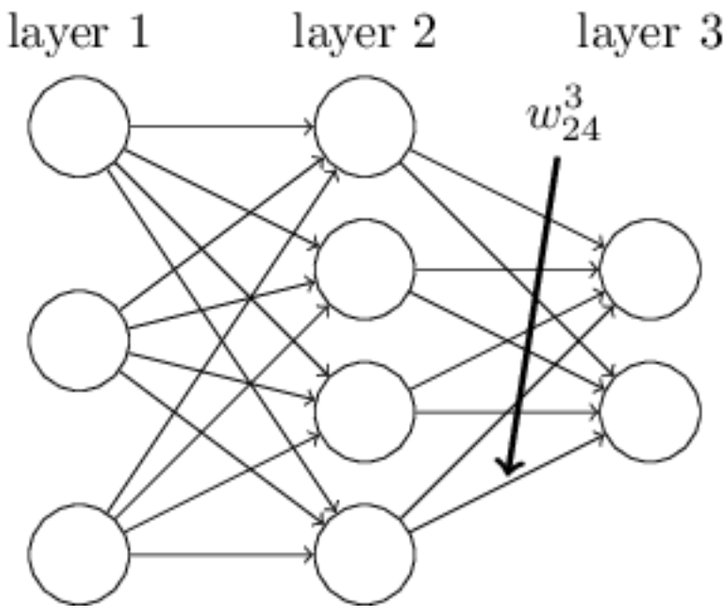
Deep learning primer

The archetype



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

What is deep learning

What can it do

Flexibility
capable of fitting complex functions

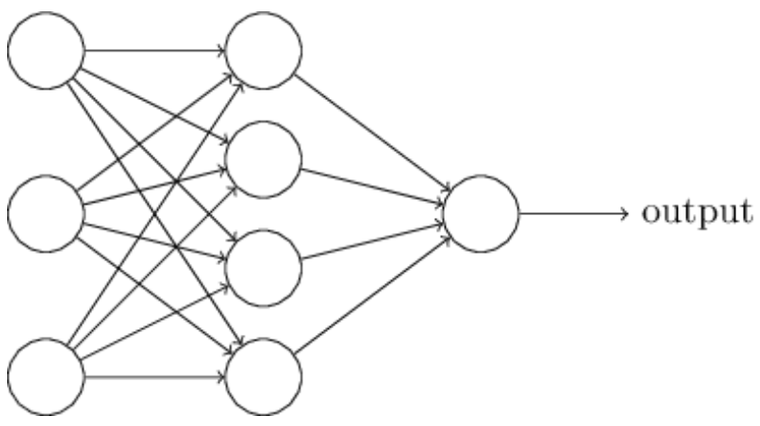
How to train it

Computable gradient
function *largely* smooth

Flexibility

- Universal representation theorem:

Any function in finite dimensions can be approximated arbitrarily well with a two-layer neural network with finite number of hidden unit

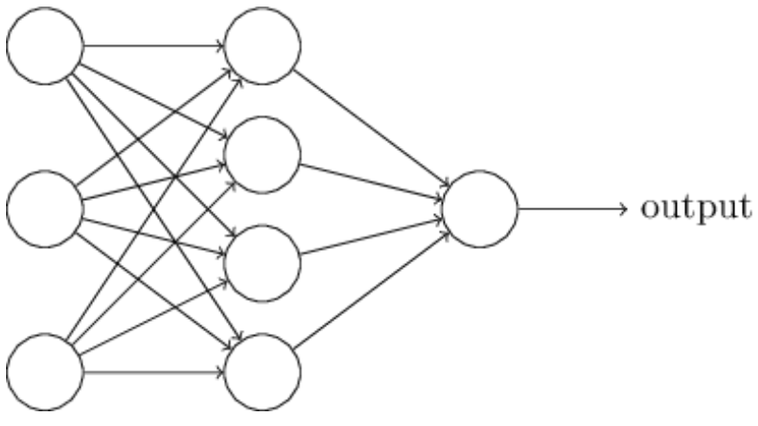


Flexibility

- Universal representation theorem (improved):

There exists a two-layer neural network with ReLU activations and 2n+d weights that can represent any function on a sample of size n in d dimensions.

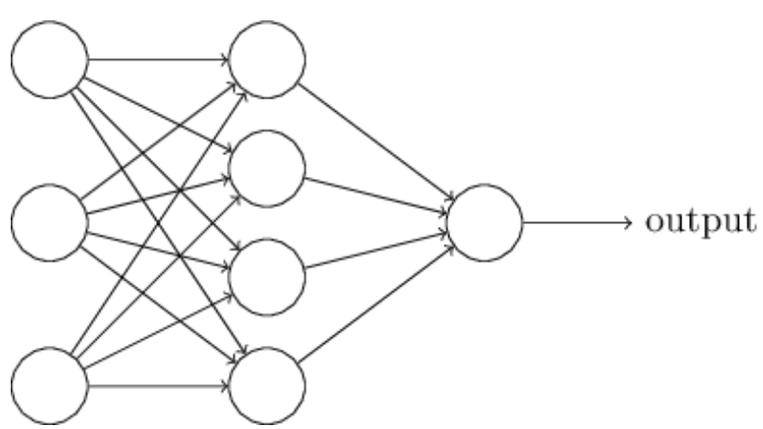
Zhang, et al. 2016, Understanding deep learning requires rethinking generalization



Flexibility

- Depth efficiency hypothesis
  (widely held belief + proof for certain models):

Some functions expressed in multi-layer models requires super-polynomial sized units to express in shallow models



Flexibility

- Flexible model does not generalize?

Rademacher complexity-based generalization bound

$$\mathrm{Rad}_S(F) = \frac{1}{m}\mathbb{E}_\sigma\left[\sup_{f\in F}\left|\sum_{i=1}^{m}\sigma_i f(z_i)\right|\right]$$

$$L_P(h) - L_S(h) \leq 2\,\mathrm{Rad}(F\circ S) + 4\sqrt{\frac{2\ln(4/\delta)}{m}}$$

with probability at least 1- sigma

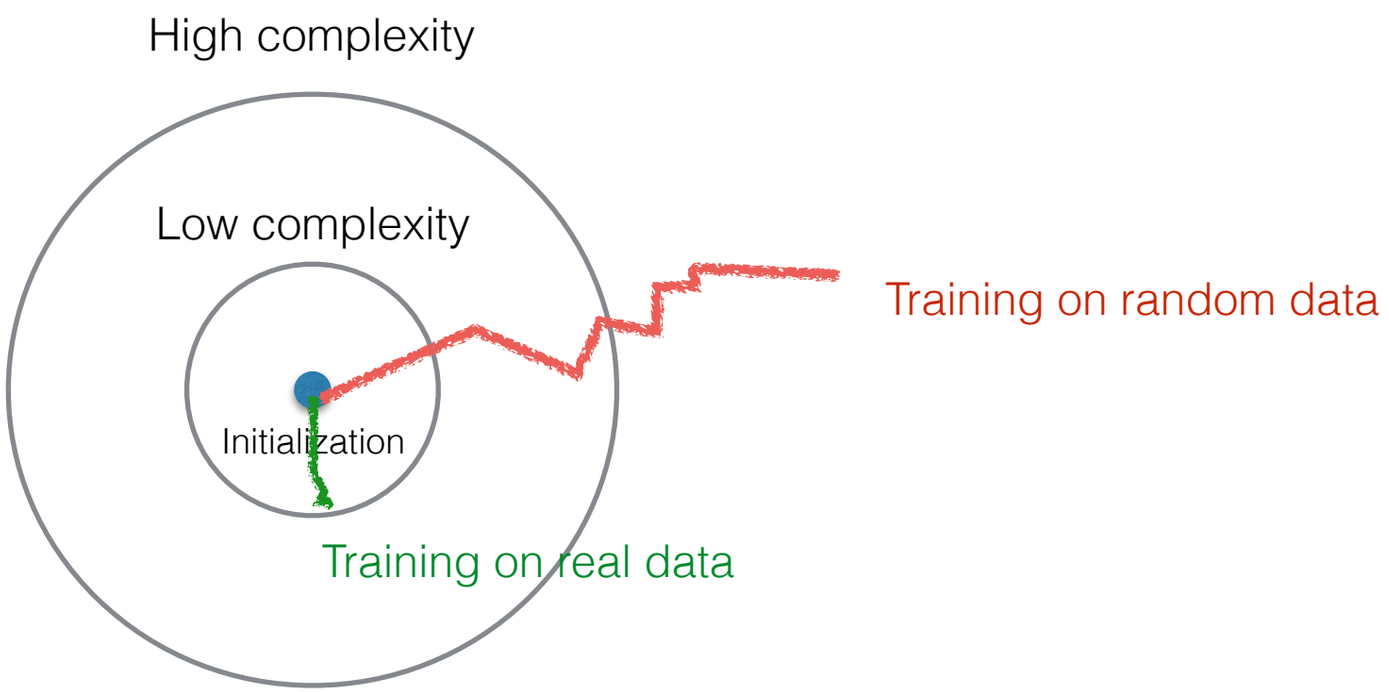
Fun fact: neural network usually has the capacity to memorize random labels perfectly

Zhang et al. Understanding deep learning requires rethinking generalization.

Flexibility

- Flexible model does not generalize?

In practice, models are never trained to obtain the minimal training loss



Notion of generalization based on the ‘length’ of training path?

Gradient

- Implicit assumption is that deep learning models can be learned by simply gradient descent

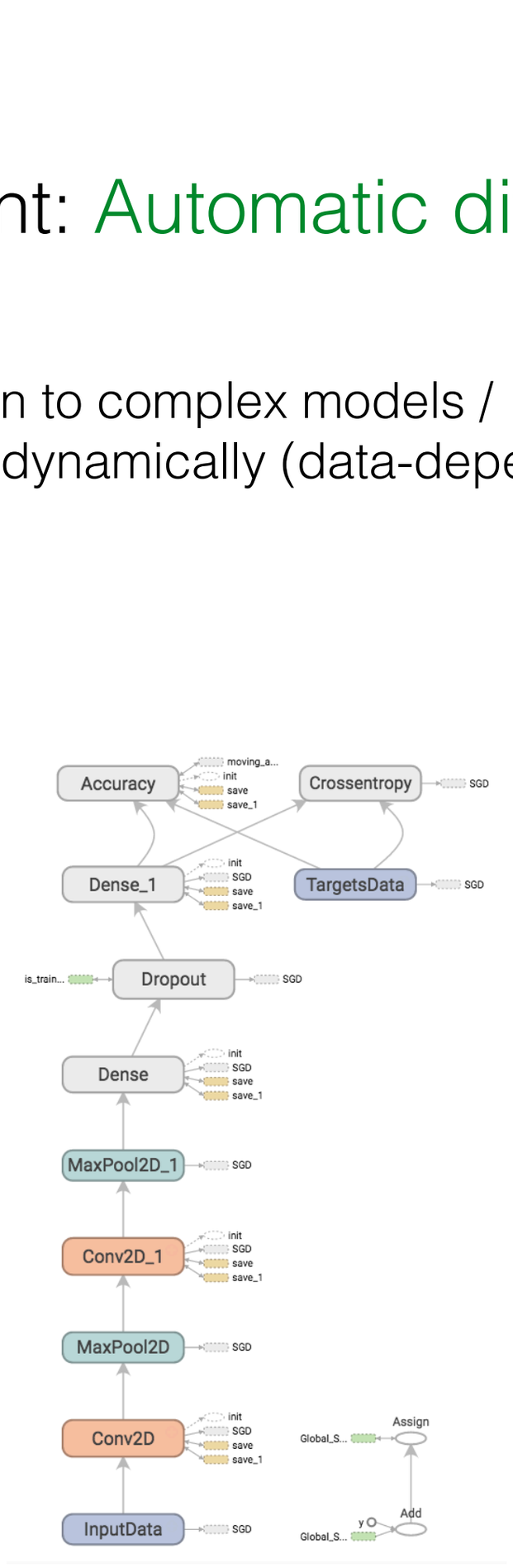
It will be interesting to know theoretically when this assumption fails
(e.g. I guess, prime factorization)

Computation of Gradient: Automatic differentiation

Allow trivial solution to complex models /
changing model structure dynamically (data-dependent)

- The basics: $\frac{dy}{dx} = \frac{dy}{dw}\frac{dw}{dx}$

- Computational graph:



Computation of Gradient: Automatic differentiation

Allow trivial solution to complex models /
changing model structure dynamically (data-dependent)

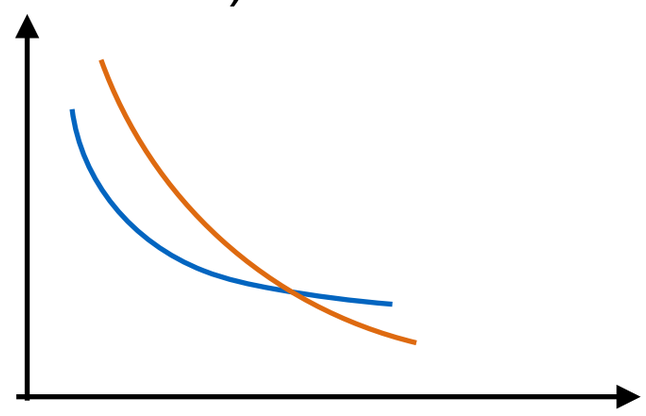
- The basics: $\frac{dy}{dx} = \frac{dy}{dw}\frac{dw}{dx}$
- Two modes: forward mode and backward mode

(optimal traversal path for arbitrary computational graph is NP-complete)

- The future?
  - Compiler for mathematical expressions that achieves acceleration and numeric stability
  - Mixing programing language (conditionals, loops, etc with mathematical functions)
  - Higher-order derivative (e.g. Hessian)

Use gradient efficiently: Stochastic gradient descent

$1/\sqrt{t}$ error rate (stochastic) vs 1/t error rate (batch)



‘High optimization error’ is tolerable:

No need to optimize beyond the statistical limits

Is SGD adaptive to the data uncertainty?

Connection between Stochastic Gradient Descent and Bayesian inference

SGD as MCMC

Stochastic gradient Langevin dynamics, Welling and Teh, 2011

SGD

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta_t)\right)$$

MCMC by Stochastic gradient Langevin dynamics

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta_t)\right) + \eta_t$$

$$\eta_t \sim N(0, \epsilon_t) \tag{4}$$

MCMC by Langevin dynamics

$$\Delta\theta_t = \frac{\epsilon}{2}\left(\nabla \log p(\theta_t) + \sum_{i=1}^{N} \nabla \log p(x_i|\theta_t)\right) + \eta_t$$

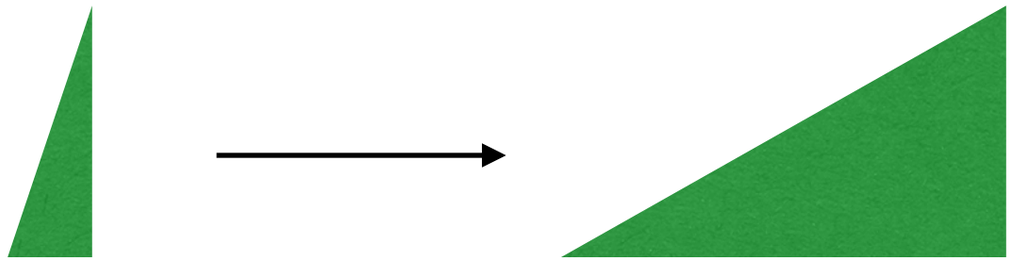
$$\eta_t \sim N(0, \epsilon) \tag{3}$$

SGD as VI

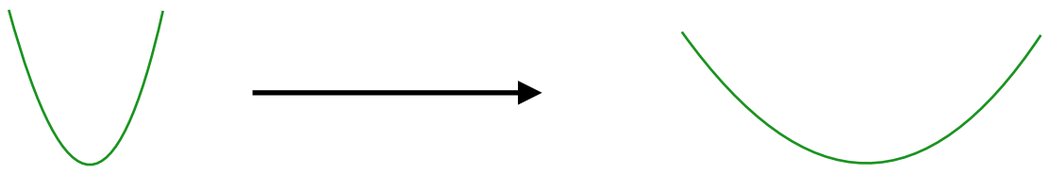
Stochastic Gradient Descent as Approximate Bayesian Inference, Mandt, 2017

Optimization: scale invariance

Naive gradient descent is not scale-invariant

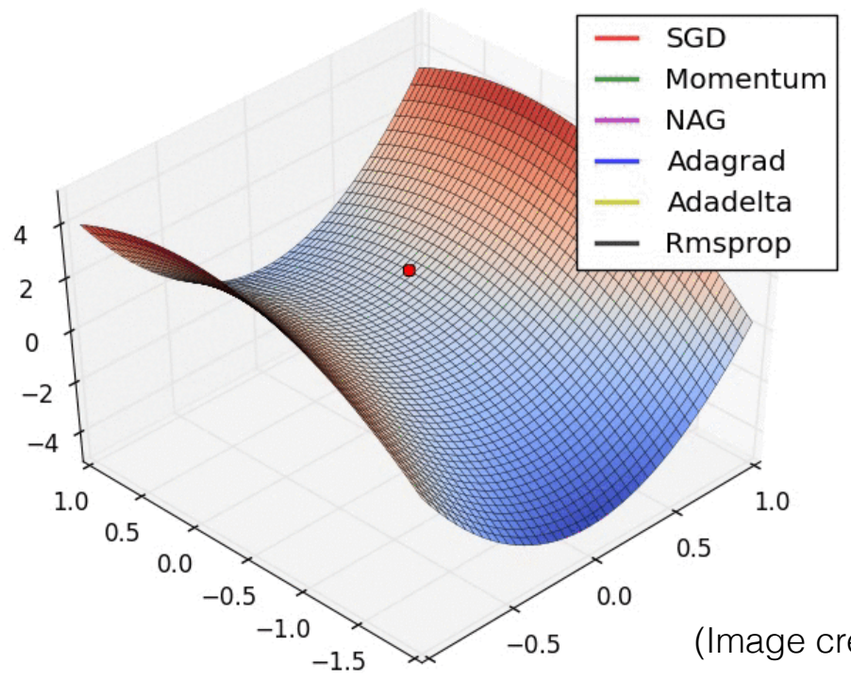


Known solution: use curvature of the surface (second order methods)



The exact way: compute Hessian matrix (second order derivatives) / Newton's method

The cheap way : approximation using the history of gradients



(Image credit: Alec Radford)

Optimization: variance reduction and scale invariance

SGD+momentum

$$g_t = 0.9 * g_{t-1} + 0.1 * g$$

RMSprop

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t.$$

Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t.$$

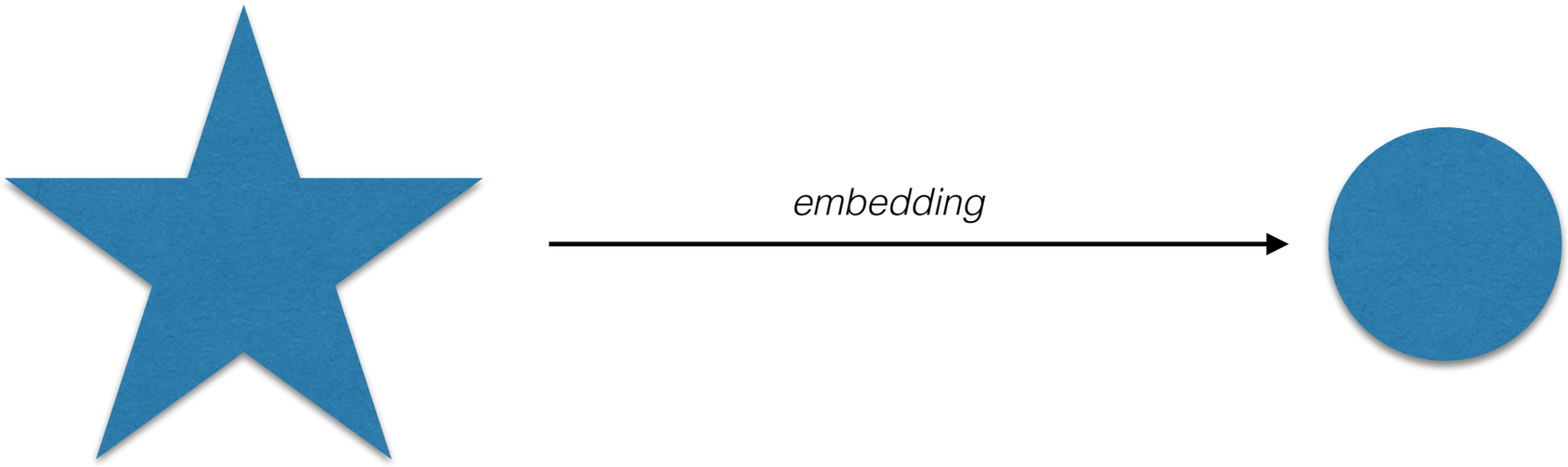
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2.$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

http://sebastianruder.com/optimizing-gradient-descent/

Learning representations

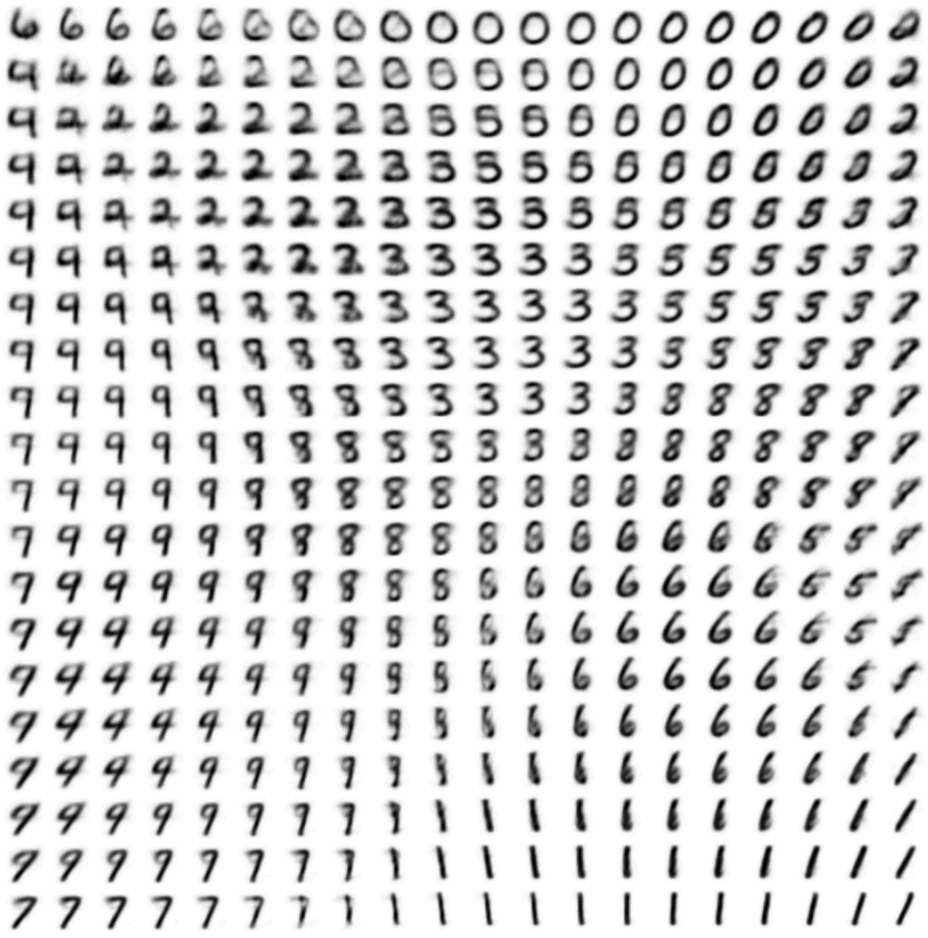
Raw data that lives in some arbitrary (high-dimensional) space



