

Introduction to Deep Learning: Lecture II

Michael Kagan

SLAC

IN2P3 School of Statistics 2021
January 26, 2021

- From Logistic Regression to Neural Networks
- Basics of Neural Networks
- Deep Neural Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
 - And a bit about Graph Neural Networks
- AutoEncoders and Generative Models

- Many types of data are not fixed in size
- Many types of data have a temporal or sequence-like structure
 - Text
 - Video
 - Speech
 - DNA
 - ...
- MLP expects fixed size data
- How to deal with sequences?

- Given a set \mathcal{X} , let $S(\mathcal{X})$ be the set of sequences, where each element of the sequence $x_i \in \mathcal{X}$
 - \mathcal{X} could be reals \mathbb{R}^M , integers \mathbb{Z}^M , etc.
 - Sample sequence $x = \{x_1, x_2, \dots, x_T\}$
- Tasks related to sequences:
 - Classification $f: S(\mathcal{X}) \rightarrow \{\mathbf{p} \mid \sum_{c=1}^N p_c = 1\}$
 - Generation $f: \mathbb{R}^d \rightarrow S(\mathcal{X})$
 - Seq.-to-seq. translation $f: S(\mathcal{X}) \rightarrow S(\mathcal{Y})$

- Input sequence $\mathbf{x} \in S(\mathbb{R}^m)$ of *variable* length $T(\mathbf{x})$
- Standard approach: use recurrent model that maintains a **recurrent state** $\mathbf{h}_t \in \mathbb{R}^q$ updated at each time step t . For $t = 1, \dots, T(\mathbf{x})$:

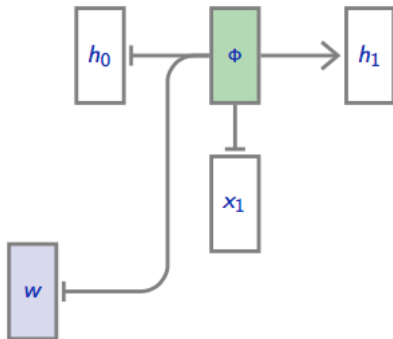
$$\mathbf{h}_{t+1} = \phi(\mathbf{x}_t, \mathbf{h}_t; \theta)$$

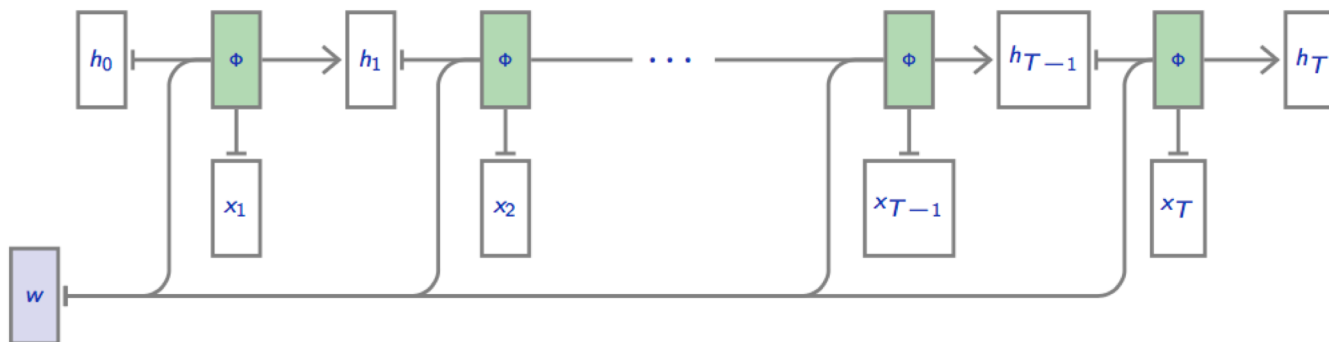
- Simplest model:

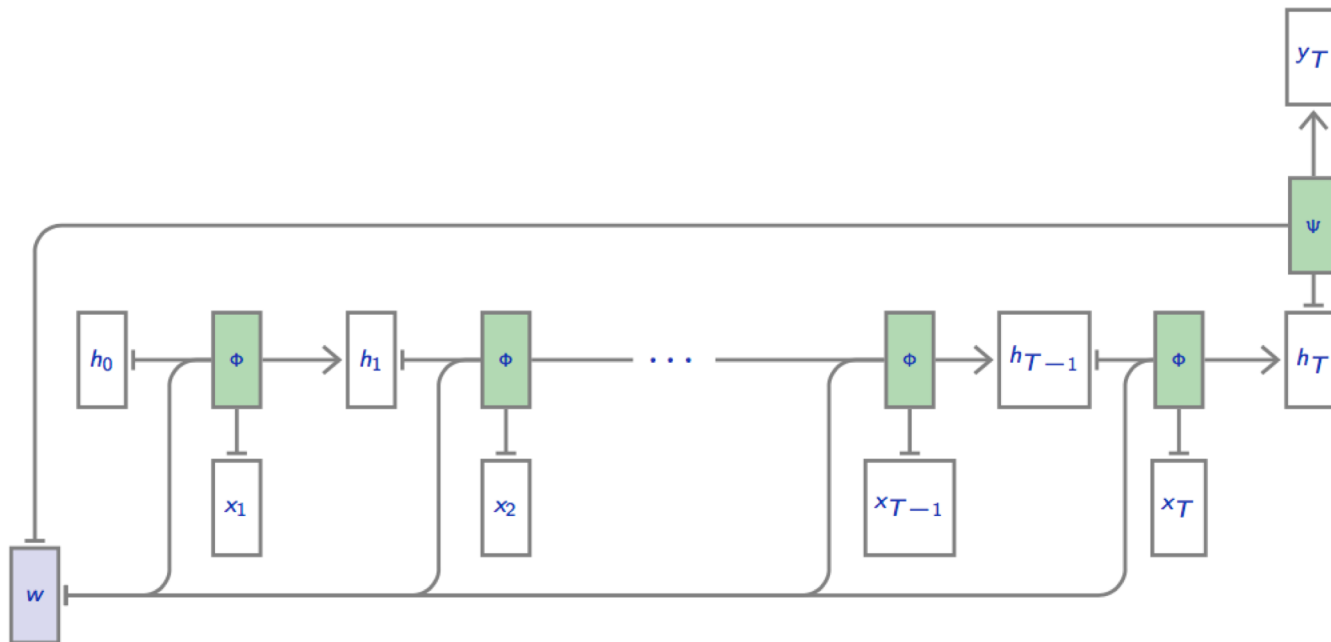
$$\phi(\mathbf{x}_t, \mathbf{h}_t; W, U) = \sigma(W\mathbf{x}_t + U\mathbf{h}_t)$$

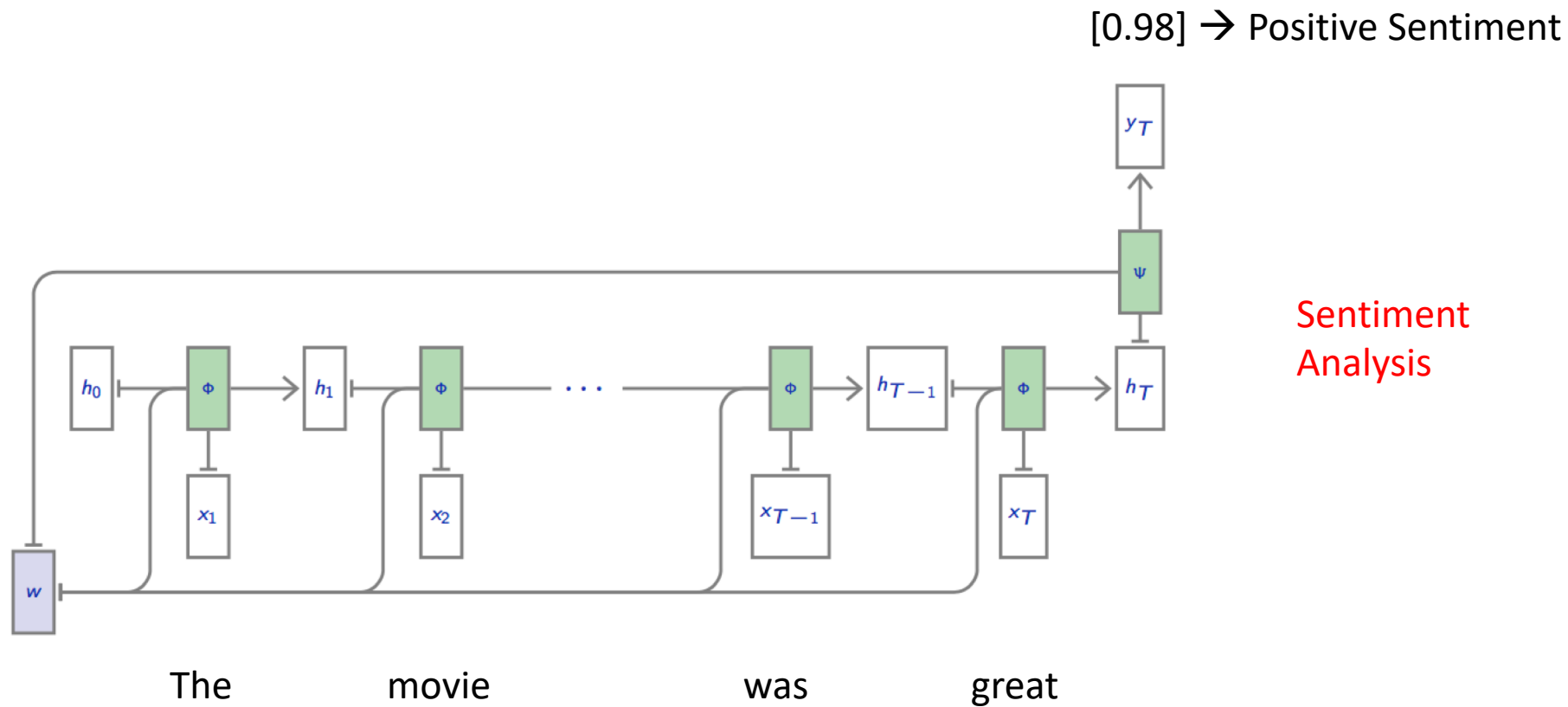
- Predictions can be made at any time t from the recurrent state

$$\mathbf{y}_t = \psi(\mathbf{h}_t; \theta)$$

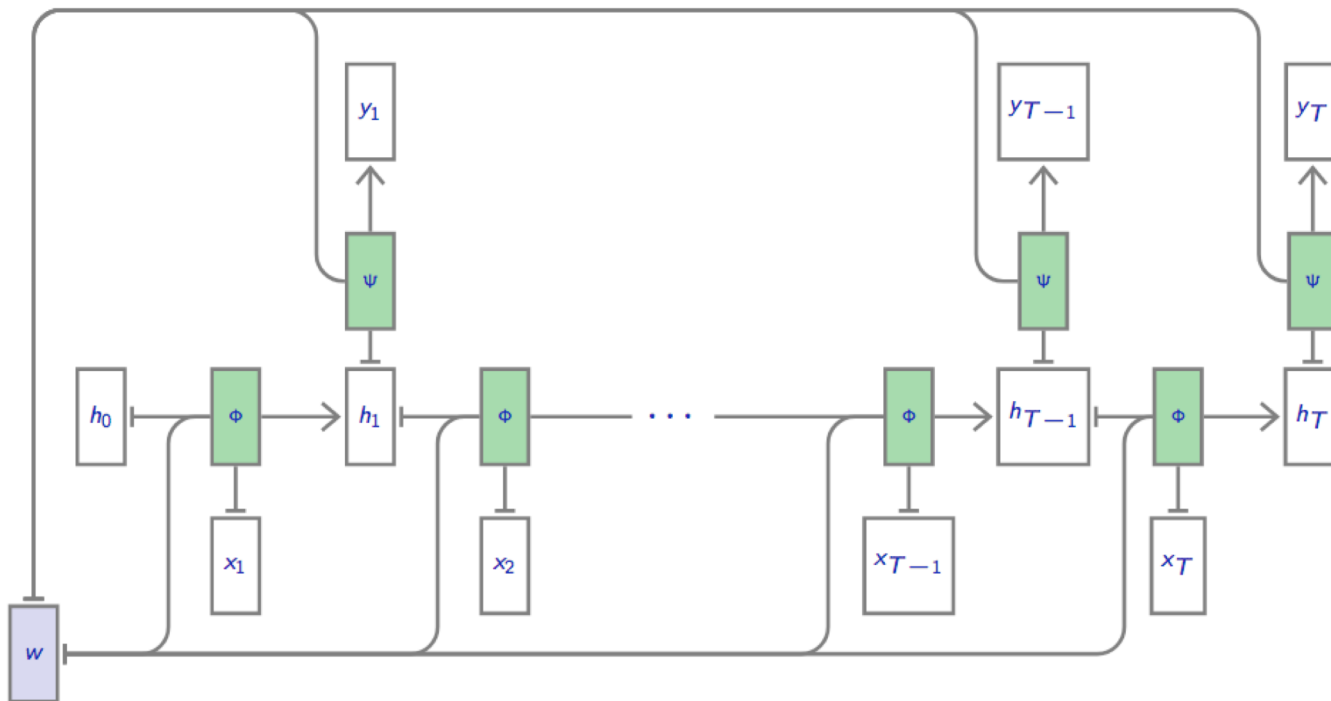




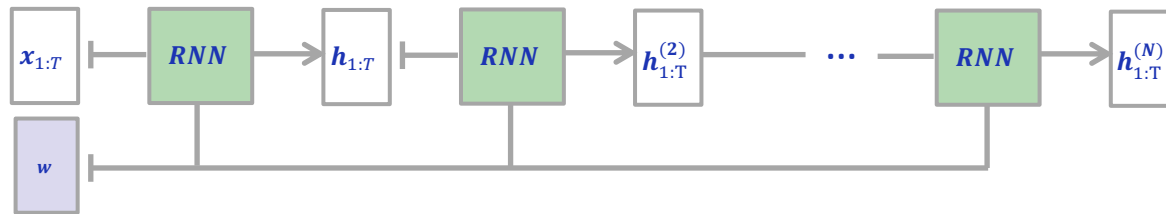


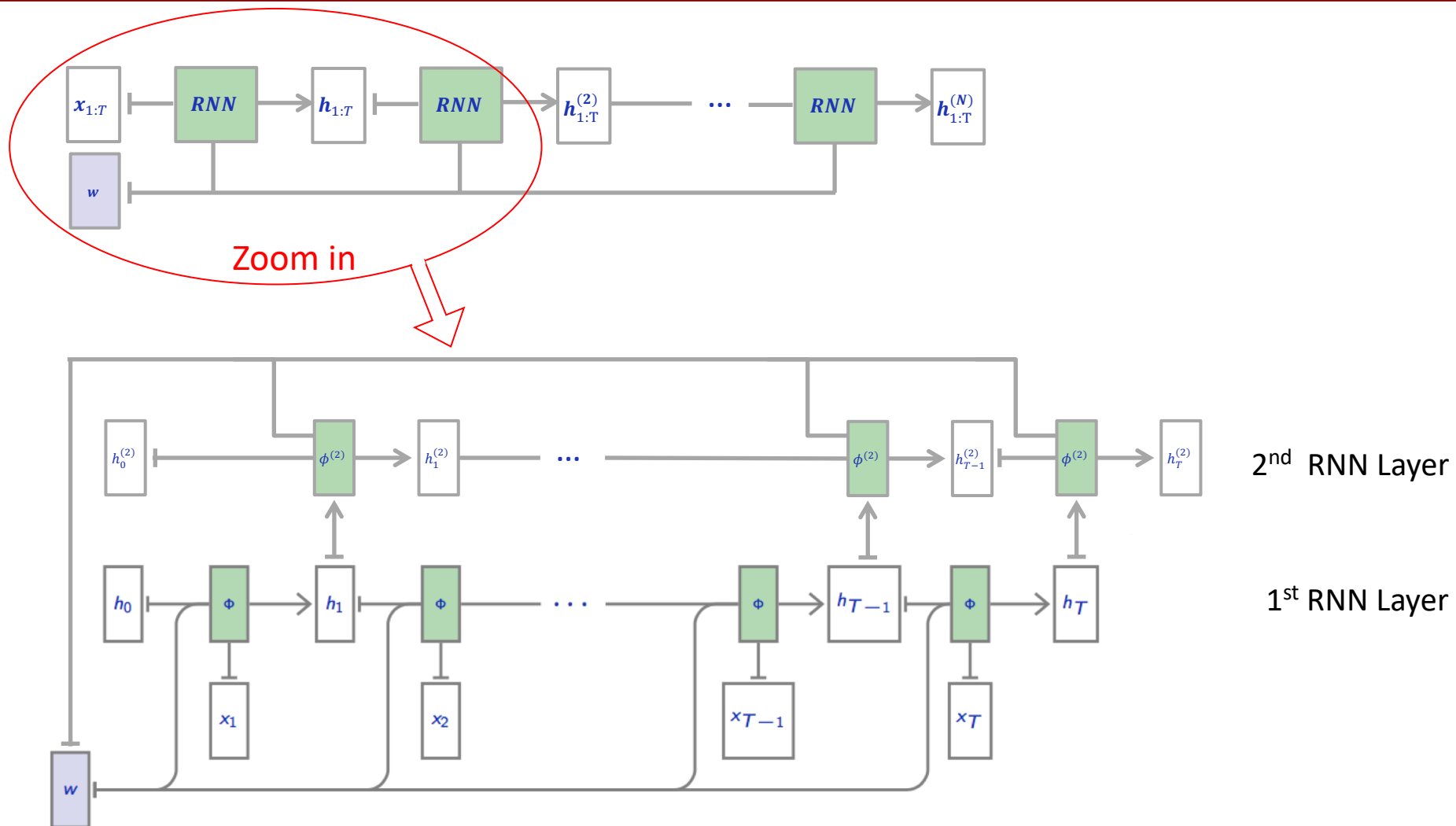


Prediction per sequence element

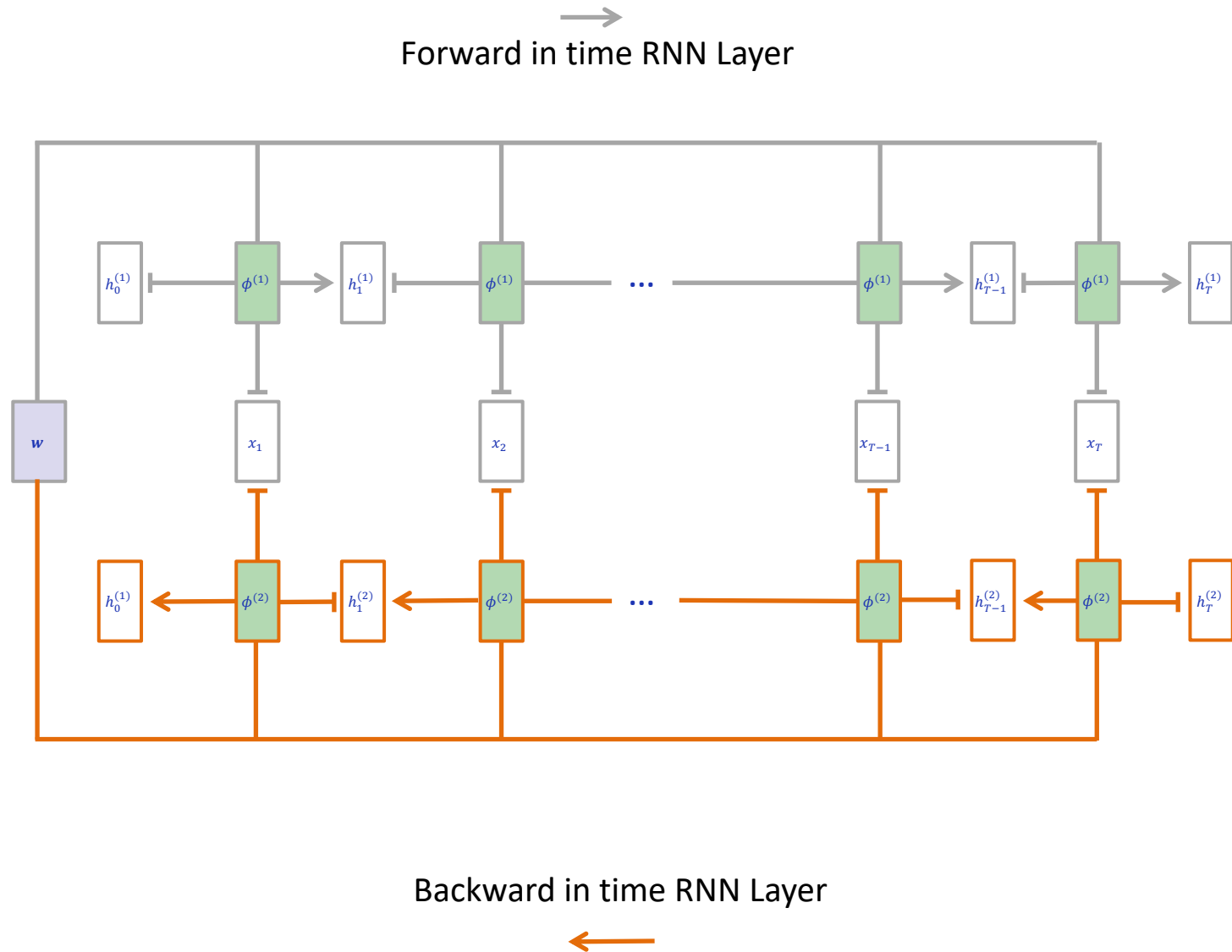


Although the number of steps $T(x)$ depends on x , this is a standard computational graph and automatic differentiation can deal with it as usual. This is known as “backpropagation through time” (Werbos, [1988](#))

