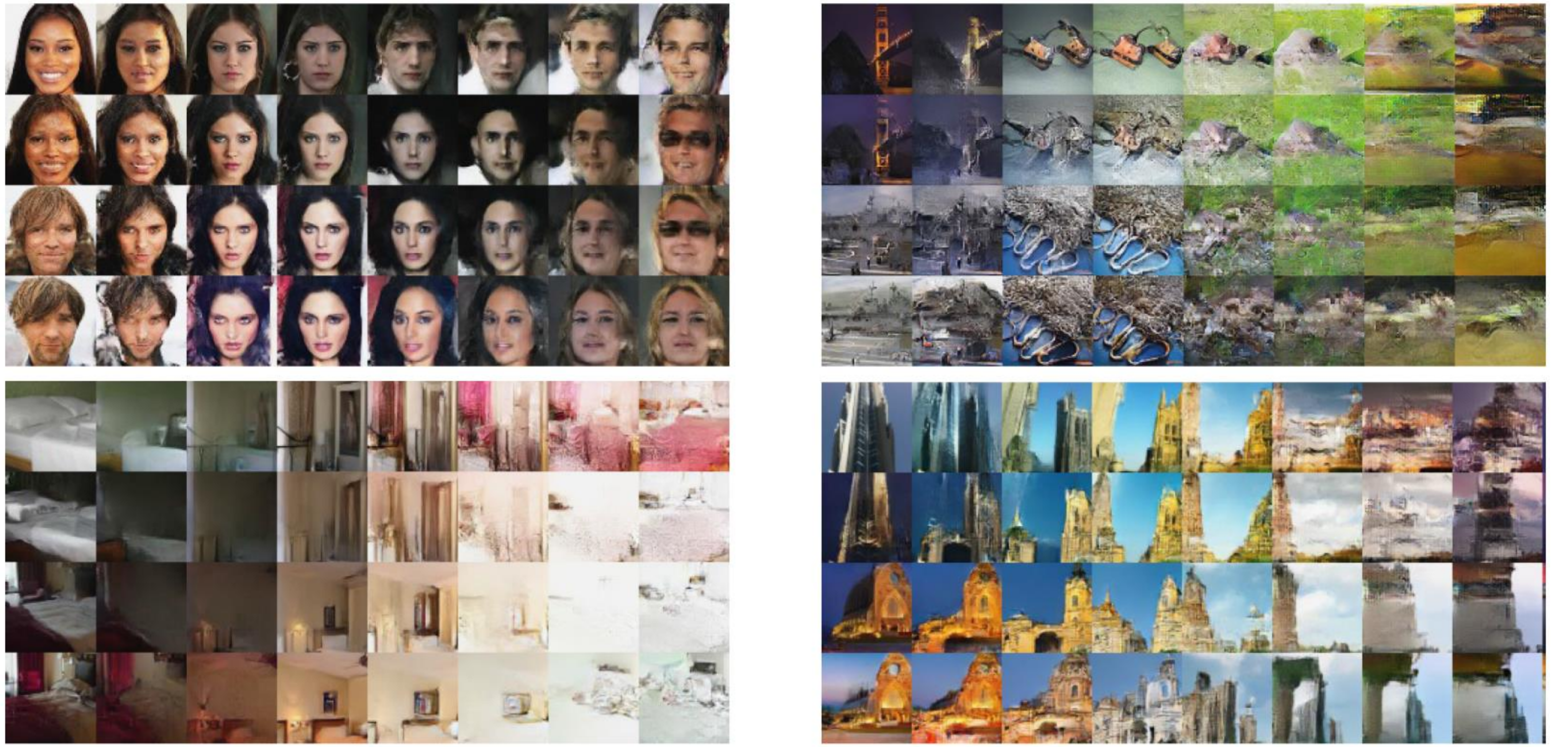
- Generation Results
  - Left: training data; Right: generated samples





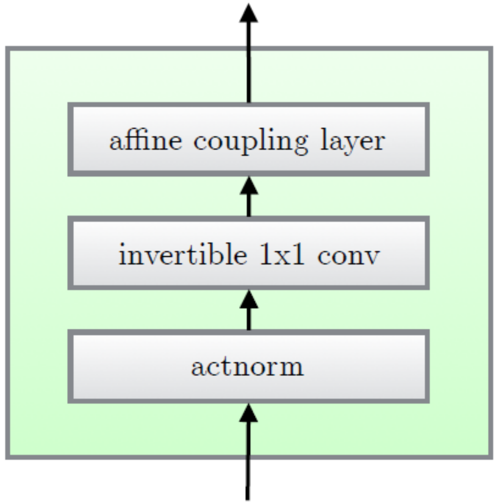
Real-NVP

- Explore the latent space
  - 4 samples selected: z^0, z^1, z^2, z^3, two interpolation parameters α, β
  - $z = \cos(\alpha)\,(\cos(\beta)z^1 + \sin(\beta)z^2) + \sin(\alpha)(\cos(\beta)\,z^3 + \sin(\beta)z^4)$