Representation space with
smooth and linear structure

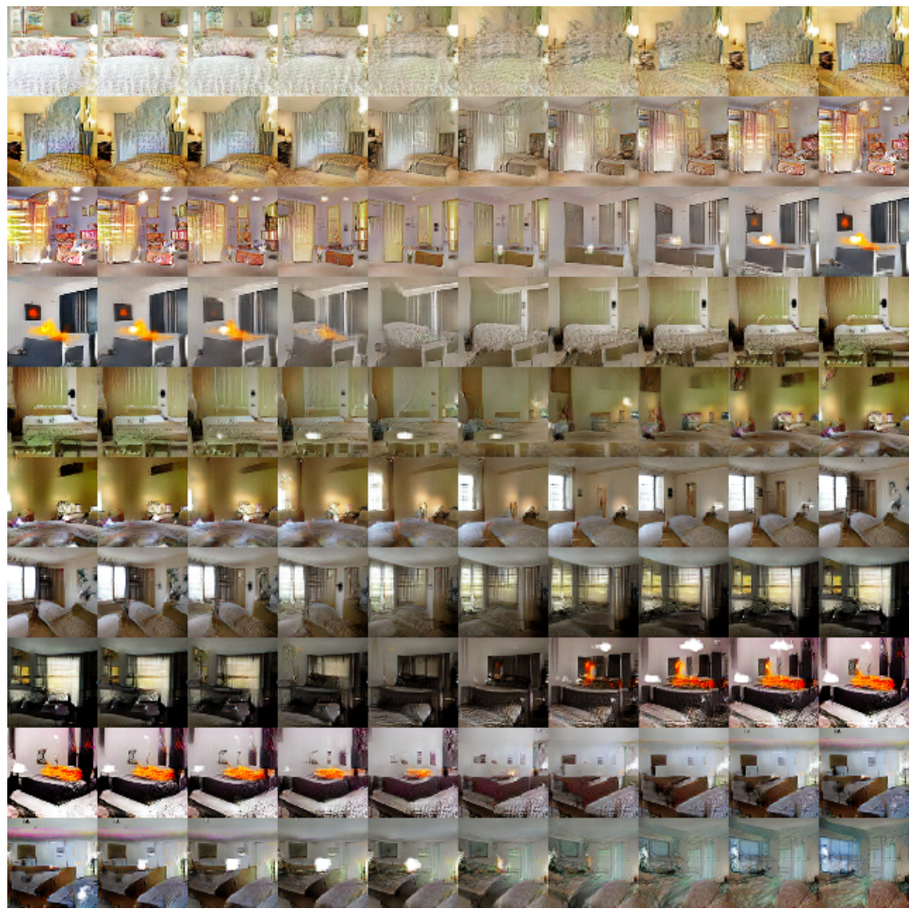
Representation: smoothness

Digits (MNIST)



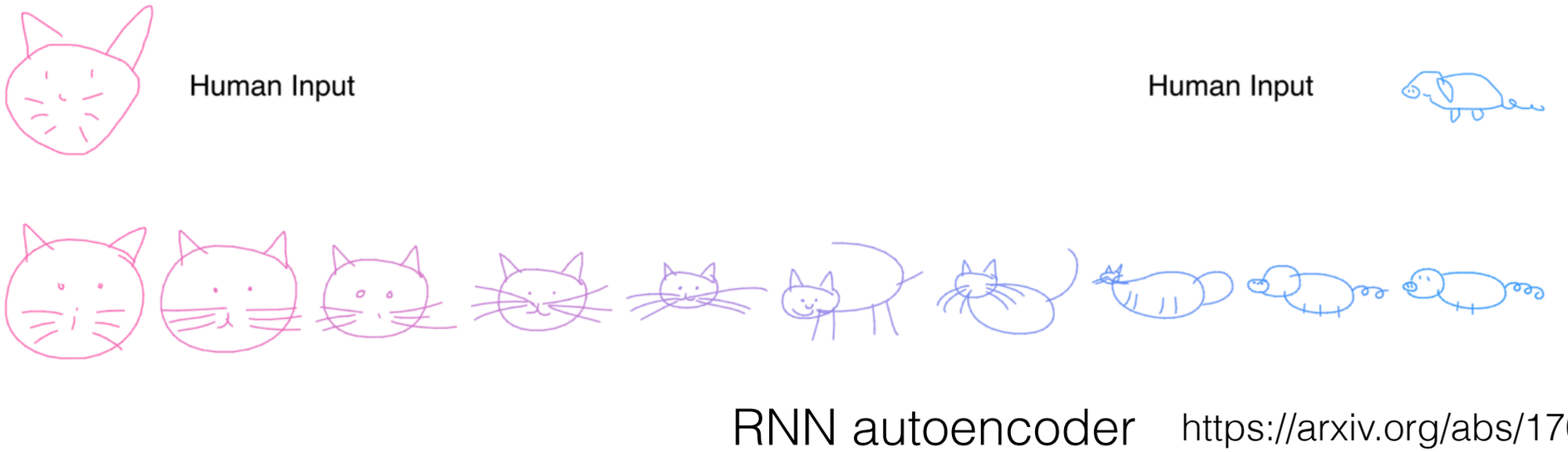
Embedding learned by
variational autoencoder (VAE)

Bedroom (LSUN)

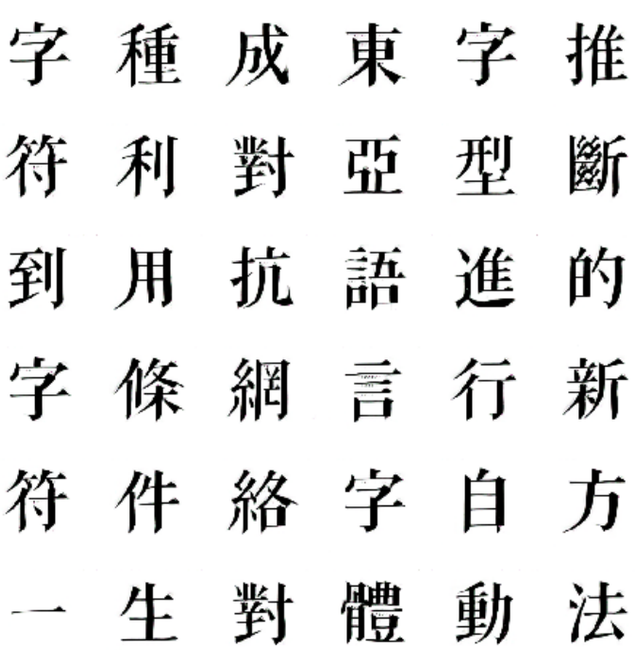


Embedding learned by
generative adversarial networks (GAN)

Representation: smoothness

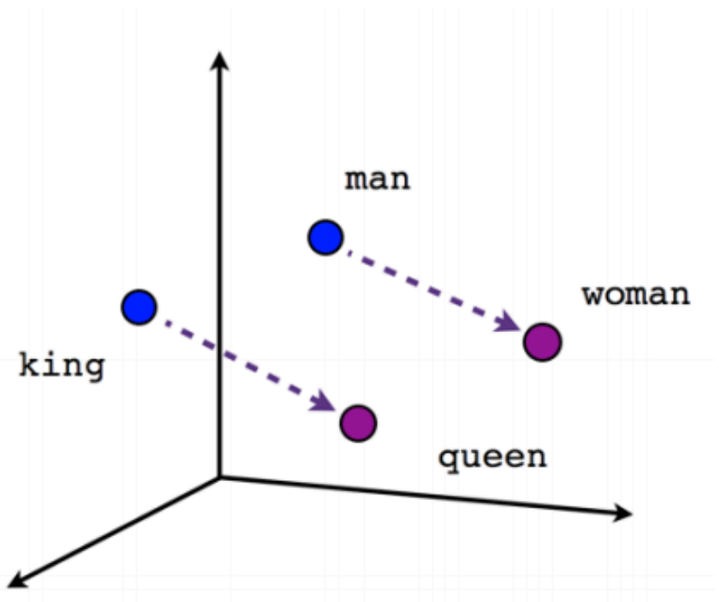


RNN autoencoder https://arxiv.org/abs/1704.03477

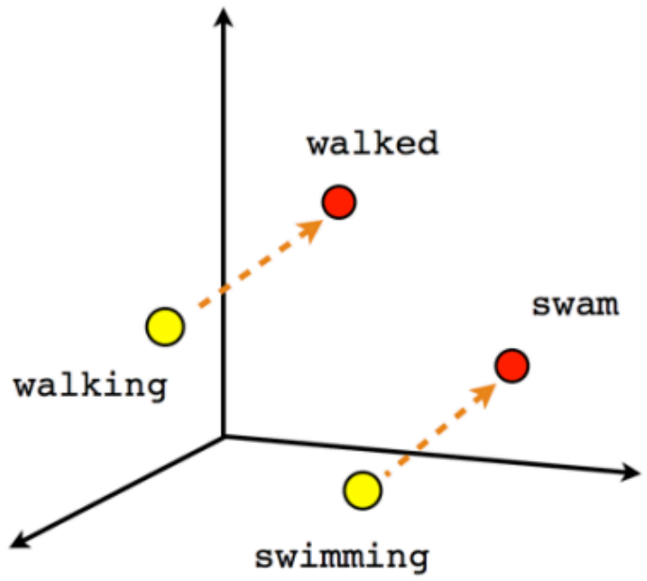


GAN github.com/kaonashi-tyc/zi2zi

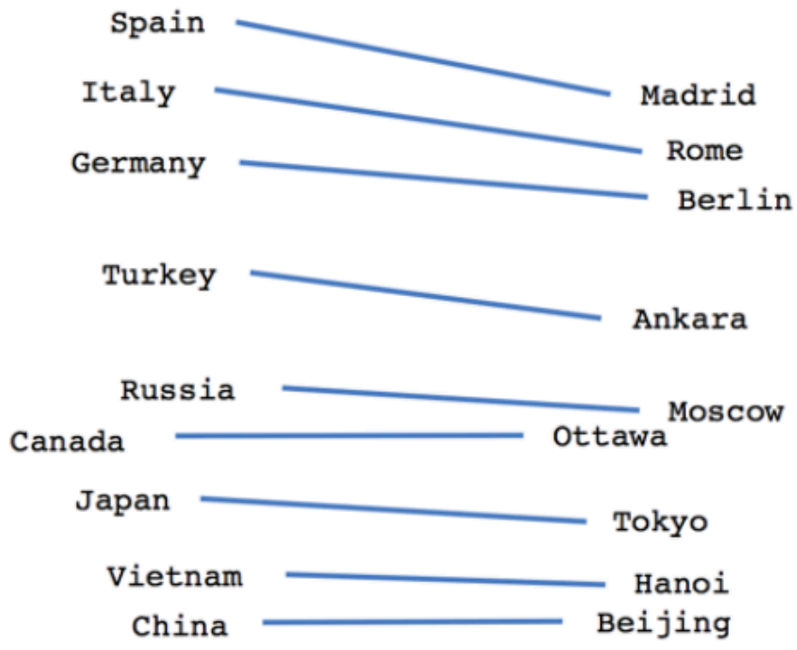
Representation: linearity



Male-Female



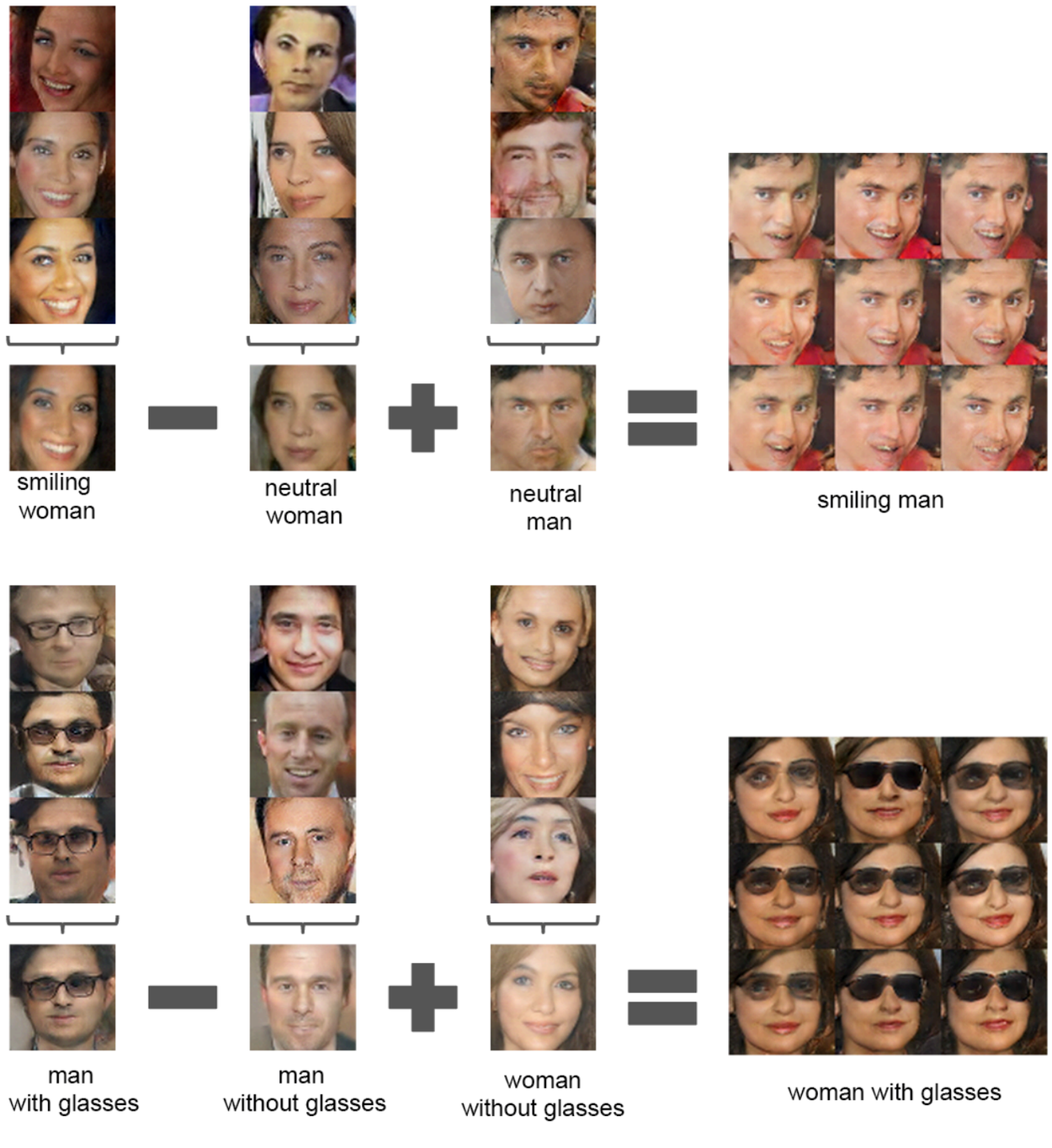
Verb tense



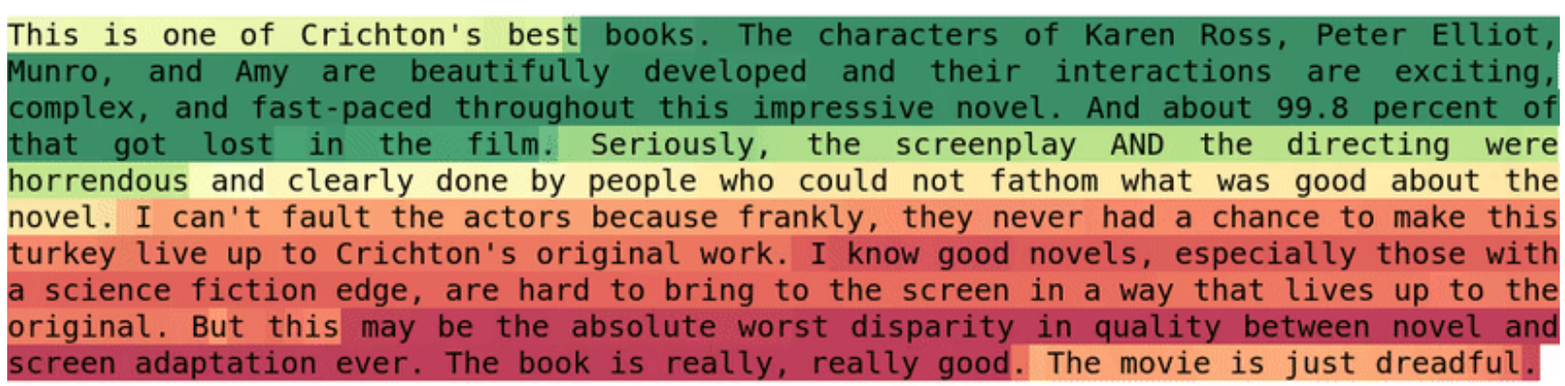
Country-Capital

Pretrained word vectors for >70 languages are publicly available

Representation: linearity



Representation learning



“Sentiment neuron”