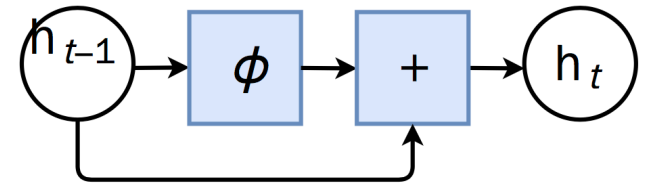




Two Stacked LSTM Layers

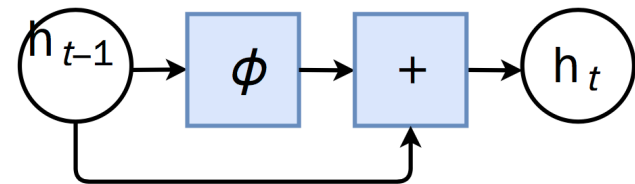


- Gating:
 - network can grow very deep, in time \rightarrow vanishing gradients.
 - *Critical component*: add pass-through (additive paths) so recurrent state does not go repeatedly through squashing non-linearity.



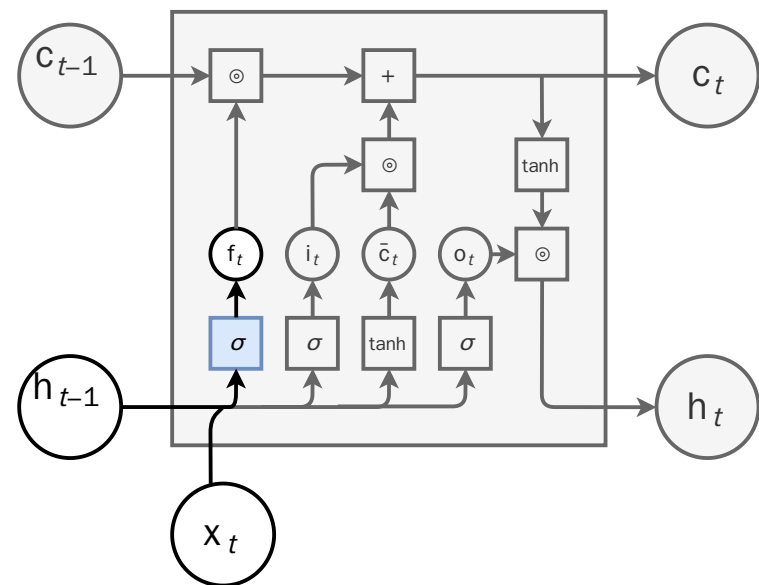
- Gating:

- network can grow very deep, in time \rightarrow vanishing gradients.
- *Critical component*: add pass-through (additive paths) so recurrent state does not go repeatedly through squashing non-linearity.



- LSTM:

- Add internal state separate from output state
- Add input, output, and forget gating

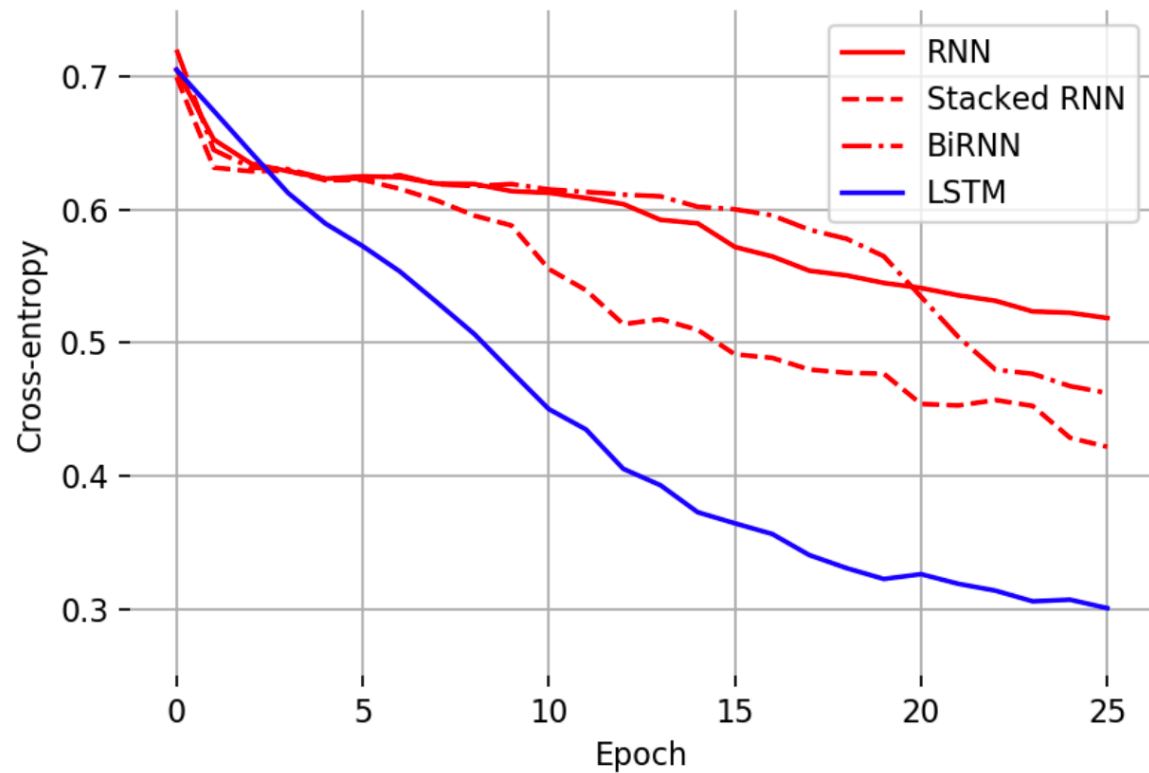


Comparison on Toy Problem

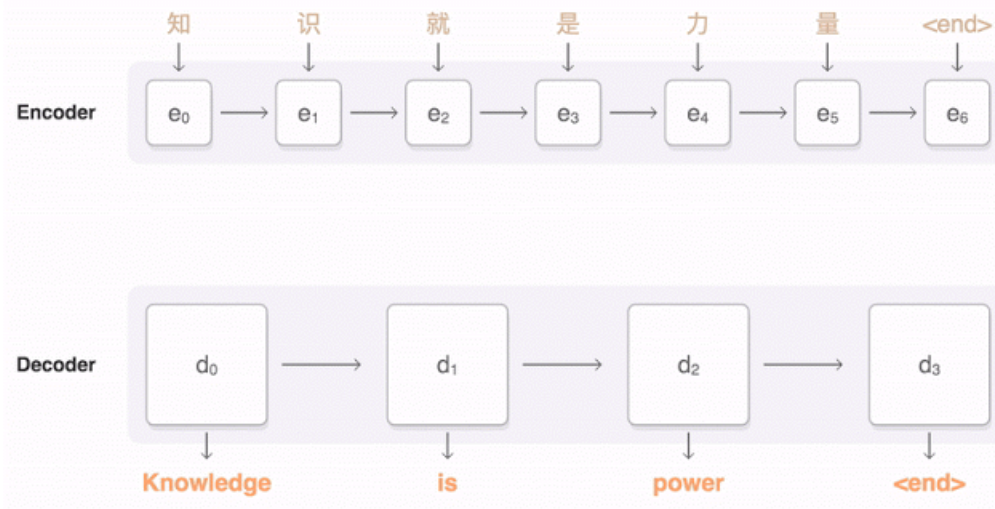
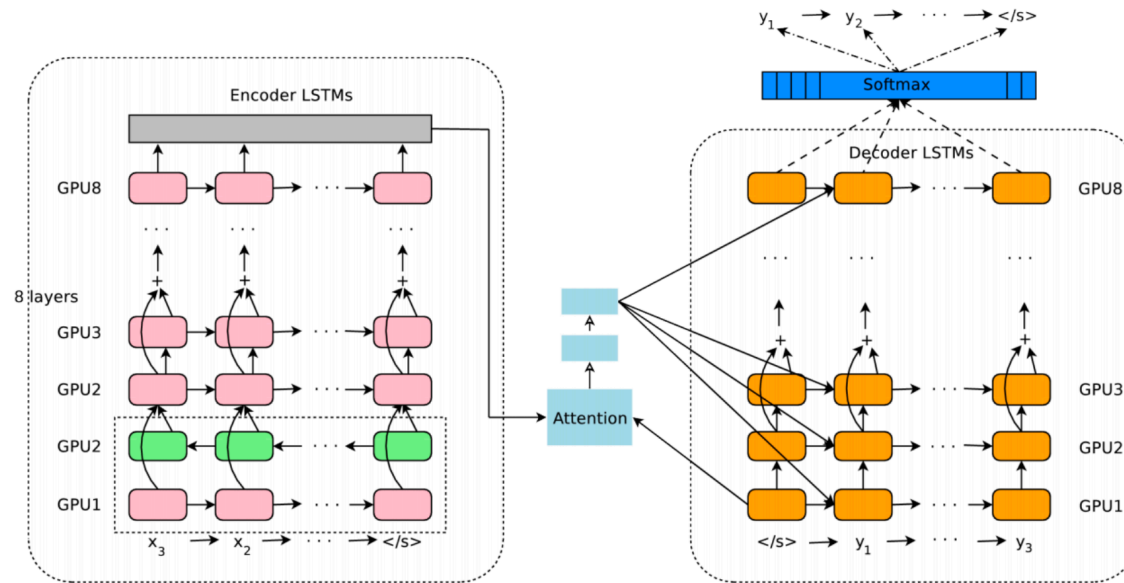
16

Learn to recognize palindrome
Sequence size between 1 to 10

x	y
(1, 2, 3, 2, 1)	1
(2, 1, 2)	1
(3, 4, 1, 2)	0
(0)	1
(1, 4)	0



Neural machine translation





MariFlow - Self-Driving Mario Kart w/Recur...

This is a recurrent neural network that I've trained to play Mario Kart like me. This NN is very different from Mari/O, because its goal is not to win, but rather to predict what controller inputs I would use in any given situation. The display on the bottom shows what the neural network sees, and its internal state and controller predictions.

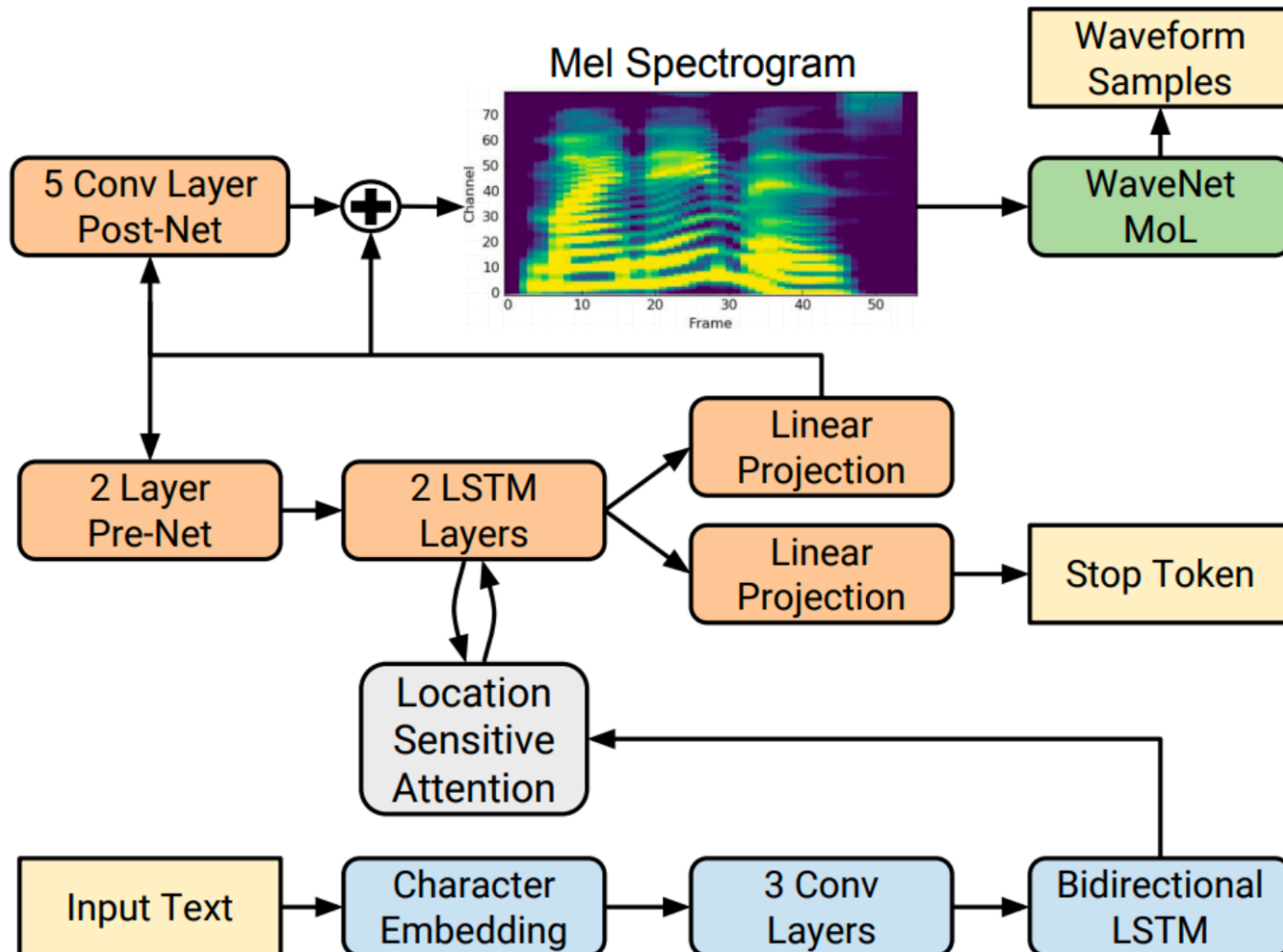
It's currently trying every cup in 50cc on repeat. The goal is to see if it can get medals in each cup. I'm not actually present. If you see something interesting clip it so I can see it later and potentially include it in my video!

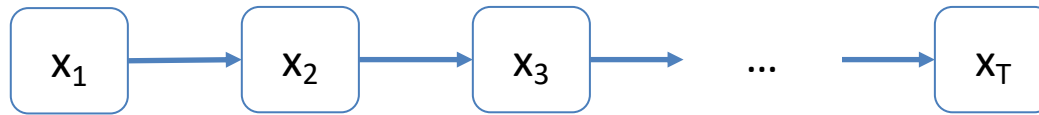
MORE VIDEOS

01:34:58

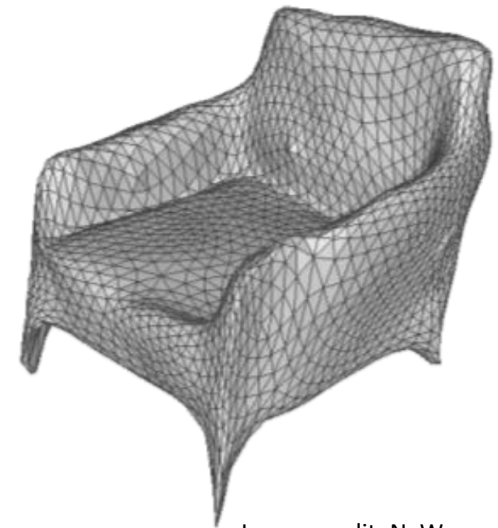
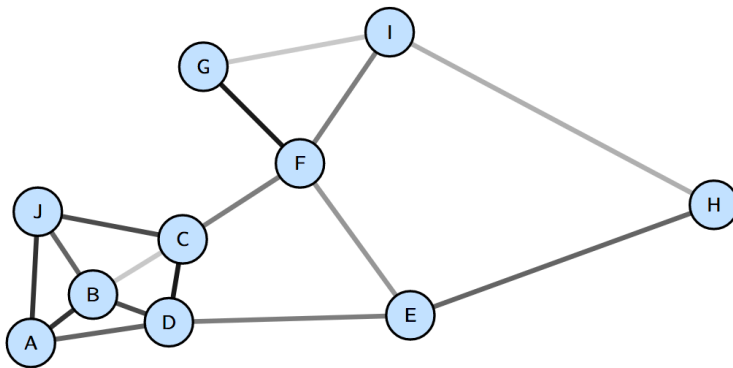
YouTube