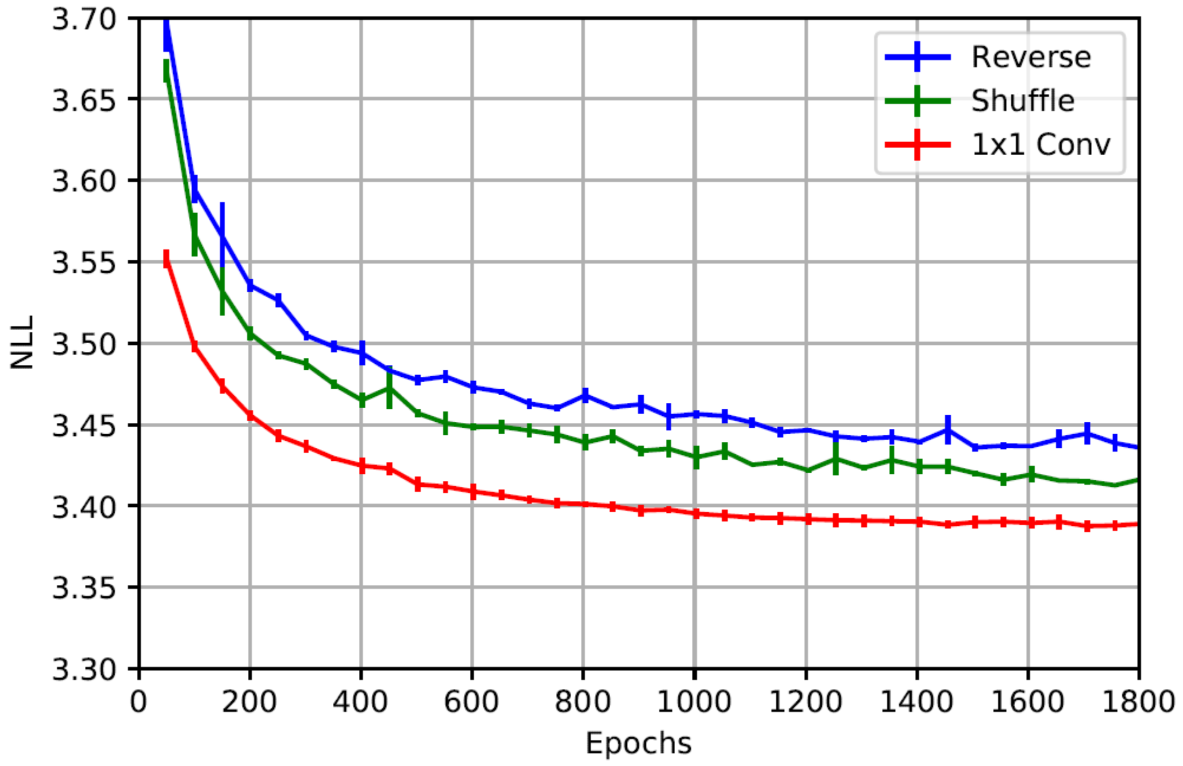
GLOW

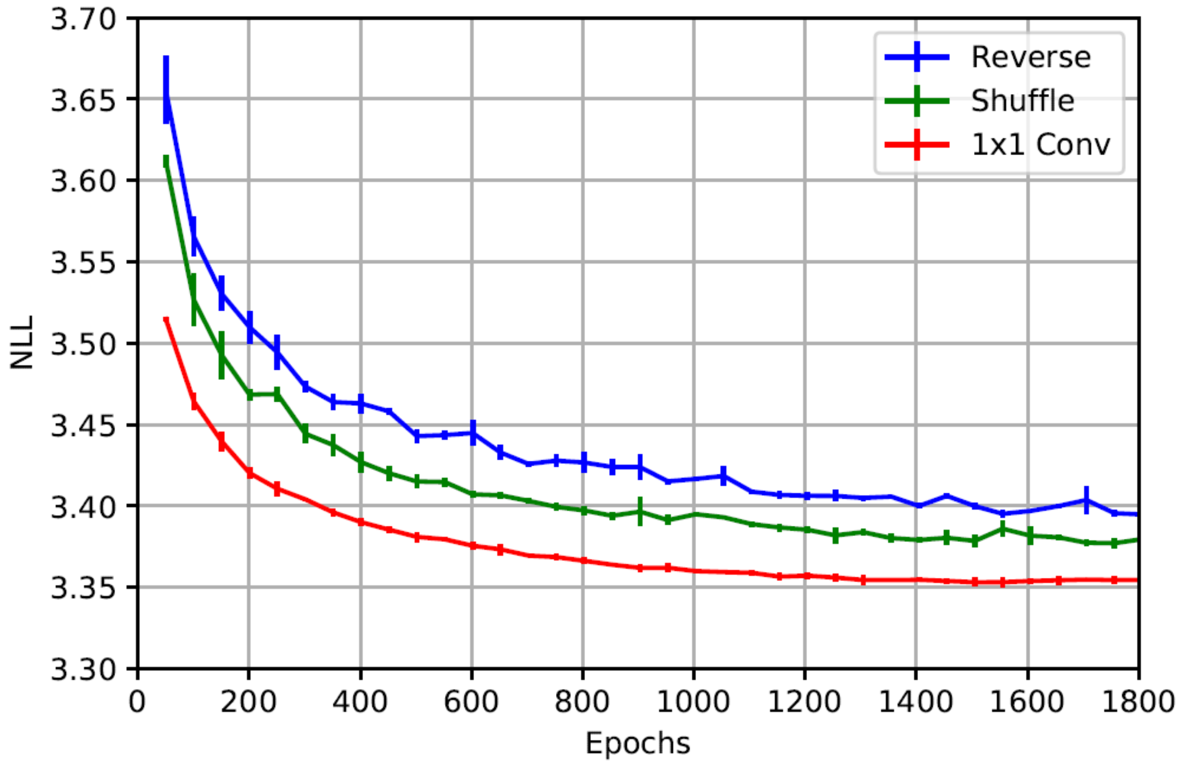
- Limited expressiveness of previous coupling layers
  - But a general non-linear transformation can be too expensive…
- Generative Flow with Invertible 1x1 Convolutions (Kingma et al. 2018)
  - Input: $x = h \times w \times c$ (height, width, channel) (assume c is small)
  - Key idea: apply a 1x1 convolution when number of channels is small
  - 1x1 conv: a linear transformation for each pixel
    - Forward mapping: $z_{ij:} = W x_{ij:} + b$
    - Inverse mapping: simply compute the inverse matrix of W
  - Computation $O(c^3)$
    - $\log|\det J| = h \cdot w \cdot \log|\det W|$
  - Also use normalization layer to stabilizing training
    - Architecture details can be found in the paper



GLOW

- Generative Flow with Invertible 1x1 Convolutions (Kingma et al. 2018)
  - 1x1 convolution generalizes channel-wise partition





GLOW

- Generation Results

Triangular Jacobian (Revisit)

- Design $x_i = f_i(z)$ only depends on $z_{\leq i}$, then

$$\det J = \det\left|\frac{\partial f}{\partial z}\right| = \det\begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d}{\partial z_1} & \cdots & \frac{\partial f_d}{\partial z_d} \end{bmatrix} = \det diag(J) = \prod_i \frac{\partial f_i}{\partial z_i}$$

  - Tractable likelihood
- Idea
  - Sequential generation $x_1, x_2, \ldots, x_d$
  - x_i only depends on $x_{<i}$, the likelihood can be easily tractable

$$p(x) = \prod_{i=1}^{d} p(x_i | x_{<i})$$

Autoregressive Flow

- Autoregressive model
  - $p(x) = \prod_{i=1}^{d} p(x_i | x_{<i})$
  - Fully tractable probability, easy to generate
    - Remark: AR model be re-visited in sequence modeling lectures

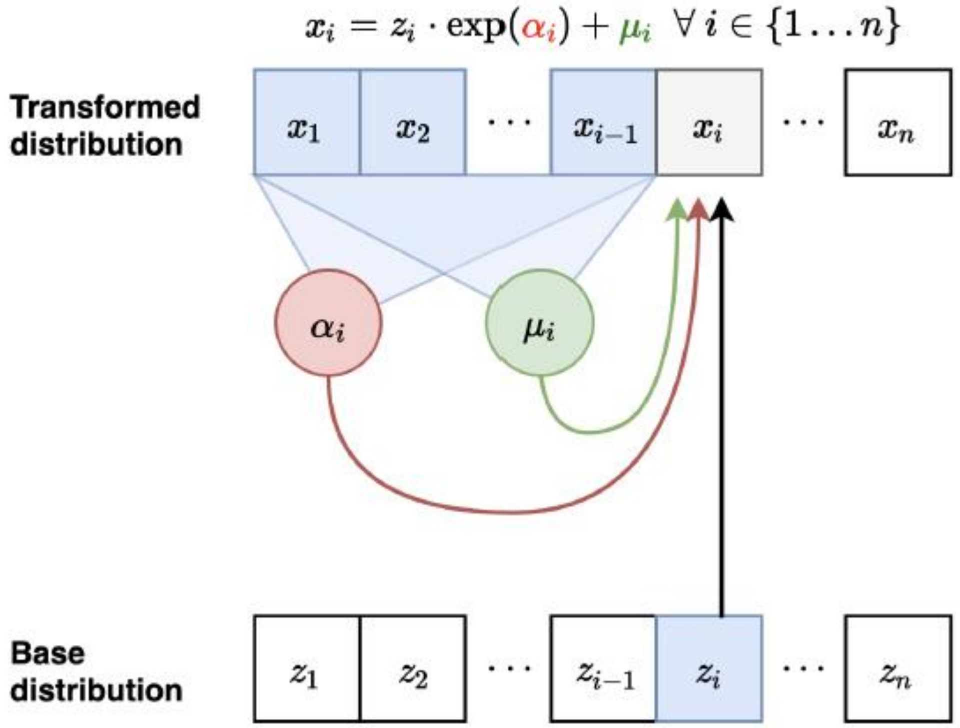
- Example: Gaussian Autoregressive model
  - $p(x_i | x_{<i}) = N\left(\mu_i(x_1, \dots, x_{i-1}), \exp\left(\alpha_i(x_1, \dots, x_{i-1})\right)^2\right)$
  - μ_i, α_i are neural networks for $i > 1$ and constant for $i = 1$
  - Connection to flow models?

Autoregressive Flow

- Autoregressive models as flow models
  - $p(x_i|x_{<i}) = N\left(\mu_i(x_1, \dots, x_{i-1}), \exp\left(\alpha_i(x_1, \dots, x_{i-1})\right)^2\right)$
  - Forward mapping (sampling)
    - Sample $z_i \sim N(0,1)$ for $i = 1 \dots d$
    - Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$
    - Let $x_2 = \exp(\alpha_2)z_2 + \mu_2$ Compute μ_2, α_2 based on x_1
    - Let $x_3 = \exp(\alpha_3)z_3 + \mu_3$ Compute μ_3, α_3 based on x_1, x_2
    - ……
  - x_i and z_i are bijections
  - **Flow interpretation**
    - For each x_i, we have an invertible transformation using $z_i \sim N(0,1)$ and $x_{<i}$
    - $z \sim N(0, I) \rightarrow x \sim p(x)$ with non-linear transformations μ_i and α_i