Trained on 82 million Amazon reviews to
predict the next character

method: multiplicative LSTM

Reference: https://blog.openai.com/unsupervised-sentiment-neuron/

Practical neural network structure

Convolutional network

- Image recognition
- Image super-resolution / in-painting
- Play Go

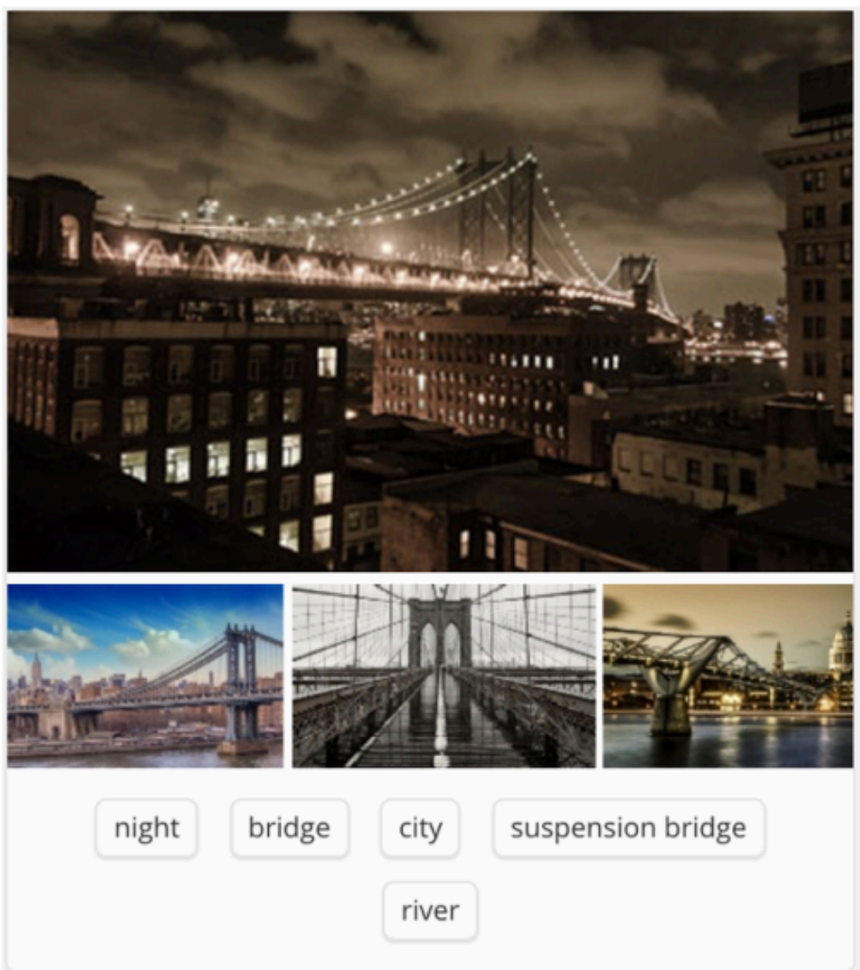
Main application area: CV

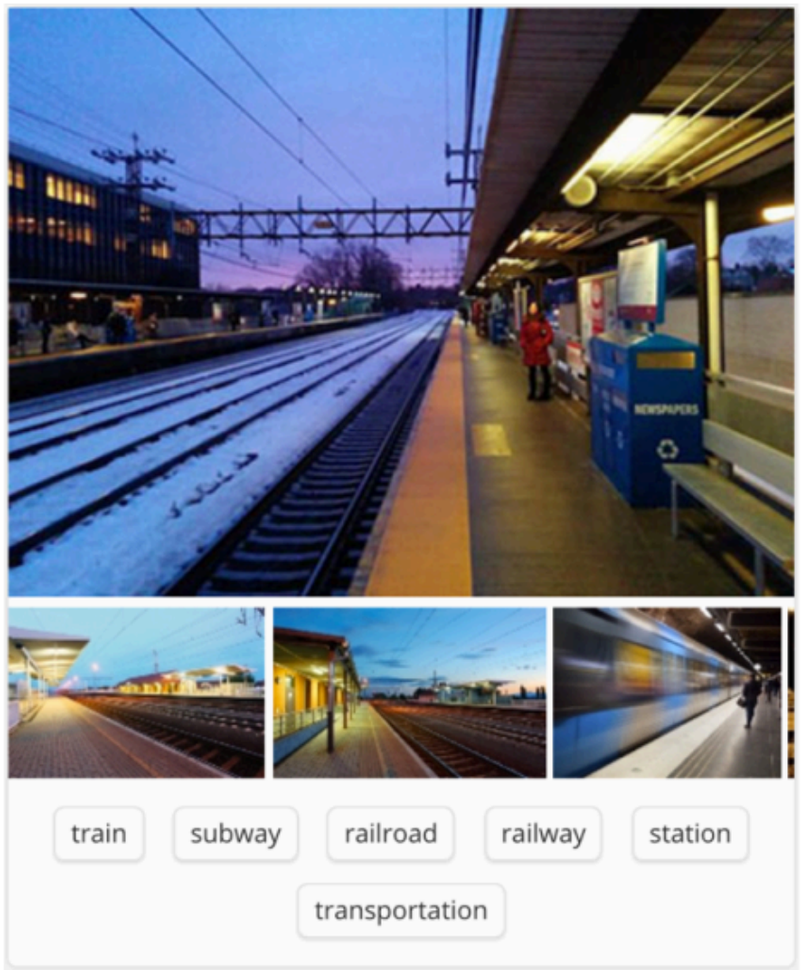
Long Short Term Memory

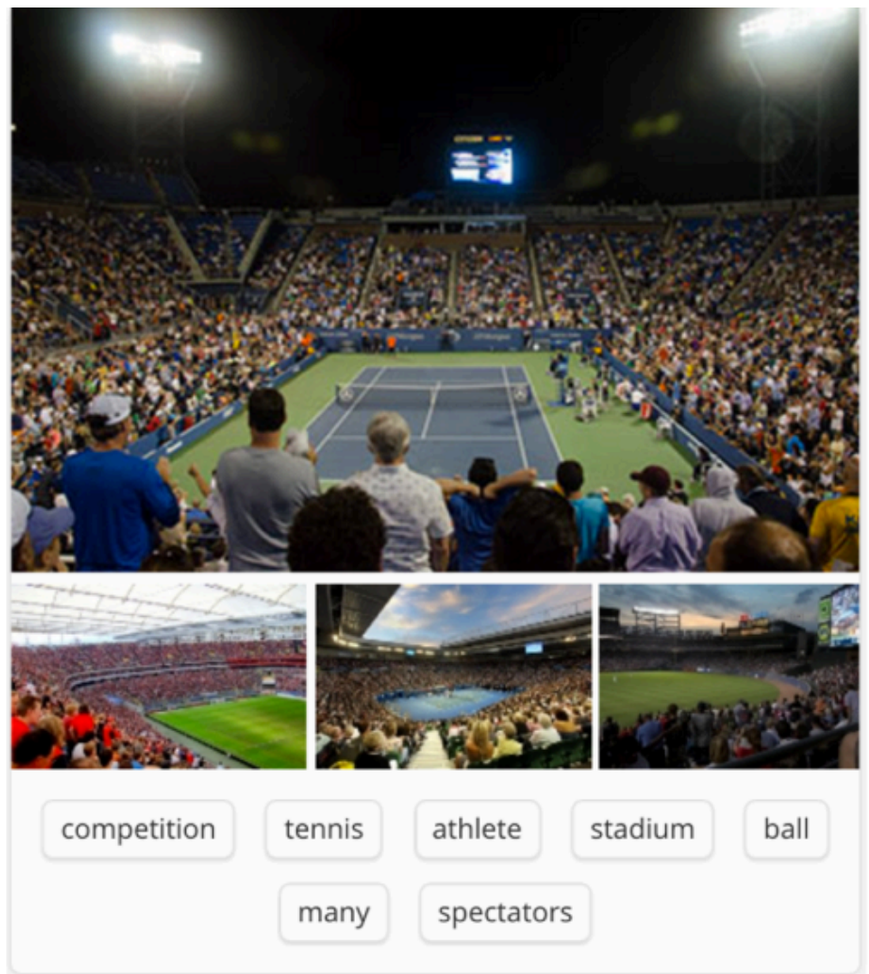
- Machine translation
- Basic question answering
- Text understanding e.g. sentiment analysis

Main application area: NLP

Convolutional network / Convnet





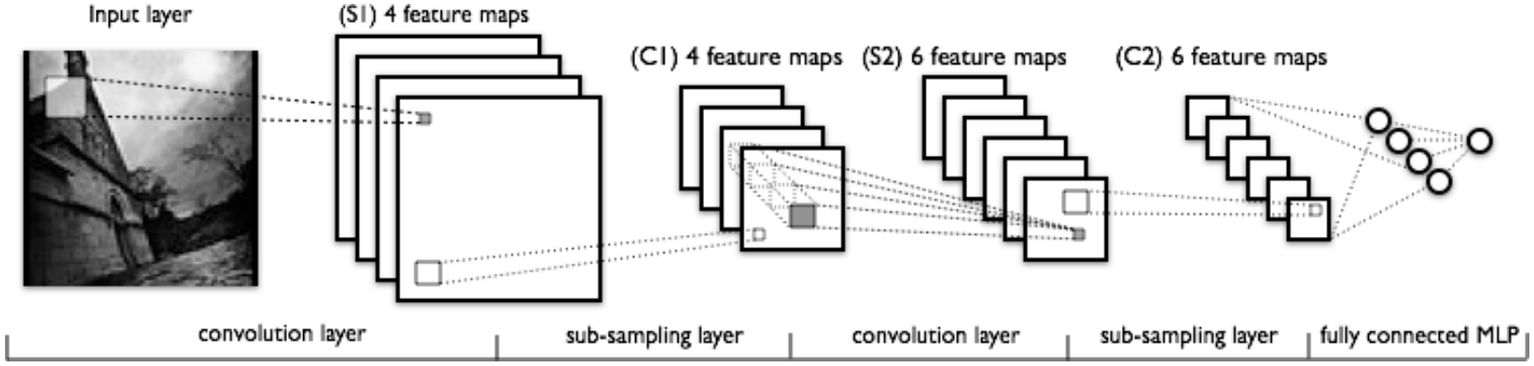


Human level image recognition

https://www.clarifai.com/

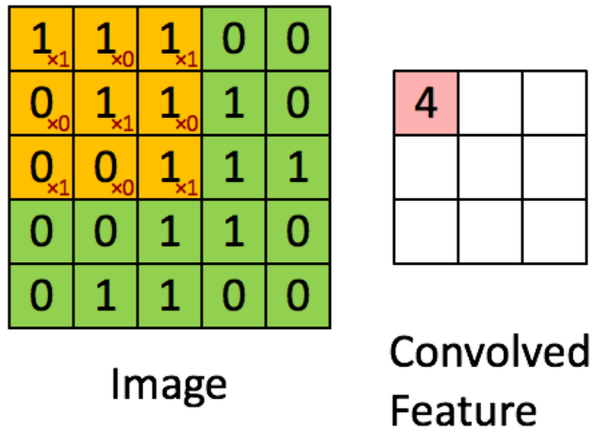
Convolutional network / Convnet

Traditional architecture



The essential part

Convolution
(Spatial weight sharing)



The recipe that won ImageNet 2012 competition
(1.3M images in 1000 categories):

ConvNet + ReLU + dropout + GPU

16.4% error vs 26.1% 2nd place

Neural style transfer

Pretrained convnet
(e.g. Imagenet classifier)

Objective 1:

Minimize distance of
high level representations
to the content image

Objective 2:

Minimize distance of
correlation matrix of
low level representations
to the style image

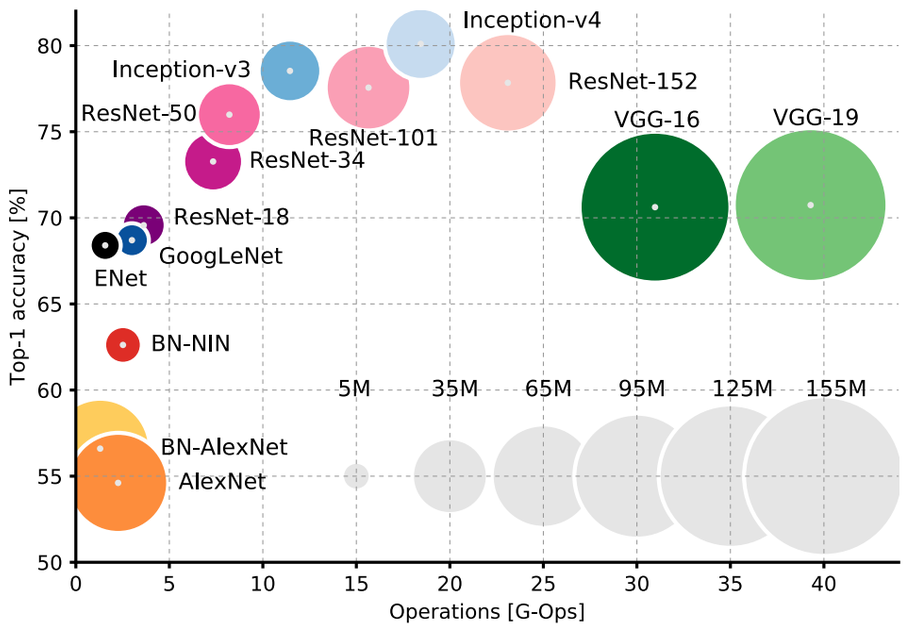
Scene from 'Visit to Picasso', a documentary by Paul Haesaert

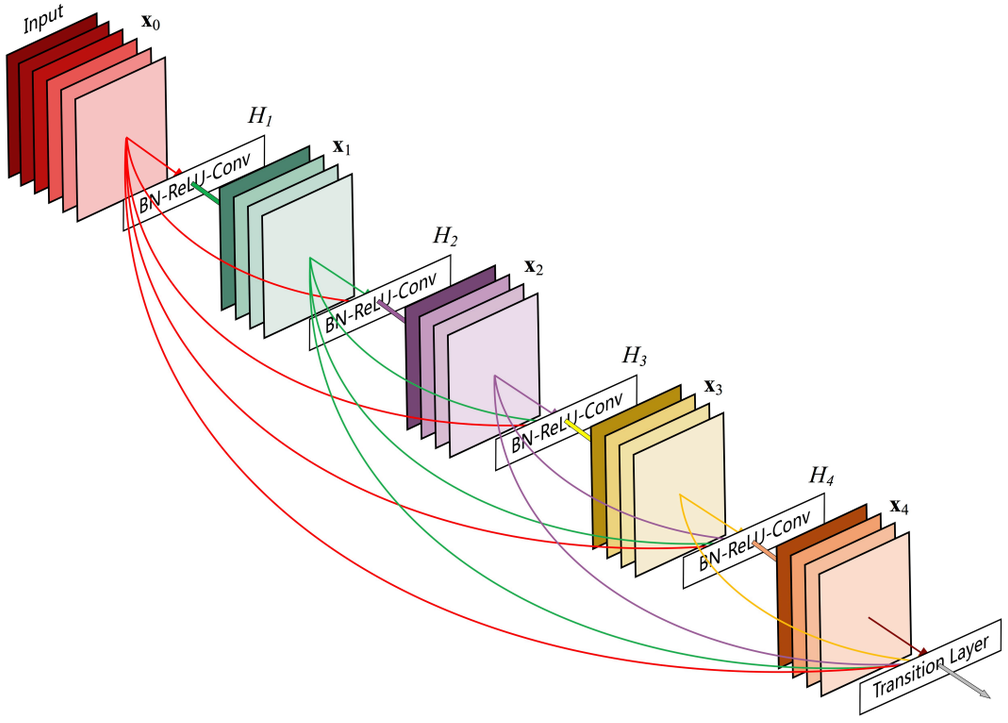
Photo style transfer: Improving photorealism

Deep Photo Style Transfer: arXiv:1703.07511

additional regularization term to encourage transformation to be locally affine

State-of-the-art Computer Vision ConvNet at 2017





Huang et al. Densely Connected Convolutional Networks

Batch normalization layer:
standardization of representations
(Loffe and Szegedy, 2015)

'Shortcut' connections:
address vanishing gradients

Reduce number of weights:
Use exclusively 1x1 convolution and 3x3 convolution.
More layers.
No fully connected layer

Convolution + Pooling is a general technique for enforcing **invariance** in representations

Can be extended to introduce translation, rotation, or scaling invariance etc.

Mathematical perspective: invariant transformations as symmetry groups

Cohen and Welling, 2016 Group Equivariant Convolutional Networks

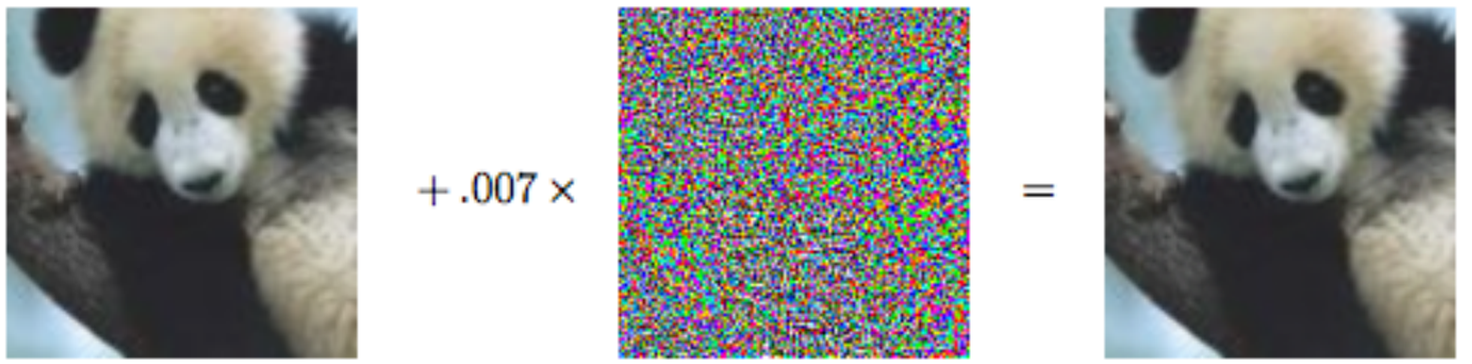
Mallat, 2012 Group Invariant Scattering

Computational challenge: how to compute efficiently?

Possible transformations grow multiplicatively if we stack invariances

Stochastic approximation (one random transformation at a time)?

Convnet can be easily tricked



$\boldsymbol{x}$

"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

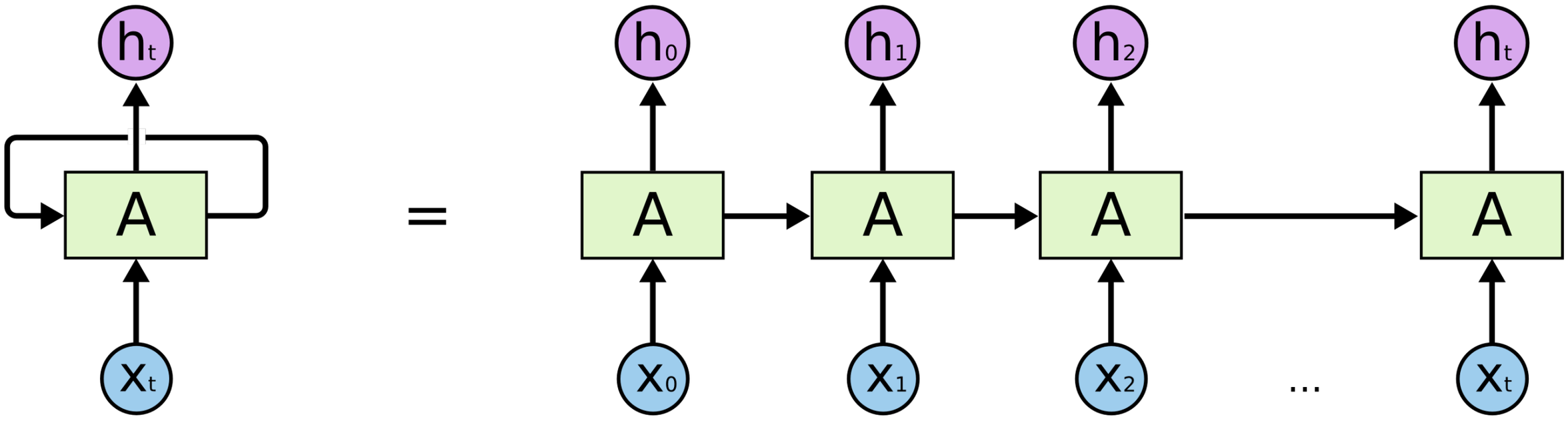
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

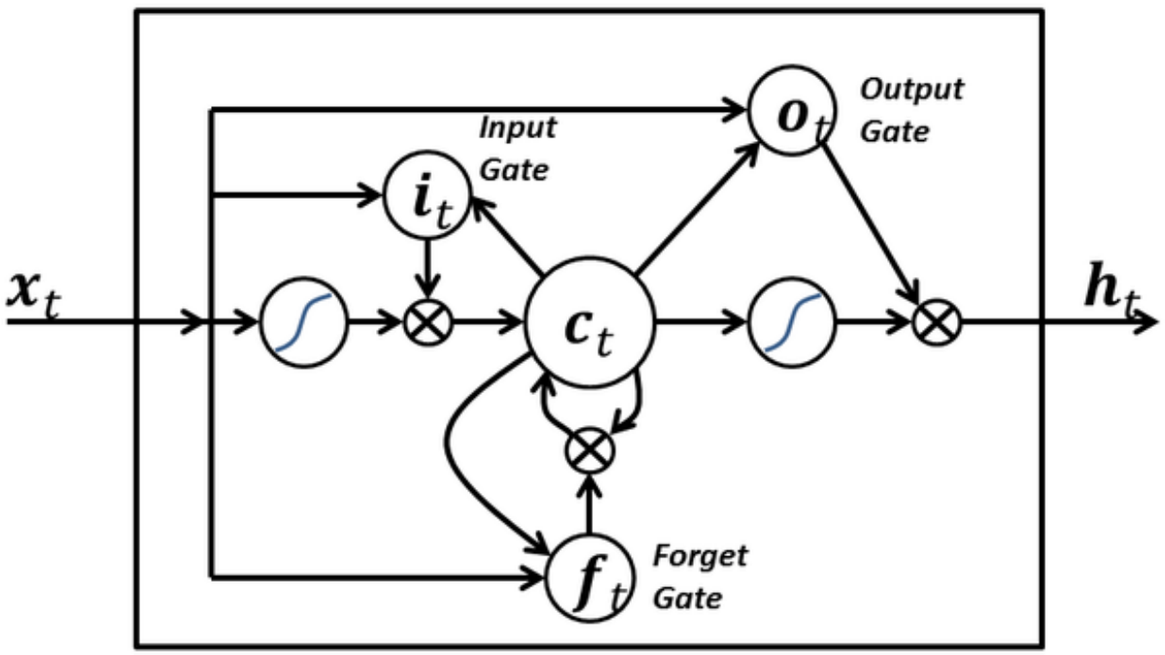
So do most high dimensional models

LSTM - recurrent neural networks



Long Short-term Memory (LSTM) network - Gate

Standard structure



The essential part

Forget gate

Greff et al., 2015, LSTM: A Search Space Odyssey

Forget gate $f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$

Input gate $i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$

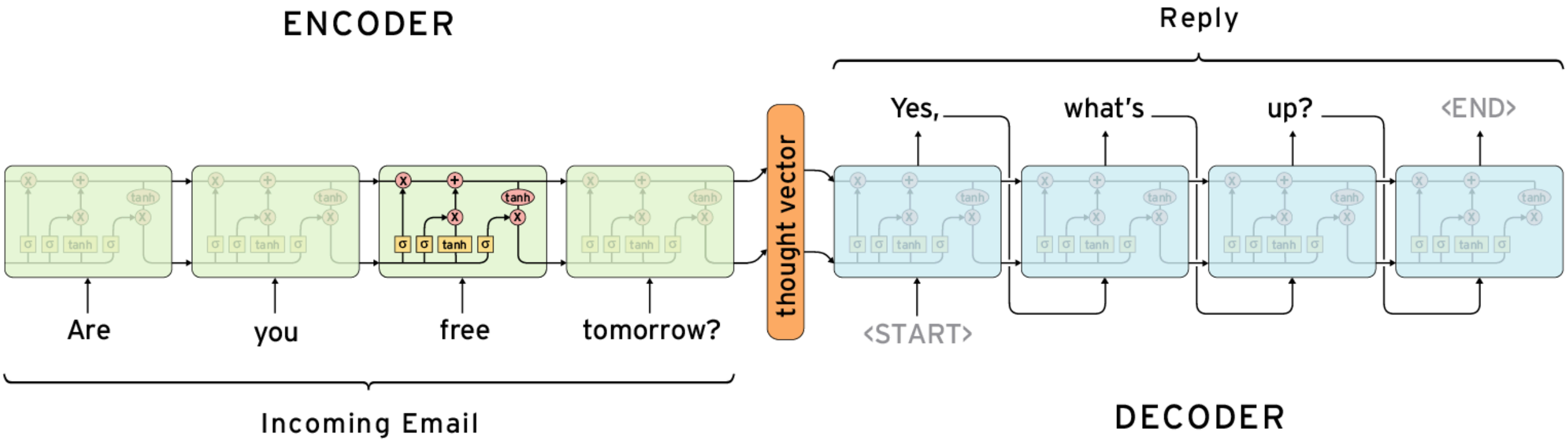
Output gate $o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$