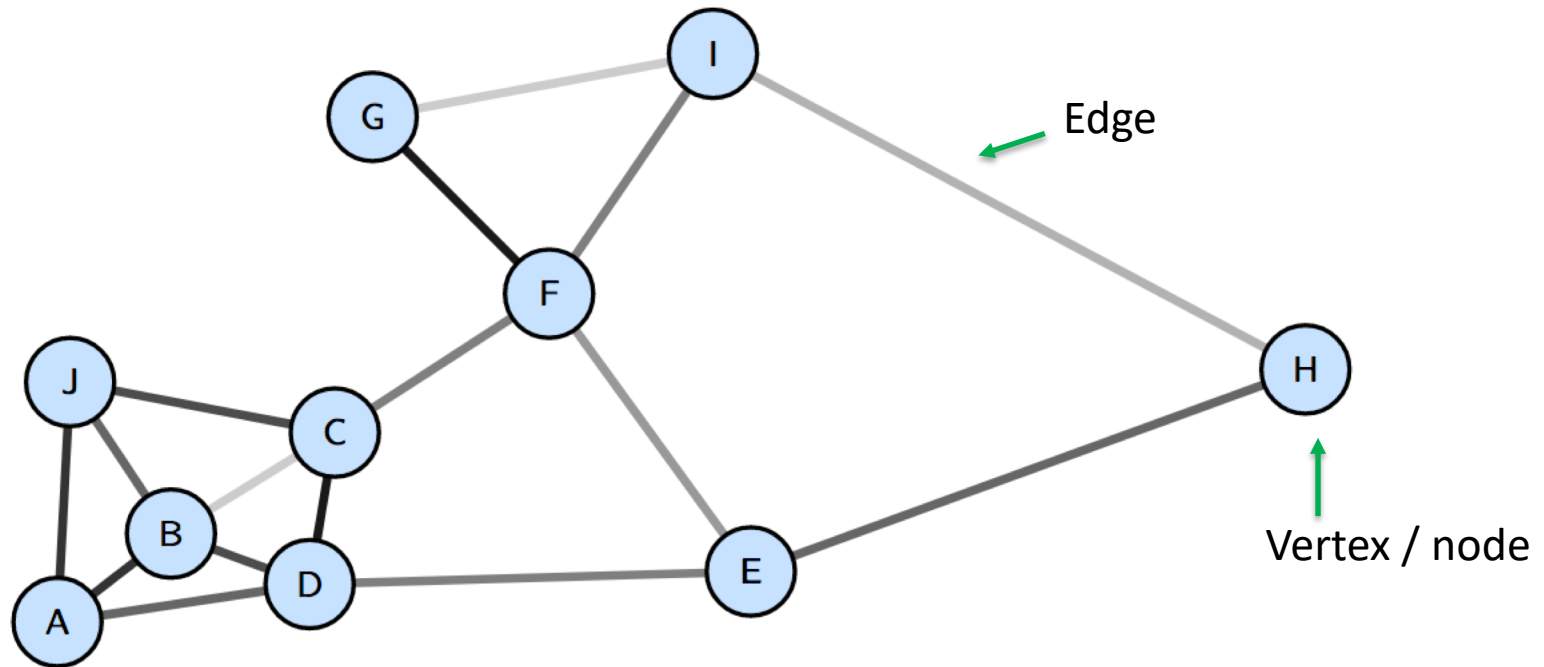
Text-to-speech synthesis



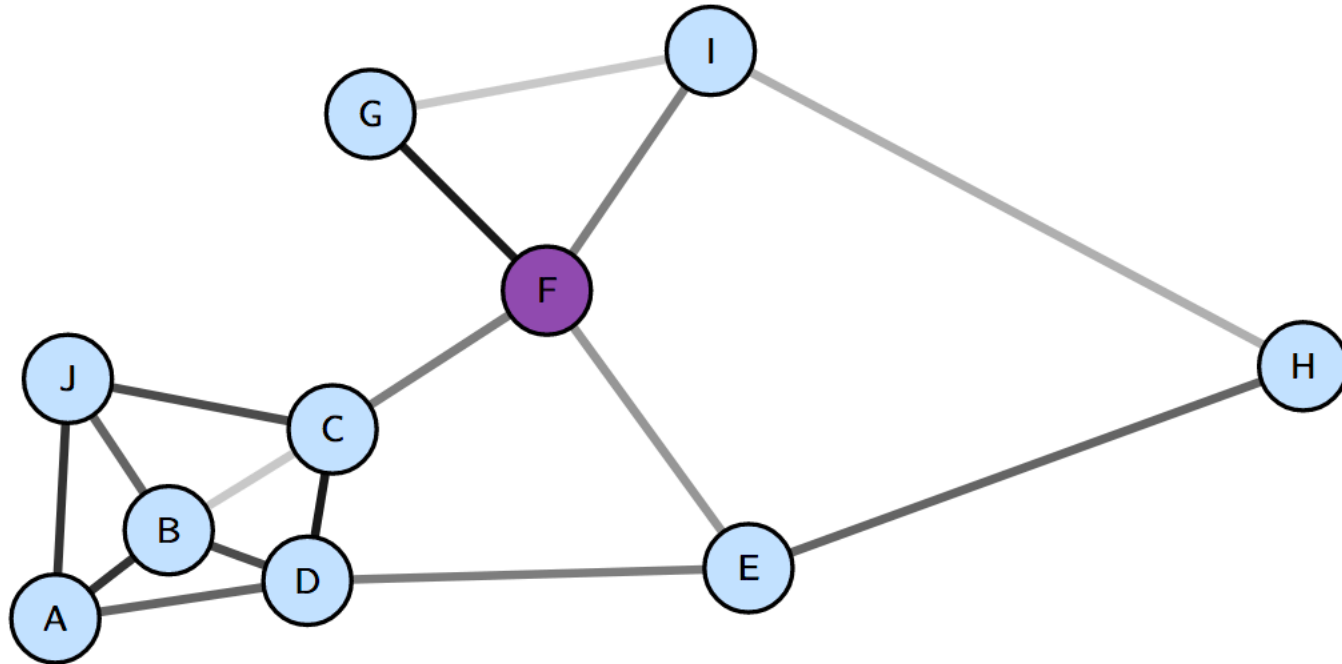


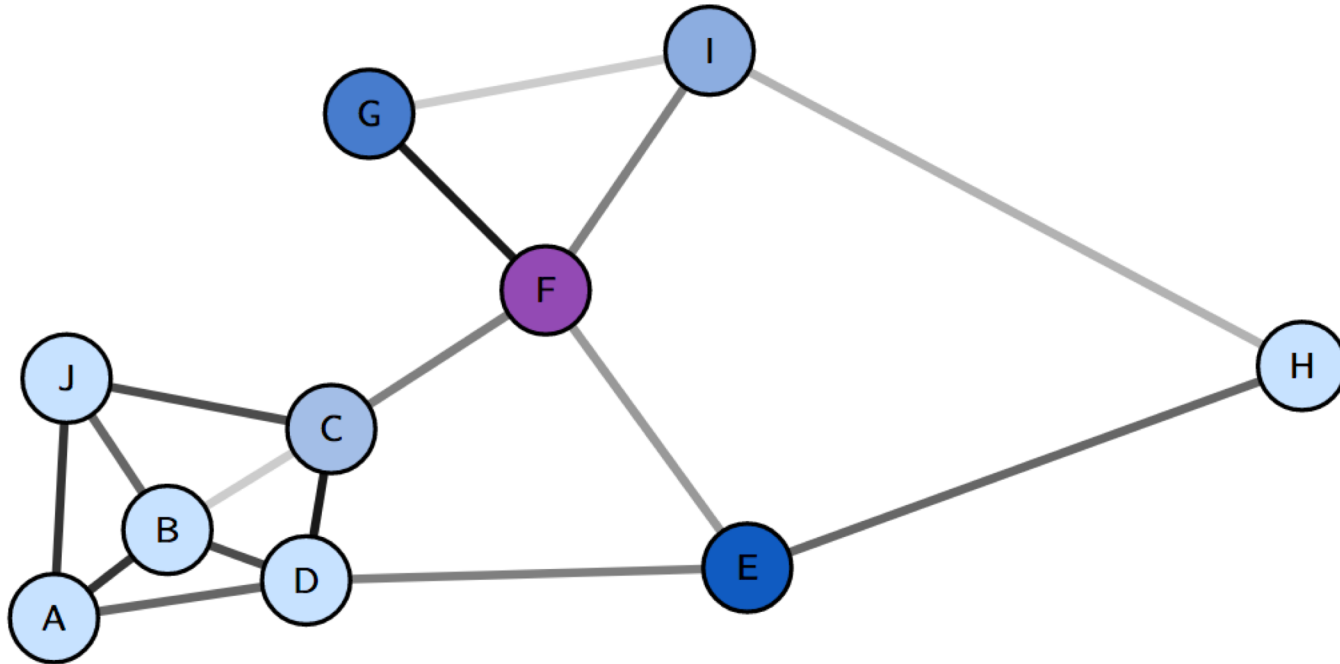
- Sequential data has single (directed) connections from data at current time to data at next time
- What about data with more complex dependencies



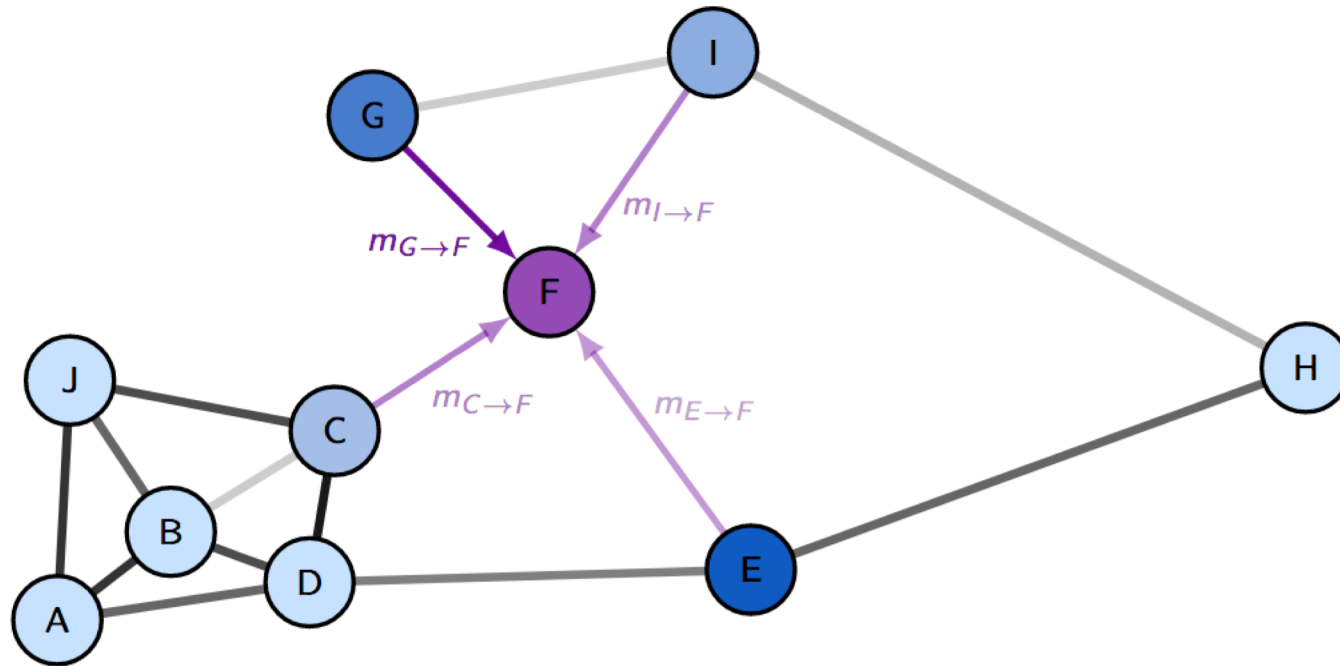


- Adjacency matrix: $A_{ij} = \delta(\text{edge between vertex } i \text{ and } j)$
- Each node can have features
- Each edge can have features, e.g. distance between nodes

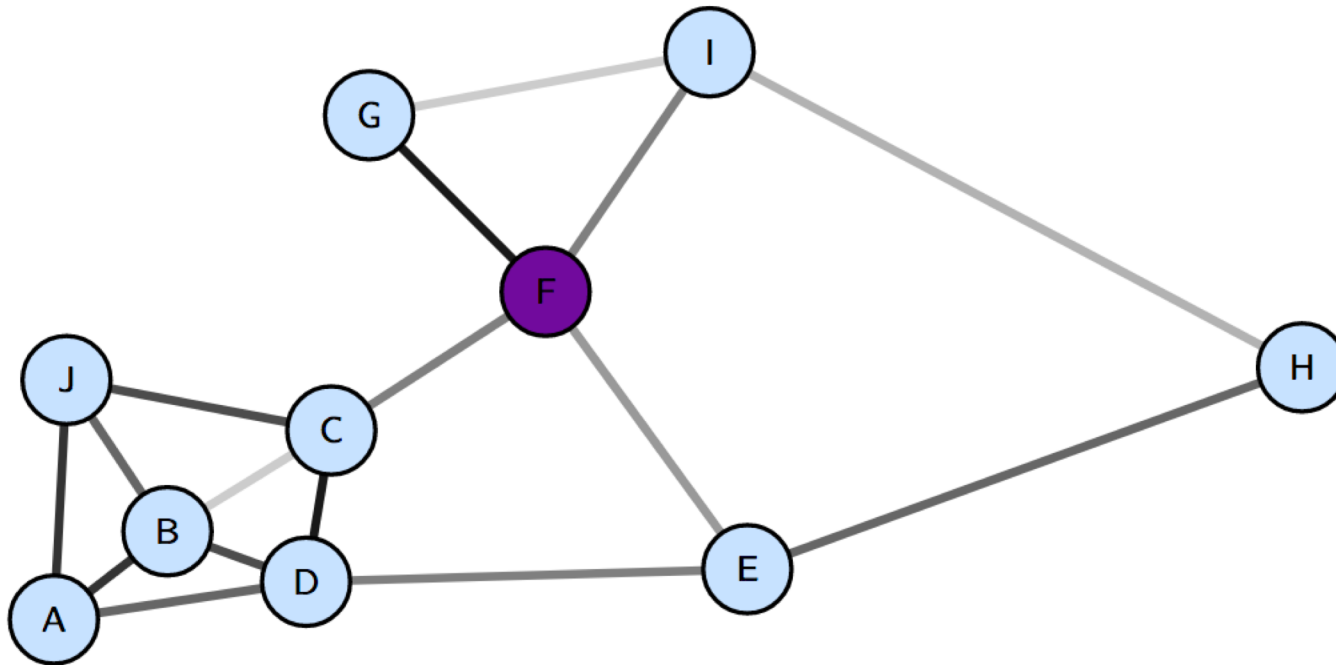




$$\tilde{m}_j^t = f(h_j^{t-1})$$



$$\tilde{m}_j^t = f(h_j^{t-1})$$
$$m_{j \rightarrow i}^t = \sigma(A_{ij} \tilde{m}_j^t)$$



$$\begin{aligned}\tilde{m}_j^t &= f(h_j^{t-1}) \\ m_{j \rightarrow i}^t &= \sigma(A_{ij} \tilde{m}_j^t) \\ h_i^t &= \text{GRU}(h_i^{t-1}, \sum_j m_{j \rightarrow i}^t)\end{aligned}$$

Algorithm 1 Message passing neural network

Require: $N \times D$ nodes \mathbf{x} , adjacency matrix A

$\mathbf{h} \leftarrow \text{Embed}(\mathbf{x})$

for $t = 1, \dots, T$ **do**

$\mathbf{m} \leftarrow \text{Message}(A, \mathbf{h})$

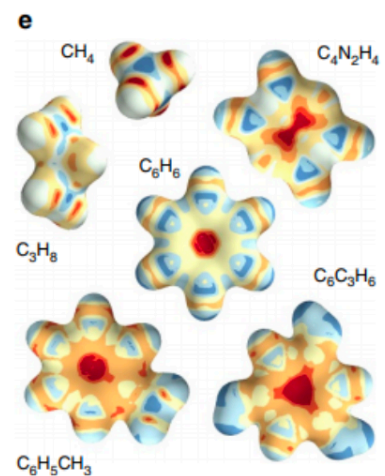
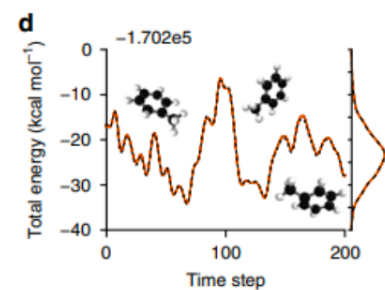
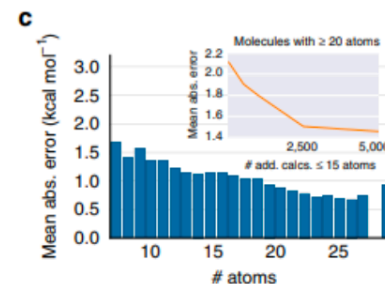
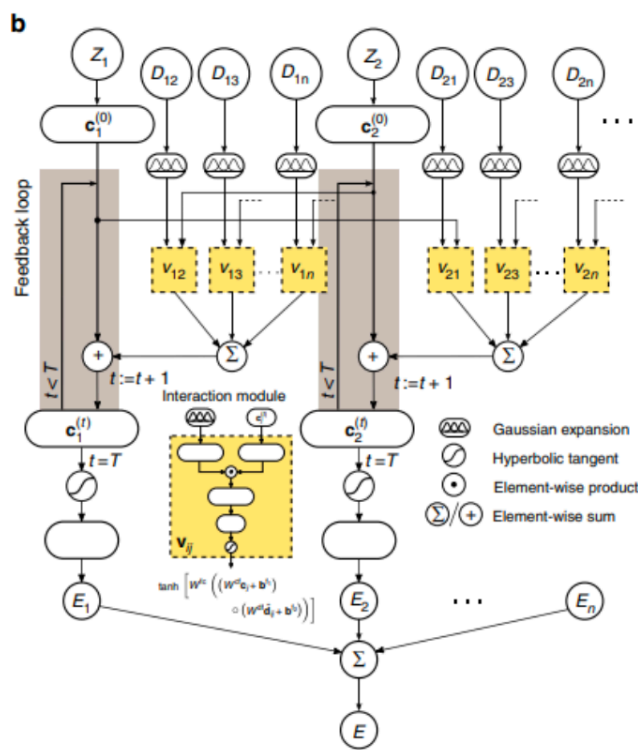
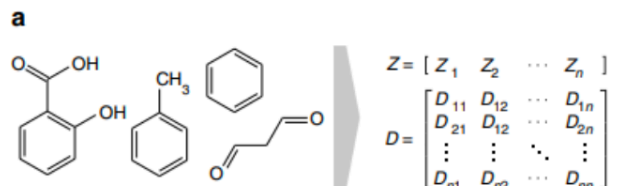
$\mathbf{h} \leftarrow \text{VertexUpdate}(\mathbf{h}, \mathbf{m})$