Autoregressive Flow

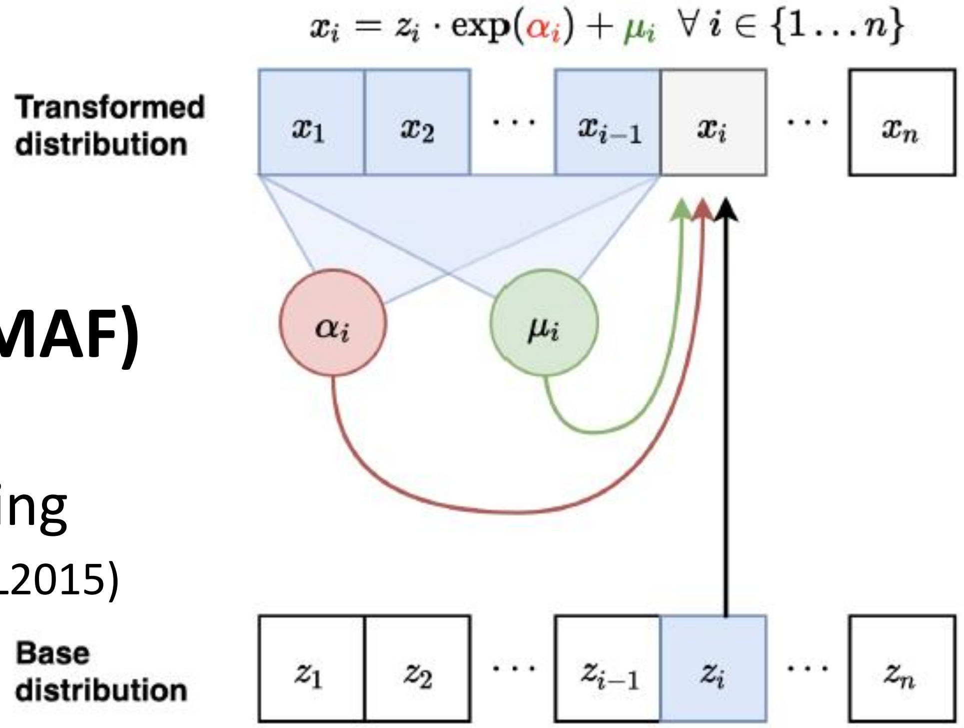
- Autoregressive Flow (AF)
  - $z \sim N(0, I)$
  - $x_i = \exp\big(\alpha_i(x_{<i})\big)\, z_i + \mu_i(x_{<i})$
- Forward mapping (sampling)
  - $O(d)$ time: sequential generation
- Inverse mapping: $x \to z$ (evaluate $p(x)$)
  - Evaluate α_i and μ_i in parallel (since x is given)
  - $z_i \leftarrow (x_i - \mu_i)/\exp(\alpha_i)$ in parallel
  - $p(x) = \prod_i p(x_i | x_{<i}) = p(z) \prod_i \exp(-\alpha_i)$
- Remark
  - Factorized Distribution (joint → sequential)
  - 1-D transformation for each dim, easy to inverse



Autoregressive Flow

- Autoregressive Flow (AF)
  - $z \sim N(0, I), x_i = \exp\left(\alpha_i(x_{<i})\right) z_i + \mu_i(x_{<i})$
- Forward mapping (sampling)
  - $O(d)$ time: sequence generation
- Inverse mapping: $x \to z$ (evaluate $p(x)$)
  - $z_i \leftarrow (x_i - \mu_i) / \exp(\alpha_i)$ in parallel
- **Extension: Masked Autoregressive Flow (MAF)**
  - Stack AF for multiple layers
  - Use MADE architecture for fast forward sampling

    MADE: Masked Autoencoder for Distribution Estimation (ICML2015)
  - Remark: relatively low-dimensional data



Autoregressive Flow: Quantization

- Discrete values for x?
  - Pixel values are integers in $\{0,1,\ldots,255\}$
  - If we apply Gaussian AR flow on image data, how to interpret $p(x_i = 1.5)$?
    - Quantization at inference time!
      - E.g., $[x, x+1) \to x$
    - Training? We only have discrete valued-x
      - No constraint at all on non-integer input value to the network!
- Goal: we want MLE training over the integral $[x, x+1)$ for x
  - $\hat{p}(x)$ an uncalibrated Gaussian AR model (i.e., neural network)

$$p(x) = \int_{[0,1)^d} \hat{p}(x+u)du$$

  - We want to optimize $p(x)$!

Autoregressive Flow: Quantization

- Goal: optimize the density over the integral of $[v, v+1)$
  - $\hat{p}(x)$ an uncalibrated Gaussian AR model

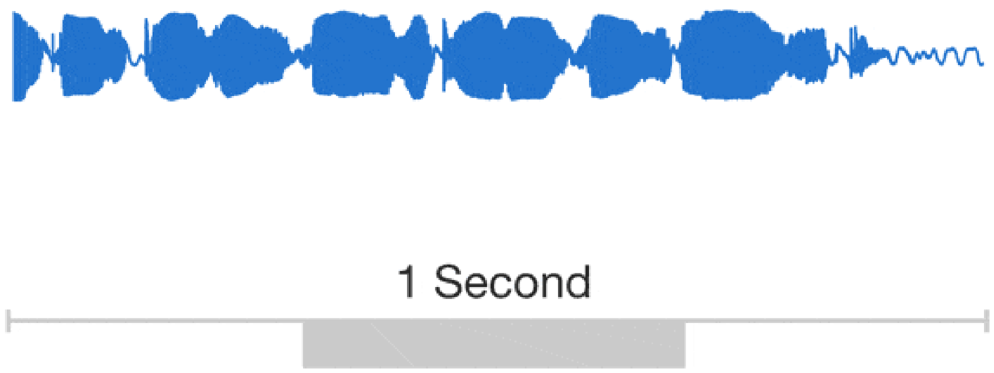
$$p(x) = \int_{[0,1)^d} \hat{p}(x+u)du$$

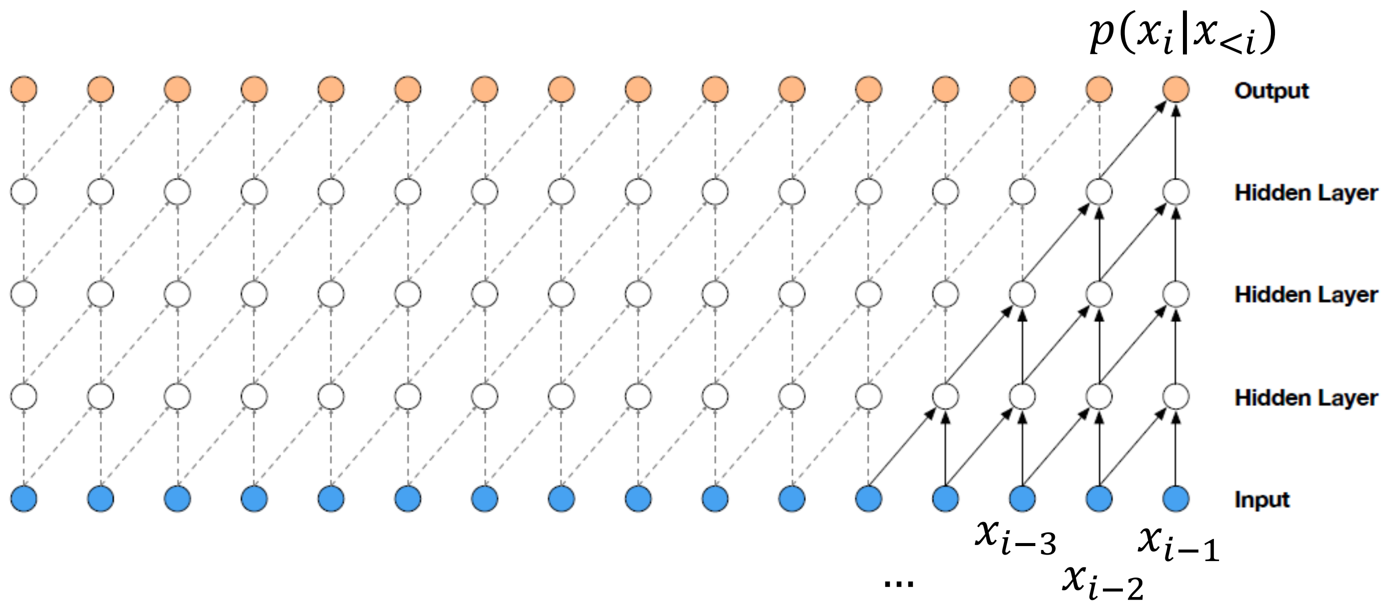
- Solution: Dequantization (Theis, Oord, Bethge, 2016)
  - Data augmentation: add noise to data by randomly drawing $u \sim [0,1)^d$
  - $x' \leftarrow x + u$
  - $L(\theta) = \mathrm{E}_{x'}[\log \hat{p}(x')] = \sum_x p_{data}(x) \int_{[0,1)^d} \log \hat{p}(x+u)\, du$
  - $\leq \sum_x p_{data}(x) \log \int_{[0,1)^d} \hat{p}(x+u) du$
  - $= \mathrm{E}_x[\log p(x)]$

We are optimizing a lower-bound of true objective!