Memory cell $c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$

Hidden units $h_t = o_t \circ \sigma_h(c_t)$

from: wikipedia

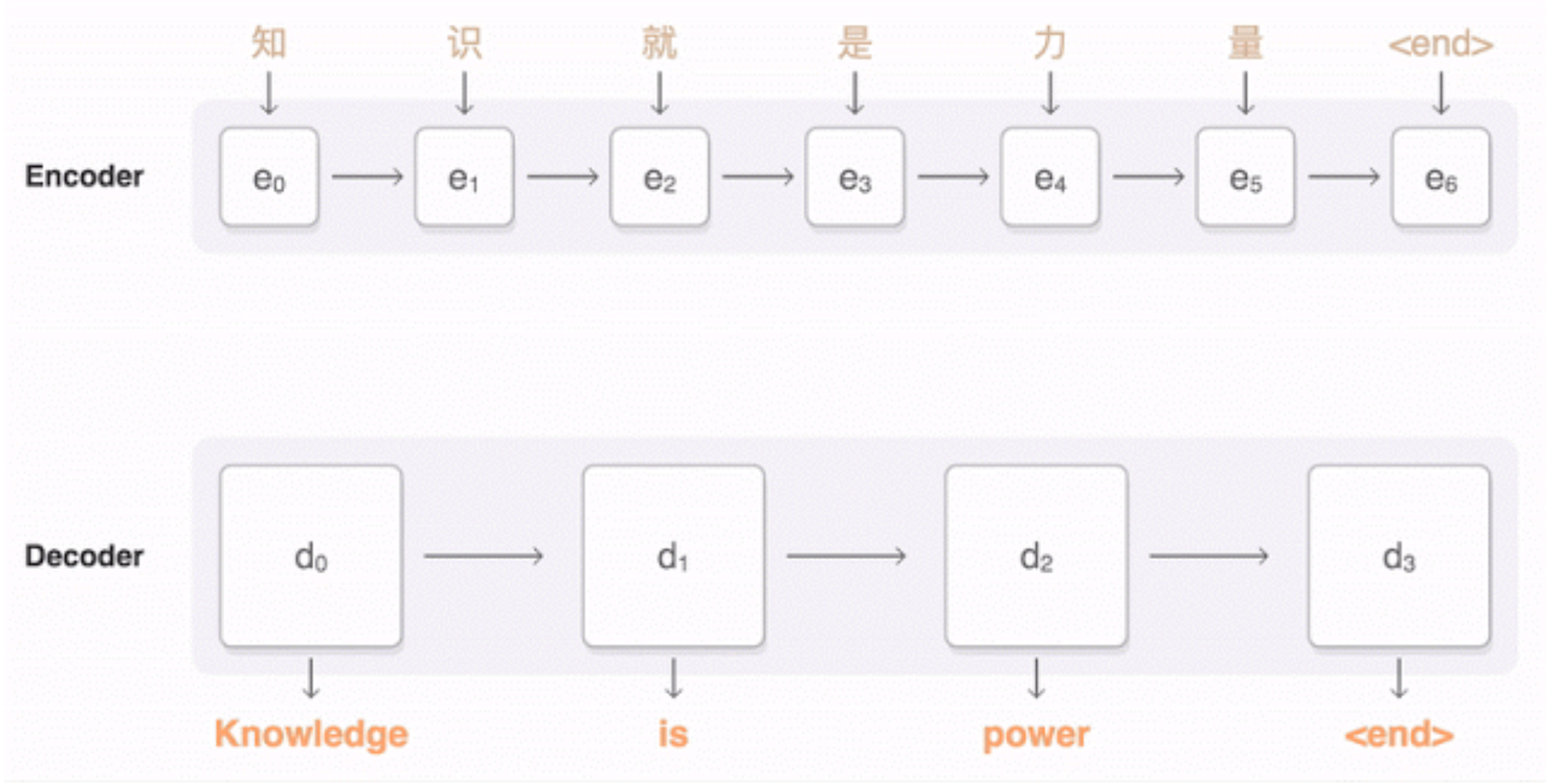
Sequence to sequence (Seq2Seq)



Question-answering
Machine translation
…

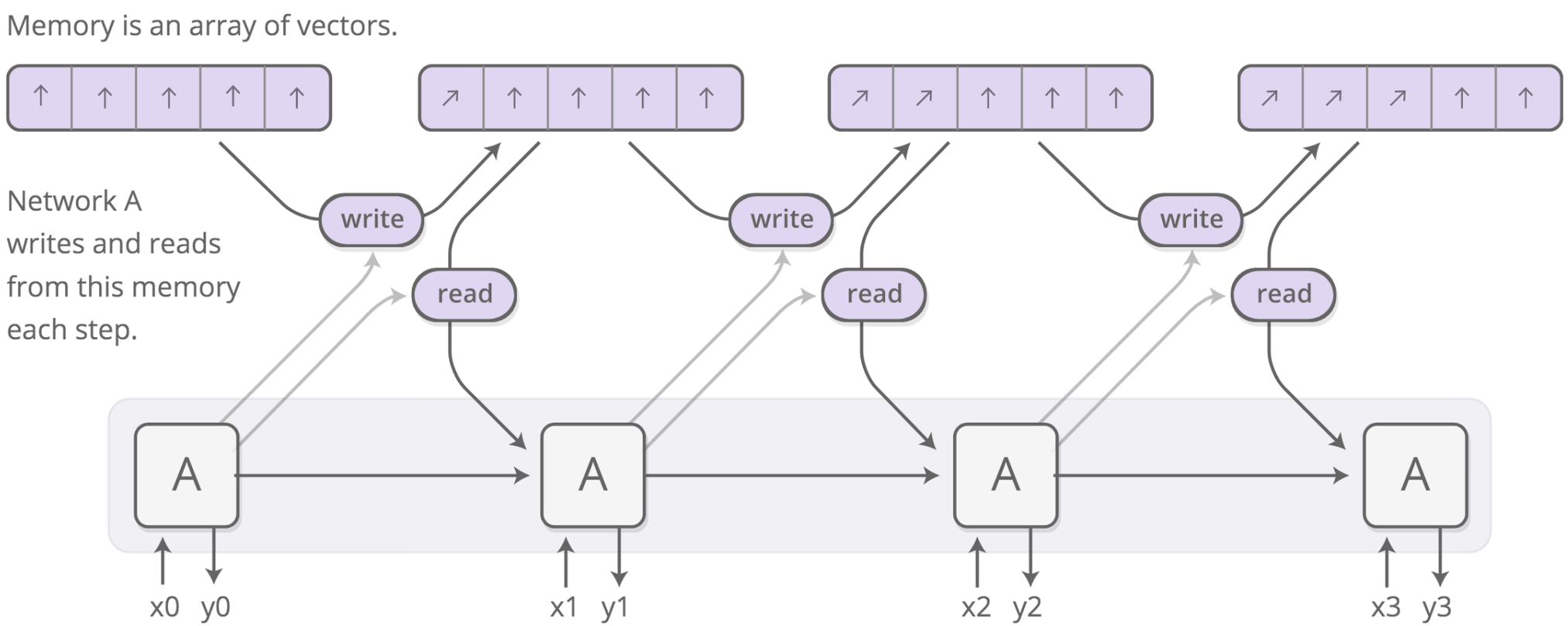
http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/

Seq2Seq + Attention



Attention - dynamic weighting of the input

Neural Turing machine / Differentiable Neural Computer



http://distill.pub/2016/augmented-rnns/

Graves et al. 2014, Neural Turing machines

Reinforcement learning

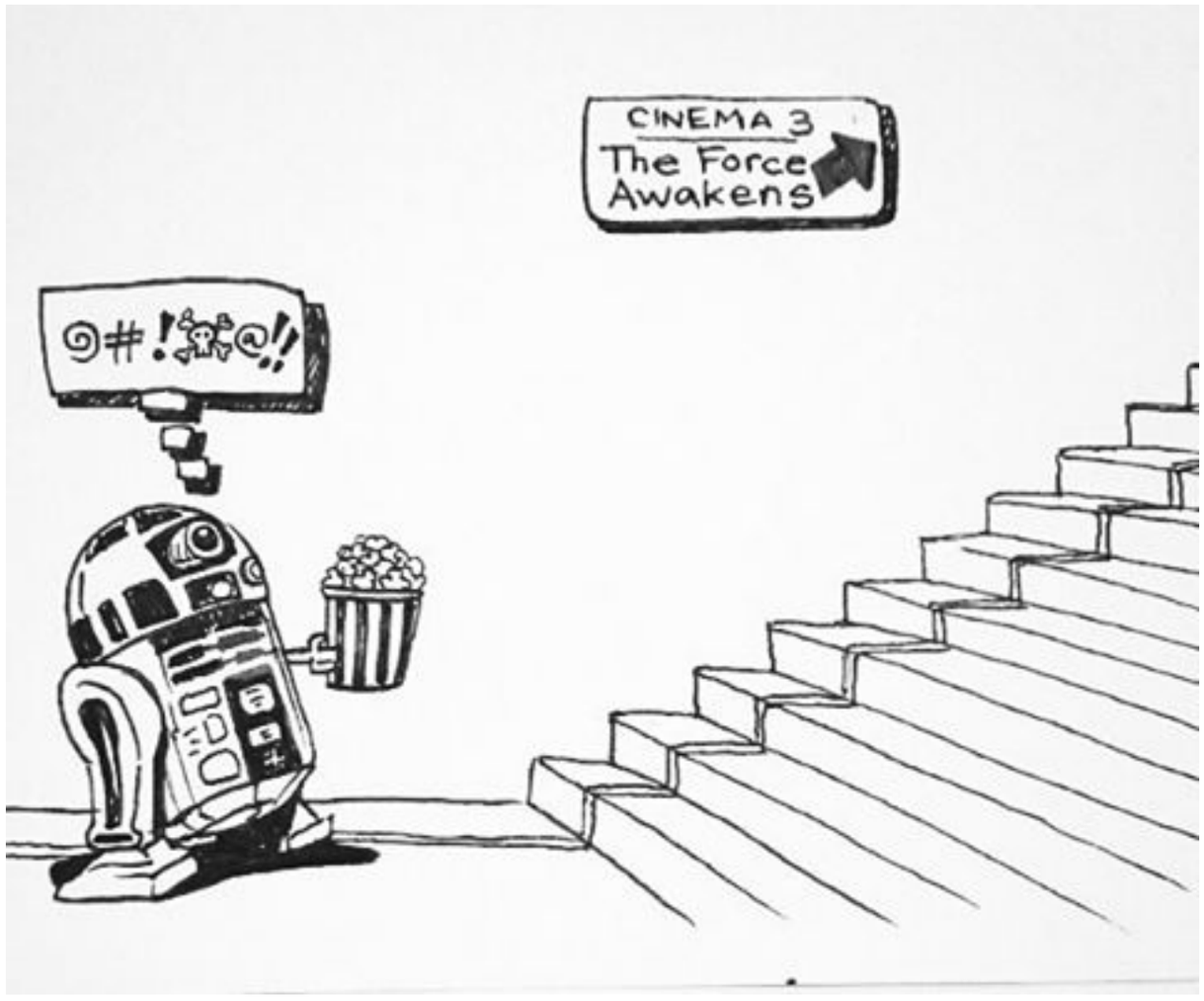
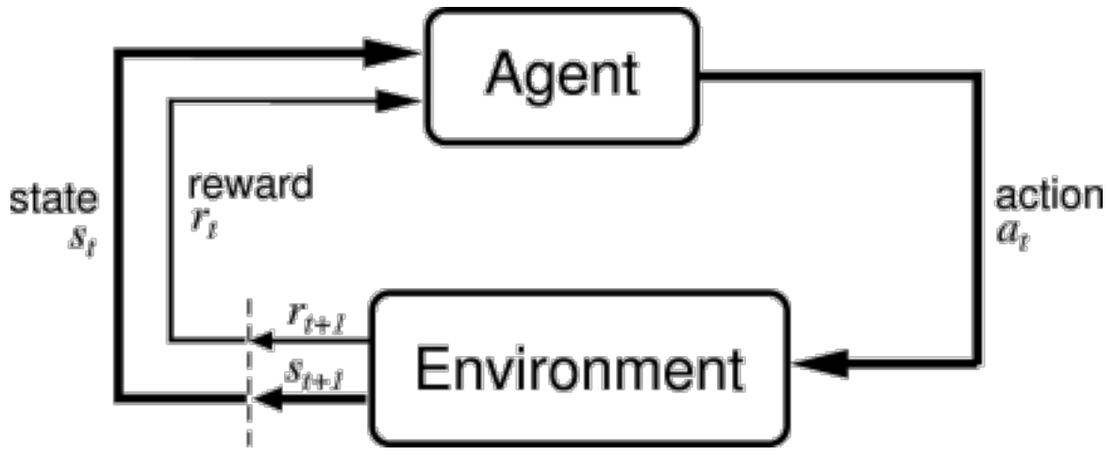


Image credit: daily.doodl @ instagram



Given state, choose action, get reward

Deep Q learning:
Predict future rewards with deep networks

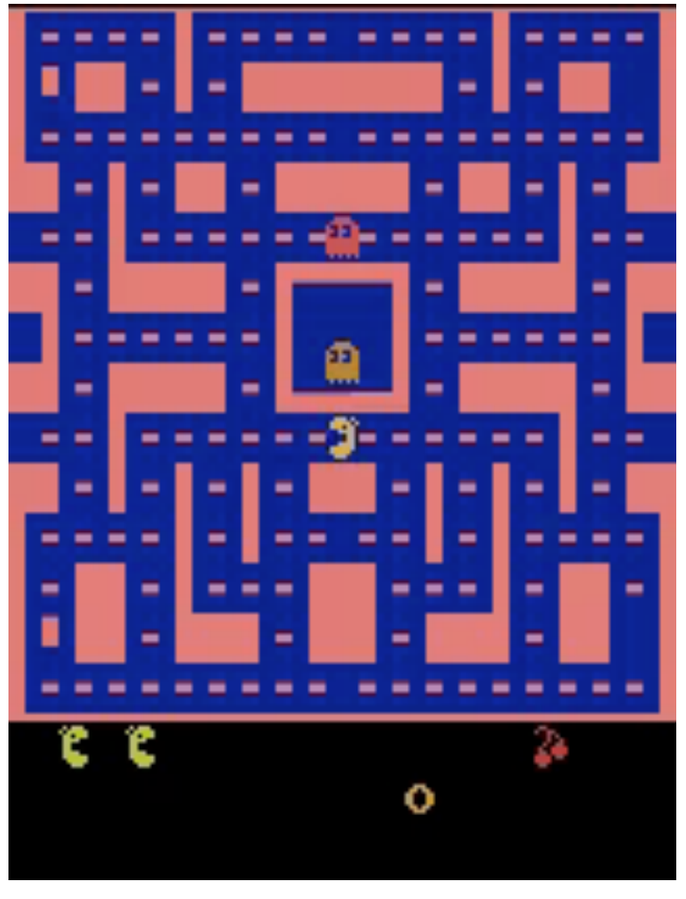
Q Learning

Q(state, action) = maximal future rewards (with the optimal actions)

Bellman equation

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

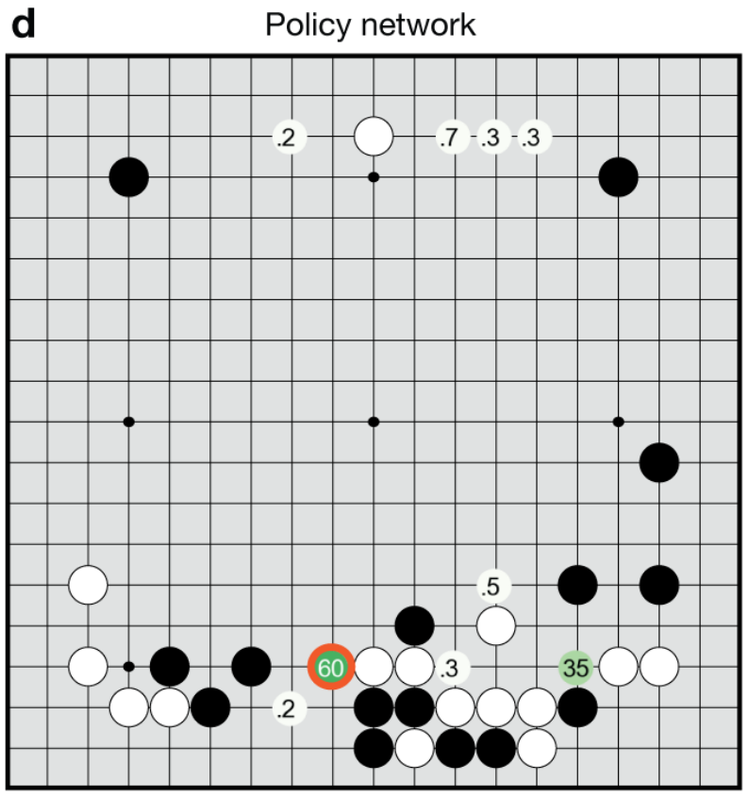
Training: minimize MSE

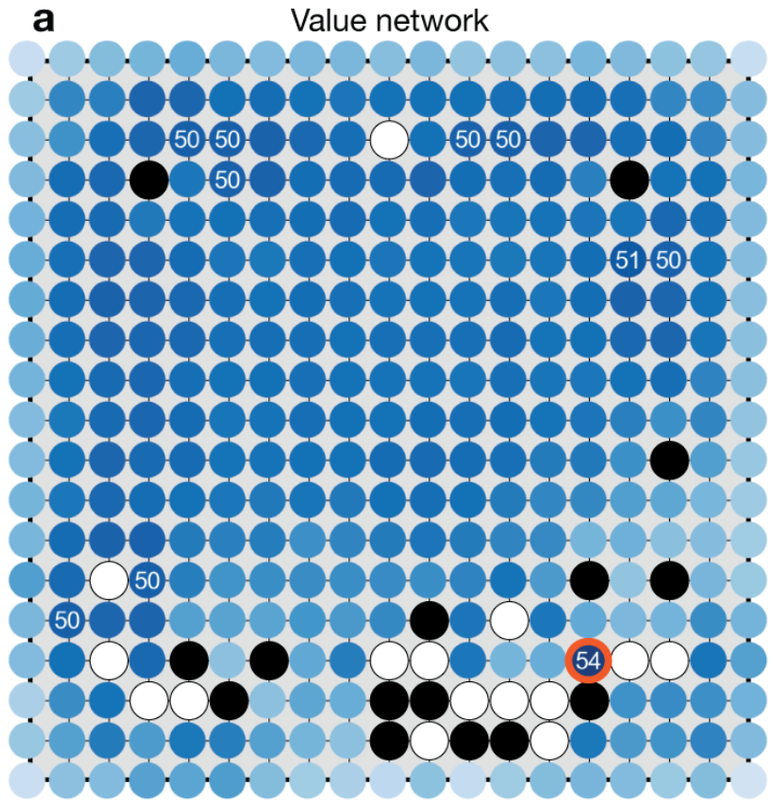


Minh et al, 2013 Playing Atari with Deep Reinforcement Learning

AlphaGo - surpass human-level game playing in Go

(the nature publication version)





SL policy network: predict expert human moves

convnet / GLM

RL policy network: optimized by self-play

convnet

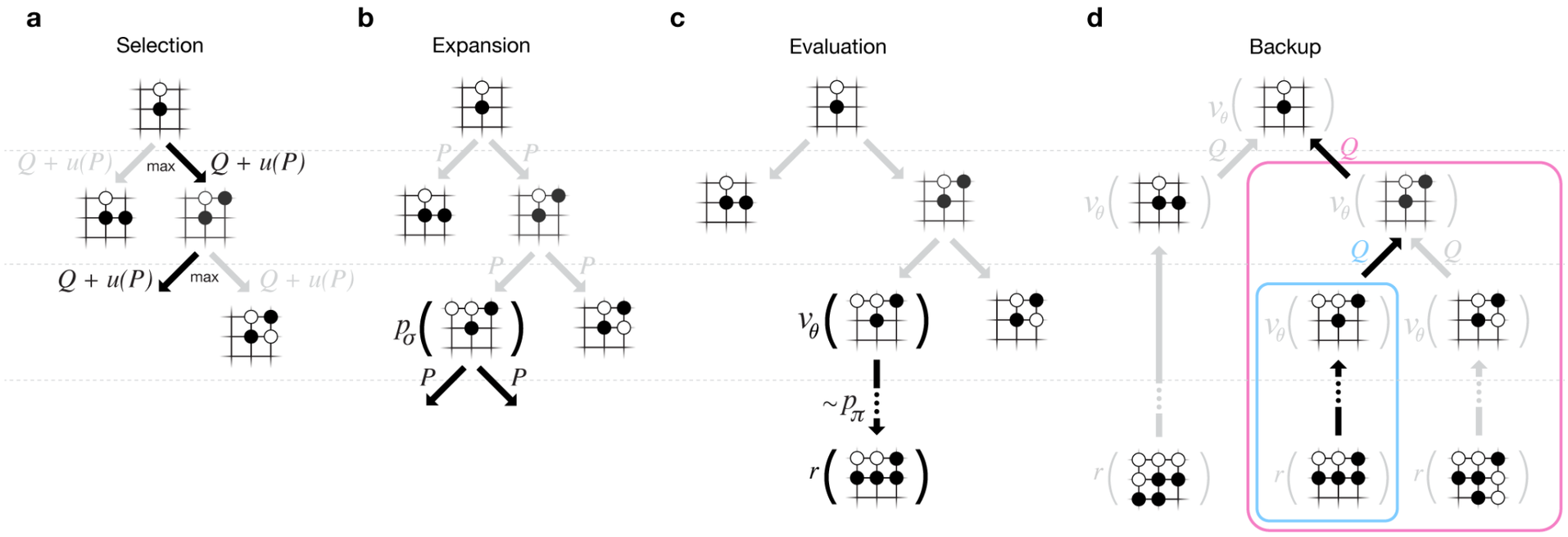
REINFORCE algorithm (Williams, 1992)