end for

$\mathbf{r} = \text{Readout}(\mathbf{h})$

return $\text{Classify}(\mathbf{r})$

Quantum chemistry with graph networks



Learning to simulate physics with graph networks

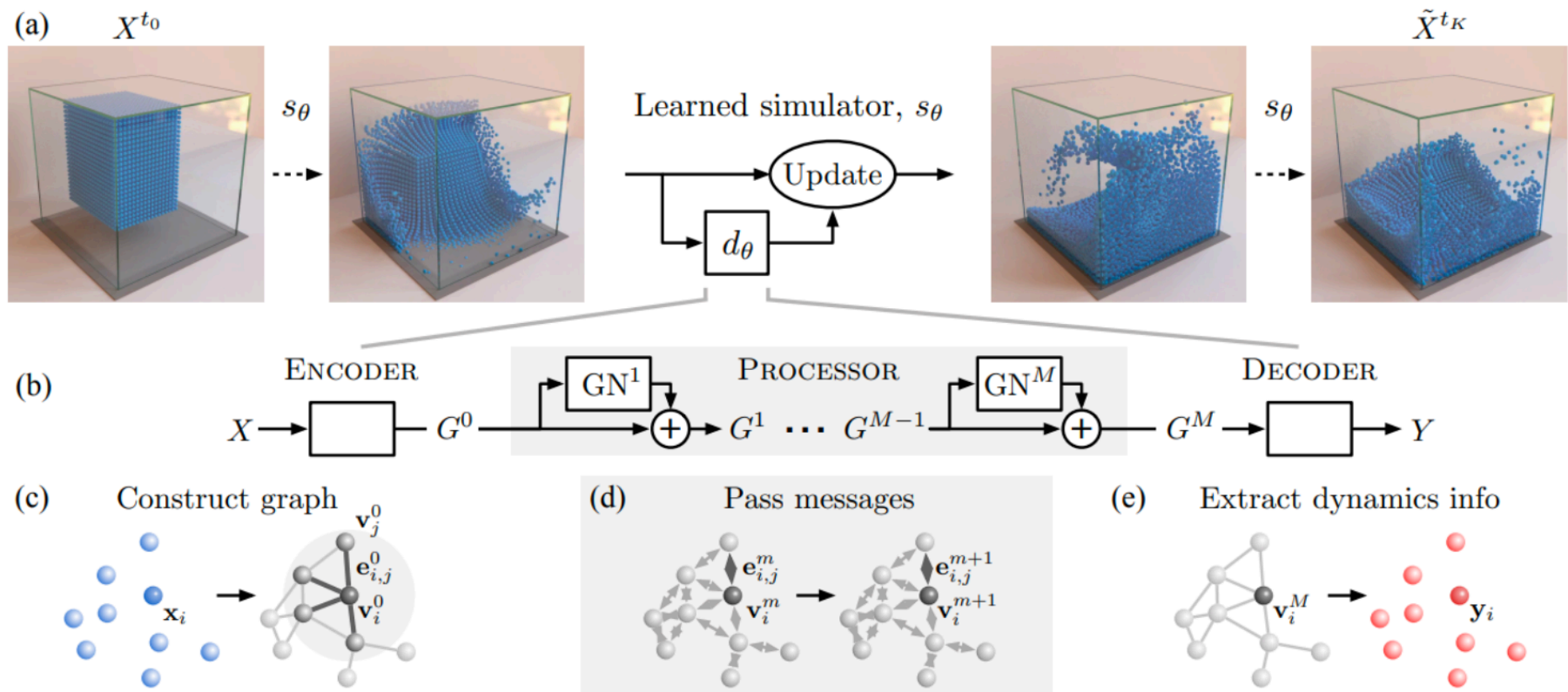
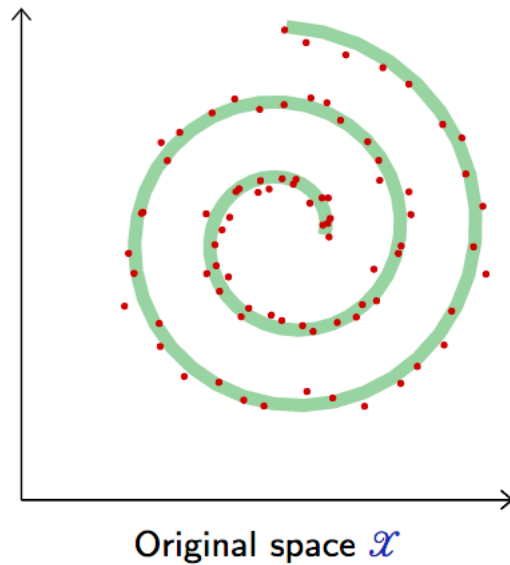
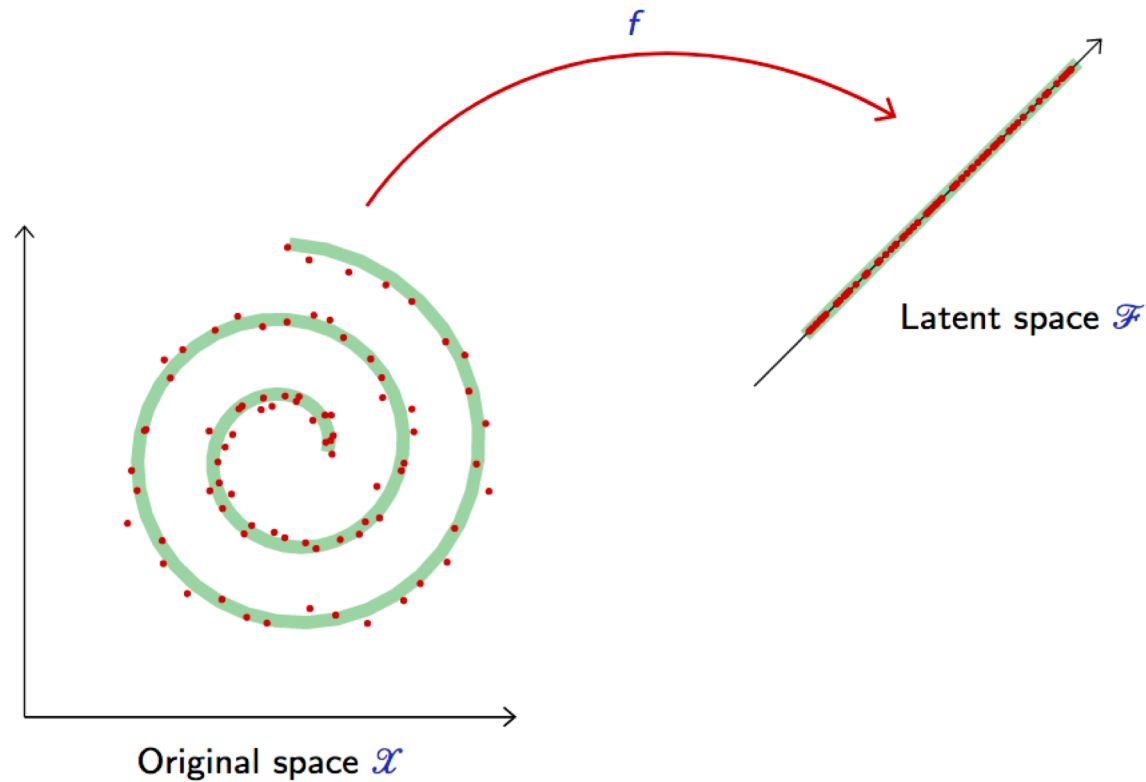


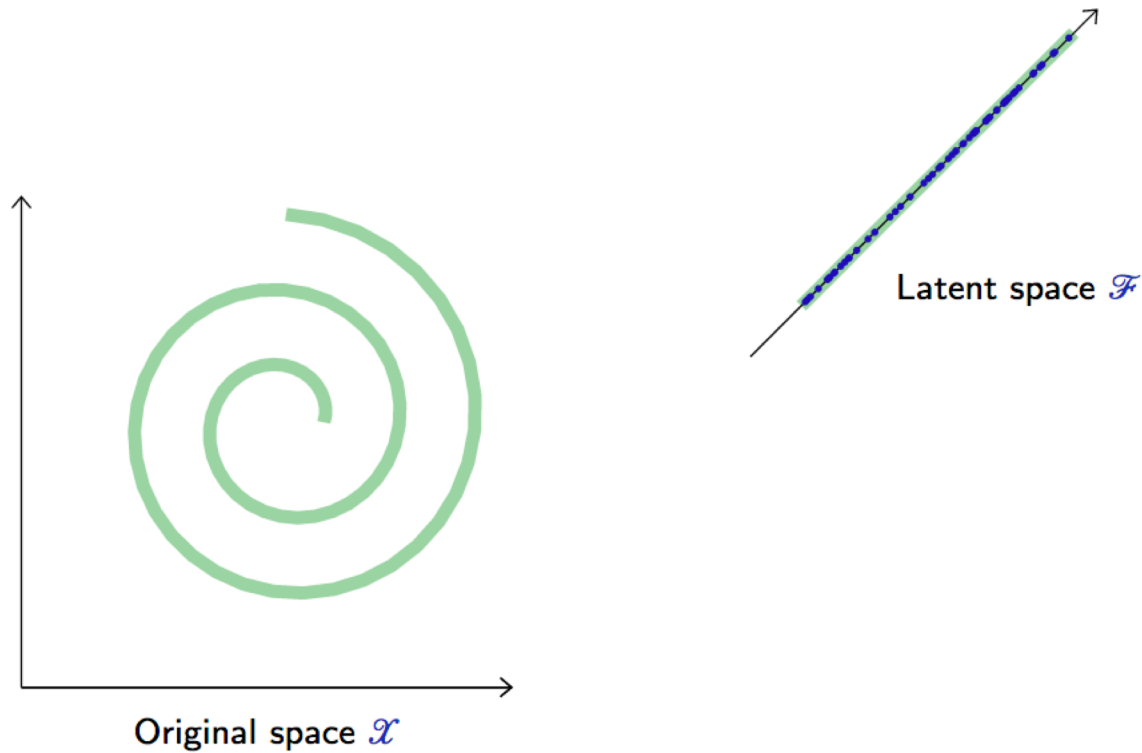
Figure 2. (a) Our GNS predicts future states represented as particles using its learned dynamics model, d_θ , and a fixed update procedure. (b) The d_θ uses an “encode-process-decode” scheme, which computes dynamics information, Y , from input state, X . (c) The ENCODER constructs latent graph, G^0 , from the input state, X . (d) The PROCESSOR performs M rounds of learned message-passing over the latent graphs, G^0, \dots, G^M . (e) The DECODER extracts dynamics information, Y , from the final latent graph, G^M .

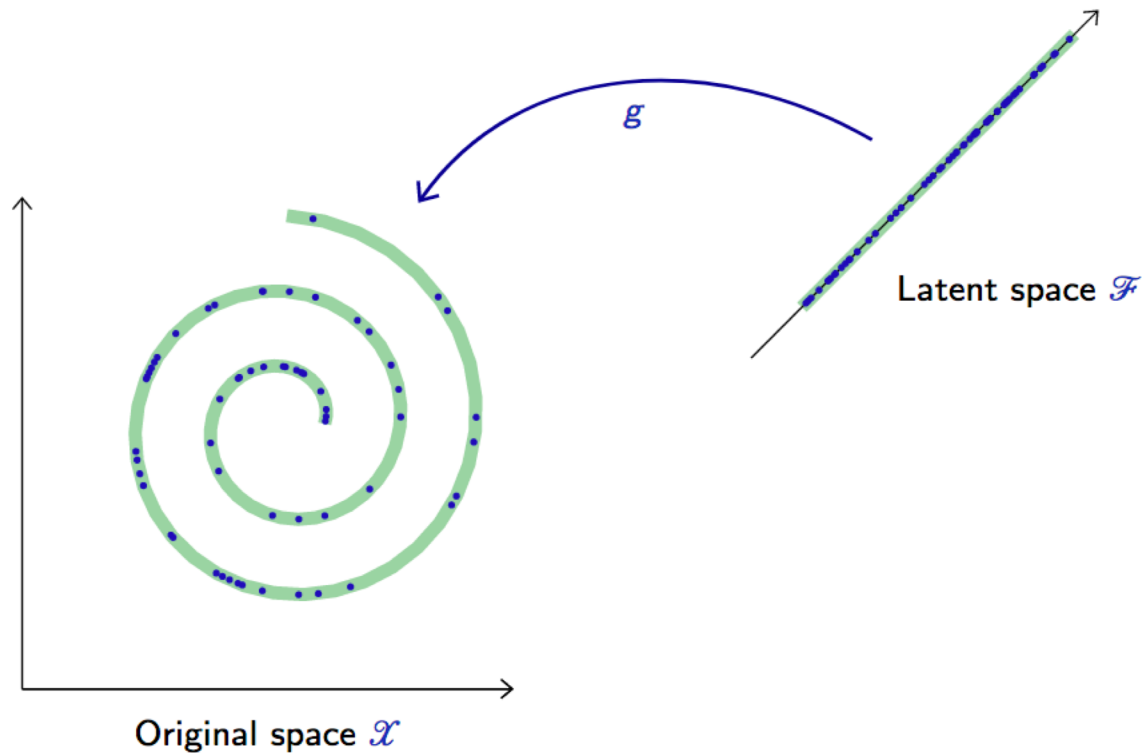
Beyond Regression and Classification

- Not all tasks are predicting a label from features, as in classification and regression
- May want / need to explicitly model a high-dim. signal
 - Data synthesis / simulation
 - Density estimation
 - Anomaly detection
 - Denoising, super resolution
 - Data compression
 - ...
- Often don't have labels → Unsupervised Learning
- Often framed as **modeling the lower dimensional “meaningful degrees of freedom”** that describe the data









- Must first determine the question we want to ask, and formulate an appropriate loss function
 - Loss function encodes the quality of model prediction
 - Parameterize models with neural networks
- Will have many of the same theoretical and practical issues as in classification and regression
 - What is the right class and structure of the model (CNN, RNN, graph, etc.)?
 - How do we stably optimize the loss w.r.t. parameters?

Autoencoders

- How can we find the “meaningful degrees of freedom” in the data?
- Dimensionality Reduction / Compression
 - Can we compress the data to a *latent space* with smaller number of dimensions, and still recover the original data from this latent space representation?
 - Latent space must encode and retain the important information about the data
 - Can we learn this compression and latent space

- Autoencoders map a space to itself through a compression, $x \rightarrow z \rightarrow \hat{x}$, and should be close to the identity on the data
 - Data: $x \in \mathcal{X}$ Latent space: $z \in \mathcal{F}$
 - **Encoder**: Map from \mathcal{X} to a lower dimensional latent space \mathcal{F}
 - Parameterize as neural network $f_{\theta}(x)$ with parameters θ
 - **Decoder**: Map from latent space \mathcal{F} back to data space \mathcal{X}
 - Parameterize as neural network $g_{\psi}(z)$ with parameters ψ

- Autoencoders map a space to itself through a compression, $x \rightarrow z \rightarrow \hat{x}$, and should be close to the identity on the data
 - Data: $x \in \mathcal{X}$ Latent space: $z \in \mathcal{F}$
 - **Encoder**: Map from \mathcal{X} to a lower dimensional latent space \mathcal{F}
 - Parameterize as neural network $f_{\theta}(x)$ with parameters θ
 - **Decoder**: Map from latent space \mathcal{F} back to data space \mathcal{X}
 - Parameterize as neural network $g_{\psi}(z)$ with parameters ψ
- What is the latent space? What are $f(x)$ and $g(z)$?
 - Choose a latent space dimension D
 - Learn mappings $f(x)$ to representation of size D , and back with $g(z)$

- Loss: mean *reconstruction loss* (MSE) between data and encoded-decoded data

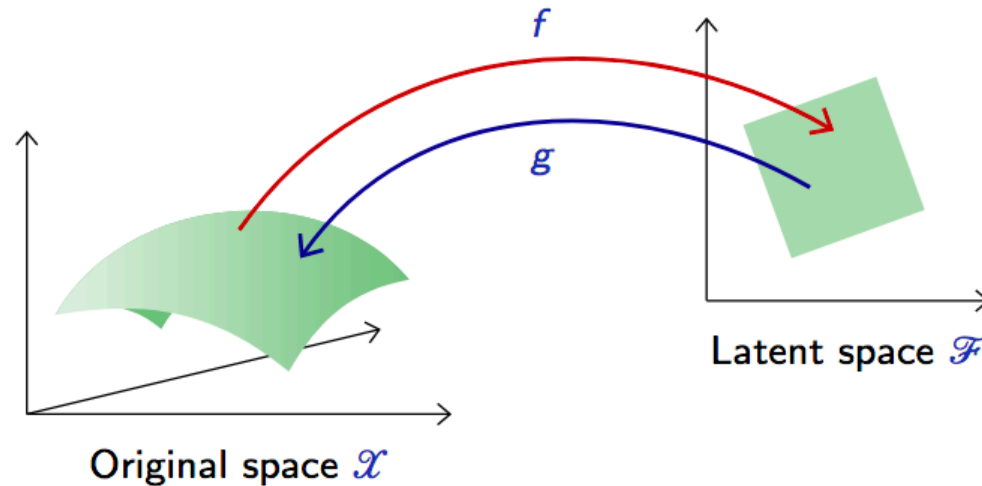
$$L(\theta, \psi) = \frac{1}{N} \sum_n \|x_n - g_\psi(f_\theta(x_n))\|^2$$

- Minimize this loss over parameters of encoder (θ) and decoder (ψ).

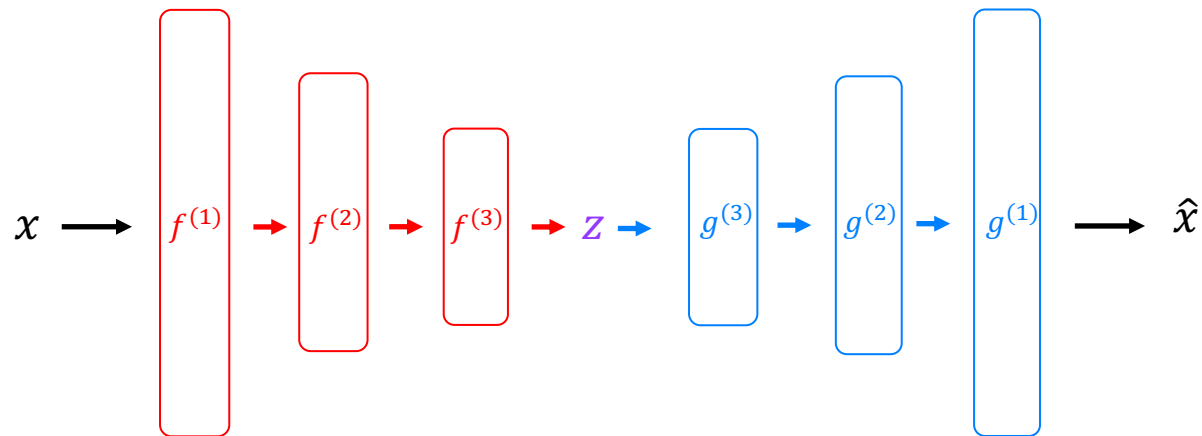
- Loss: mean *reconstruction loss* (MSE) between data and encoded-decoded data

$$L(\theta, \psi) = \frac{1}{N} \sum_n \|x_n - g_\psi(f_\theta(x_n))\|^2$$

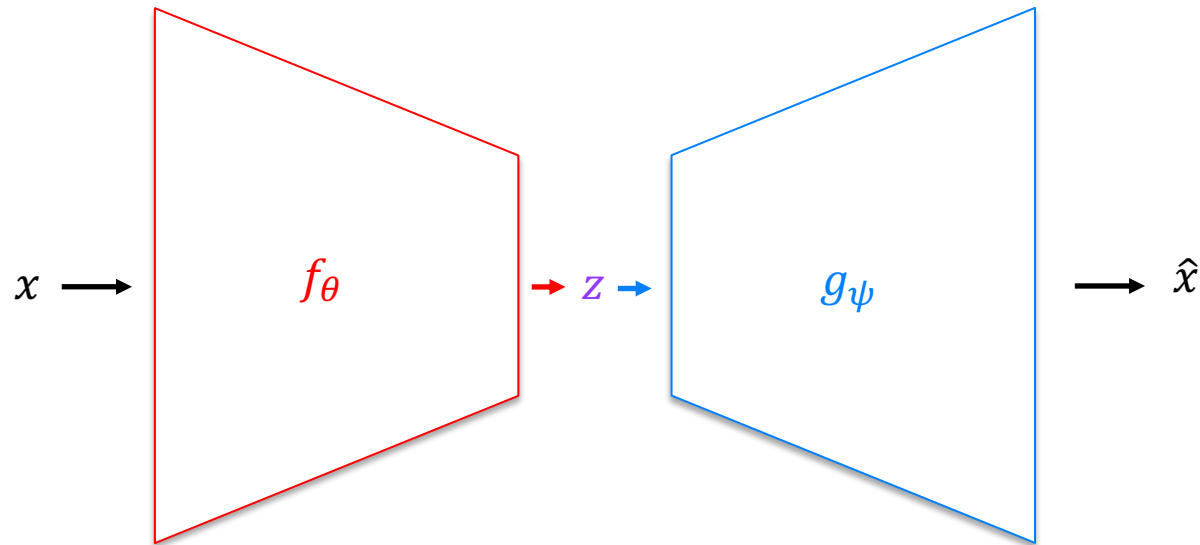
- Minimize this loss over parameters of encoder (θ) and decoder (ψ).
- NOTE: if $f_\theta(x)$ and $g_\psi(z)$ are linear, optimal solution given by Principle Components Analysis