Autoregressive Flow: Example

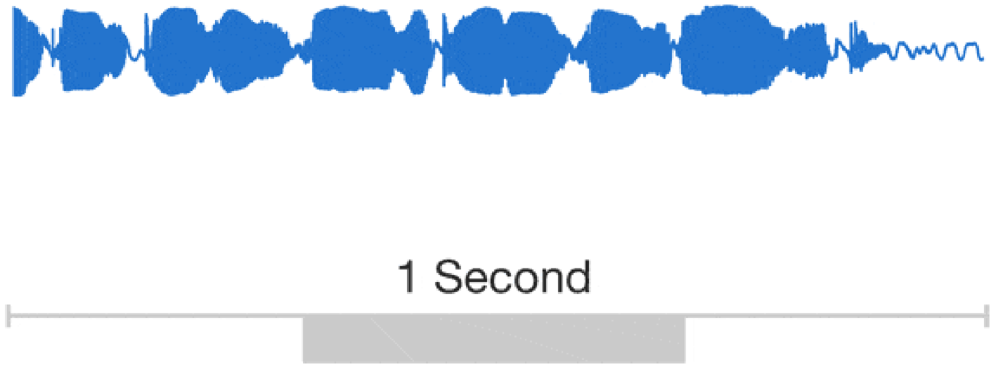
- WaveNet (DeepMind, 2016)
  - Goal: voice synthesis
  - $p(x) = \prod_i p(x_i | x_1, \dots, x_{i-1})$
  - Idea: temporal convolution
    - **Issue: how many layers do you need?**

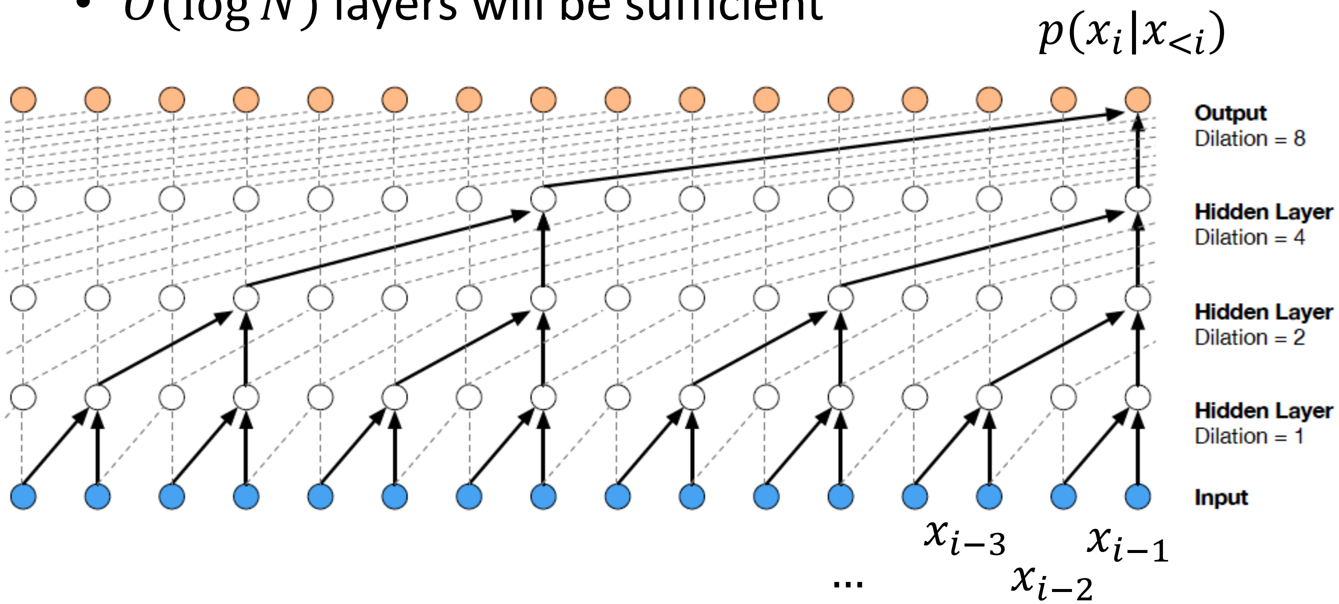




Autoregressive Flow: Example

- WaveNet (DeepMind, 2016)
  - Goal: voice synthesis
  - $p(x) = \prod_i p(x_i | x_1, \dots, x_{i-1})$
  - Idea: temporal convolution
    - **Dilated Convolution!**
    - $O(\log N)$ layers will be sufficient





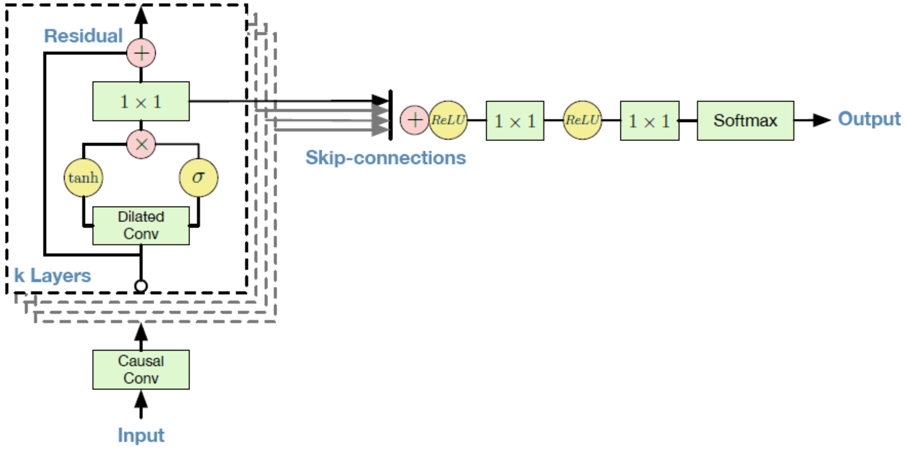
Computation Cost

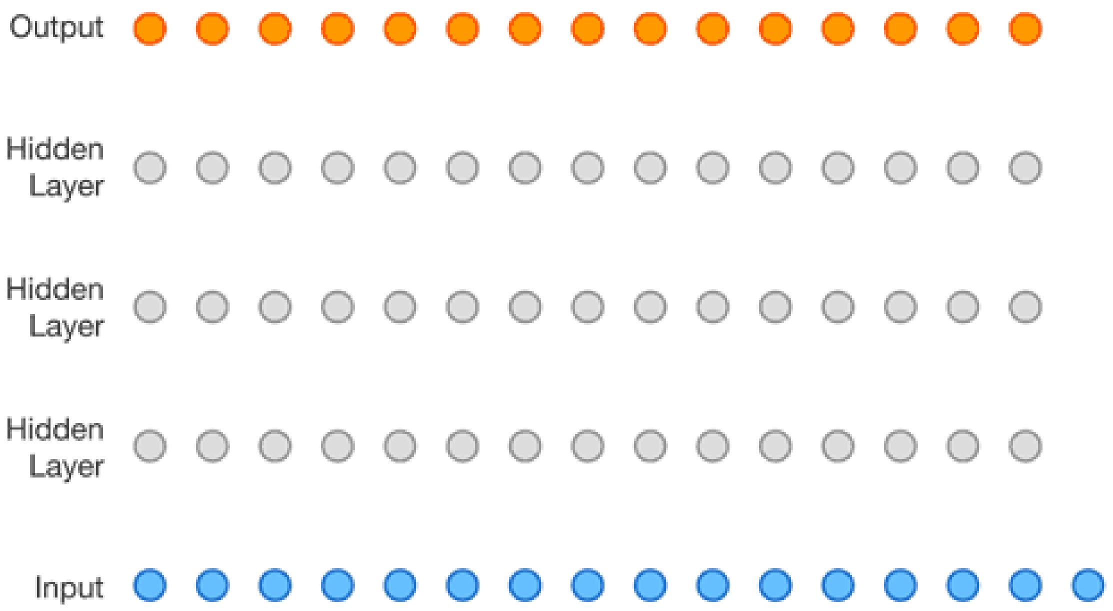
- Generation
  - Sequential: $O(N)$
- Likelihood:
  - fully parallel (CNN)

Autoregressive Flow: Example

- WaveNet (DeepMind, 2016)
  - $p(x) = \prod_i p(x_i|x_1, \dots, x_{i-1})$
  - Dilated Temporal Convolution
  - And more
    - Quantization
    - Residual connection
    - Conditioned generation
      - $p(x|h)$
  - Remark
    - First deep generative model that can generate raw signals

(also check newer ones Jukebox & Suno)

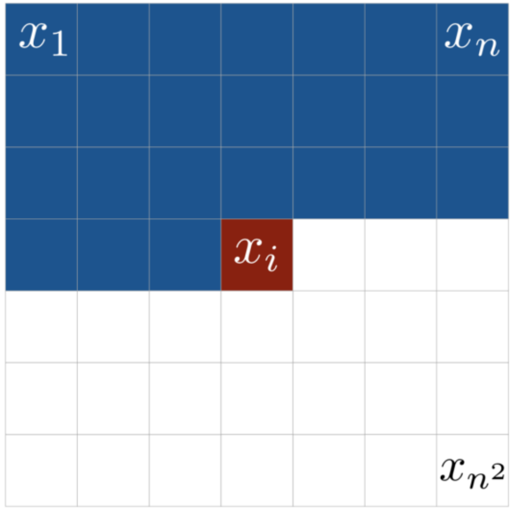




https://deepmind.com/blog/article/wavenet-generative-model-raw-audio
https://openai.com/blog/jukebox/
https://www.suno.ai/