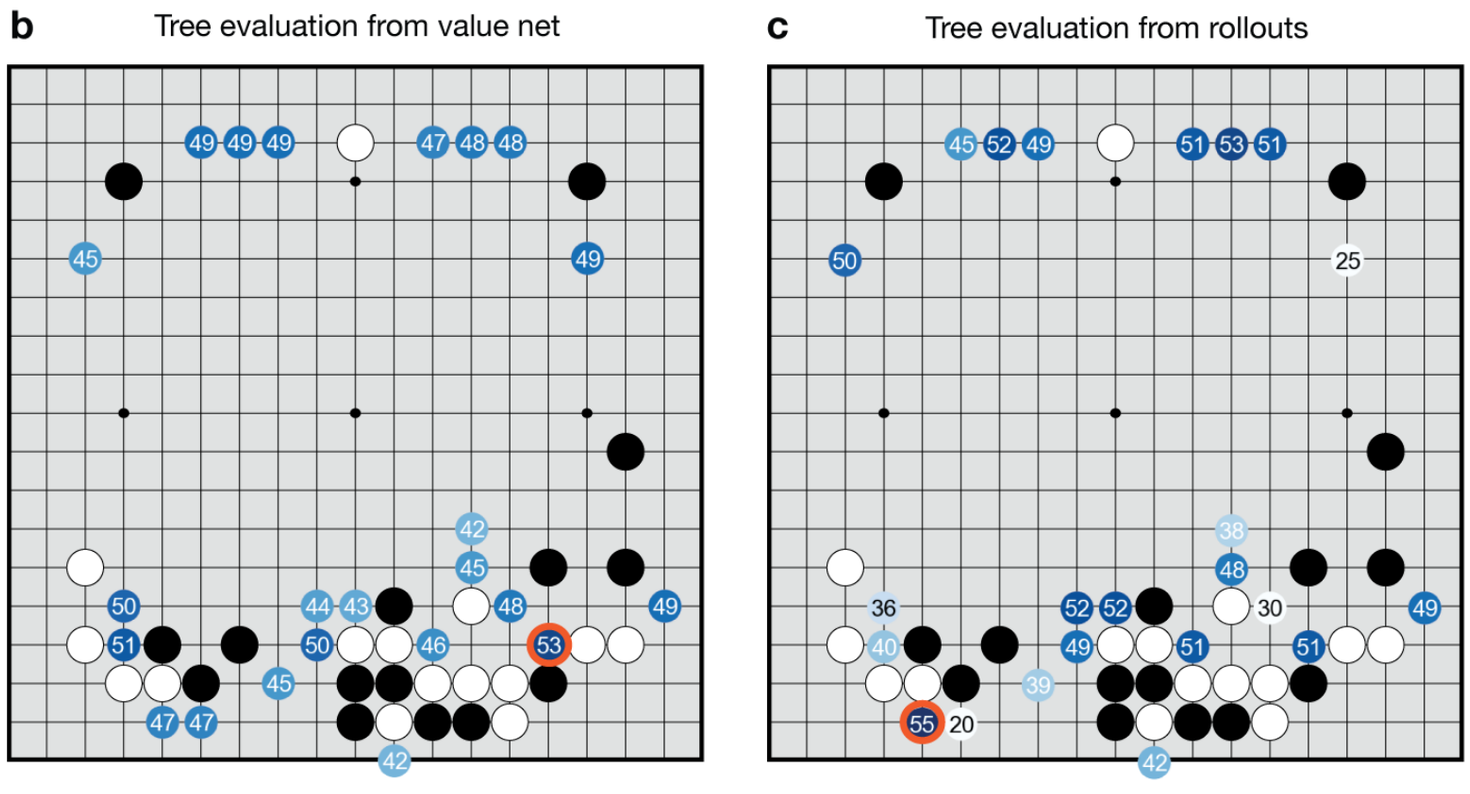
Value network: predict outcome of self-play

convnet

Silver et al., 2016, Mastering the game of Go with deep neural networks and tree search

AlphaGo - Monte carlo tree search





Deep learning for probabilistic models

Why

- Toward tractable inference for more expressive probabilistic models

intractable distributions (unnormalized distributions)

Posterior distribution $p(\theta \mid \mathbf{X}, \alpha) = \frac{p(\mathbf{X} \mid \theta)p(\theta \mid \alpha)}{p(\mathbf{X} \mid \alpha)} \propto p(\mathbf{X} \mid \theta)p(\theta \mid \alpha)$

Energy-based models $P(x) = \frac{1}{Z} \exp f(x)$

Similar to deep learning, inference method are often gradient based

- Variational inference
- MCMC (e.g. Hamiltonian Monte Carlo uses gradient to speed up sampling)

http://arogozhnikov.github.io/2016/12/19/markov_chain_monte_carlo.html

Deep learning for probabilistic models

Why

- Toward tractable inference for more expressive probabilistic models

  intractable distributions (unnormalized distributions)

Posterior distribution	$p(\theta \mid \mathbf{X}, \alpha) = \frac{p(\mathbf{X} \mid \theta) p(\theta \mid \alpha)}{p(\mathbf{X} \mid \alpha)} \propto p(\mathbf{X} \mid \theta) p(\theta \mid \alpha)$
Energy-based models	$P(x) = \frac{1}{Z} \exp f(x)$

Potential approaches for NN-assisted inference

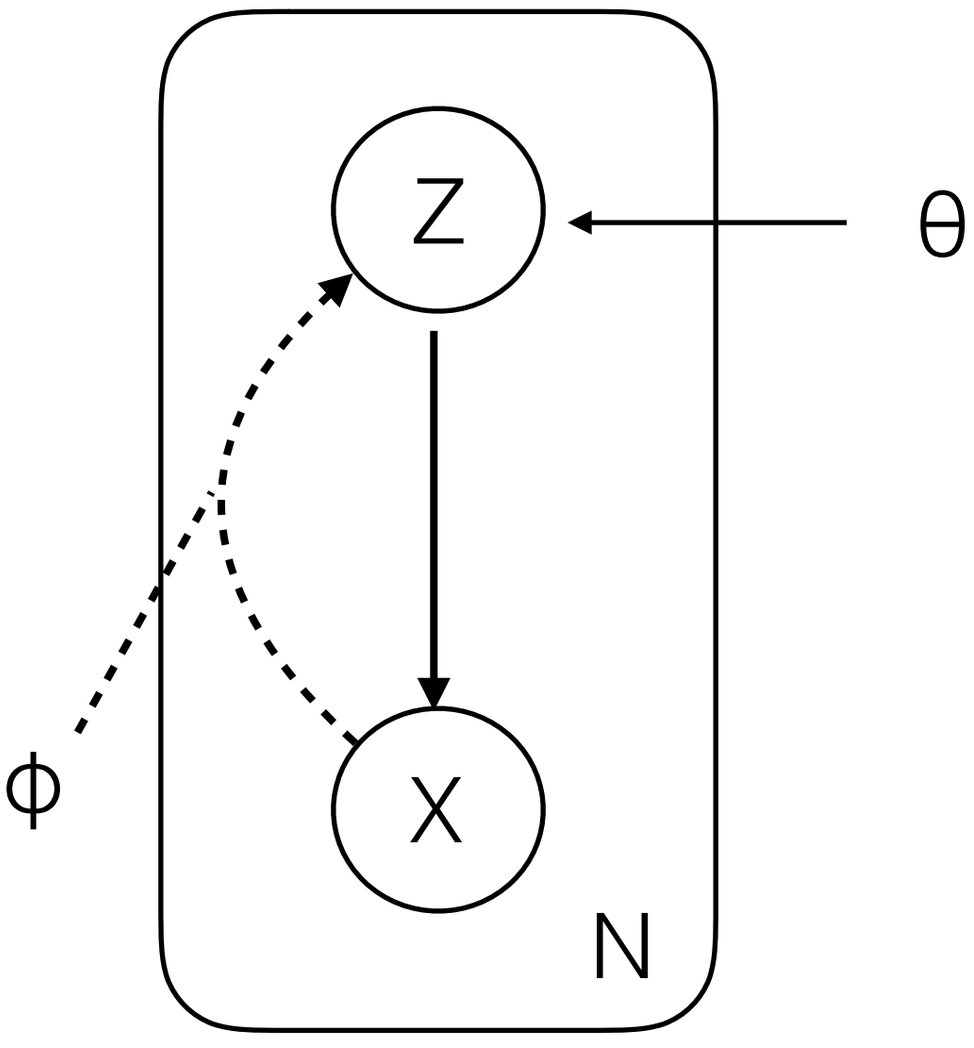
- Neural variational inference
- Neural sampler

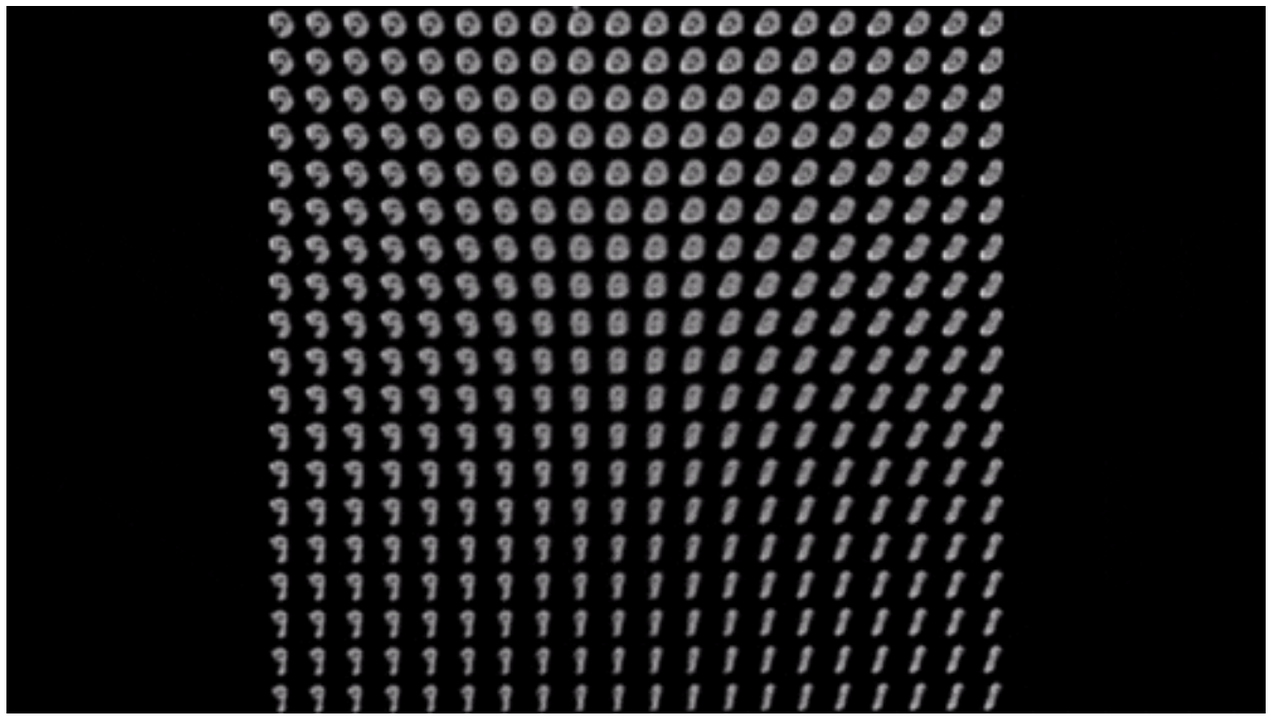
Probabilistic modeling with neural networks: **NVI**

Neural variational inference

(Kingma and Welling, 2014)

Use neural network for describing P(X|Z) or Q(Z|X)



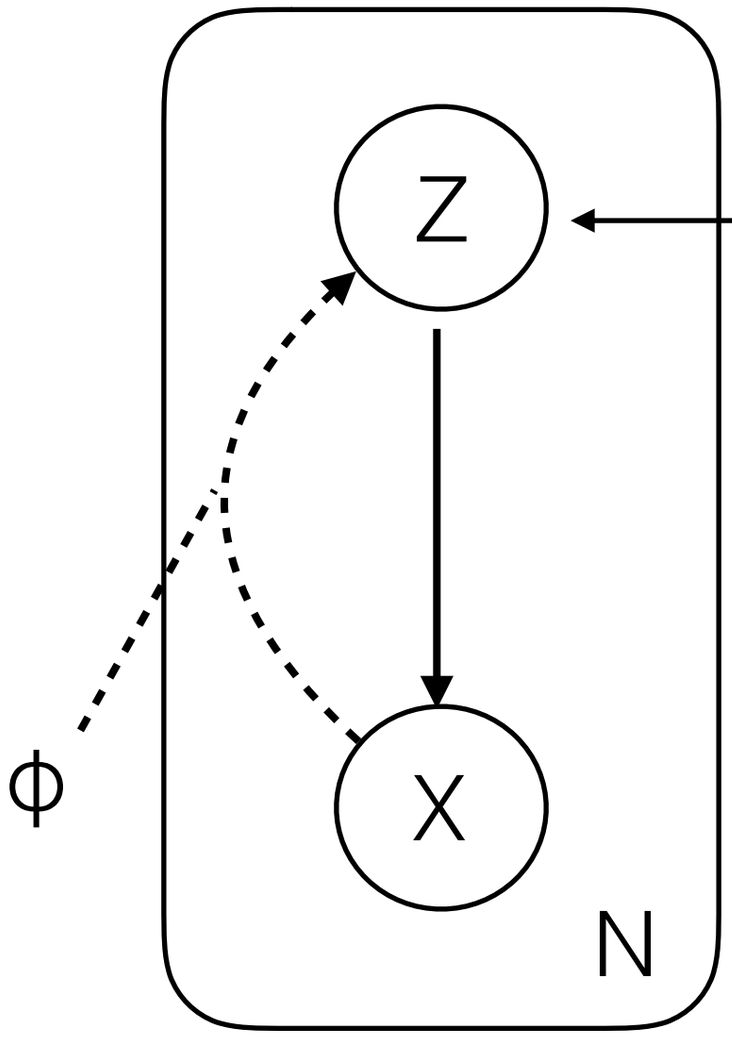


Probabilistic modeling with neural networks: **NVI**

Neural variational inference

(Kingma and Welling, 2014 Auto-Encoding Variational Bayes)

Use neural network for describing P(X|Z) or Q(Z|X)



$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]$$

$$= \boxed{-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]}$$

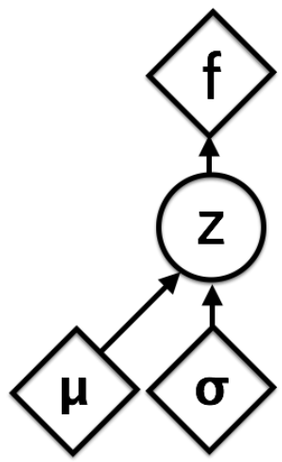
The variational objective

Backpropagation over stochastic units:
Reparametrization trick

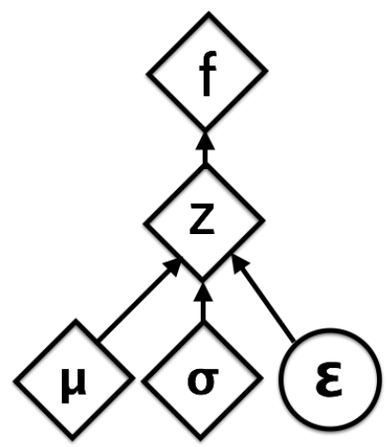
How to compute good gradient estimate of

$$-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

Gradient of expectation -> expectation of stochastic gradient



Original



Reparametrized

$$\nabla_{\mu,\sigma} E_{z\sim\mu,\sigma}[f(z)] = E_{z\sim\mu,\sigma}[f(z)\nabla_{\mu,\sigma}\log\left(p(z|\mu,\sigma)\right)]$$

Williams, 1992 “REINFORCE” estimator

$$\nabla_{\mu,\sigma} E_{\epsilon\sim p(\epsilon)}[f(z)] = E_{\epsilon\sim p(\epsilon)}[\nabla_{\mu,\sigma} f\left(g(\mu,\sigma,\epsilon)\right)]$$

Kingma , et al. 2014. Auto-Encoding Variational Bayes

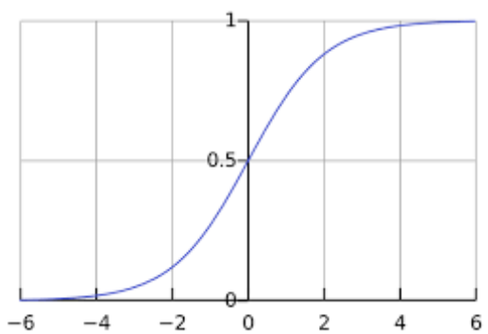
Backpropagation over stochastic units:
Reparametrization trick for discrete variables

The Gumbel trick for sampling from discrete distributions $P(X = k) \propto \alpha_k$

$$G = -\log(-\log(U)) \text{ with } U \sim \text{Unif}[0, 1]$$

$$X = \arg\max_k \left(\log \alpha_k + G_k\right).$$

Softmax function for approximating the max operation with a differentiable function



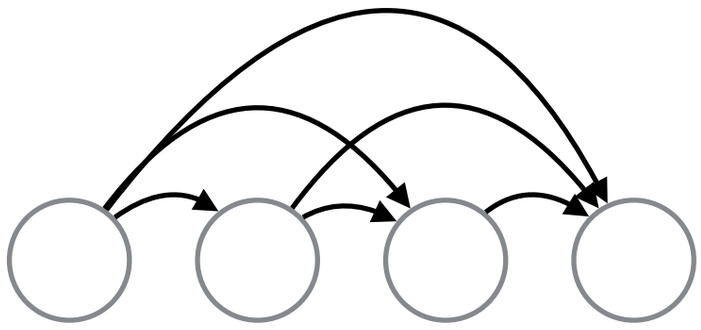
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Discrete variables can always be represented by binary vectors

Toward flexible and normalized probabilistic models

$$-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

1.Fully factorized models

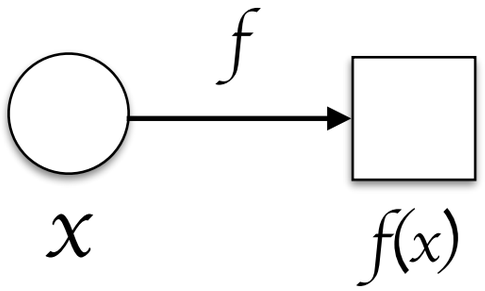


Neural autoregressive models

Probability function is fully factorial
However, it has to commit to a sequence

Toward flexible and normalized probabilistic models

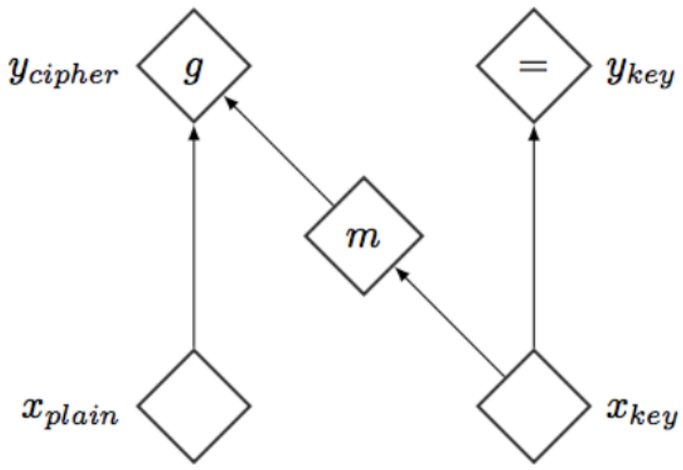
2. Invertible transformations



$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

Examples:

NICE



determinant fixed

Dinh 2015, NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

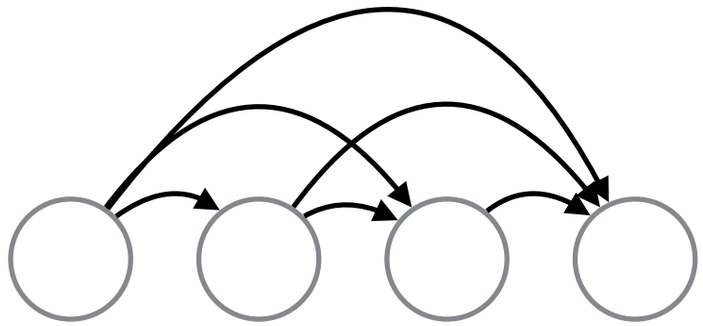
Normalizing flow

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$

determinant O(D) time

Rezende 2016. Variational Inference with Normalizing Flows

Invertible autoregressive flow



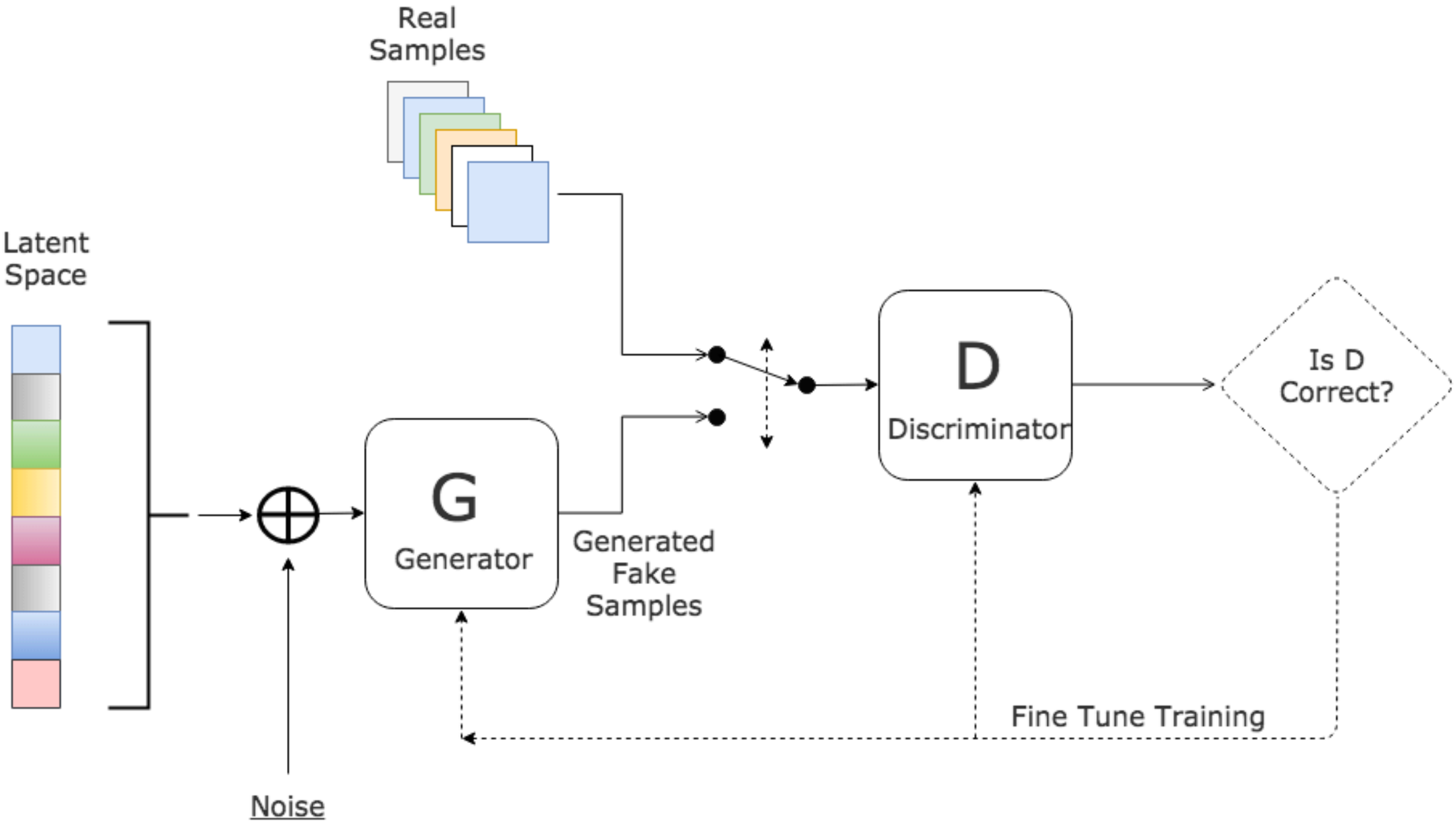
determinant O(D) time

Kingma, 2017 Invertible autoregressive flow

Hidden variables are equal in dimensionality.