- If the latent space is of lower dimension, the autoencoder has to capture a “good” parametrization, and in particular dependencies between components



- When f_{θ} and g_{ψ} are multiple neural network layers, can learn complex mappings between \mathcal{X} and \mathcal{F}
 - f_{θ} and g_{ψ} can be Fully Connected, CNNs, RNNs, etc.
 - Choice of network structure will depend on data



- When f_θ and g_ψ are multiple neural network layers, can learn complex mappings between \mathcal{X} and \mathcal{F}
 - f_θ and g_ψ can be Fully Connected, CNNs, RNNs, etc.
 - Choice of network structure will depend on data

X (original samples)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

$g \circ f(X)$ (CNN, $d = 16$)



f_θ and g_ψ are each
5 convolutional layers

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

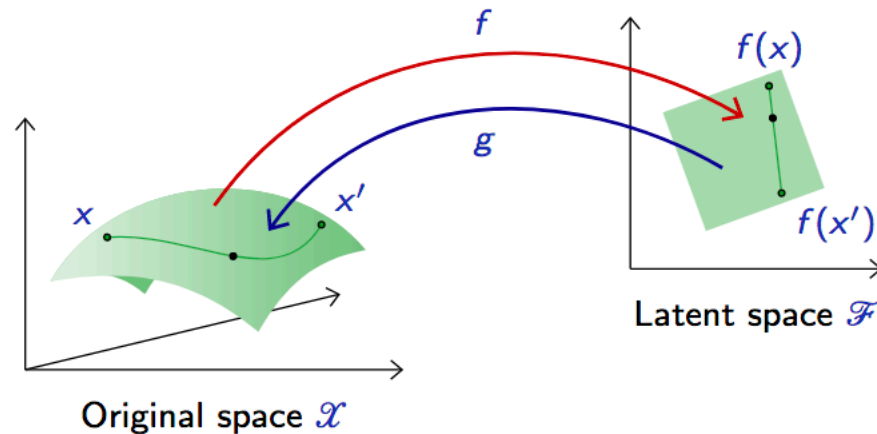
$g \circ f(X)$ (PCA, $d = 16$)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Interpolating in Latent Space

47

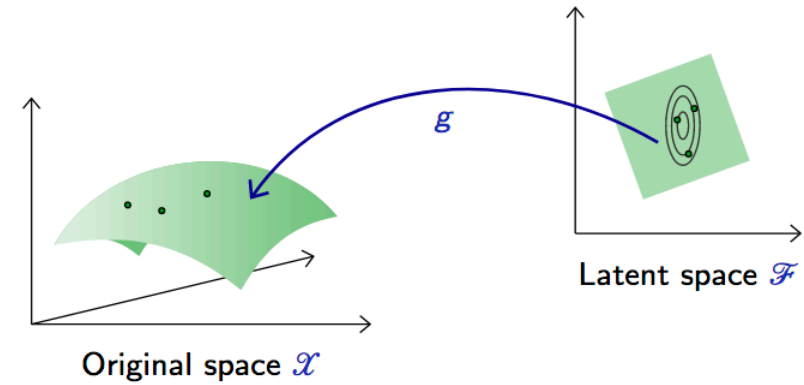
$$\alpha \in [0, 1], \quad \xi(x, x', \alpha) = g((1 - \alpha)f(x) + \alpha f(x')).$$



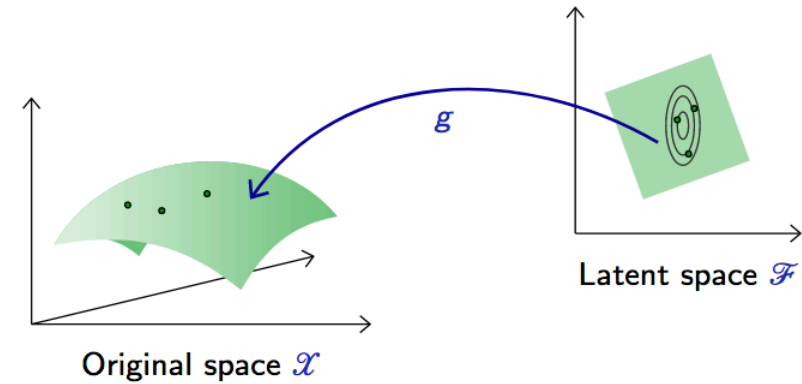
Autoencoder interpolation ($d = 8$)



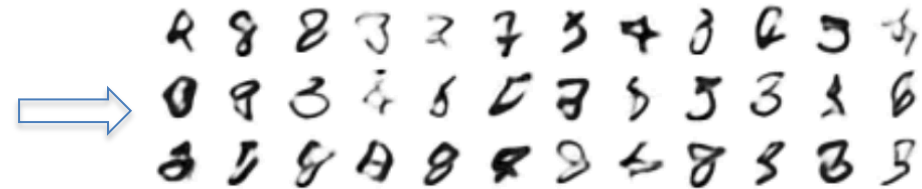
- Can we sample in latent space and decode to generate data?



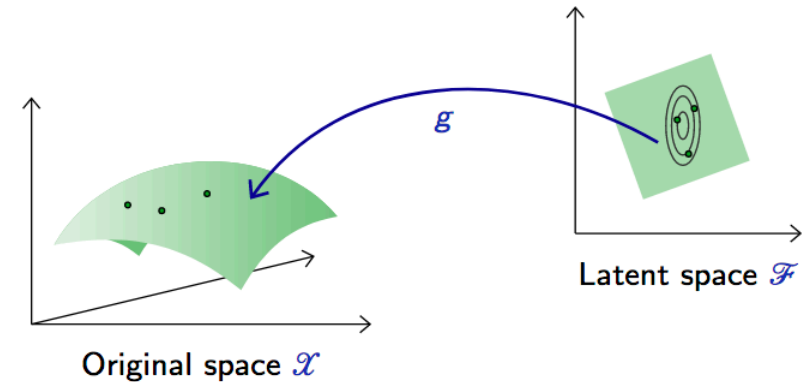
- Can we sample in latent space and decode to generate data?
- What distribution to sample from in latent space?
 - Try Gaussian with mean and variance from data



Autoencoder sampling ($d = 16$)

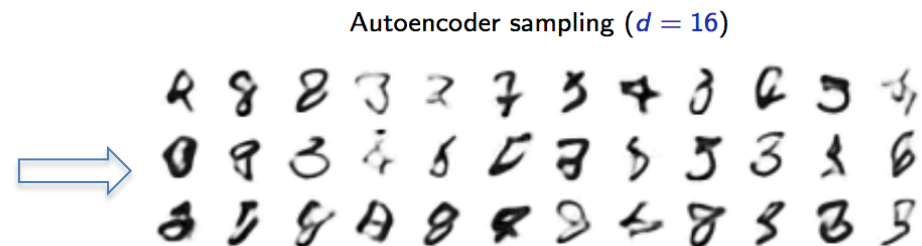


- Can we sample in latent space and decode to generate data?



- What distribution to sample from in latent space?

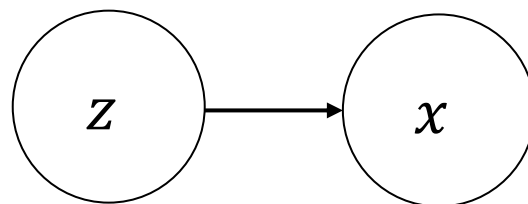
- Try Gaussian with mean and variance from data



- Doesn't work! Don't know the right latent space density
 - Don't have model of where the encoder encodes!

Generative Models

- Generative models aim to:
 - Learn a distribution $p(x)$ that explains the density of the data
 - Draw samples of plausible data points
- Explicit Models
 - Can evaluate the density $p(x)$ of a data point x
- Implicit Models
 - Can only sample from $p(x)$, but not evaluate density



- Observed random variable x depends on unobserved latent random variable z
 - Interpret z as the causal factors for x
- Joint probability: $p(x, z) = p(x|z)p(z)$
- $p(x|z)$ is a stochastic generation process from $z \rightarrow x$
- Inference from posterior:
$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$
 - Usually can't compute marginal $p(x) = \int p(x|z)p(z)dz$

- Consider probabilistic relationship between data and latent variables

$$x, z \sim p(x, z) = p(x|z)p(z)$$



Decoding data x
from latent z

Prior over latent space

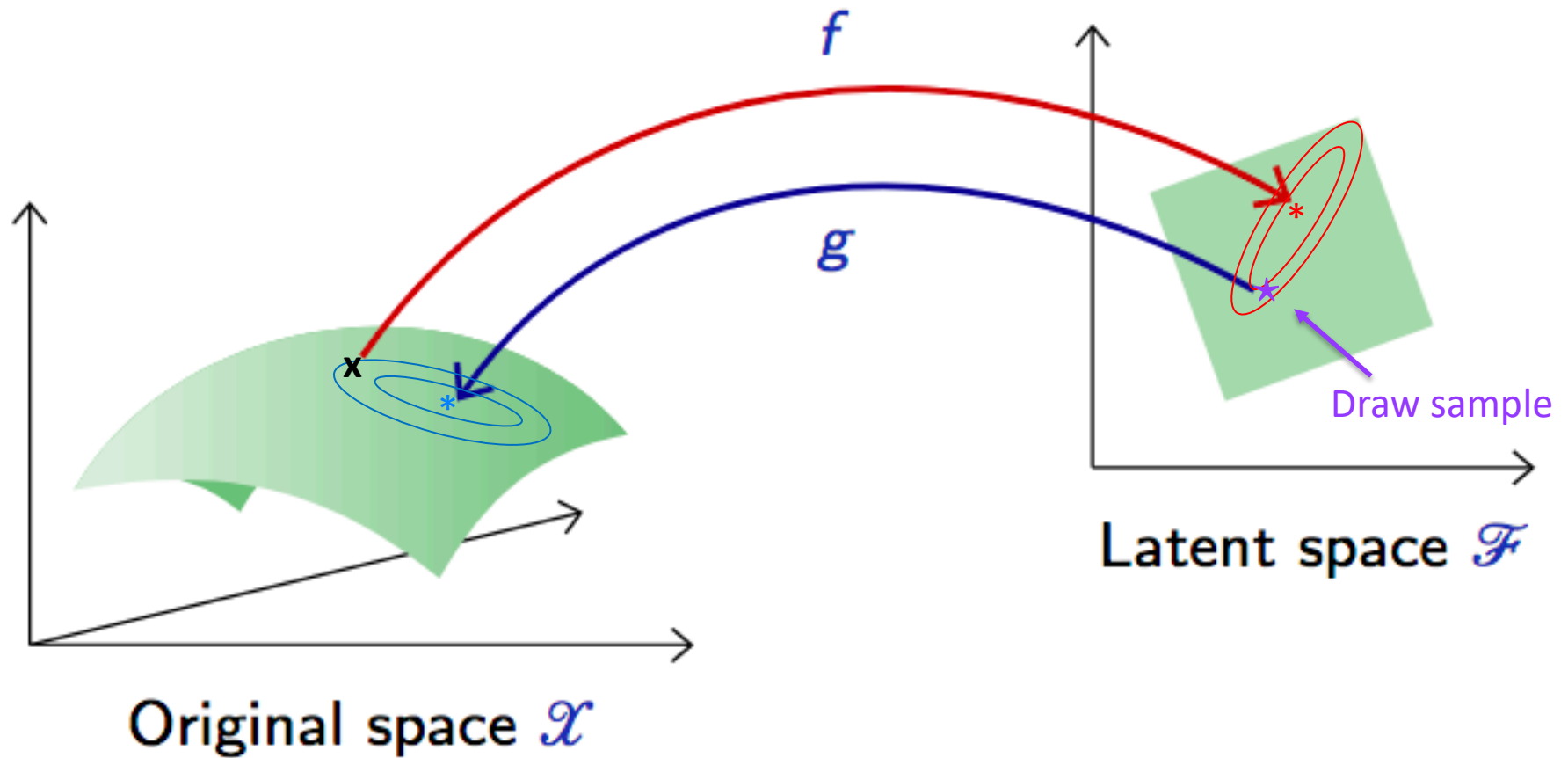
- Consider probabilistic relationship between data and latent variables

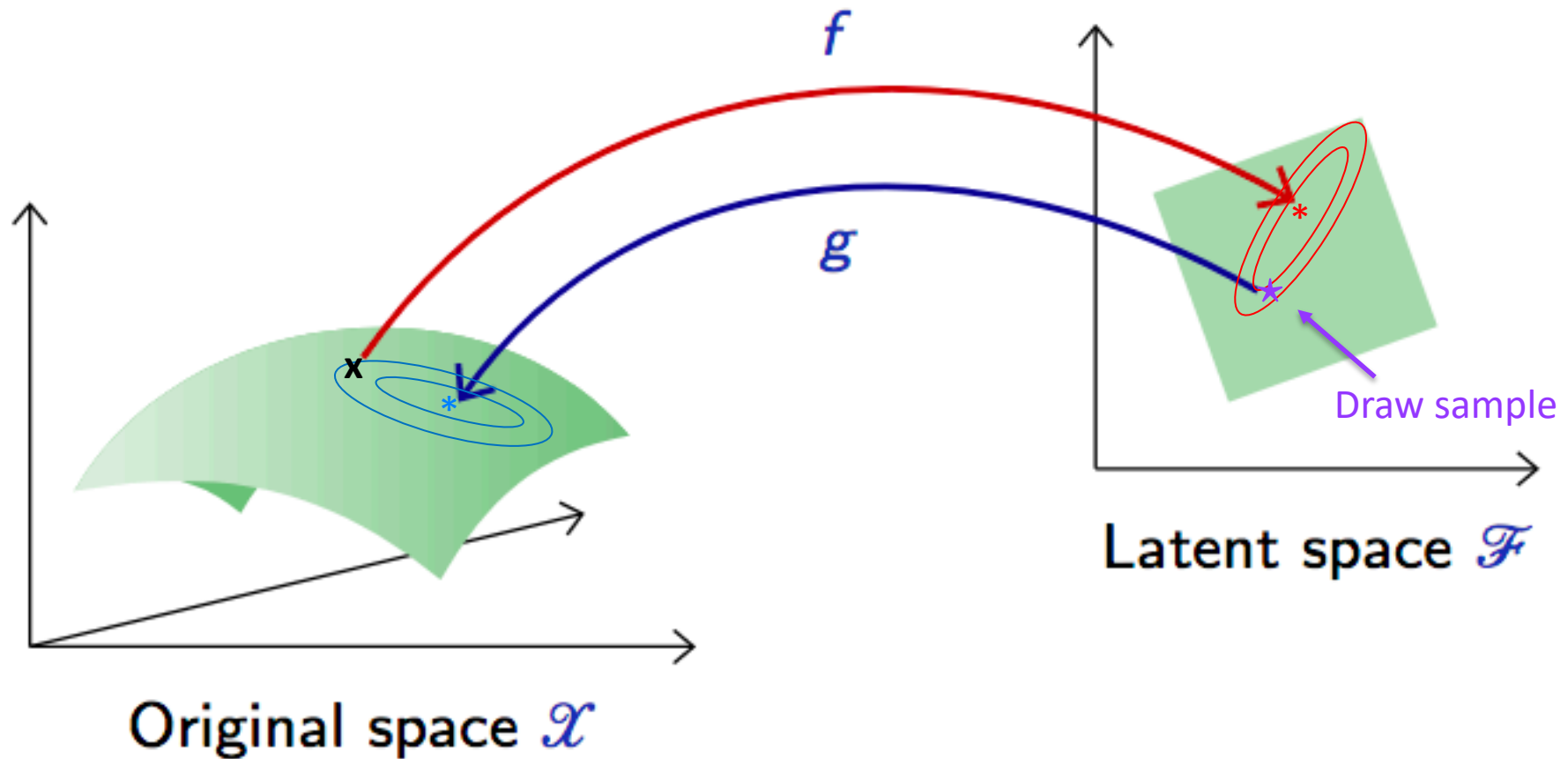
$$x, z \sim p(x, z) = p(x|z)p(z)$$

- Autoencoding

$$x \rightarrow q(z|x) \xrightarrow[\text{sample}]{} z \rightarrow p(x|z)$$

- Choose simple prior distribution
- **Encoder:** Learn what latents can produced data: $q(z|x)$
- **Decoder:** Learn what data is produced by latent: $p(x|z)$

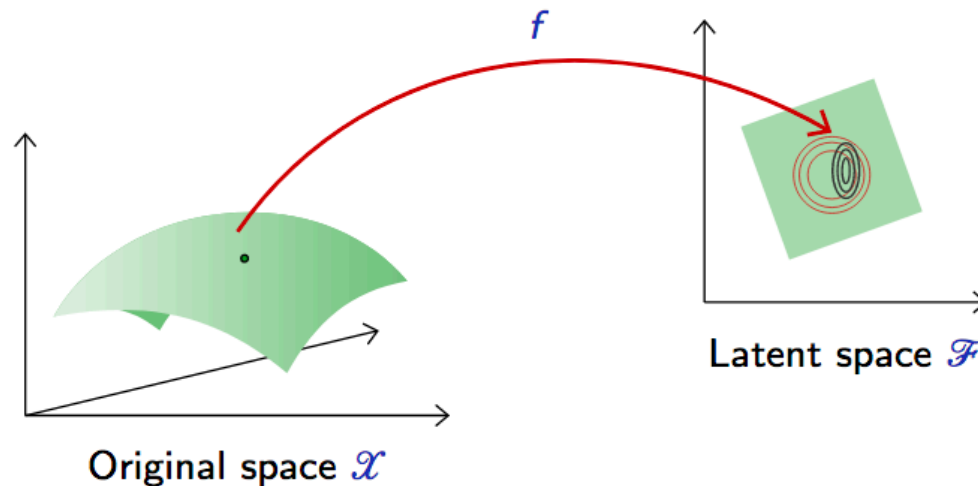




Reconstruction Loss: Maximize expected likelihood of decoding x from encodings of x

$$L_{reco} = \mathbb{E}_{z \sim q(z|x)} [\log p(x|z)] \approx \frac{1}{N} \sum_{z_i \sim q(z|x)} \log p(x|z_i)$$

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_{\psi}(z|x)} \log p_{\theta}(x|z_i)$
- Prior $p(z)$ describes the latent space distribution, **need to ensure the encoder is consistent with prior**



- $L_{reco} = \frac{1}{N} \sum_{z \sim q_{\psi}(z|x)} \log p_{\theta}(x|z_i)$
- Prior $p(z)$ describes the latent space distribution,
need to ensure the encoder is consistent with prior
- Constrain difference between distributions with
Kullback–Leibler divergence

$$D_{KL}[q(z|x)|p(z)] = \mathbb{E}_{q(z|x)} \left[\log \frac{q(z|x)}{p(z)} \right] = \int q(z|x) \log \frac{q(z|x)}{p(z)} dz$$

$$- D_{KL}[q|p] \geq 0 \quad \text{and is only 0 when } q = p$$

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_{\psi}(z|x)} \log p_{\theta}(x|z_i)$
- Prior $p(z)$ describes the latent space distribution,
need to ensure the encoder is consistent with prior
- VAE full objective

$$\max_{\theta, \psi} L(\theta, \psi) = \max_{\theta, \psi} \left[\mathbb{E}_{q_{\psi}(z|x)} [\log p_{\theta}(x|z)] - D_{KL}[q_{\psi}(z|x) | p(z)] \right]$$

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_{\psi}(z|x)} \log p_{\theta}(x|z_i)$
- Prior $p(z)$ describes the latent space distribution,
need to ensure the encoder is consistent with prior
- VAE full objective

$$\max_{\theta, \psi} L(\theta, \psi) = \max_{\theta, \psi} \left[\mathbb{E}_{q_{\psi}(z|x)} [\log p_{\theta}(x|z)] - D_{KL}[q_{\psi}(z|x) | p(z)] \right]$$

NOTE: there is a formal derivation using *variational inference*

- Relies on the fact that $\log p(x) \geq \mathbb{E}_{q_{\psi}(z|x)} [\log p(x|z)] - D_{KL}[q_{\psi}(z|x) | p(z)] \equiv ELBO(x; \psi)$
- $q_{\psi}(z|x)$ is a variational approximation of posterior $p(z|x)$
- Maximize ELBO w.r.t. ψ to get closer to $p(x)$

- Classification / regression models make single predictions...