Autoregressive Flow: Example

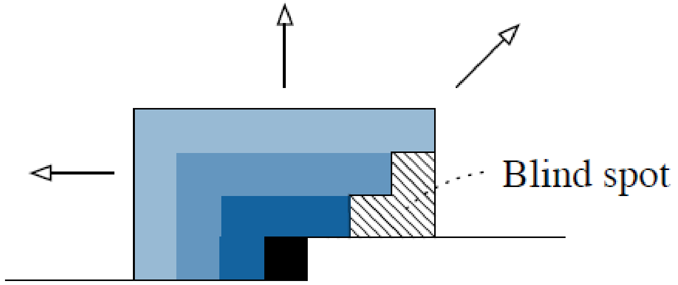
- PixelCNN (DeepMind, ICML 2016)
  - Autoregressive model over images
  - $p(x) = \prod_i^{N^2} p(x_i | x_1, \dots, x_{i-1})$
  - CNN?
    - How to design the convolution filter?
    - Goal: the convolution filter only takes in previous values

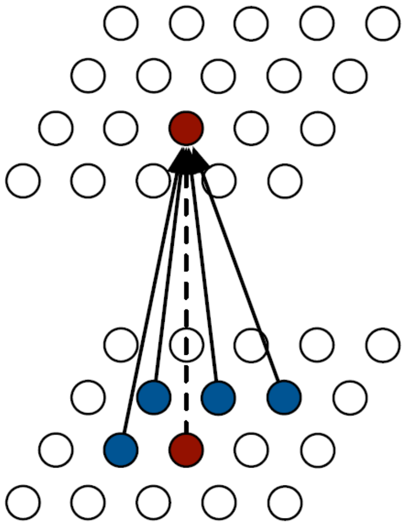


Context

Autoregressive Flow: Example

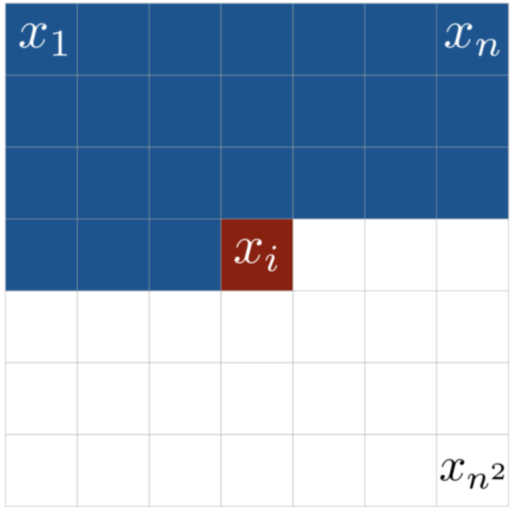
- PixelCNN (DeepMind, ICML 2016)
  - Autoregressive model over images
  - $p(x) = \prod_i^{N^2} p(x_i | x_1, \dots, x_{i-1})$
  - **Masked Convolution**
    - Each pixel only takes in previous values
  - Likelihood evaluation is in perfect parallel
  - Issues?
    - Receptive fields have blind spots!



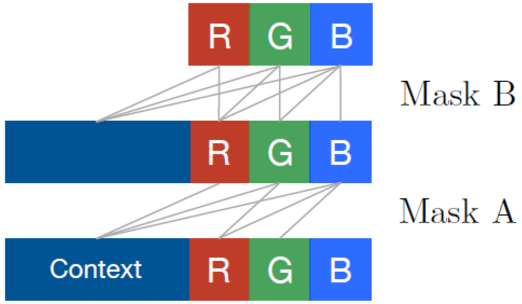


1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Mask

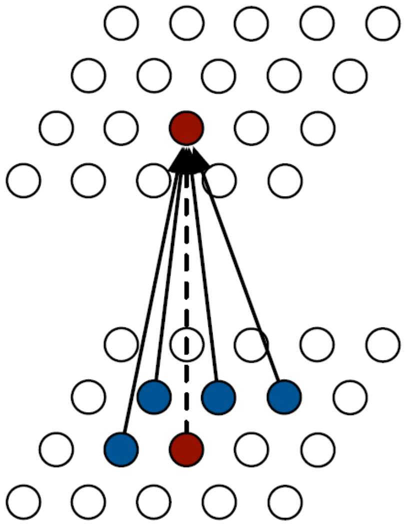


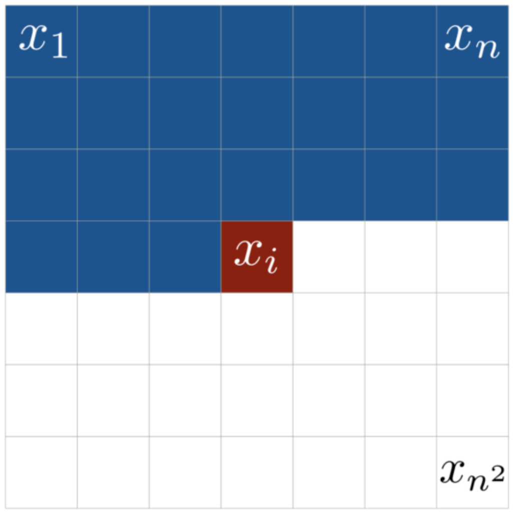
Context



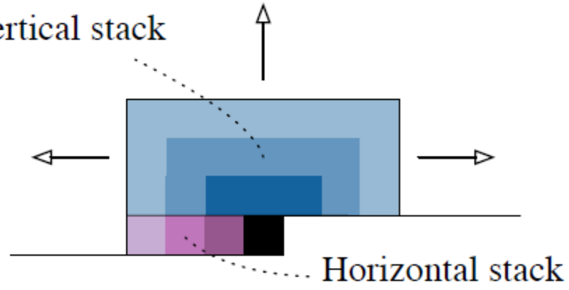
Autoregressive Flow: Example

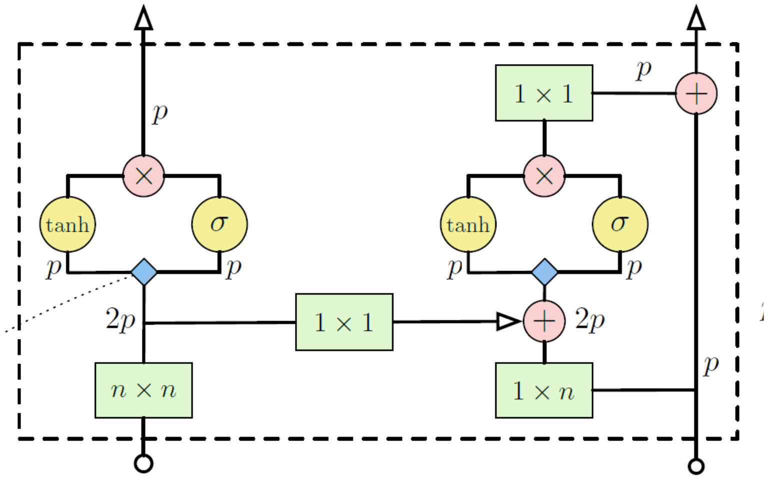
- PixelCNN (DeepMind, ICML 2016)
  - Autoregressive model over images
  - $p(x) = \prod_i^{N^2} p(x_i | x_1, \ldots, x_{i-1})$
  - **Masked Convolution**
    - Each pixel only takes in previous values
  - Likelihood evaluation is in perfect parallel





- Gated PixelCNN (DeepMind, NIPS 2016)
  - Corrected receptive field (homework ☺)
  - Gated convolution
    - "Gating" technique
    - Inspired by LSTM
      - More in future lectures