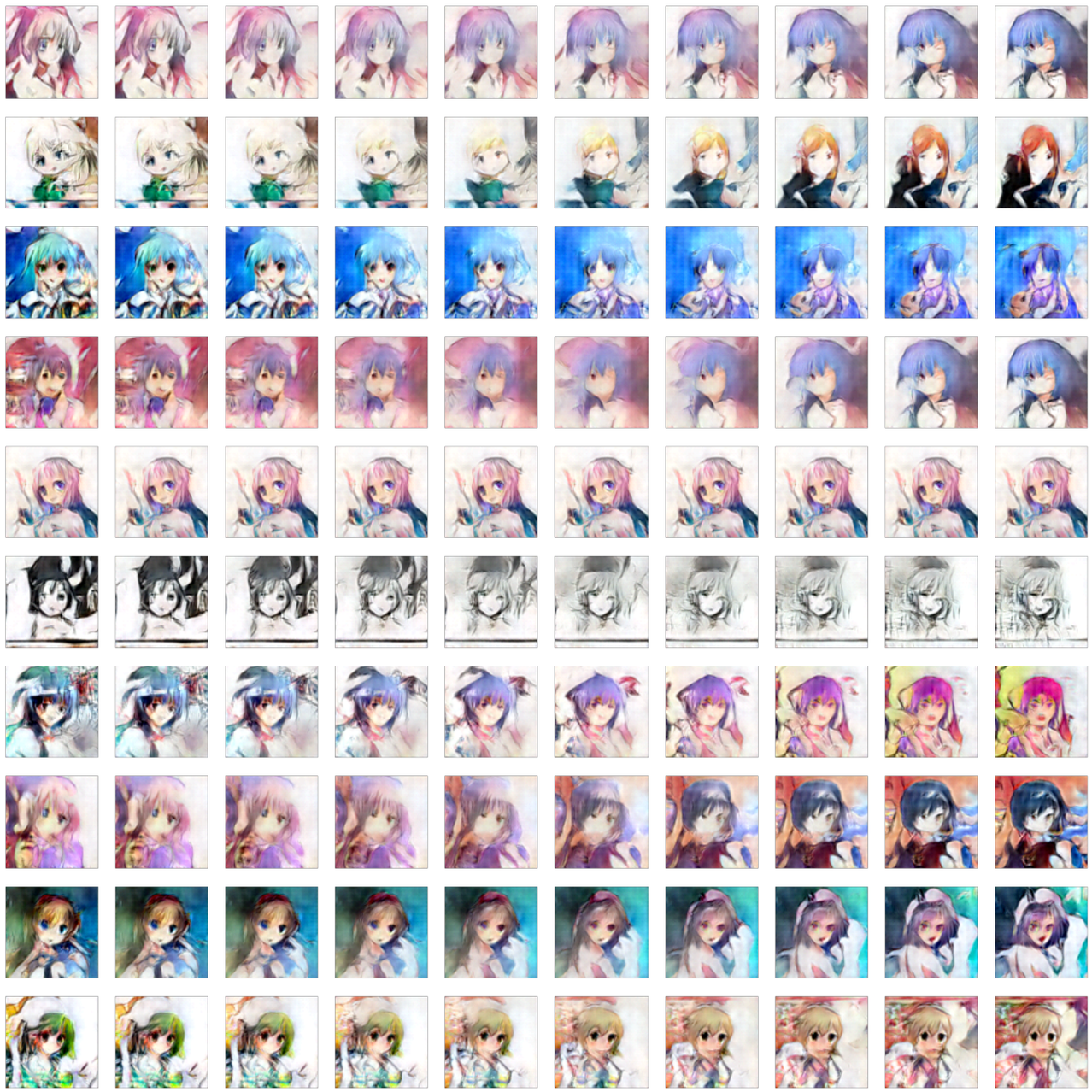
Probabilistic modeling with neural networks: Learn to sample

Generative adversarial networks

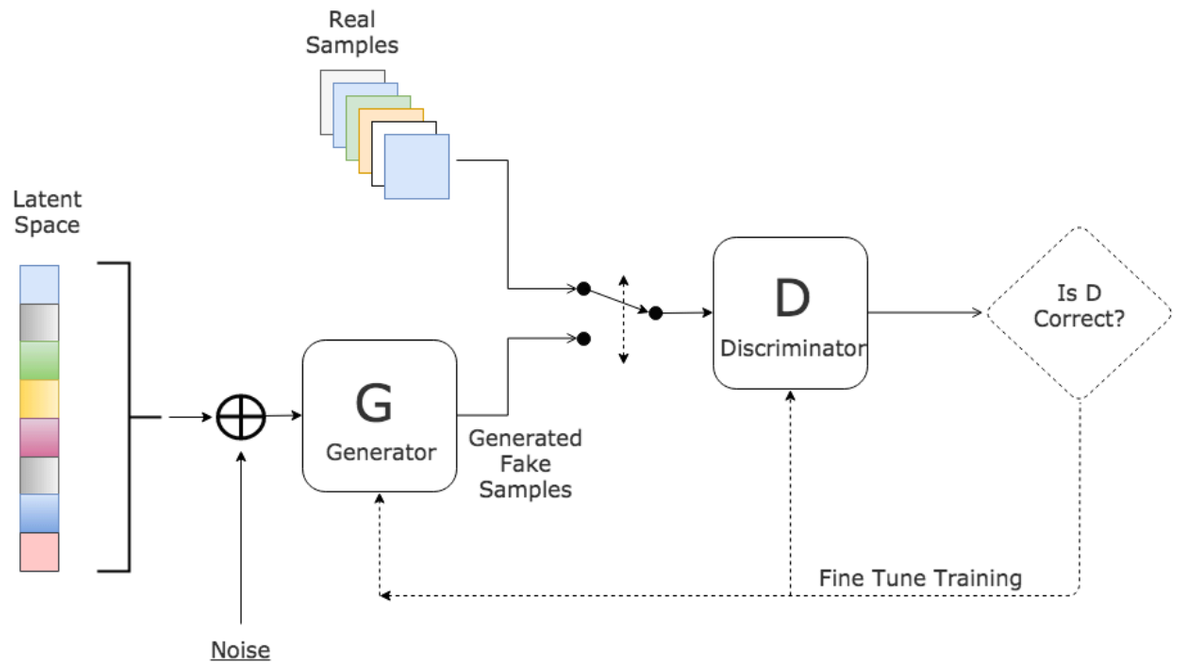


Probabilistic modeling with neural networks: Learn to sample

Generative adversarial networks



Formulating Generative adversarial networks as a proper probabilistic model



$$P(x) = \frac{1}{Z} \exp f(x)$$

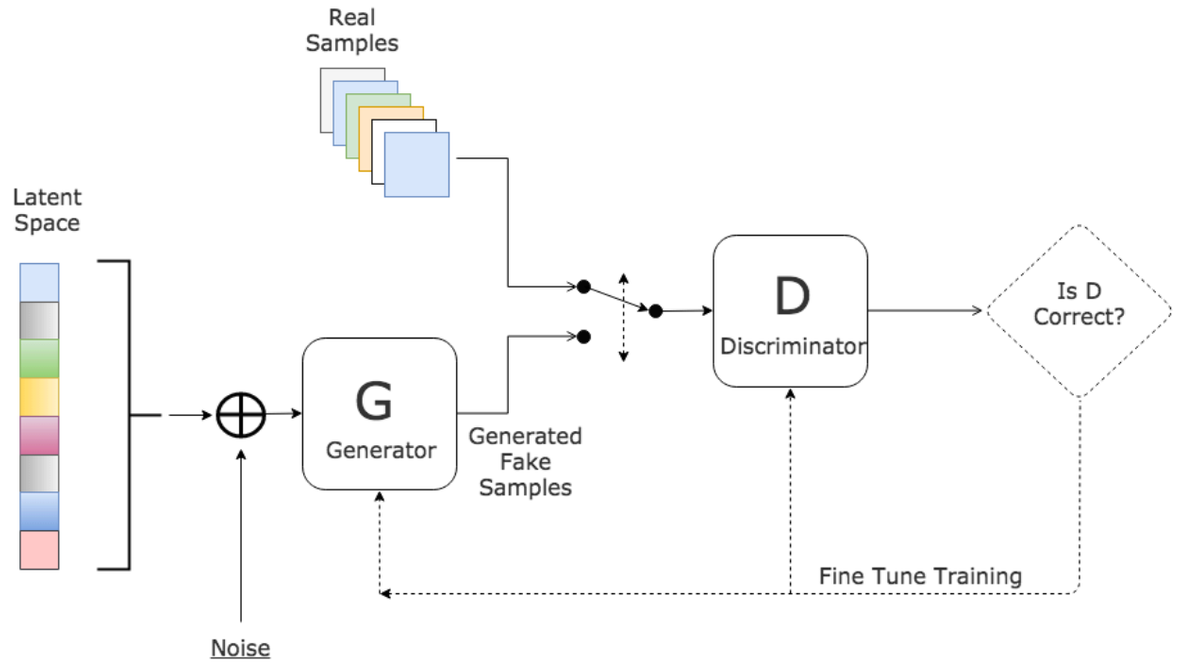
$$\frac{\partial \log P}{\partial \theta} = E_{x \sim data}(\frac{\partial f(x)}{\partial \theta}) - E_{x \sim model}(\frac{\partial f(x)}{\partial \theta})$$

Generator network: use x~Generator instead of x~model

Discriminator network: $f(x)$

Question: can we do exact sampling instead of approximate sampling?

Formulating Generative adversarial networks as a proper probabilistic model



$$P(x) = \frac{1}{Z} \exp f(x) \qquad \frac{\partial \log P}{\partial \theta} = E_{x \sim data}(\frac{\partial f(x)}{\partial \theta}) - E_{x \sim model}(\frac{\partial f(x)}{\partial \theta})$$

Two proposals :

1. Design Generator that generate reversible sequence of samples (allows Metropolis-Hasting sampling)
2. Allows computing (unnormalized) probability for generator networks (allows Metropolis sampling)

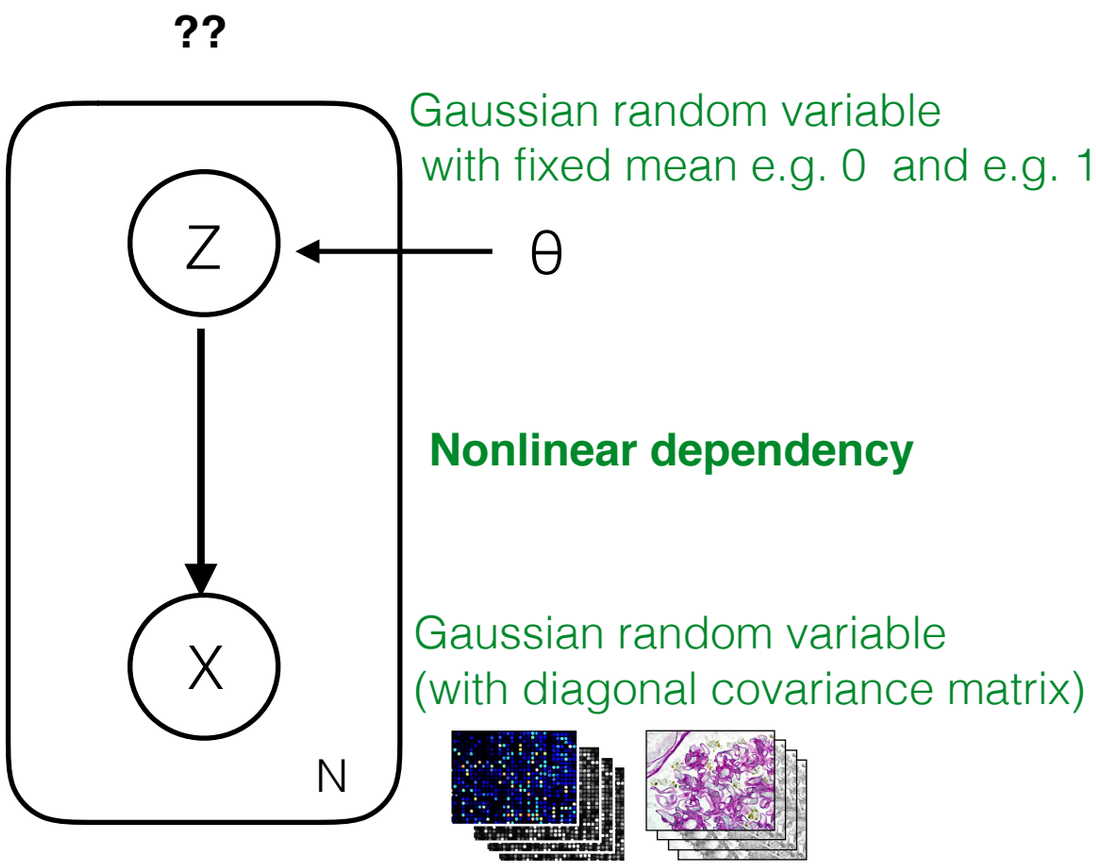
Question: can we do exact sampling instead of approximate sampling?

Coding: choice of packages

	Pros	Cons
pytorch	Fastest automatic differentiation engine	in beta
torch7	Fast and low level Good code base	written in lua
Keras	High-level abstraction	hard for implementing new techniques
Tensorflow	Distributed computation	Poor support for interactive mode

Coding

- https://github.com/pytorch/tutorials



Conditional distribution

$$p(x|z) = \mathcal{N}\left(f(z), \sigma(z)\right)$$

Marginal distribution

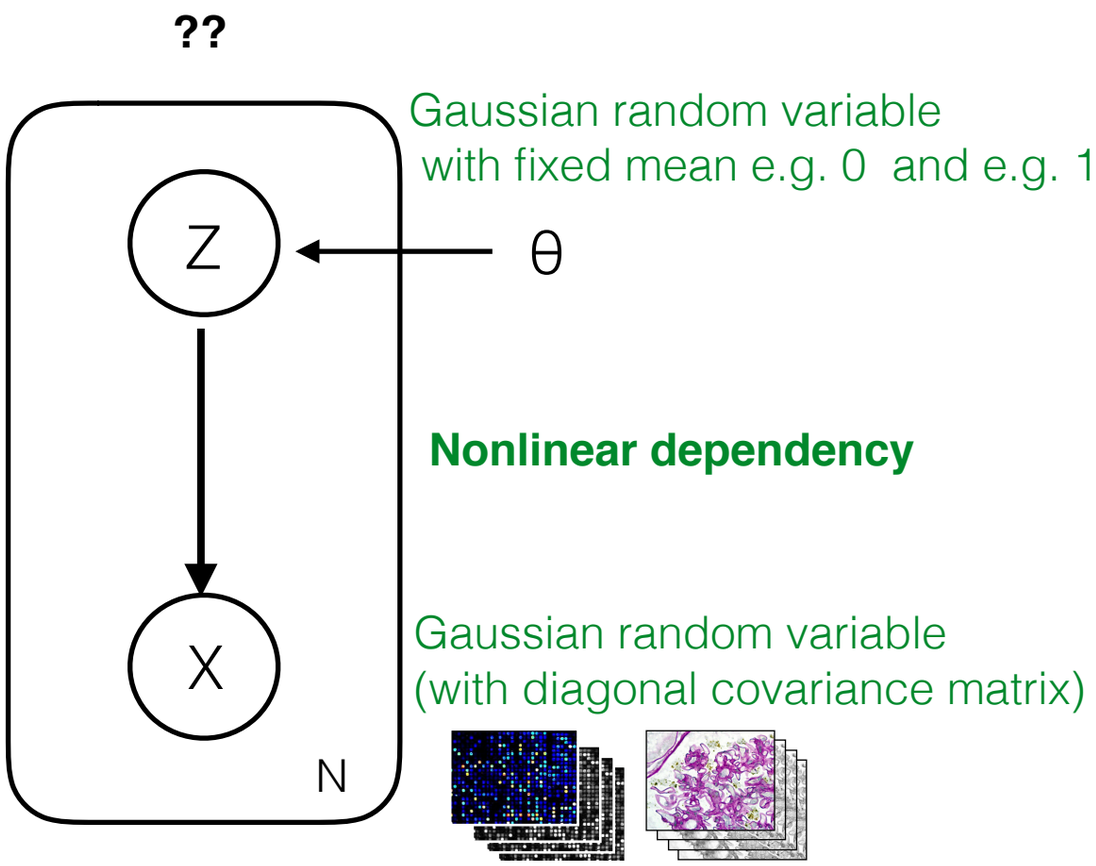
$$p(x) = \int p(x|z)p(z)dz = ?$$

Training algorithm

Proposal* 2:

$$p(x) = \frac{p(x,z)}{p(z|x)} = \frac{p(z)p(x|z)}{p(z|x)}$$

Problem: P(z|x) is generally as intractable as p(x)



Conditional distribution

$$p(x|z) = \mathcal{N}\left(f(z), \sigma(z)\right)$$

Marginal distribution

$$p(x) = \int p(x|z)p(z)dz = ?$$

Training algorithm

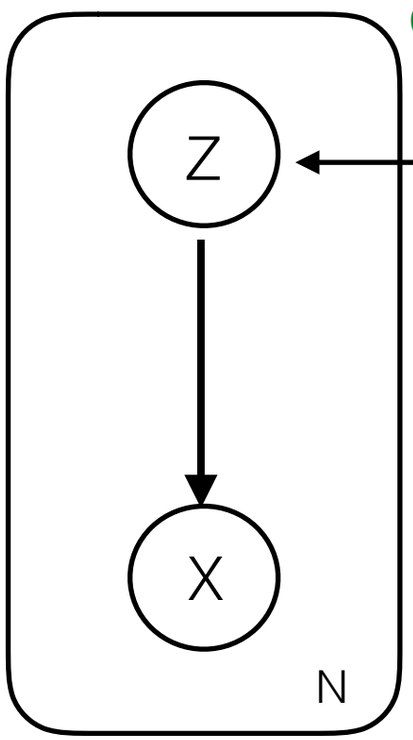
Proposal* 2:

$$p(x) = \frac{p(x,z)}{p(z|x)} = \frac{p(z)p(x|z)}{p(z|x)}$$

Problem: P(z|x) is generally as intractable as p(x)

Training algorithm

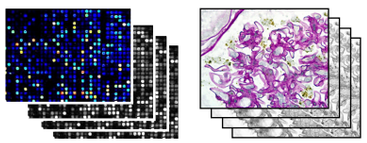
??



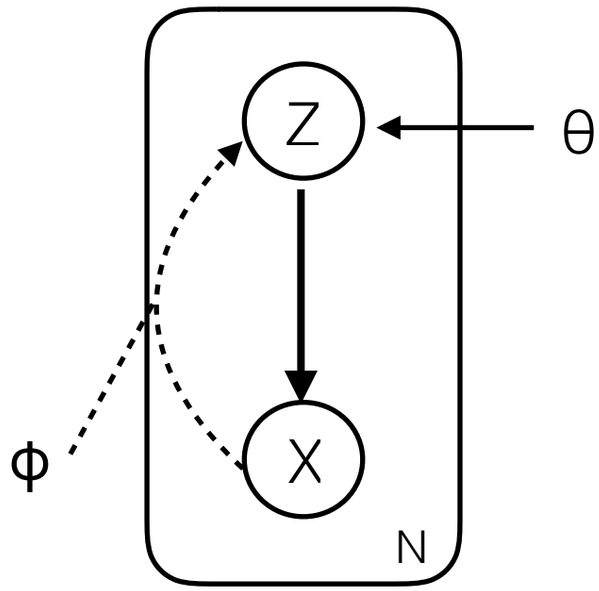
Gaussian random variable
with fixed mean e.g. 0 and e.g. 1

Nonlinear dependency

Gaussian random variable
(with diagonal covariance matrix)



Proposal 3: (Variational Autoencoder)