How to model a conditional density $p(a|b)$?

- Classification / regression models make single predictions...

How to model a conditional density $p(a|b)$?

- Assume a known form of density, e.g. normal

$$p(a|b) = \mathcal{N}(a; \mu(b), \sigma(b))$$

- Parameters of density depend on conditioned variable

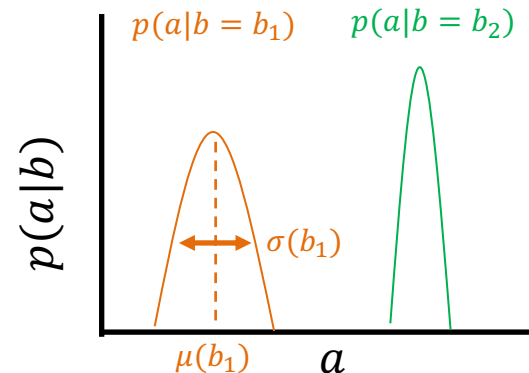
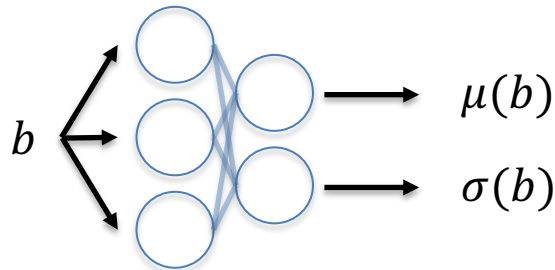
- Classification / regression models make single predictions...

How to model a conditional density $p(a|b)$?

- Assume a known form of density, e.g. normal

$$p(a|b) = \mathcal{N}(a; \mu(b), \sigma(b))$$

- Parameters of density depend on conditioned variable
- Use neural network to model density parameters



- **Decoder**
 - Neural network with parameters θ
 - Input $z \rightarrow$ output estimate of Gaussian $\mu_\theta(z)$, $\sigma_\theta(z)$
- **Likelihood of a data point x**

$$\log p(x|z) = -\log \sigma_\theta(z) - \frac{(x - \mu_\theta(z))^2}{\sigma_\theta(z)^2} + \text{const}$$

- **Encoder**
 - Neural network with parameters ψ
 - Input $x \rightarrow$ outputs estimate of Gaussian $\mu_\psi(x)$, $\sigma_\psi(x)$
- For reconstruction loss:
 - Need a value of z to evaluate decoder!
 - Need to gradient through z to encoder parameters

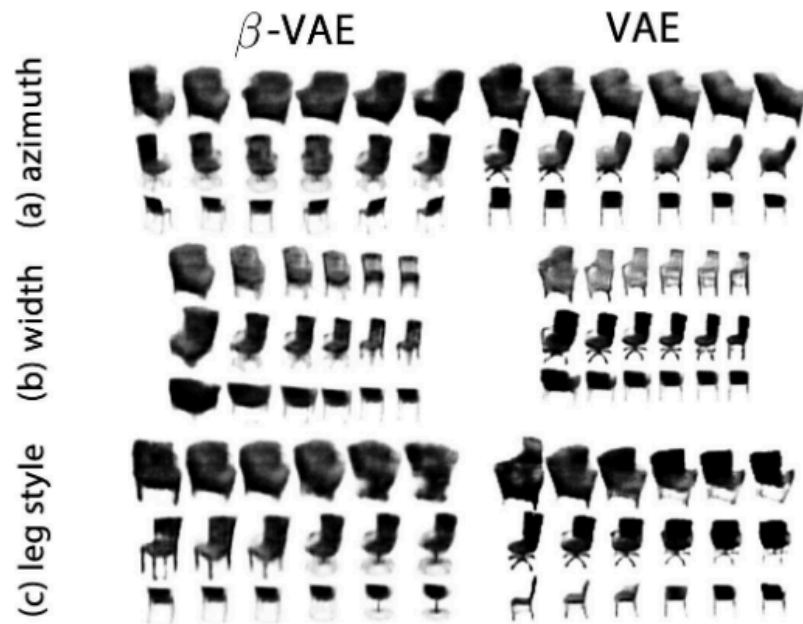
$$\max_{\theta, \psi} L(\theta, \psi) = \max_{\theta, \psi} \sum_{z_i \sim q_\psi(Z|x)} \log p_\theta(x|z_i) - \log \left[\frac{q_\psi(z_i|x)}{p(z_i)} \right]$$

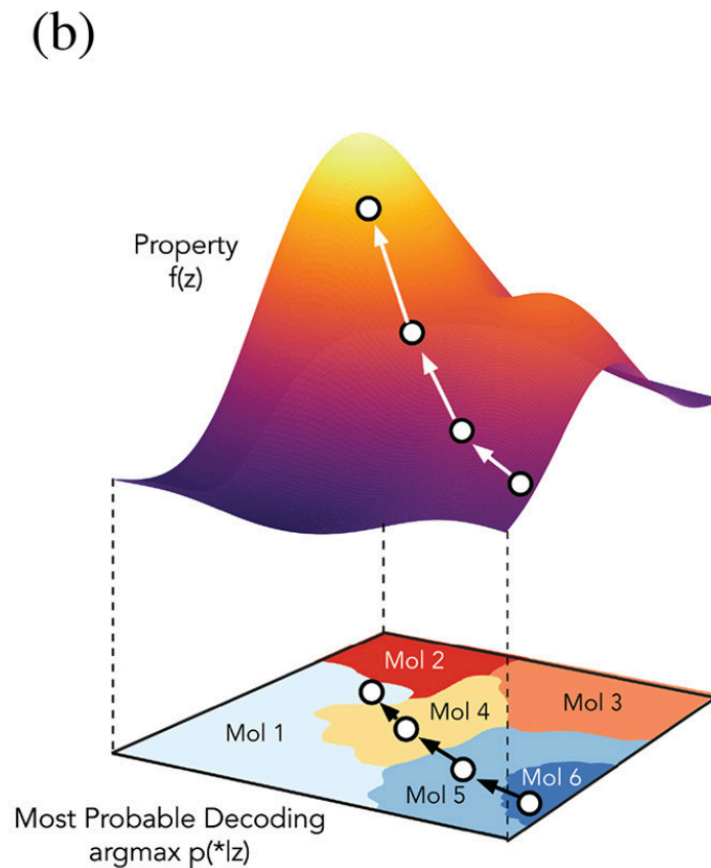
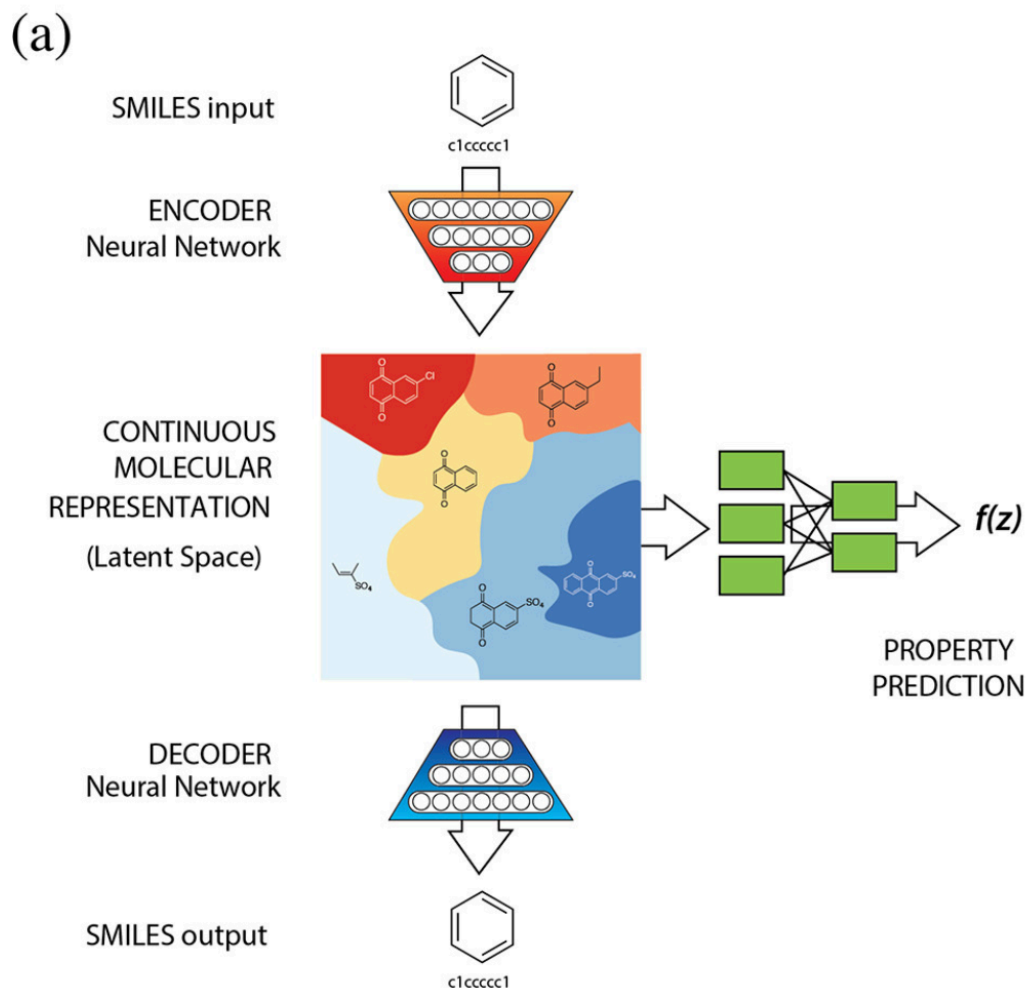
- For $z \sim p_\theta(z)$, rewrite z as a function of a random variable ϵ whose distributions $p(\epsilon)$ does not depend on θ
 - Gaussian Example:

$$z \sim \mathcal{N}(\mu, \sigma) \rightarrow z = \sigma * \epsilon + \mu \quad \text{where } \epsilon \sim \mathcal{N}(0,1)$$

- VAE Loss

$$\max_{\theta, \psi} L(\theta, \psi) = \max_{\theta, \psi} \sum_{\epsilon \sim p(\epsilon)} \log p_\theta(x | z_i = \epsilon * \sigma_\psi(x) + \mu_\psi(x)) - \log \left[\frac{q_\psi(z_i | x)}{p(z_i)} \right]$$





Design of new molecules with desired chemical properties.
(Gomez-Bombarelli et al, 2016)

- Another approach to generative modeling is to formulate the task as a two player game
- One player tries to output data that looks as real as possible
- Another player tries to compare real and fake data
- In this case we need:
 - A *generator* that can produce samples
 - A measure of *not too far from the real data*

- **Generator network $g_{\theta}(z)$** with parameters θ
 - Map sample from known $p(z)$ to sample in data space

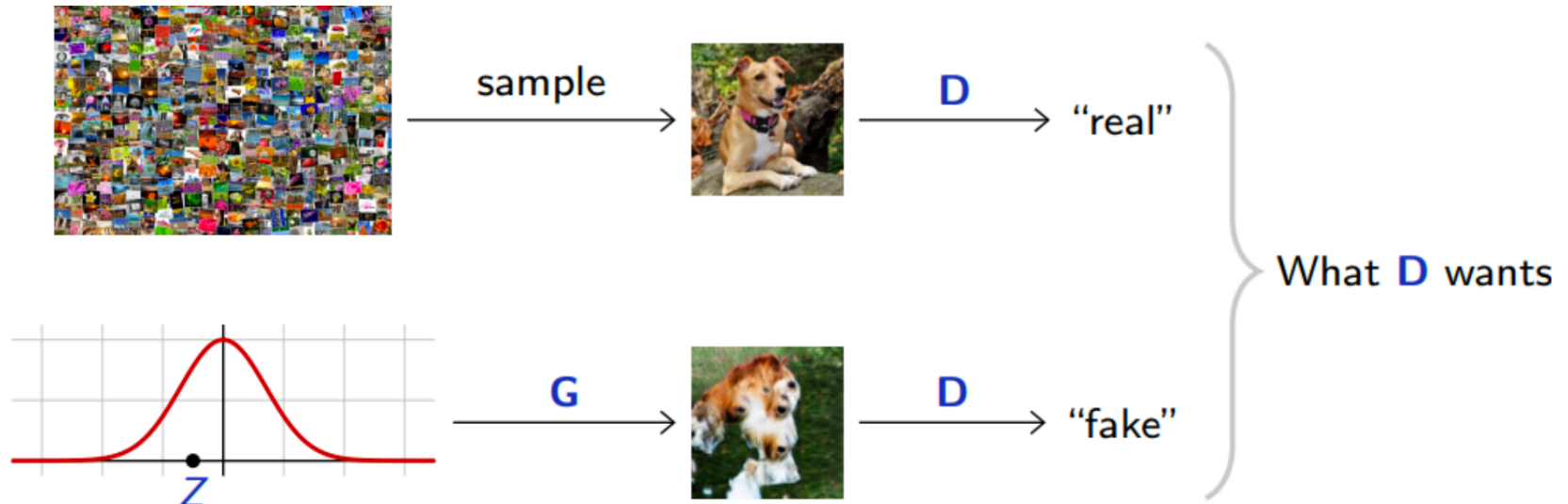
$$x = g_{\theta}(z) \quad z \sim p(z)$$

- We don't know what the generated distribution $p_{\theta}(x)$ is, but we can sample from it \rightarrow *Implicit Model*

- **Generator network $g_{\theta}(z)$** with parameters θ
 - Map sample from known $p(z)$ to sample in data space

$$x = g_{\theta}(z) \quad z \sim p(z)$$

- We don't know what the generated distribution $p_{\theta}(x)$ is, but we can sample from it \rightarrow *Implicit Model*
- **Discriminator Network $d_{\phi}(x)$** with parameters ϕ
 - Classifier trained to distinguish between real and fake data
 - Classifier is learning to predict $p(y = \textit{real} \mid x)$
 - This classifier is our measure of *not too far from the real data*



- Generator's goal is to produce *fake* data that tricks the discriminator to think it is *real* data
- Discriminator wants to miss-classify data as real or fake as little as possible
- The setup is *adversarial* because the two networks have opposing objectives

- Data
 - Real data samples: $\{x_i, y_i = 1\}$
 - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$

- Data
 - Real data samples: $\{x_i, y_i = 1\}$
 - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$
- For a fixed generator, can train discriminator by minimizing the cross entropy

$$L(\phi) = -\frac{1}{2N} \sum_{i=1}^N \left[y_i \log d_\phi(x_i) + (1 - \tilde{y}_i) \log(1 - d_\phi(\tilde{x}_i)) \right]$$

- Data
 - Real data samples: $\{x_i, y_i = 1\}$
 - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$
- For a fixed generator, can train discriminator by minimizing the cross entropy

$$\begin{aligned} L(\phi) &= -\frac{1}{2N} \sum_{i=1}^N \left[y_i \log d_\phi(x_i) + (1 - \tilde{y}_i) \log(1 - d_\phi(\tilde{x}_i)) \right] \\ &= -\frac{1}{2N} \sum_{i=1}^N \left[\log d_\phi(x_i) + \log(1 - d_\phi(g_\theta(z_i))) \right] \end{aligned}$$

- Data
 - Real data samples: $\{x_i, y_i = 1\}$
 - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$
- For a fixed generator, can train discriminator by minimizing the cross entropy

$$\begin{aligned} L(\phi) &= -\frac{1}{2N} \sum_{i=1}^N \left[y_i \log d_\phi(x_i) + (1 - \tilde{y}_i) \log(1 - d_\phi(\tilde{x}_i)) \right] \\ &= -\frac{1}{2N} \sum_{i=1}^N \left[\log d_\phi(x_i) + \log(1 - d_\phi(g_\theta(z_i))) \right] \\ &= -\mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log d_\phi(x) \right] - \mathbb{E}_{z \sim p(z)} \left[\log(1 - d_\phi(g_\theta(z))) \right] \end{aligned}$$

- However, generator isn't fixed... have to train it!