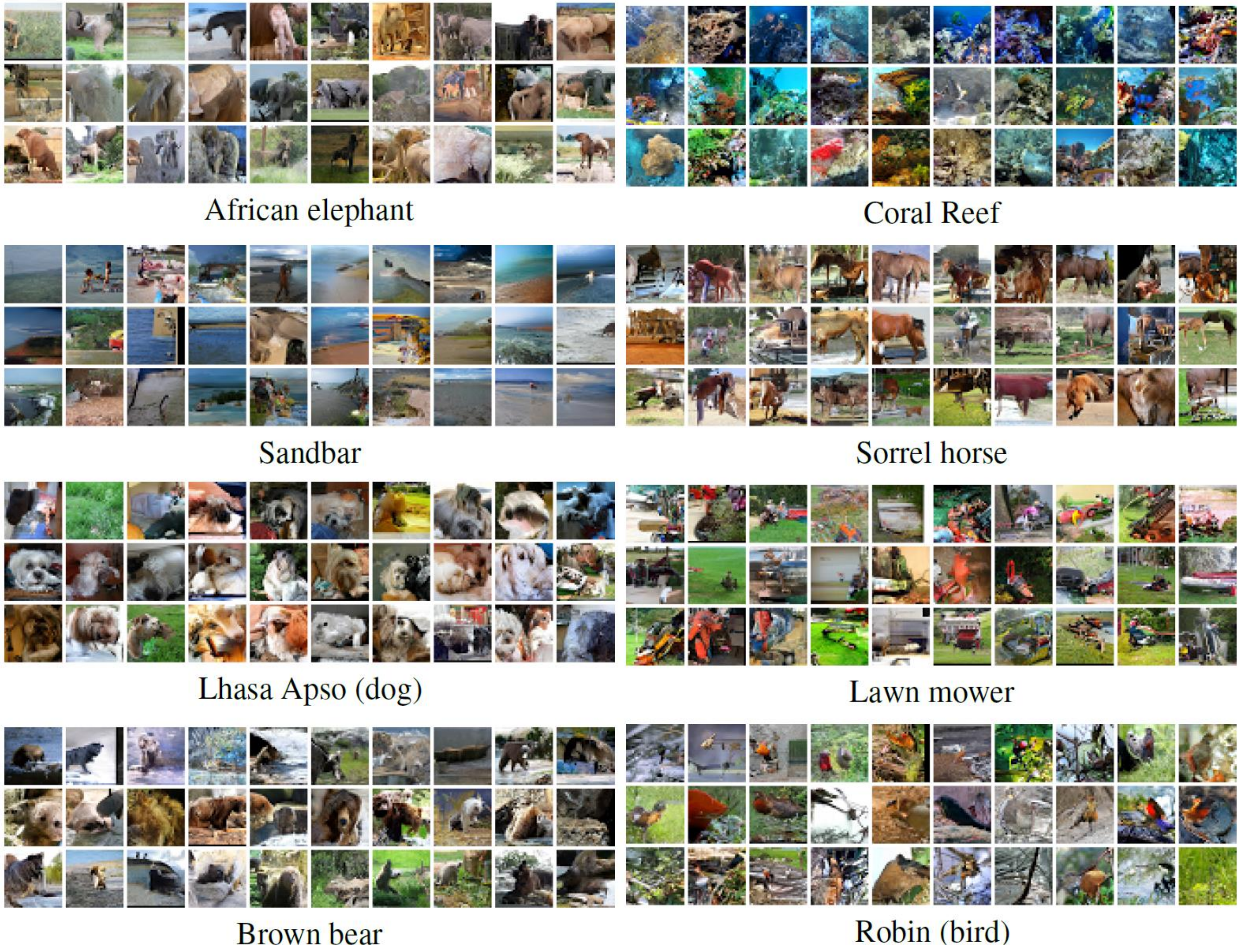




Autoregressive Flow: Example

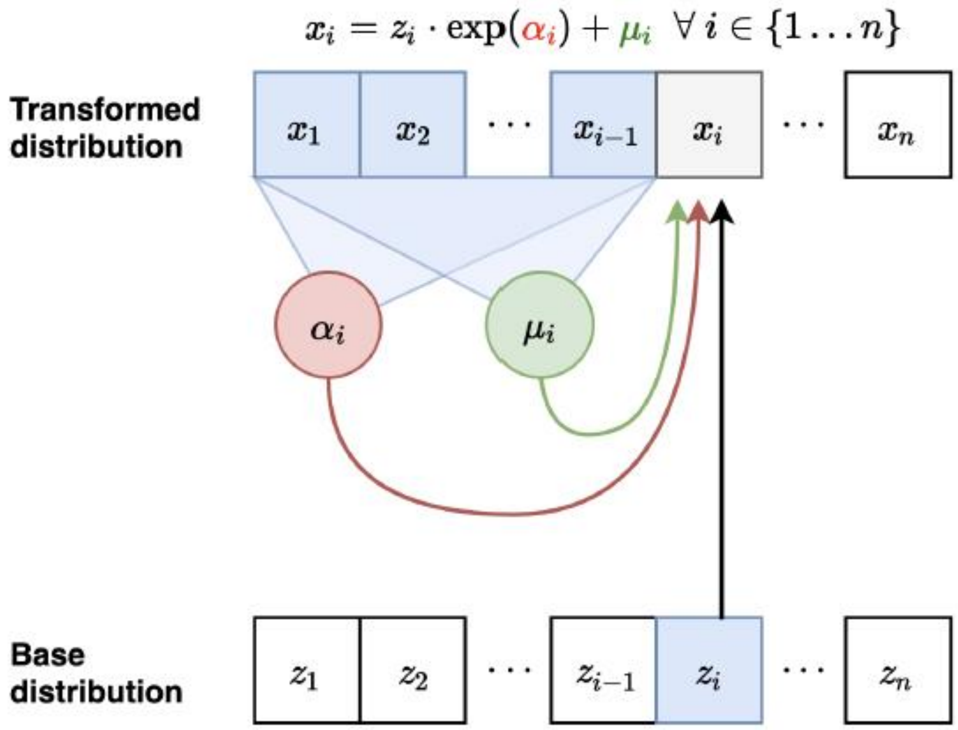
- Conditioned generation

Gated PixelCNN



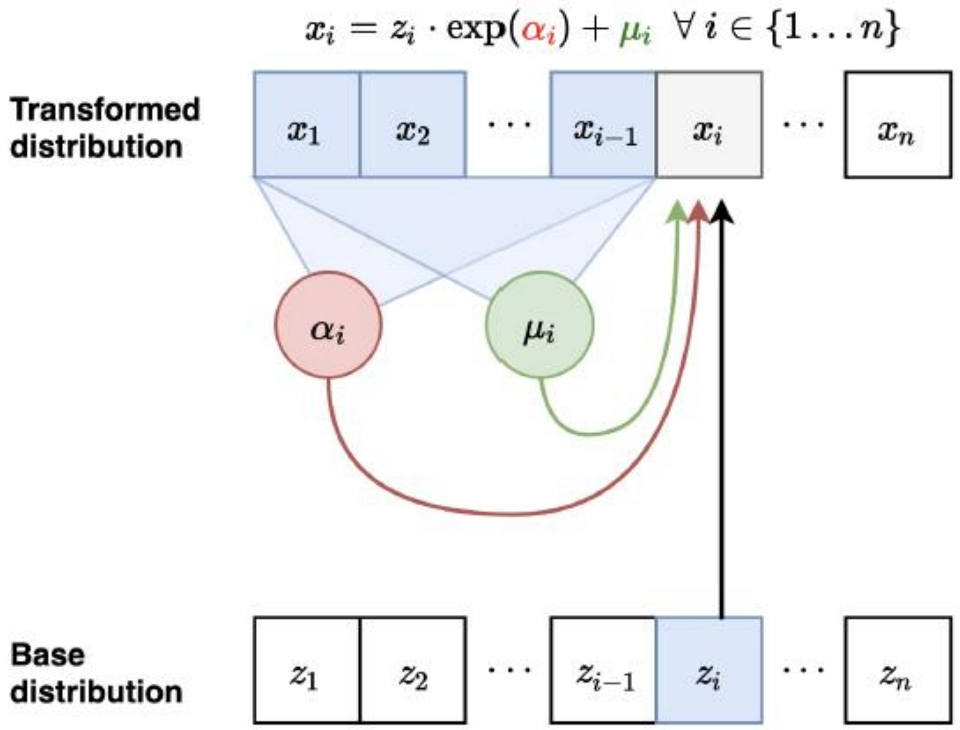
Autoregressive Flow (Recap)

- Gaussian Autoregressive Flow (AF)
  - $z \sim N(0, I)$
  - $x_i = \exp\big(\alpha_i(x_{<i})\big)\, z_i + \mu_i(x_{<i})$
- Forward mapping (sampling)
  - $O(d)$ time: sequential sampling
- Inverse mapping: $x \to z$ (likelihood)
  - Likelihood evaluation is in perfect parallel
  - **Training is fast!**
- Faster sampling?