Conditional distribution

$$p(x|z) = \mathcal{N}\left(f(z), \sigma(z)\right)$$

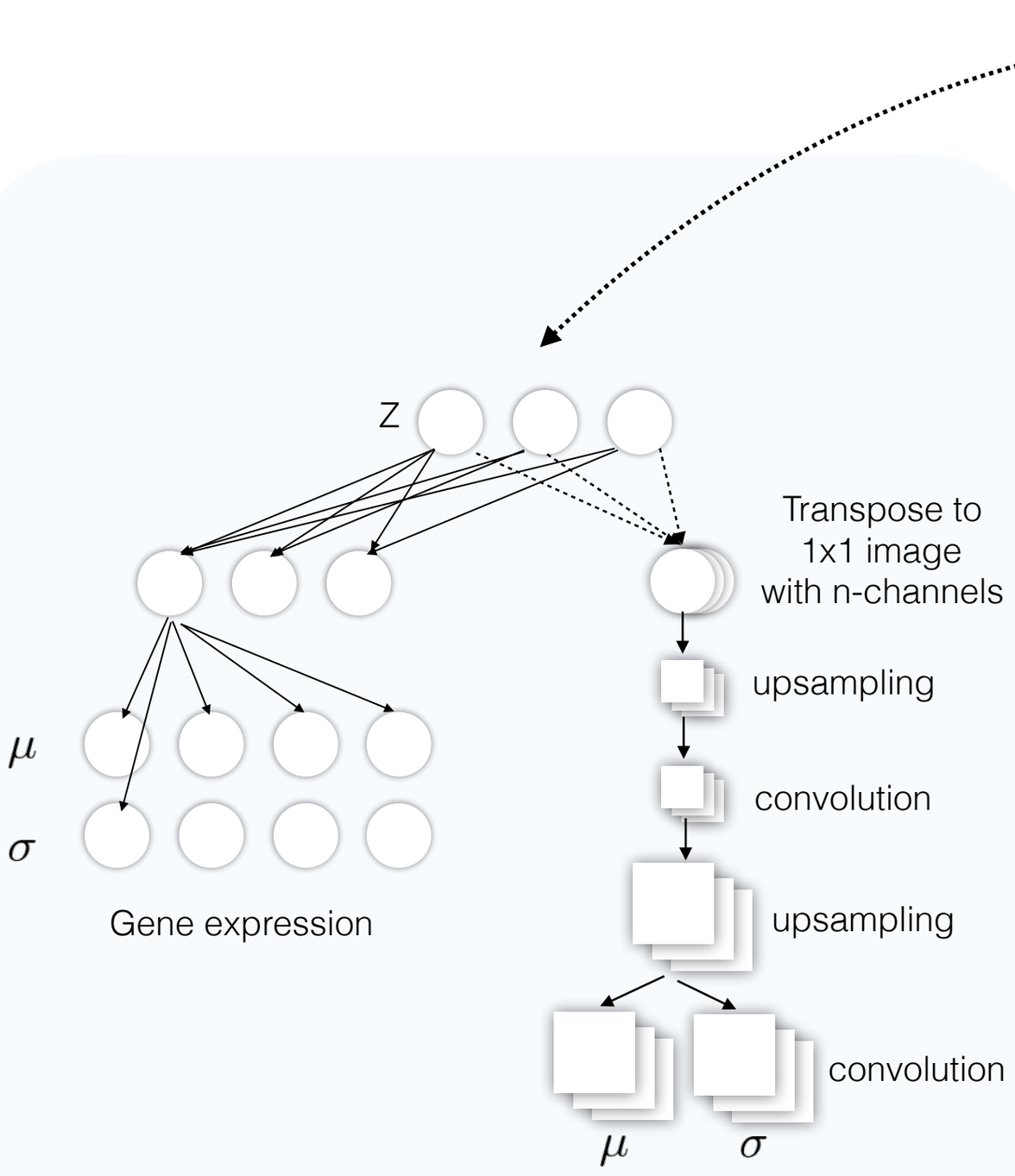
Marginal distribution

$$p(x) = \int p(x|z)p(z)dz = ?$$

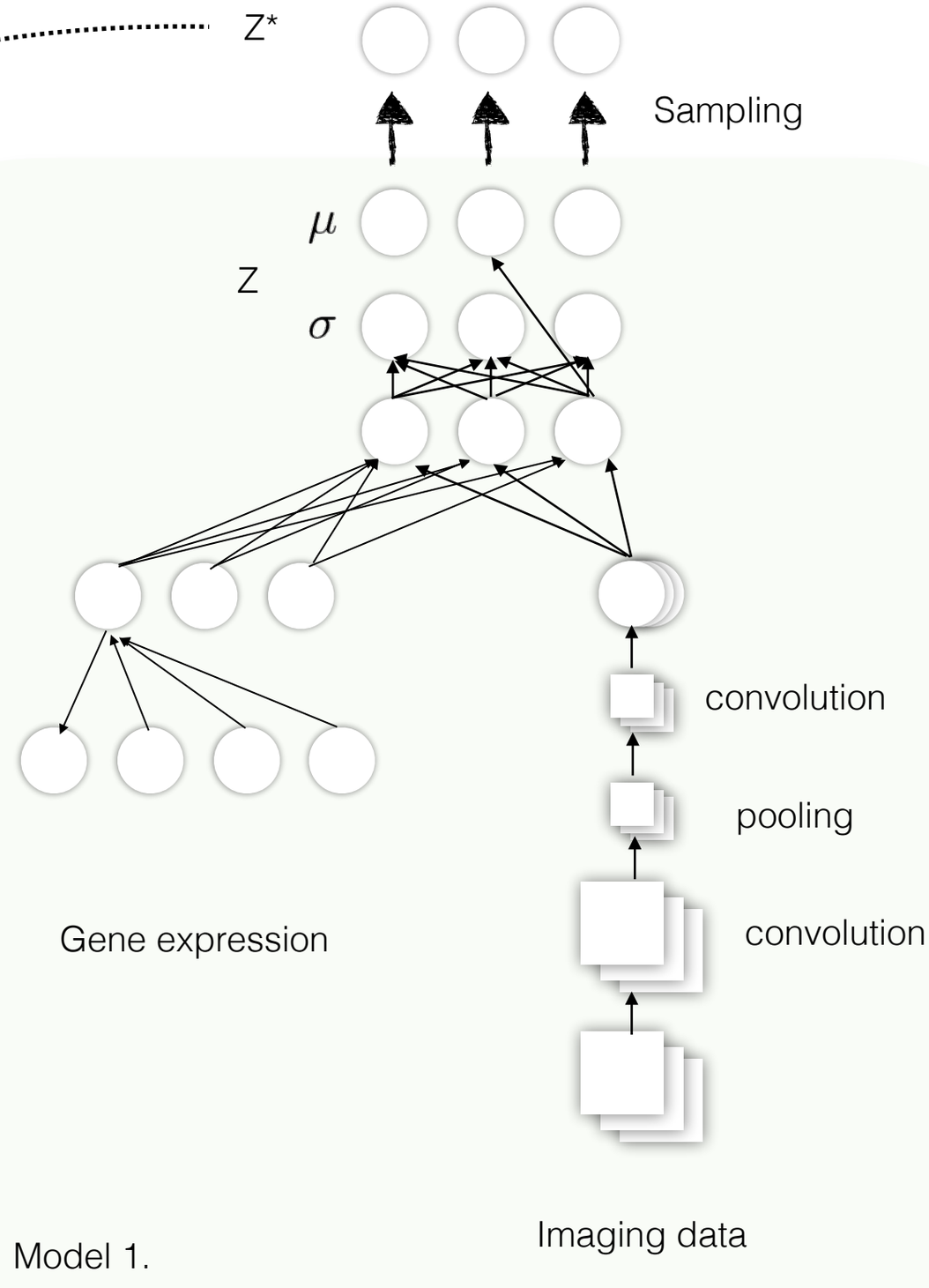
$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]$$

$$= \boxed{-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]}$$

The variational objective



P(X|Z)



Q(Z|X)

Deep probabilistic model

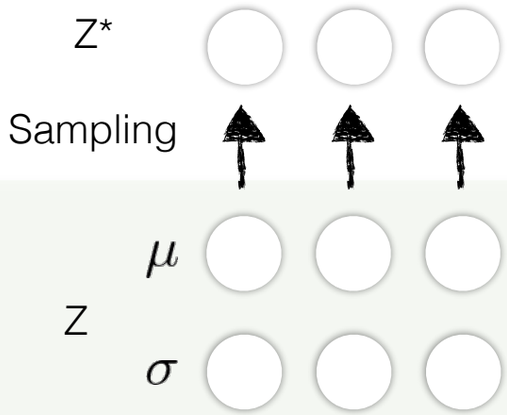
$$-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

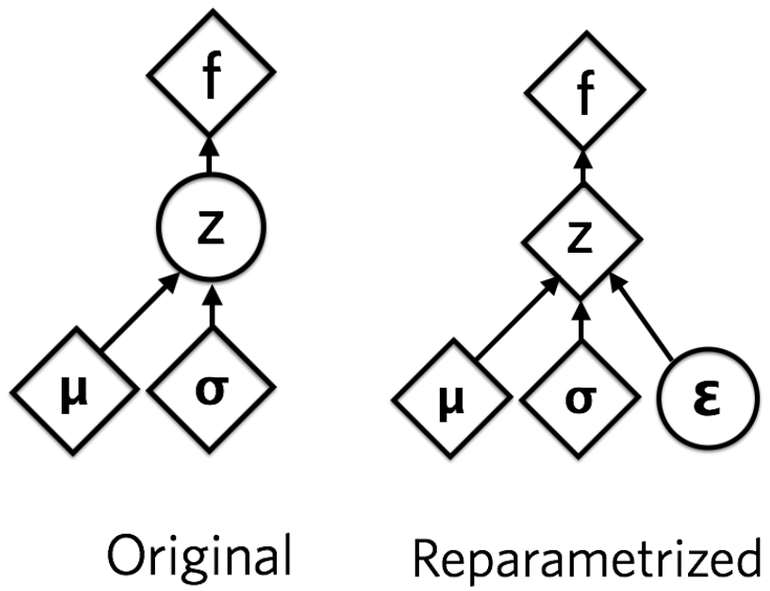
KL divergence between two Gaussians

Use stochastic gradient estimator

Stochastic gradient for p_θ is straightforward

Stochastic gradient for q_Φ:



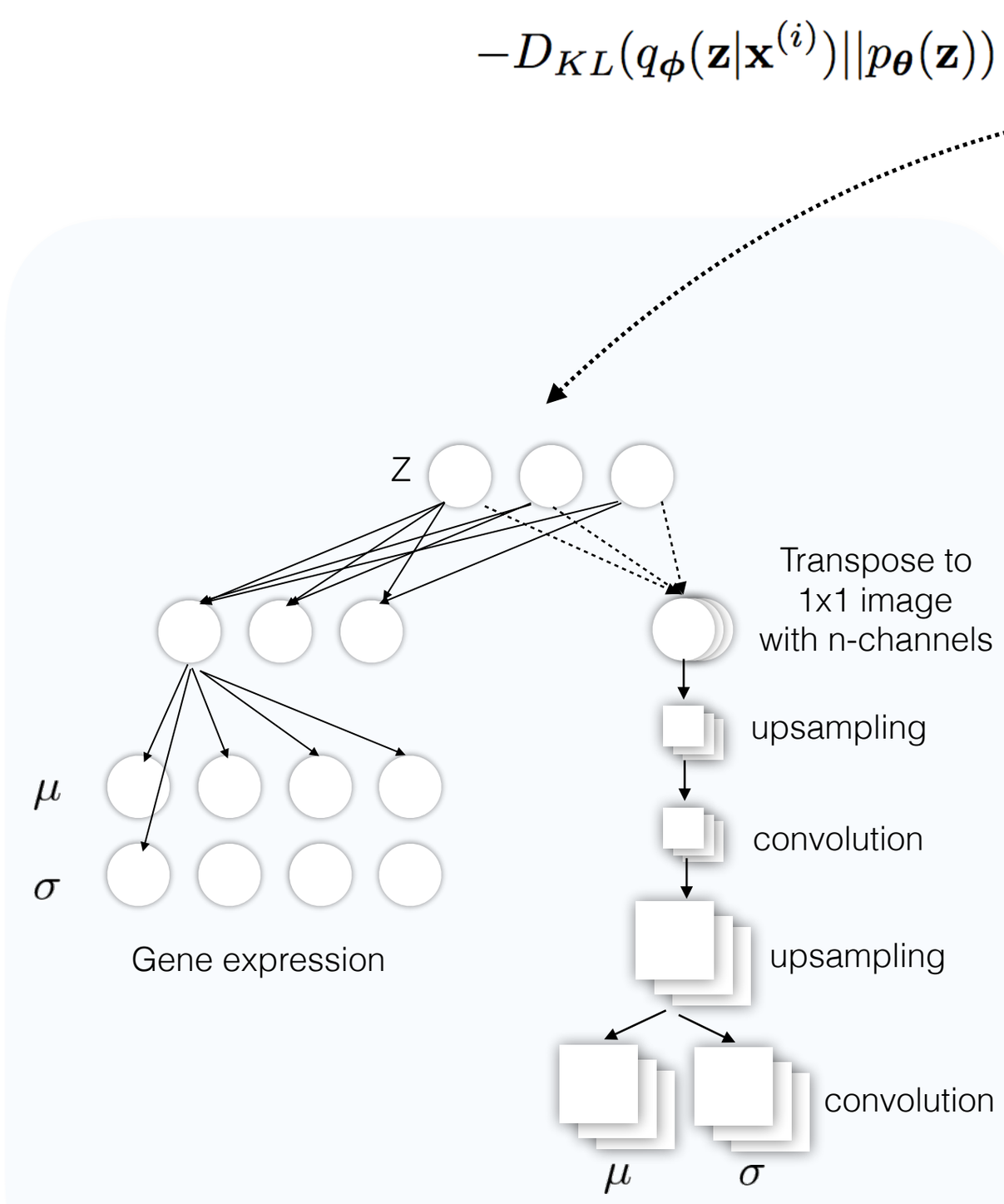


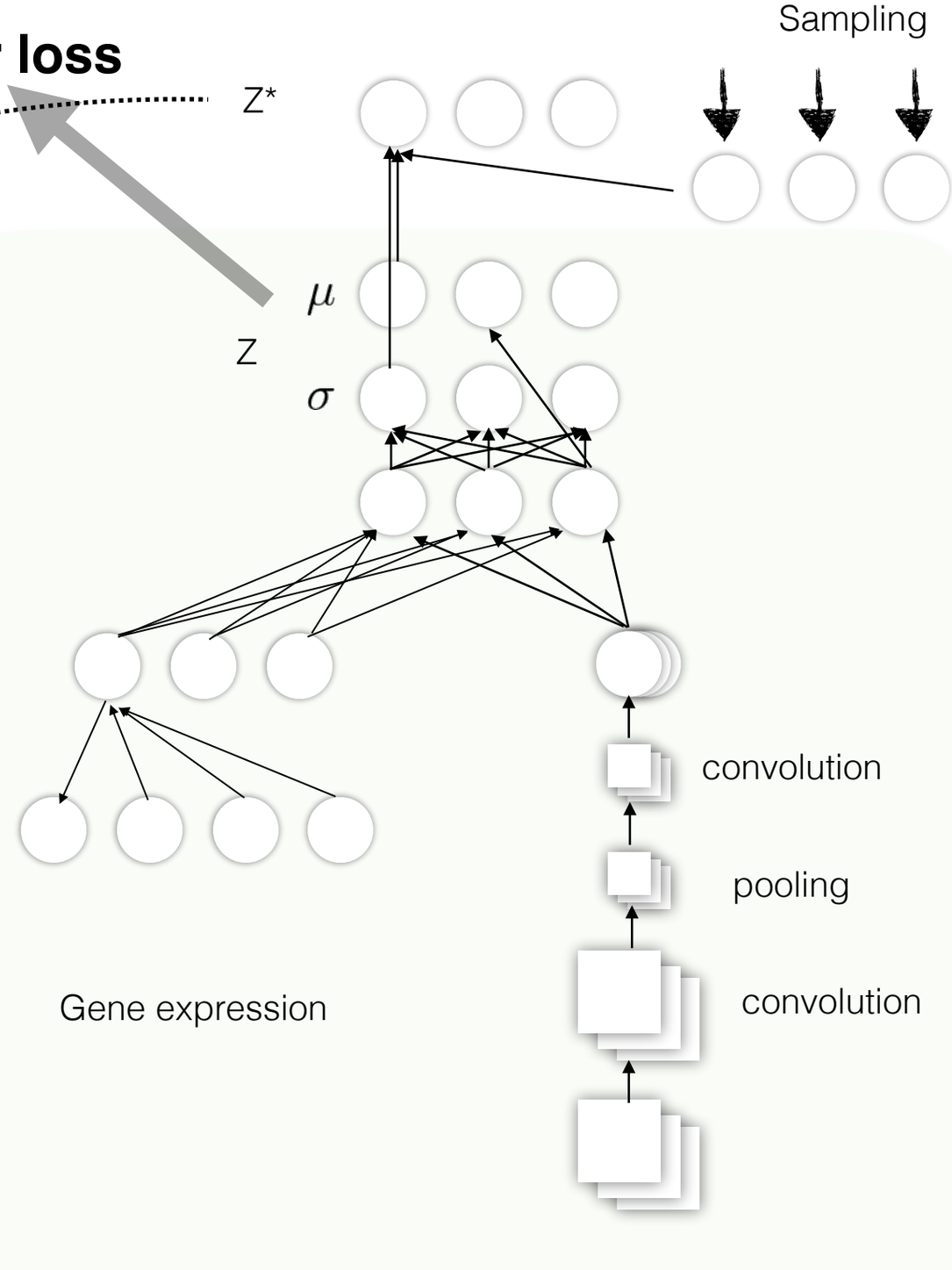
Original Reparametrized

$$\nabla_{\mu,\sigma} E_{\epsilon\sim p(\epsilon)}[f(z)] = E_{\epsilon\sim p(\epsilon)}[\nabla_{\mu,\sigma} f\ (g(\mu,\sigma,\epsilon))]$$

Kingma , et al. 2014. Auto-Encoding Variational Bayes

$$-D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z}))$$ **Prior loss**

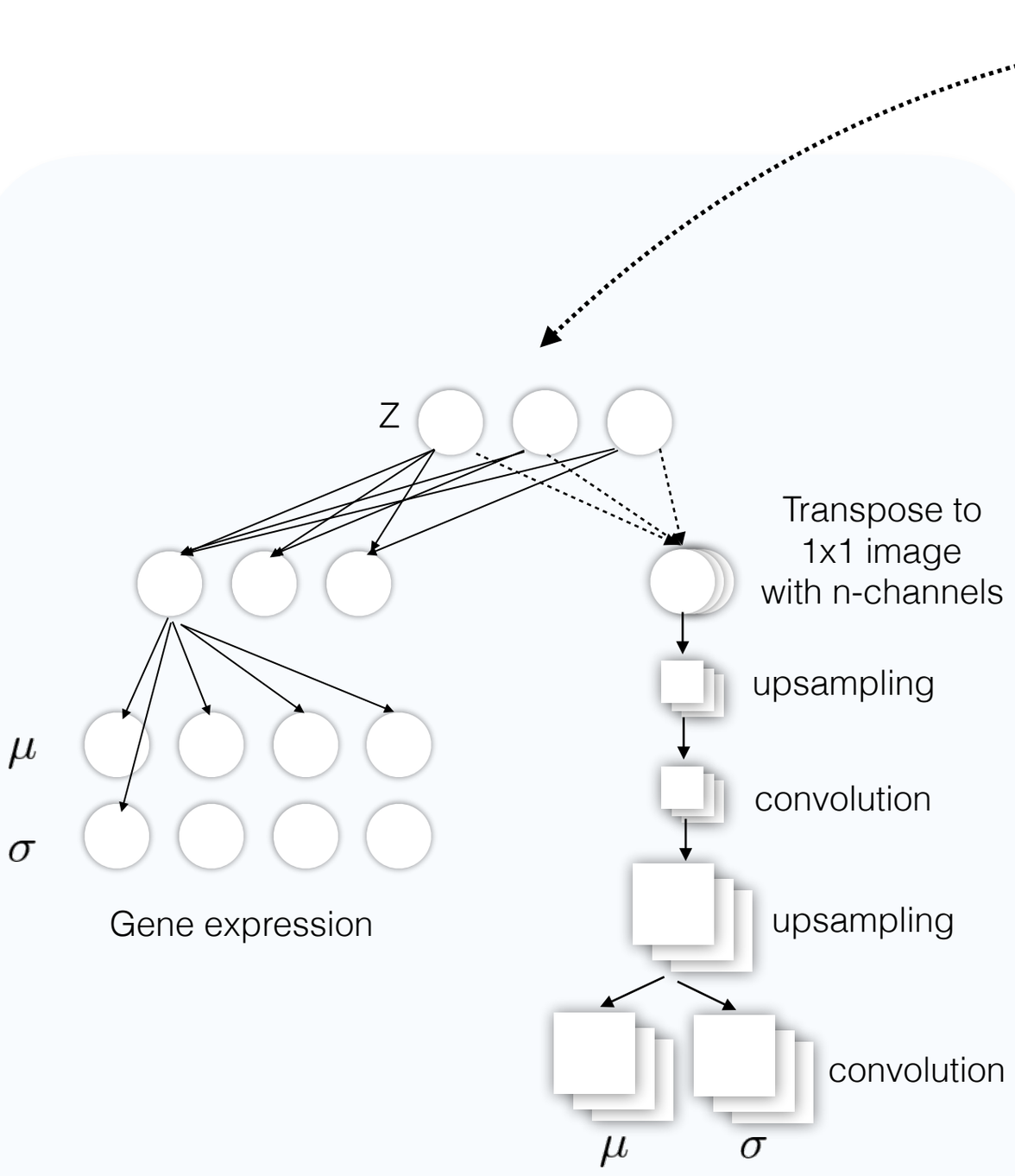


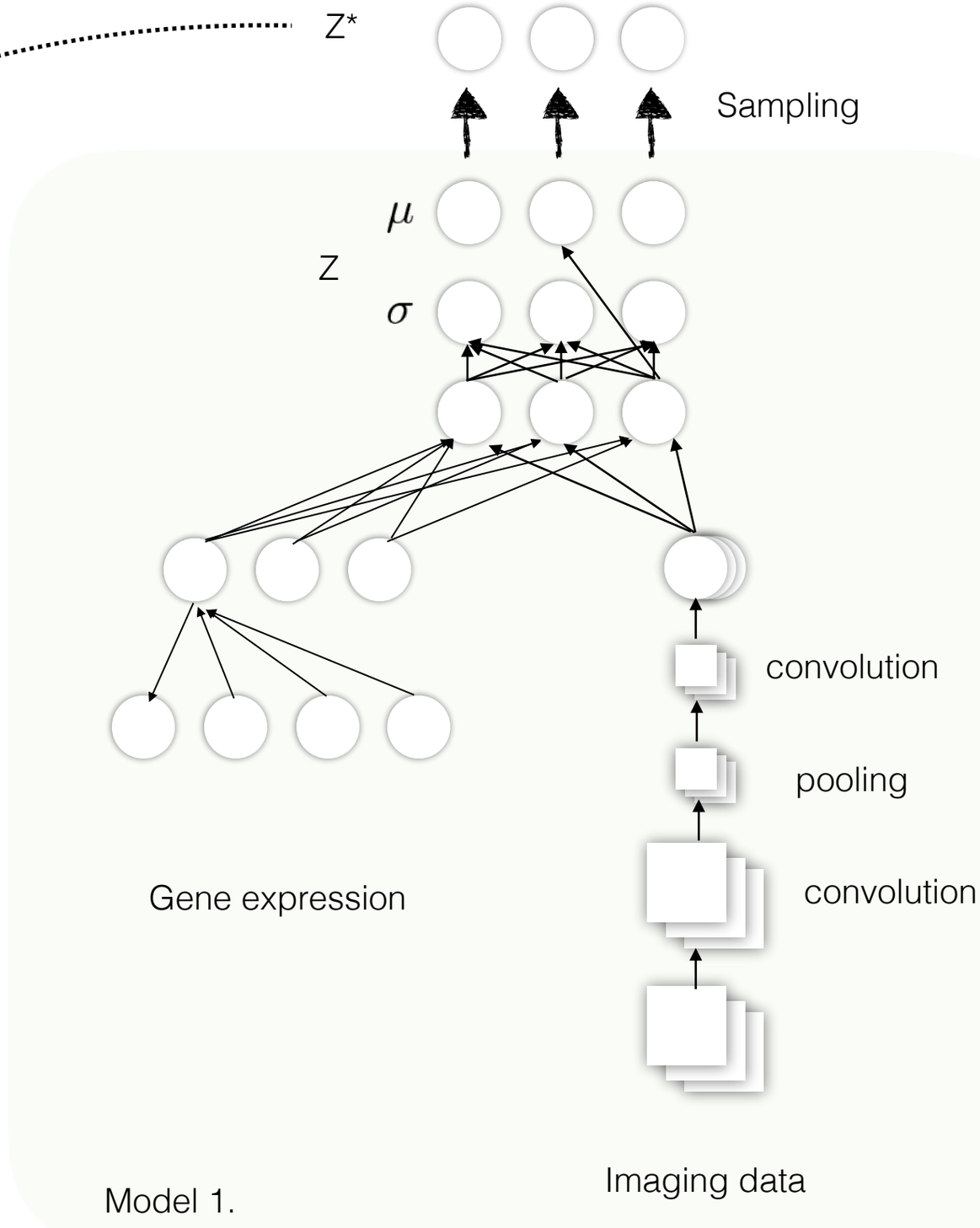


P(X|Z)