- However, generator isn't fixed... have to train it!
- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log d_{\phi}(x) \right] + \mathbb{E}_{z \sim p(z)} \left[\log(1 - d_{\phi}(g_{\theta}(z))) \right]$$

- However, generator isn't fixed... have to train it!
- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log d_{\phi}(x) \right] + \mathbb{E}_{z \sim p(z)} \left[\log(1 - d_{\phi}(g_{\theta}(z))) \right]$$

- For fixed generator, $V(\phi, \theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes
- For a perfect discriminator, a good generator will confuse discriminator and $V(\phi, \theta)$ will be low

- However, generator isn't fixed... have to train it!
- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log d_{\phi}(x) \right] + \mathbb{E}_{z \sim p(z)} \left[\log(1 - d_{\phi}(g_{\theta}(z))) \right]$$

- For fixed generator, $V(\phi, \theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes
- For a perfect discriminator, a good generator will confuse discriminator and $V(\phi, \theta)$ will be low
- So our optimization goal becomes:

$$\theta^* = \arg \min_{\theta} \max_{\phi} V(\phi, \theta)$$

- However, generator isn't fixed... have to train it!
- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log d_{\phi}(x) \right] + \mathbb{E}_{z \sim p(z)} \left[\log(1 - d_{\phi}(g_{\theta}(z))) \right]$$

- For fixed generator, $V(\phi, \theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes
- For a perfect discriminator, a good generator will confuse discriminator and $V(\phi, \theta)$ will be low
- So our optimization goal becomes:
$$\theta^* = \arg \min_{\theta} \max_{\phi} V(\phi, \theta)$$

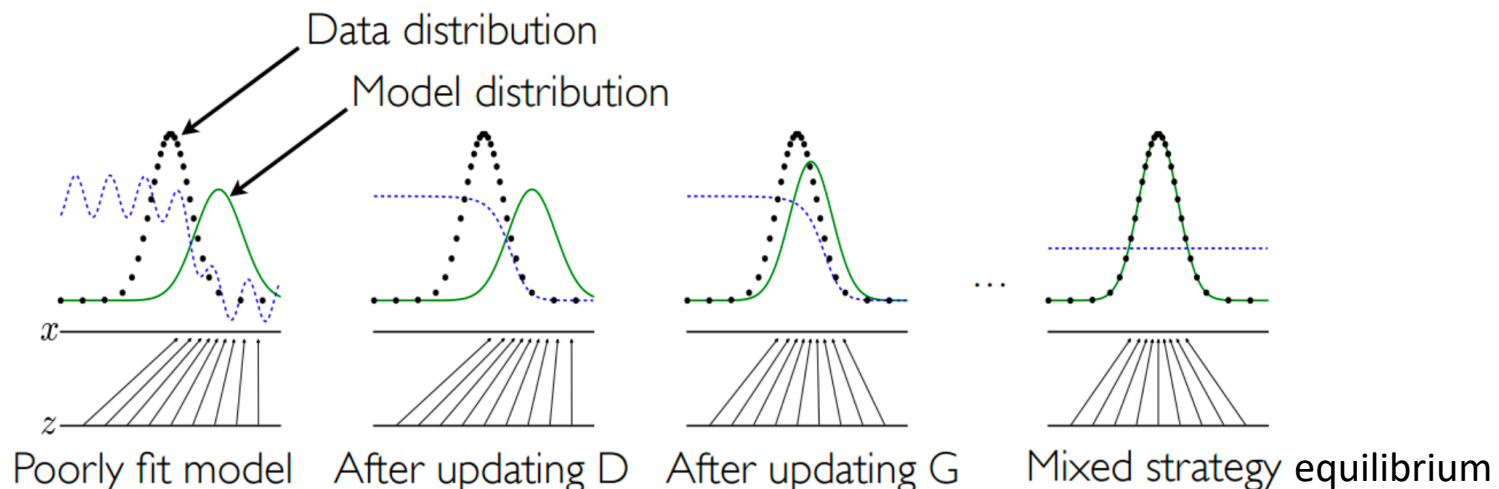
NOTE: can prove that
minimax solution
corresponds to generator
that perfectly reproduces
data distribution
 $q_{\theta^*}(x) = p_{\text{data}}(x)$

- Alternating Gradient descent to solve the min-max problem:

$$\theta \leftarrow \theta - \gamma \nabla_{\theta} V(\phi, \theta) = \theta - \gamma \frac{\partial V}{\partial d} \frac{\partial (d_{\phi})}{\partial g} \frac{\partial g_{\theta}}{\partial \theta}$$

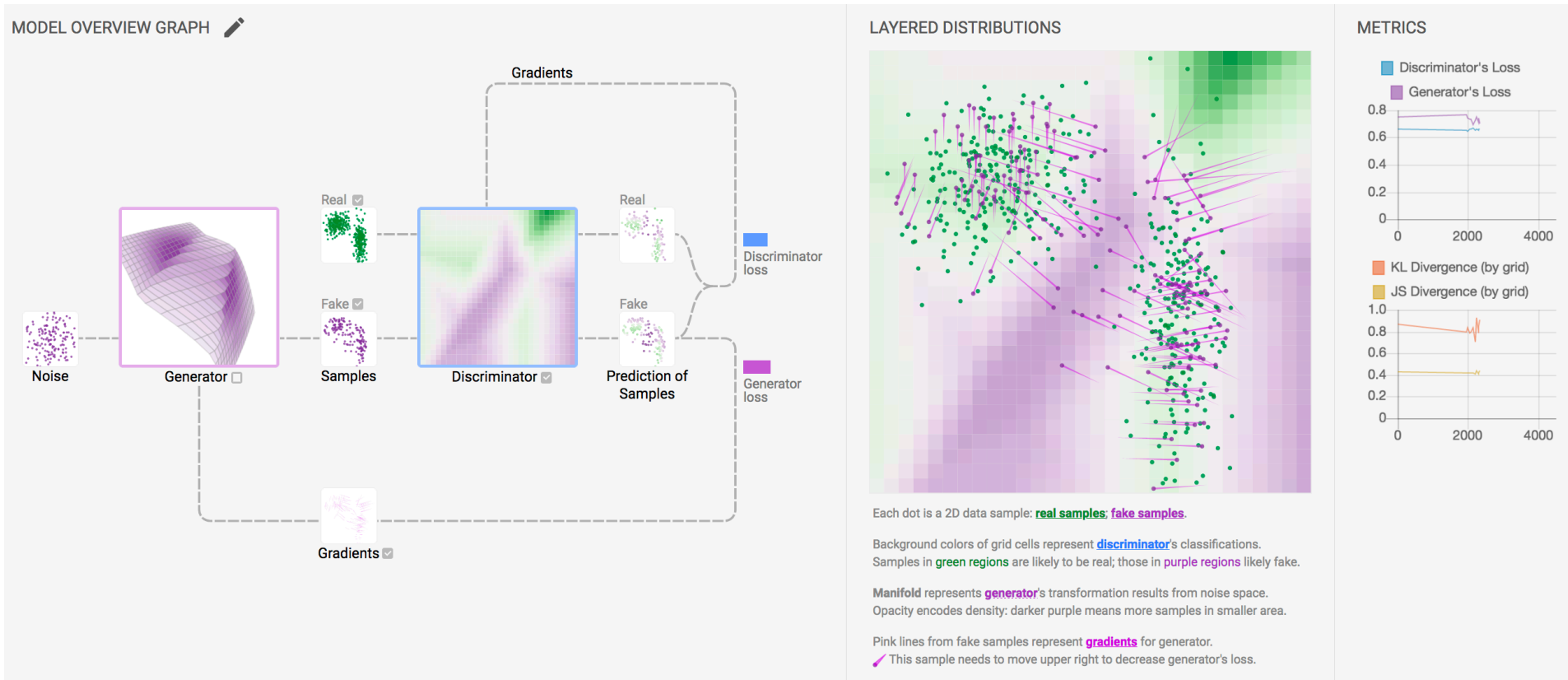
$$\phi \leftarrow \phi - \gamma \nabla_{\phi} V(\phi, \theta) = \phi - \gamma \frac{\partial V}{\partial d} \frac{d(d_{\phi})}{d\phi}$$

- For each θ step, take k steps in ϕ to keep discriminator near optimal



GAN Training Example

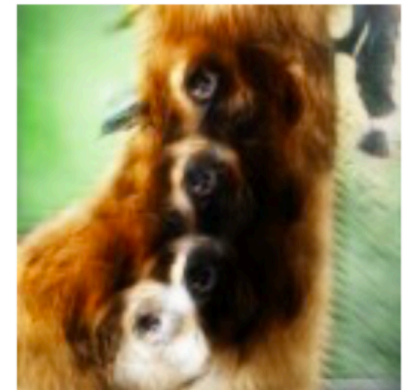
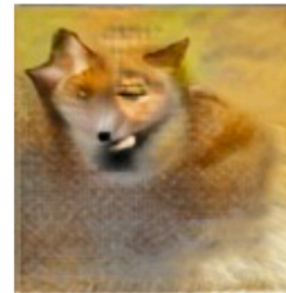
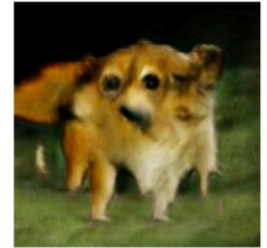
85



[GAN Lab Demo](#)

Examples

Goodfellow et. al., [2014](#)

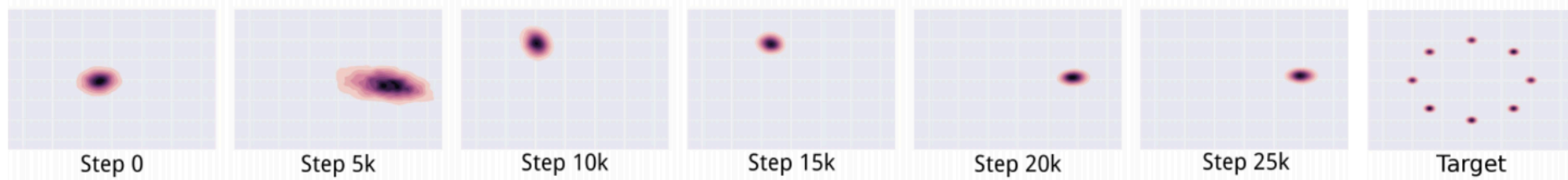


Not so good

Goodfellow 2016



- **Oscillations without convergence:** unlike standard loss minimization, alternating stochastic gradient descent has no guarantee of convergence.
- **Vanishing gradients:** if classifier is too good, value function saturates \rightarrow no gradient to update generator
- **Mode collapse:** generator models only a small sub-population, concentrating on a few data distribution modes.
- **Difficult to assess performance,** when are generated data good enough?



- Standard GANS compare real and fake distributions with Jensen-Shannon Divergence, “vertically”
- Wasserstein-GAN (Arjovsky et al, [2017](#)) compares “horizontally” with Wasserstein-1 distance (a.k.a. Earth Movers distance)
- Substantially improves *vanishing gradient* and *mode collapse* problems!

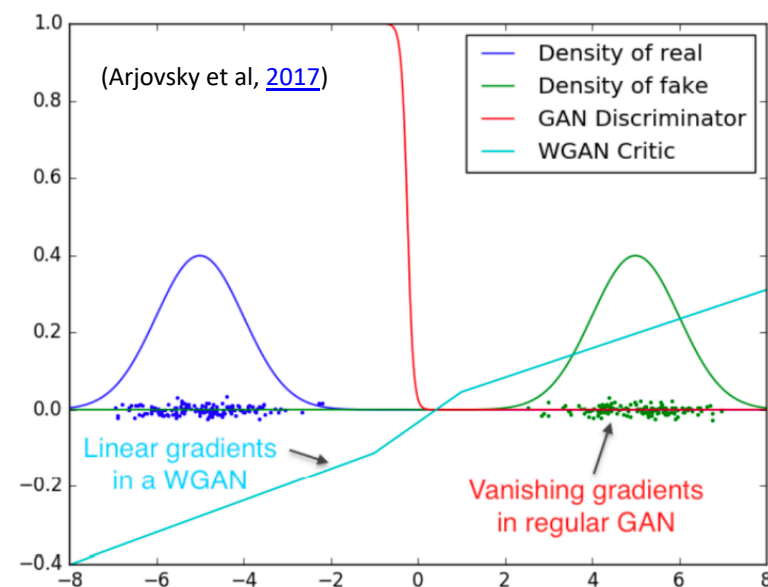
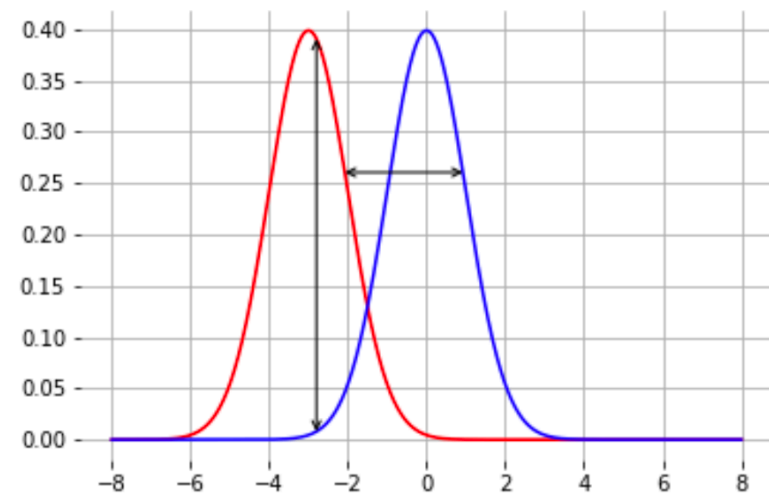
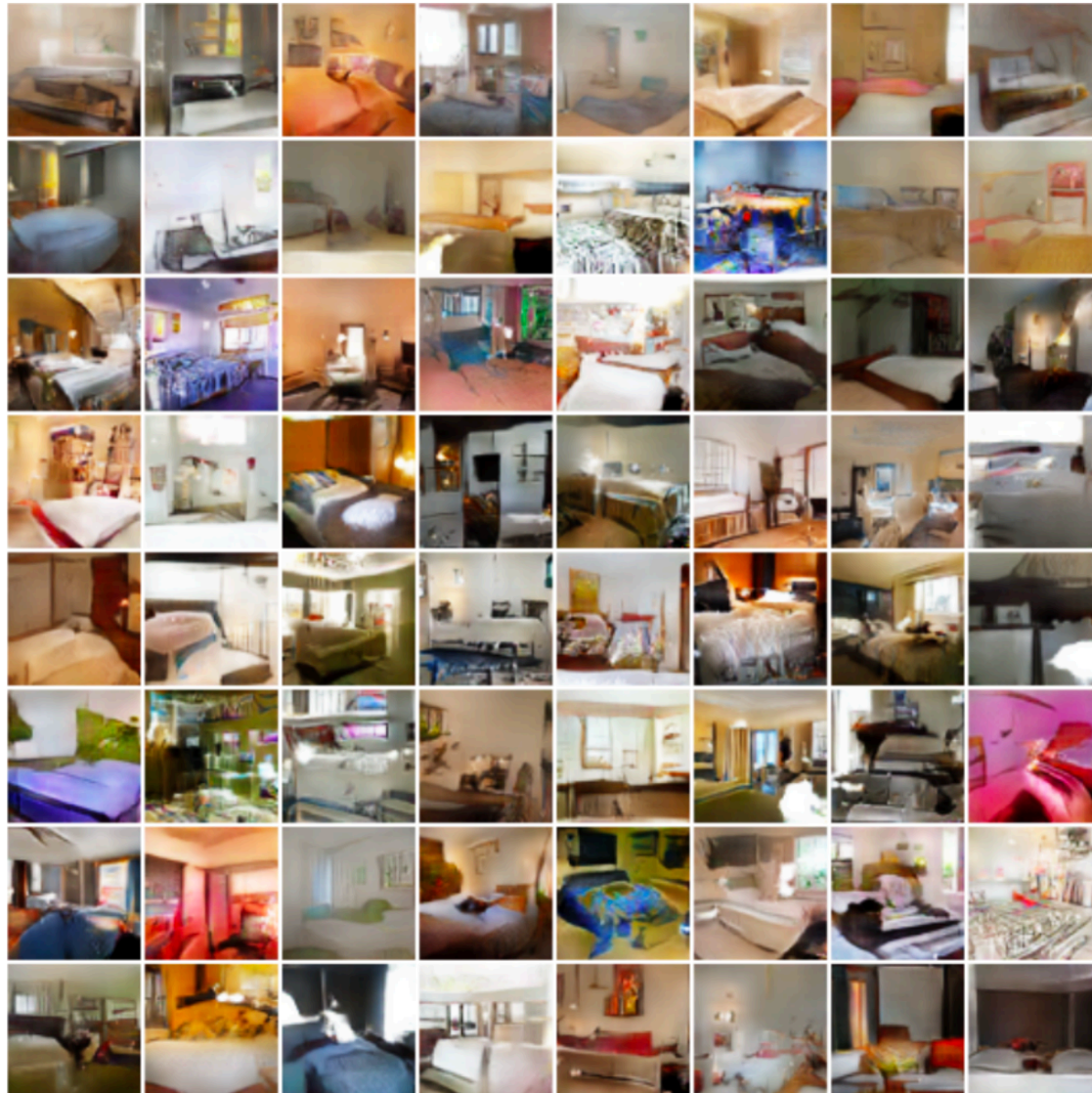
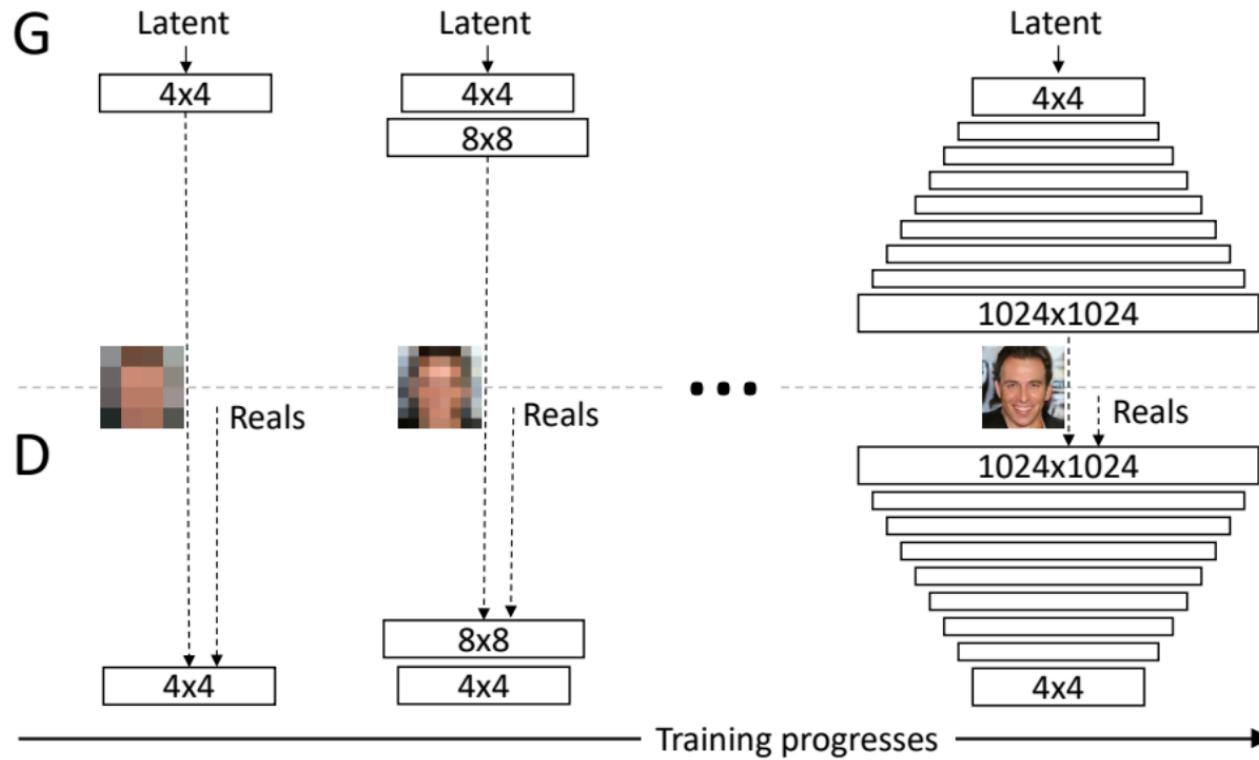


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

WGAN Examples



Progressive GAN



(Karras et al, 2017)