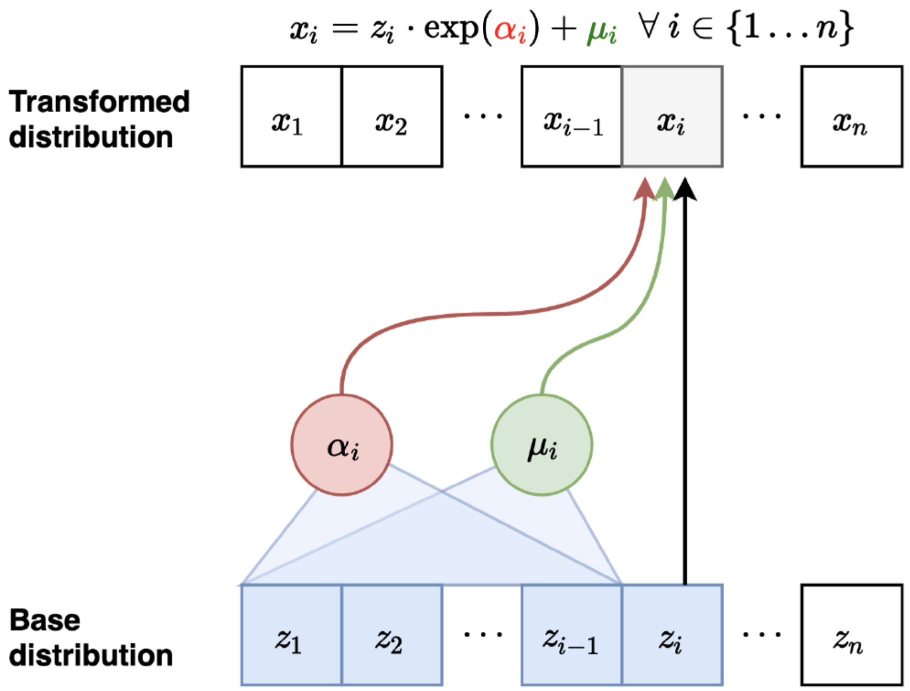
Autoregressive Flow (Recap)

- Gaussian Autoregressive Flow (AF)
  - $z \sim N(0, I)$
  - $x_i = \exp\left(\alpha_i(x_{<i})\right) z_i + \mu_i(x_{<i})$
- Forward mapping (sampling)
  - $O(d)$ time: sequential sampling
- Inverse mapping: $x \rightarrow z$ (likelihood)
  - Likelihood evaluation is in perfect parallel
  - Training is fast!
- Faster sampling?
  - **We can let $\boldsymbol{\alpha_i}, \boldsymbol{\mu_i}$ condition on $\boldsymbol{z_{<i}}$**
    - Remark: mapping between x and z is a bijection



Inverse Autoregressive Flow (IAF)

- Gaussian Inverse Autoregressive Flow (IAF)
  - $z \sim N(0, I)$
  - $x_i = \exp\big(\alpha_i(z_{<i})\big)\, z_i + \mu_i(z_{<i})$
- Forward mapping $z \to x$ (sampling)
  - α_i and μ_i can be evaluated in parallel
  - Fast sampling!
- Inverse mapping $x \to z$ (likelihood)
  - $z_i \leftarrow (x_i - \mu_i) \cdot \exp(-\alpha_i)$
  - In order to evaluate $p(x_i|z_{<i})$, we need to generate all $z_{<i}$
  - $O(d)$ computation for evaluation (slow training)



AF v.s. IAF

- Interchange x and z matches the forward mapping and inverse mapping of AF (MAF) and IAF

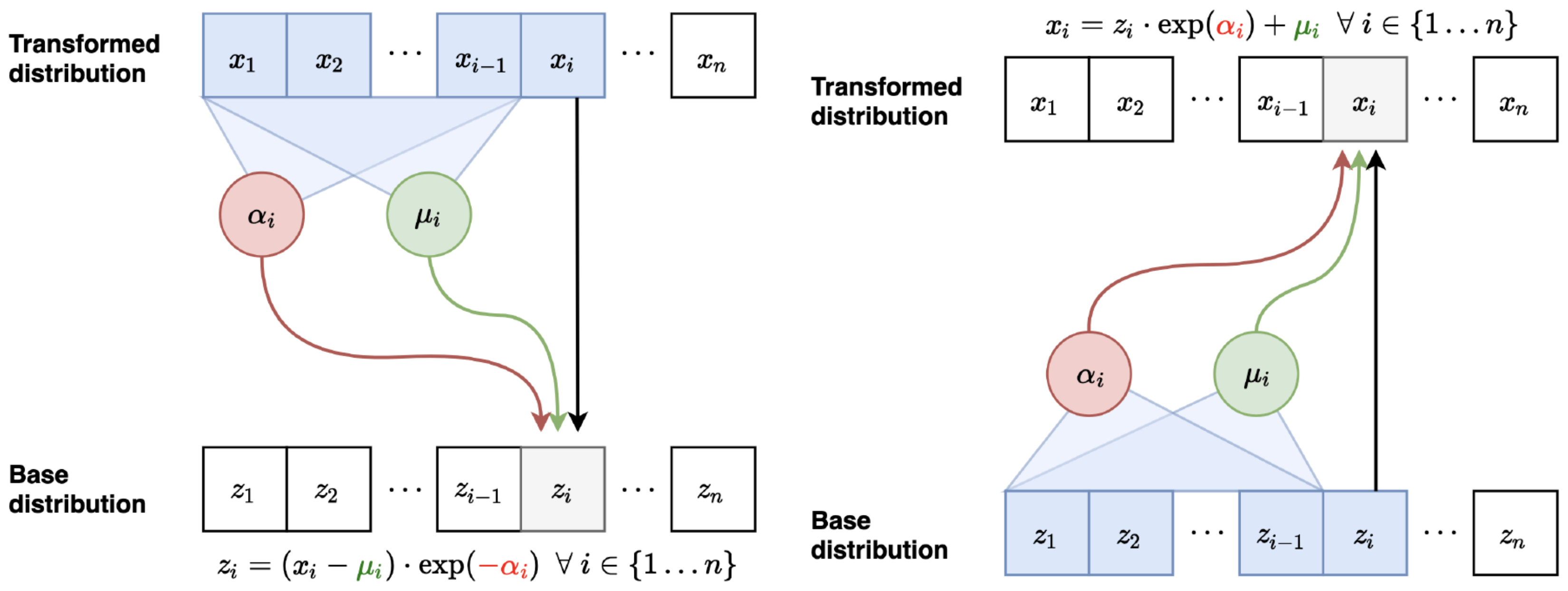


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

AF v.s. IAF

- AF: fast evaluation (inverse mapping) + slow sampling (forward)
- IAF: slow evaluation (inverse mapping) + fast sampling (forward)
- **Can we get positive sides from both frameworks?**