Q(Z|X)

$$\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$ **Likelihood loss**

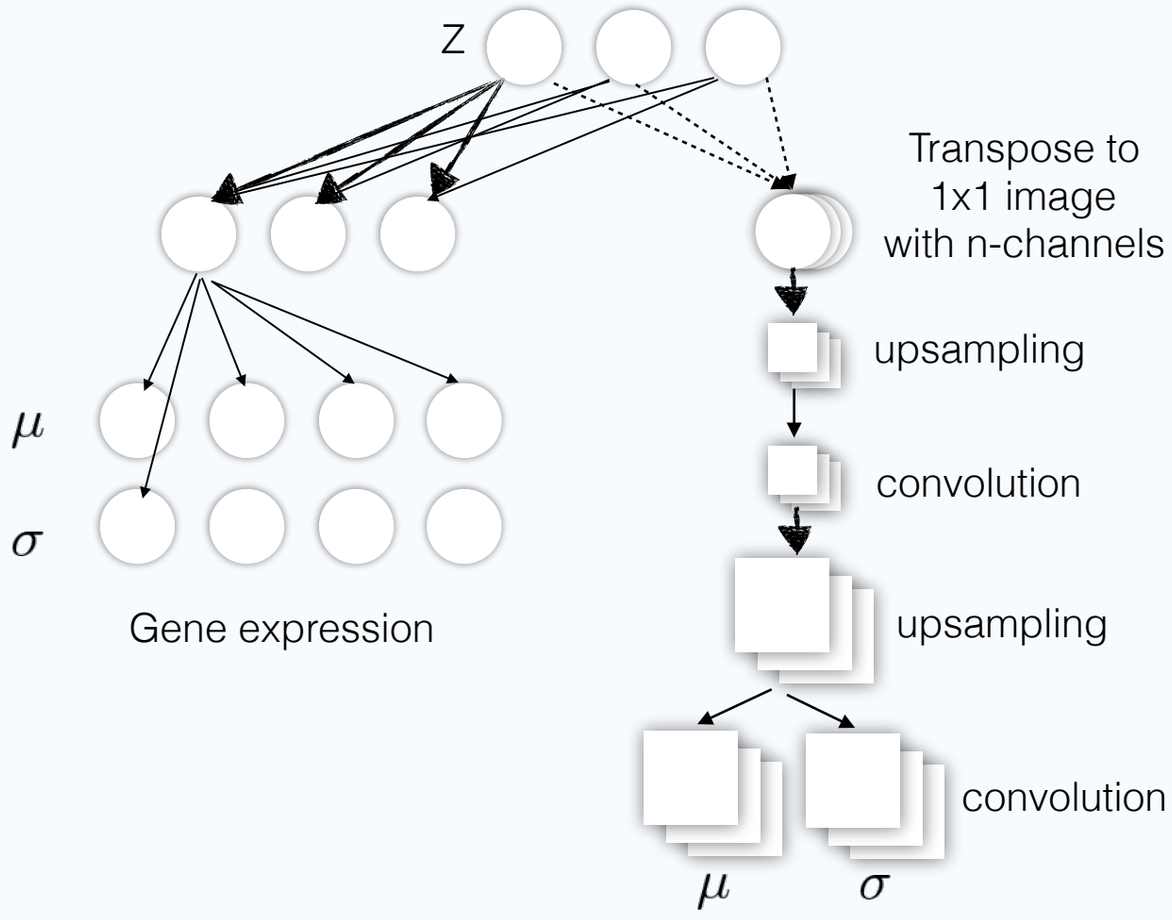




P(X|Z)

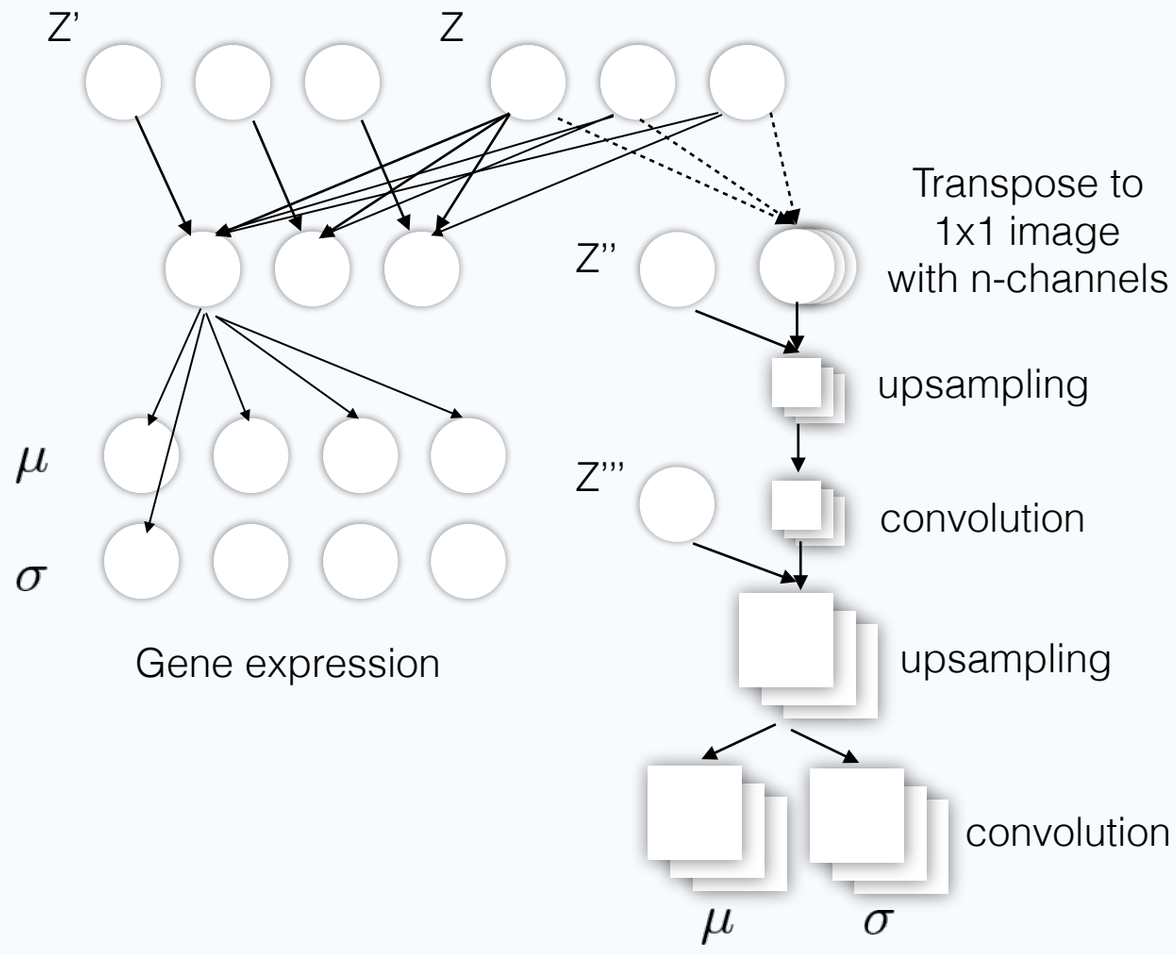
Q(Z|X)

Limitation: Q(Z|X) is still limited to independent Gaussians



Model 2.

Reparametrized



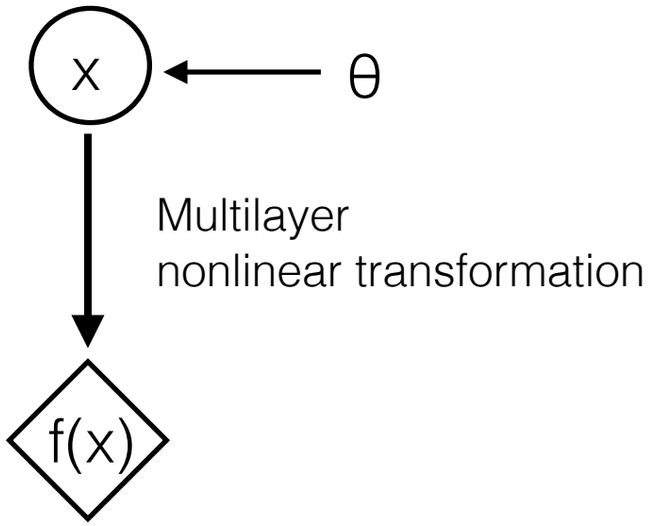
Model 2.

P(X|Z)

P(X|Z) being independent Gaussians
does not limit its flexibility

Use neural network as flexible distributions

Prototype for stochastic neural network



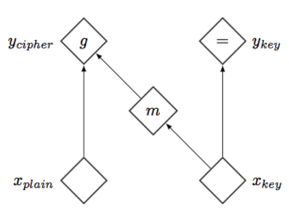
P(f(x)) is in general intractable for stochastic neural networks

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

is applicable when f is invertible

Invertible models

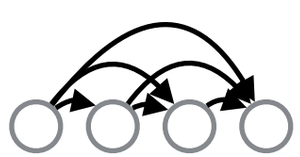
NICE

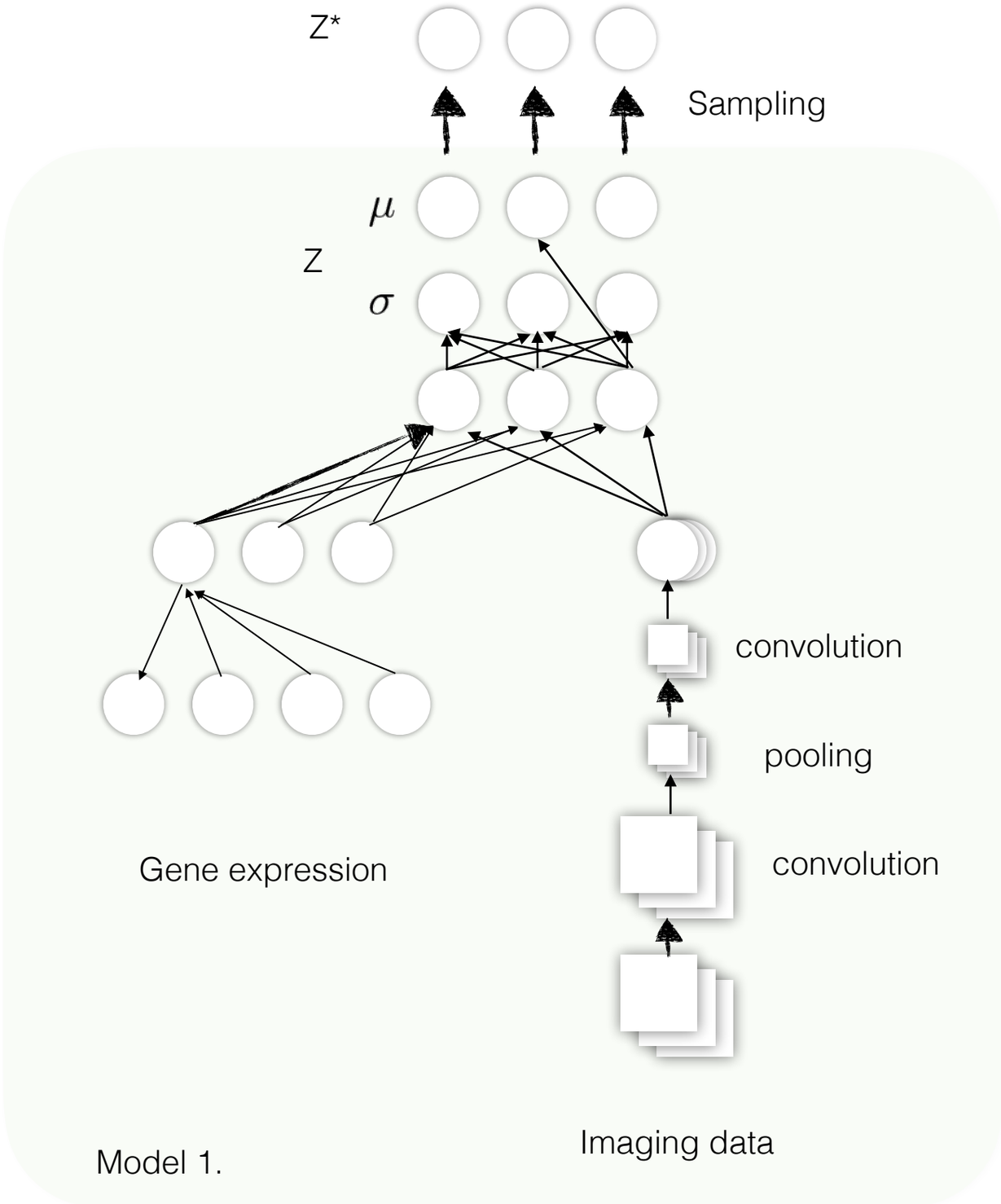


Normalizing flow

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$

Invertible autoregressive flow





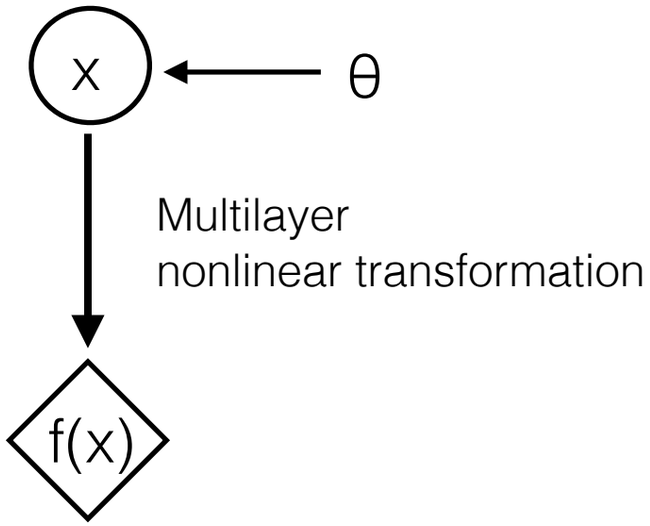
Model 1.

Q(Z|X)

Deep probabilistic model

Use neural network as flexible distributions

Prototype for stochastic neural network

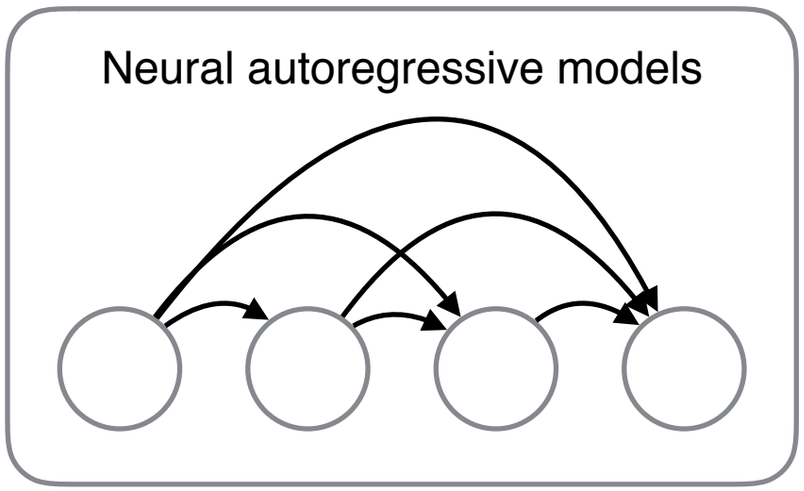


P(f(x)) is in general intractable for stochastic neural networks

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

is applicable when f is invertible

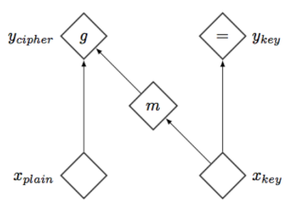
Fully factorized model



Any other family of tractable models existing?

Invertible models

NICE



Normalizing flow

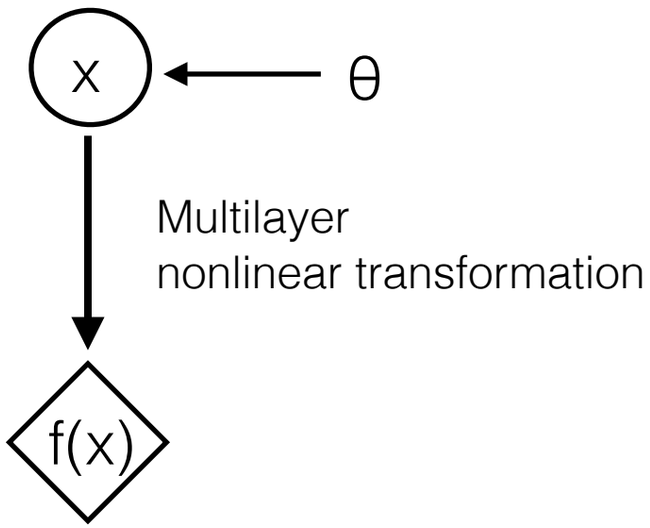
$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$

Invertible autoregressive flow

Use neural network as flexible distributions

Route 1: Easy sampling, hard probability evaluation

Prototype for stochastic neural network



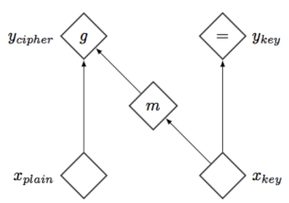
P(f(x)) is in general intractable for stochastic neural networks

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

is applicable when f is invertible

Invertible models

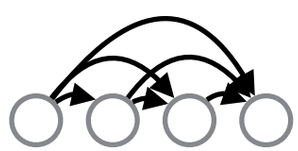
NICE



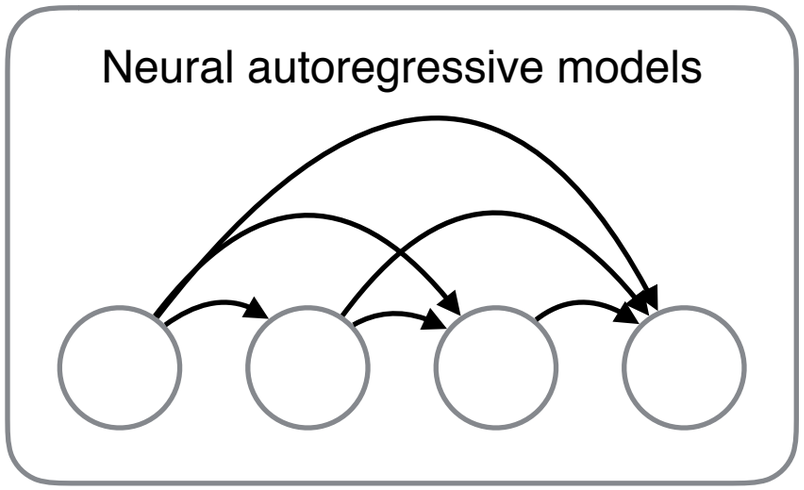
Normalizing flow

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$

Invertible autoregressive flow



Fully factorized model



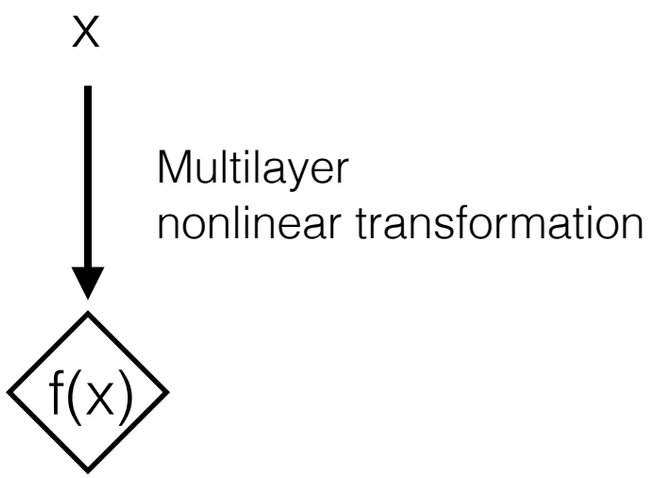
Any other more flexible tractable families existing?

Deep probabilistic model

Use neural network as flexible distributions

Route 2: Easy (unnormalized) probability calculation, hard sampling

Prototype for neural network-based energy model



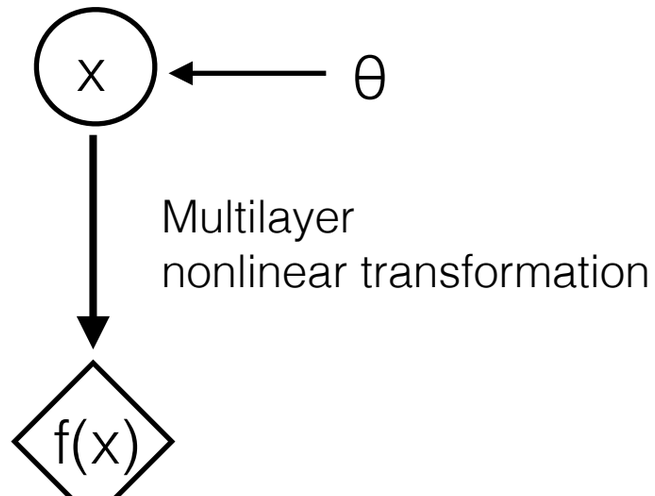
$$P(x) = \frac{1}{Z} \exp f(x)$$

Direct gradient-based MCMC sampling like HMC is feasible, but generally too slow

Is it possible to design trainable reversible conditional stochastic neural network ?

Q(x'|x)=Q(x|x')

Use stochastic neural network as proposal distribution to assist sampling



Q(x) or Q(x'|x)

Metropolis sampling

Acceptance probability $\min\left(1, \frac{P(x')}{P(x)}\frac{Q(x)}{Q(x')}\right)$

Metropolis-Hastings sampling

Acceptance probability $\min\left(1, \frac{P(x')}{P(x)}\right)$