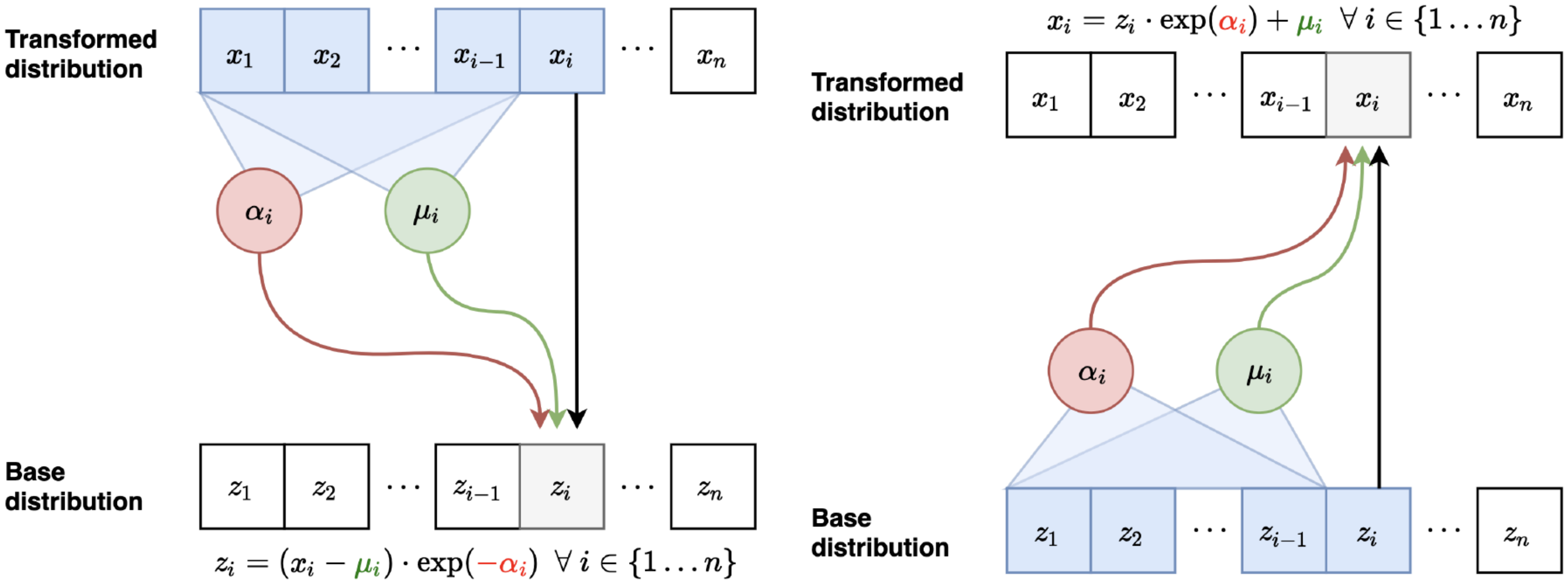


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

Parallel WaveNet

- Parallel WaveNet (DeepMind, ICML 2018)
  - Original WaveNet is AF, slow sampling but train fast
  - We can convert WaveNet to an IAF, but it will be slow to train (evaluation)
  - Idea: teacher student framework!
    - Teacher: $p_T(x_i|x_{<i})$ a standard WaveNet (fast training)
    - Student: $p_S(x_i|z_{<i})$ a IAF WaveNet, and running imitation learning w.r.t. p_T
      - i.e., minimize the difference between $p_S(x)$ and $p_T(x)$
    - **Key observation: if z is given, evaluation of IAF is fast (Gaussian density)**
  - Algorithm Sketch
    - Step 1: Train teacher $p_T(x_i|x_{<i})$ network and fix it
    - Step 2: Minimize the difference between $p_T(x)$ and $p_S(x)$
    - Finally we use $p_S(x)$ for fast sampling

Parallel WaveNet

- Parallel WaveNet (DeepMind, ICML 2018)
  - Original WaveNet is AF, slow sampling but train fast
  - We can convert WaveNet to an IAF, but it will be slow to train (evaluation)
  - Idea: teacher student framework!
    - Teacher: $p_T(x_i|x_{<i})$ a standard WaveNet (fast training)
    - Student: $p_S(x_i|z_{<i})$ a IAF WaveNet, and running imitation learning w.r.t. p_T
      - i.e., minimize the difference between $p_S(x)$ and $p_T(x)$
    - **Key observation: if z is given, evaluation of IAF is fast (Gaussian density)**
  - Algorithm Sketch
    - Step 1: Train teacher $p_T(x_i|x_{<i})$ network and fix it
    - Step 2: Minimize the difference between $p_T(x)$ and $p_S(x)$
    - Finally we use $p_S(x)$ for fast sampling

Parallel WaveNet

- Parallel WaveNet (DeepMind, ICML 2018)
  - Teacher-Student Learning
    - Pretrain $p_T(x)$ and then train $p_S(x)$ by imitation learning
  - Distance measure for two distributions
    - KL divergence: $KL(p||q) = \mathrm{E}_{x\sim p}\left[\log\frac{p(x)}{q(x)}\right]$
      - Asymmetric measure!
      - **Choose p such that p is easy to sample from**
        - more to come in the next lecture ☺

Parallel WaveNet

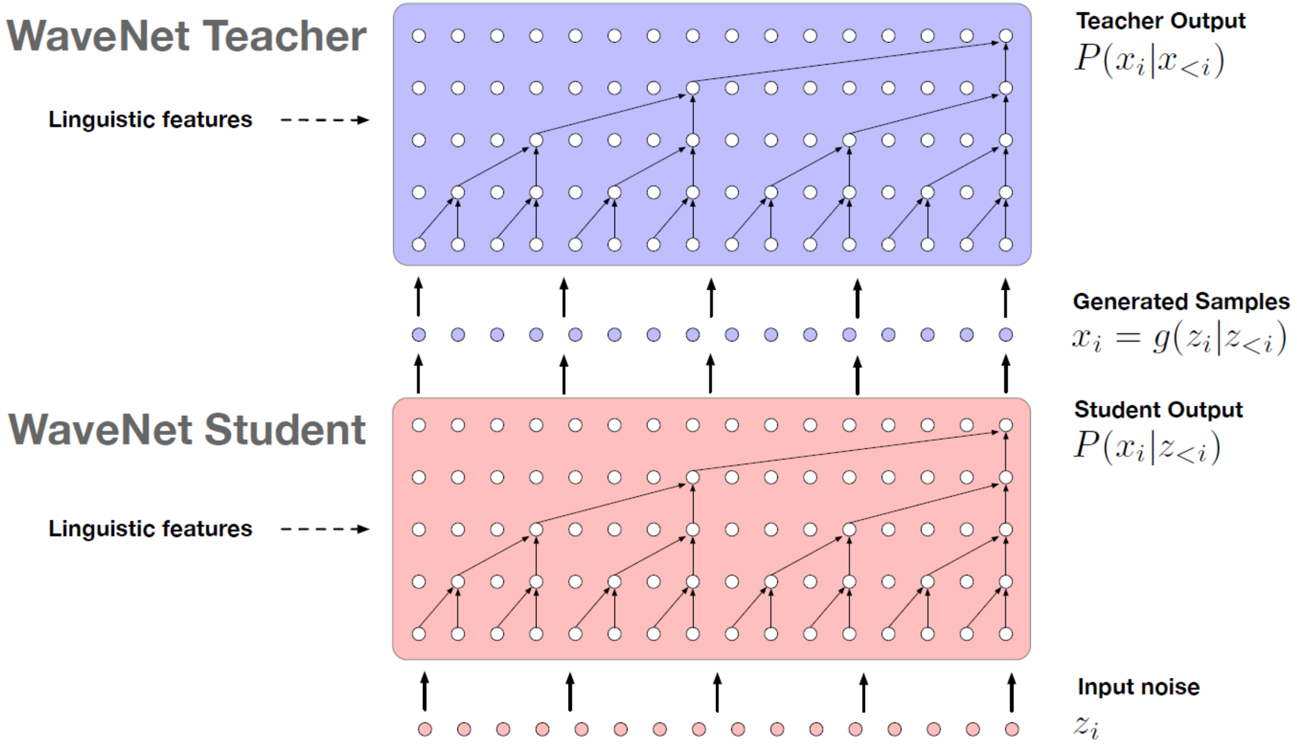
- Parallel WaveNet (DeepMind, ICML 2018)
  - Teacher-Student Learning
    - Pretrain $p_T(x)$ and then train $p_S(x)$ by imitation learning
  - Distance measure for two distributions
    - KL divergence: $KL(p||q) = \mathrm{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right]$
  - Imitation learning

$$L(\theta) = KL(p_S||p_T) = \mathrm{E}_{x\sim p_S}[\log p_S(x;\theta) - \log p_T(x)]$$

    - Monte Carlo estimates for the expectation
      - **Key: sample from the student network!**
    - Sample $z\sim N(0,I)$, generate $x\sim p_S(x|z)$ (parallel)
    - Evaluate $p_S(x|z)$ (parallel since z is known)
    - Evaluate $p_T(x)$ (parallel since $p_T(x)$ is AF)

Parallel WaveNet

- Parallel WaveNet (DeepMind, ICML 2018)
  - Teacher-Student Learning
    - Pretrain $p_T(x)$ and then train $p_S(x)$ by imitation learning
  - Combining AF & IAF
    - Also other tricks for performances
  - Speedup
    - 20x faster than real-time
    - 1000x faster than WaveNet
    - Google production



Normalizing Flow: Summary

- Normalizing Flow
  - Easy to sample by iterative transforming a simple distribution
  - Invertible transformation for tractable likelihood
    - Enable straightforward MLE learning
  - Critical idea
    - Design non-linear transformation with easy-to-compute Jacobian determinant
- Autoregressive flow
  - Assumption: a factored distribution
    - So each layer $p(x_i|x_{<i})$ results in a simple Jacobian
  - Can be interpreted as a special case of normalizing flow
  - AF v.s. IAF: trade-off between evaluation and sampling

Normalizing Flow: Summary

- Energy-Based Model
  - Flexible density function
  - Arbitrary network structure, allowing (low-dimensional) feature learning
  - Hardest to sample and learn
    - No direct sampling
    - No direct MLE learning due to unknown partition function
- Normalizing Flow
  - Easy to sample by transforming from a simple distribution
  - Most restricted network structure (trade expressiveness for tractability)
    - But autoregressive model can be great ☺ (future lectures)

Normalizing Flow: Summary

- Energy-Based Model
  - Flexible density function
  - Arbitrary network structure, allowing (low-dimensional) feature learning
  - Hardest to sample and learn
    - No direct sampling
    - No direct MLE learning due to unknown partition function
- Normalizing Flow
  - Easy to sample by transforming from a simple distribution
  - Most restricted network structure (trade expressiveness for tractability)
    - **Tractability requirement due to MLE training**
- **Is MLE objective a must? Other learning objectives?**
  - Next two lectures ☺

Thanks