StyleGAN v2



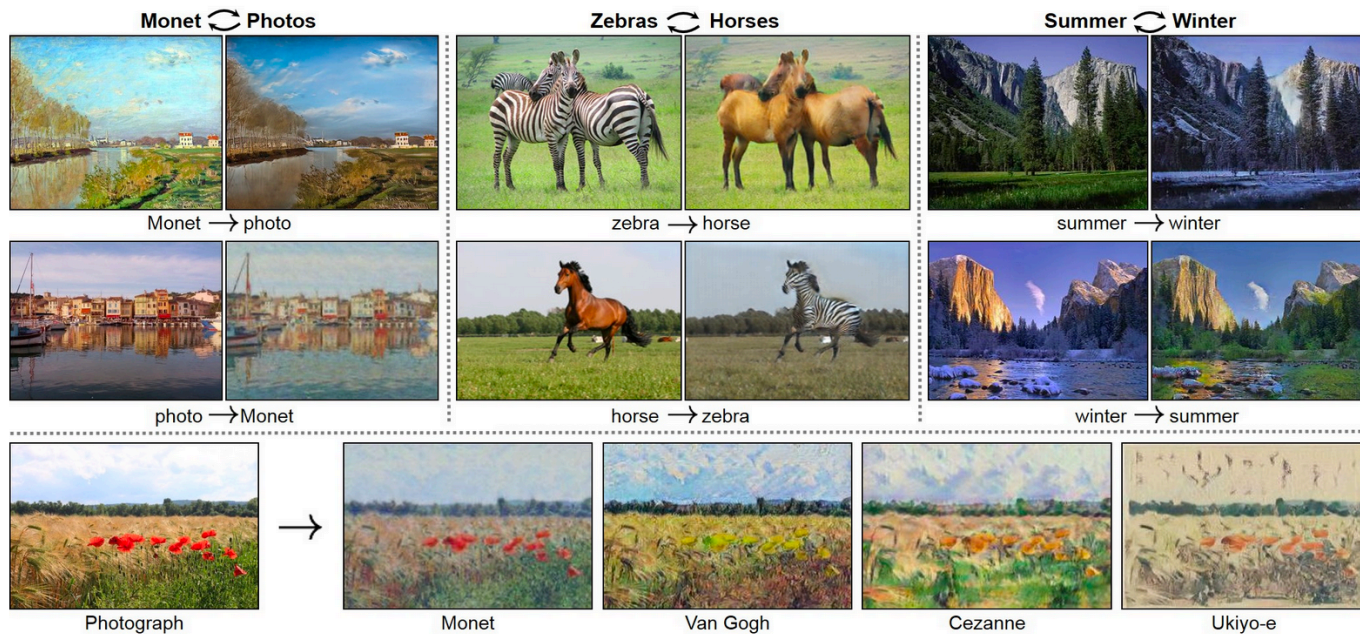
BigGAN

(Karras et al, 2019)



(Brock et al, 2018)

- $p(z)$ doesn't have to be random noise
- CycleGAN uses *cycle-consistency loss* in addition to GAN loss
 - Translating from $A \rightarrow B \rightarrow A$ should be consistent with original A




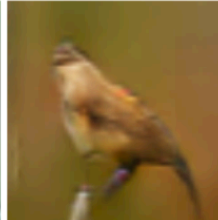
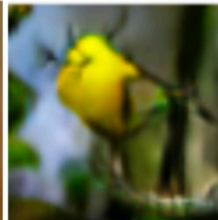


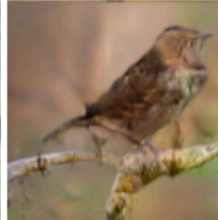
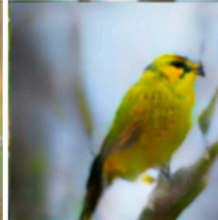


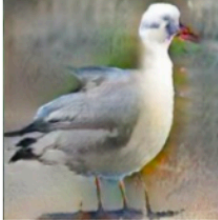
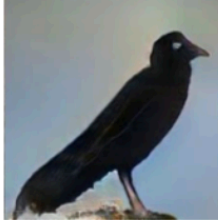
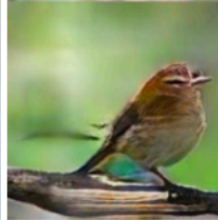

Text description	This bird is red and brown in color, with a stubby beak	The bird is short and stubby with yellow on its body	A bird with a medium orange bill white body gray wings and webbed feet	This small black bird has a short, slightly curved bill and long legs	A small bird with varying shades of brown with white under the eyes	A small yellow bird with a black crown and a short black pointed beak	This small bird has a white breast, light grey head, and black wings and tail
64x64 GAN-INT-CLS							
128x128 GAWWN							
256x256 StackGAN-v1							

Fig. 3: Example results by our StackGAN-v1, GAWWN [29], and GAN-INT-CLS [31] conditioned on text descriptions from CUB test set.

(Zhang et al, 2017)

- Deep neural networks are an extremely powerful class of models
- We can express our inductive bias about a system in terms of model design, and can be adapted to a many types of data
- Even beyond classification and regression, deep neural networks allow for powerful model schemes such as Variational Autoencoder and Generative adversarial Networks



- Autoencoders learn the latent space, but we don't know what is the latent space distribution
- Autoencoder prescribes a deterministic relationship between data space and latent space
- One set of “meaningful degrees of freedom” can only describe one data space point

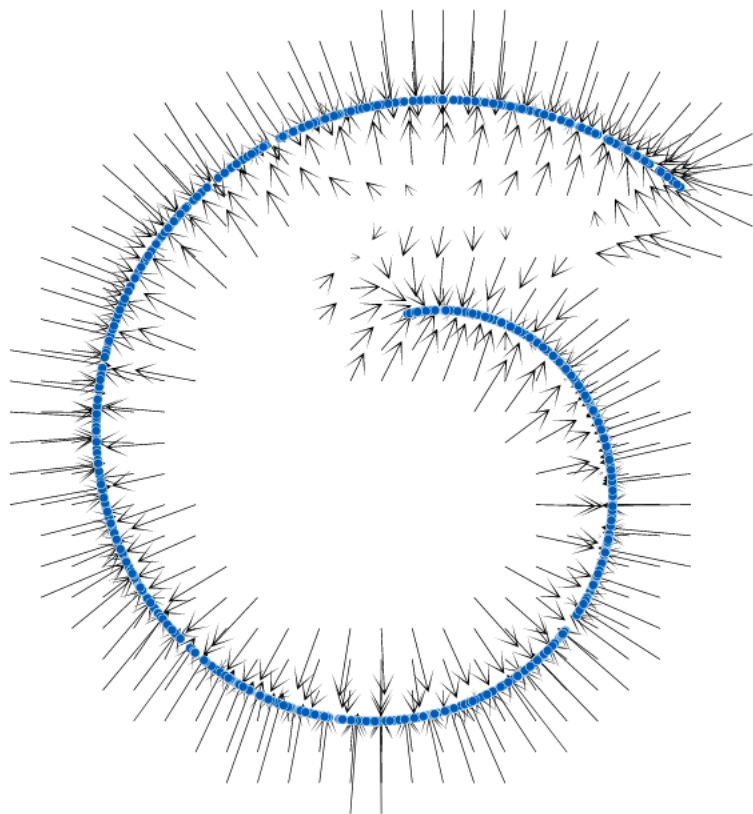
Denoising Autoencoder

- Learn a mapping from corrupted data space $\tilde{\mathcal{X}}$ back to original data space
 - Mapping $\phi_w(\tilde{x}) = x$
 - ϕ_w will be a neural network with parameters w
- Loss:

$$L = \frac{1}{N} \sum_n \|x_n - \phi_w(x_n + \epsilon_n)\|$$



Perturbation, e.g. Gaussian noise



Original

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Corrupted ($\sigma = 4$)

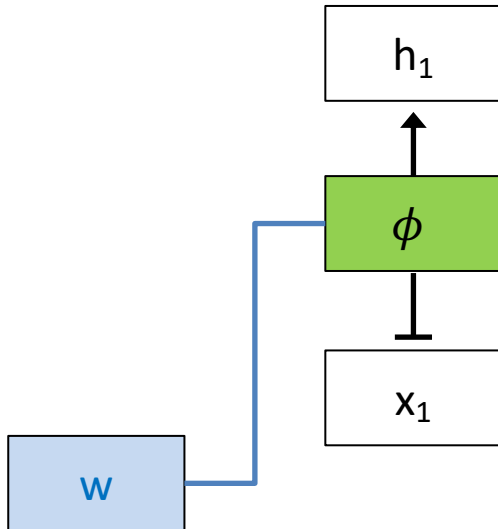
7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

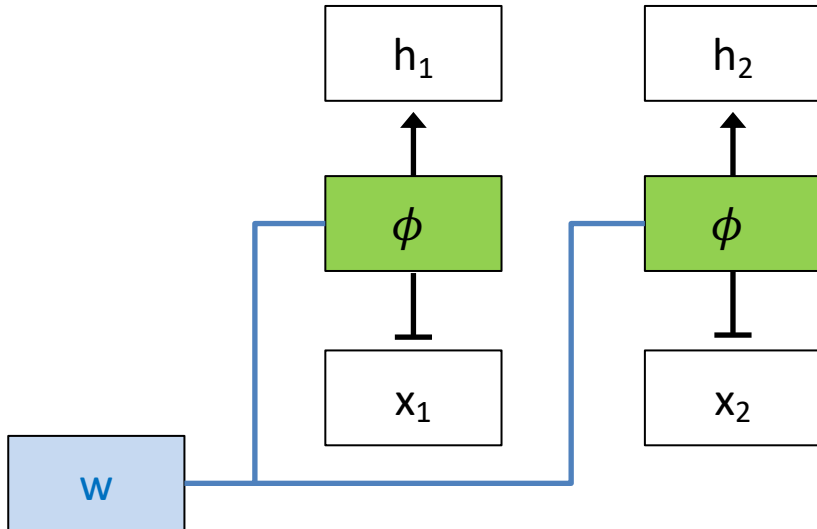
Reconstructed

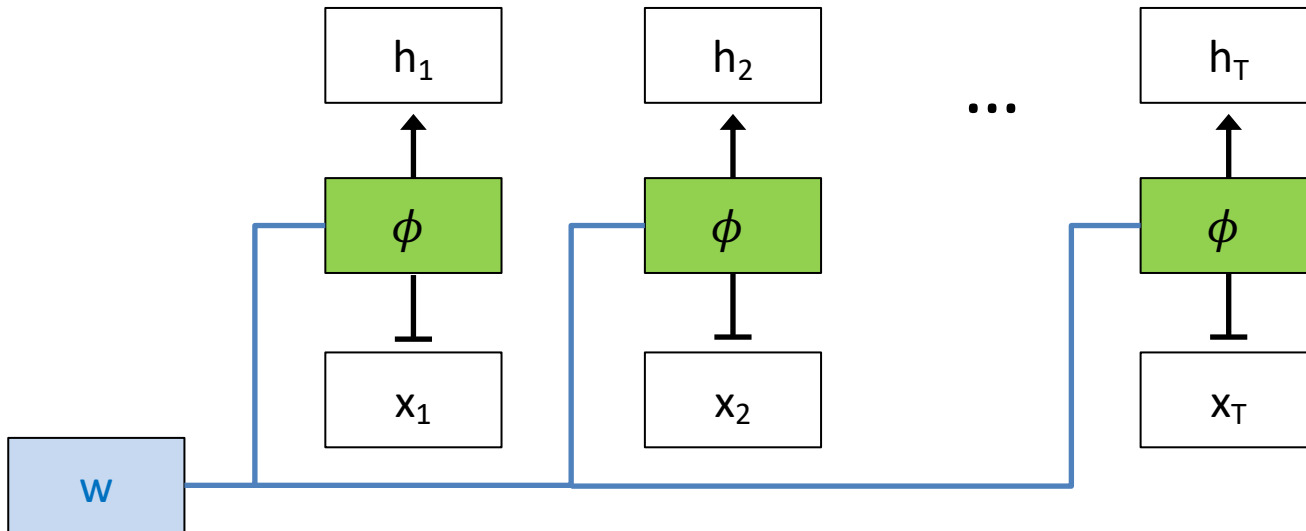
7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

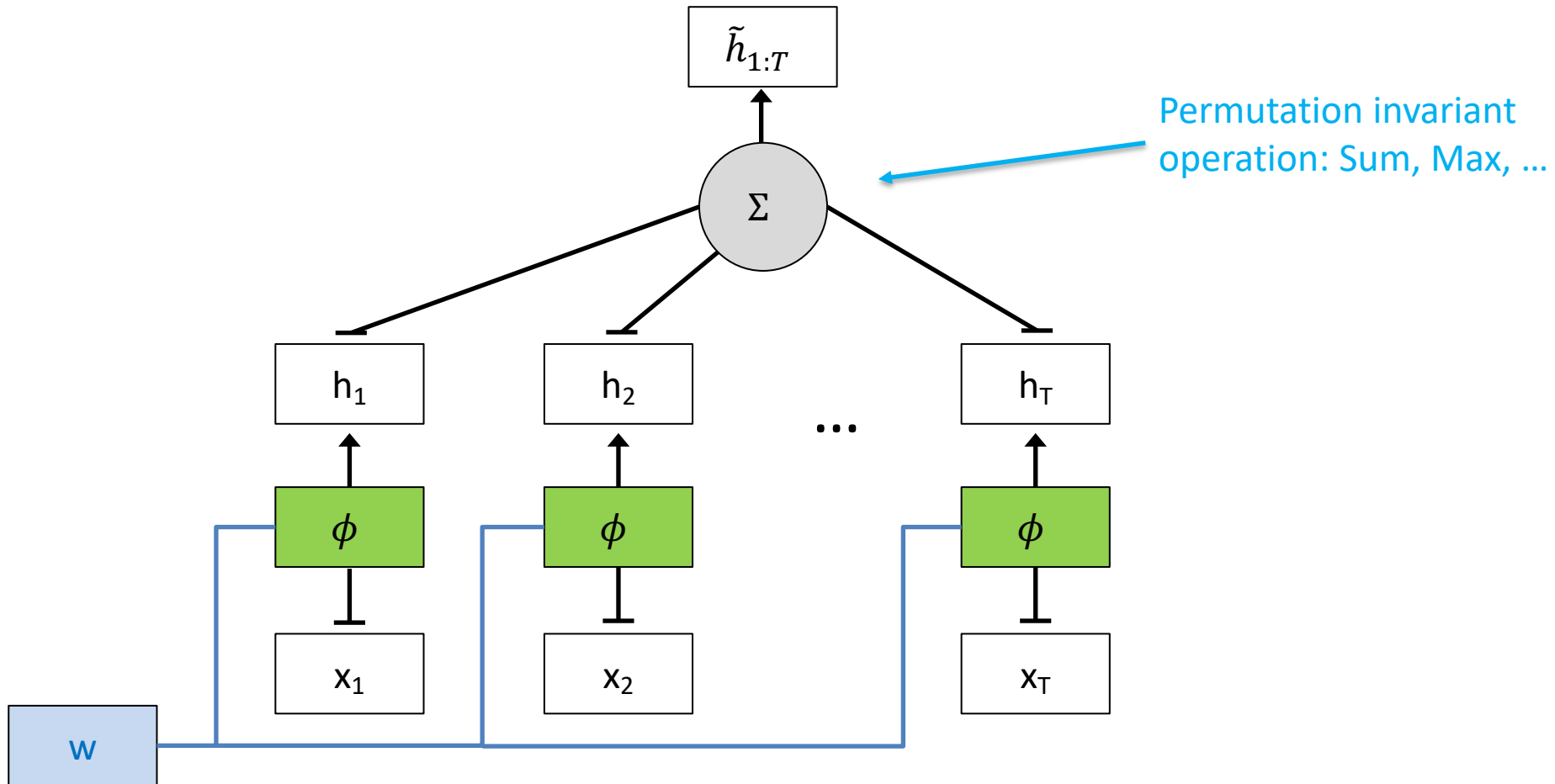
Deep Sets

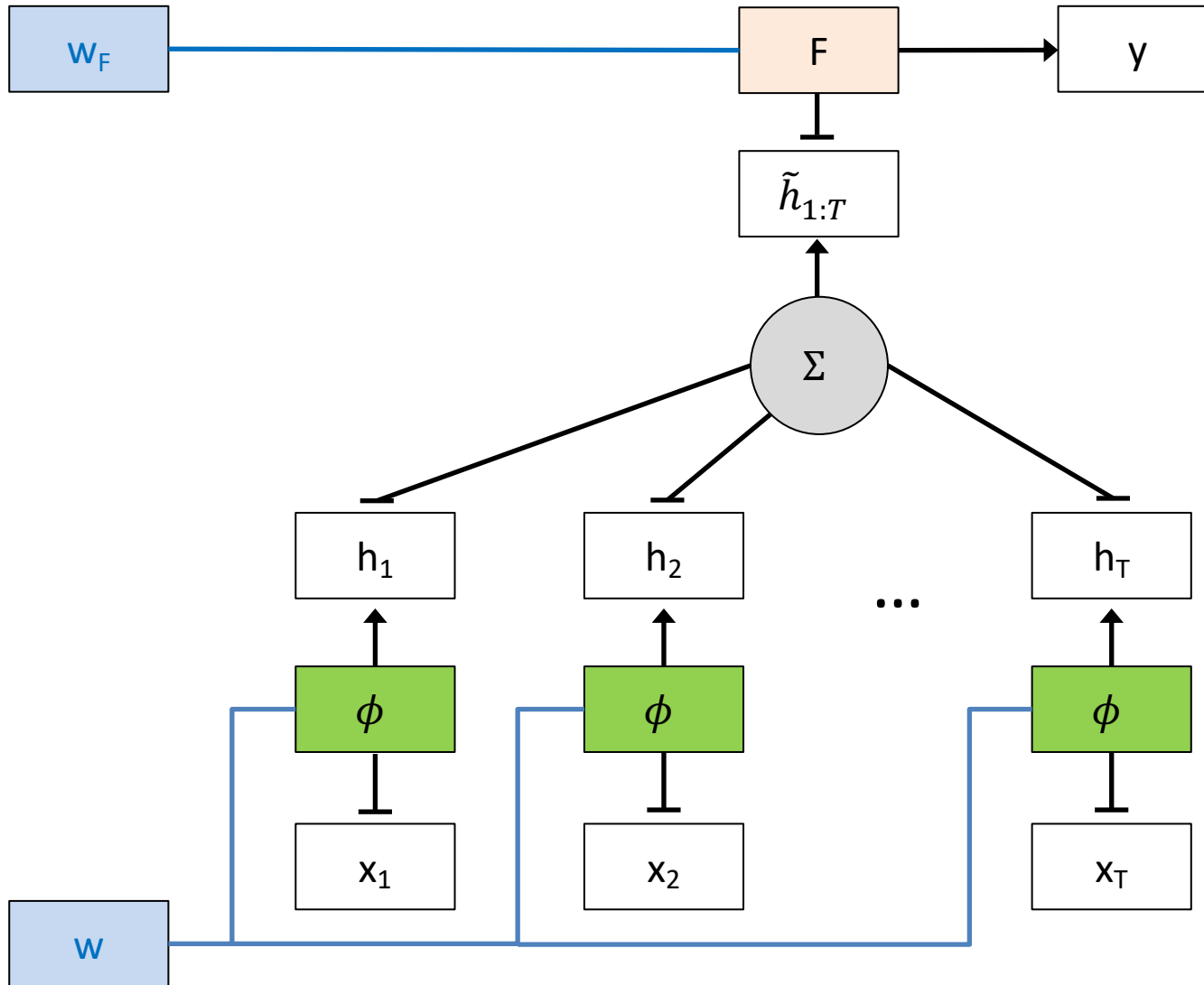
- Data may be variable in length but have no temporal structure \rightarrow *Data are sets of values*
- *One option:* If we know about the data domain, could try to impose an ordering, then use RNN
- *Better option:* use system that can operate on variable length sets in permutation invariant way
 - Why permutation invariant \rightarrow so order doesn't matter











Outlier detection



M. Zaheer et. al [2017](#)

Medical Imaging

With more complex architecture

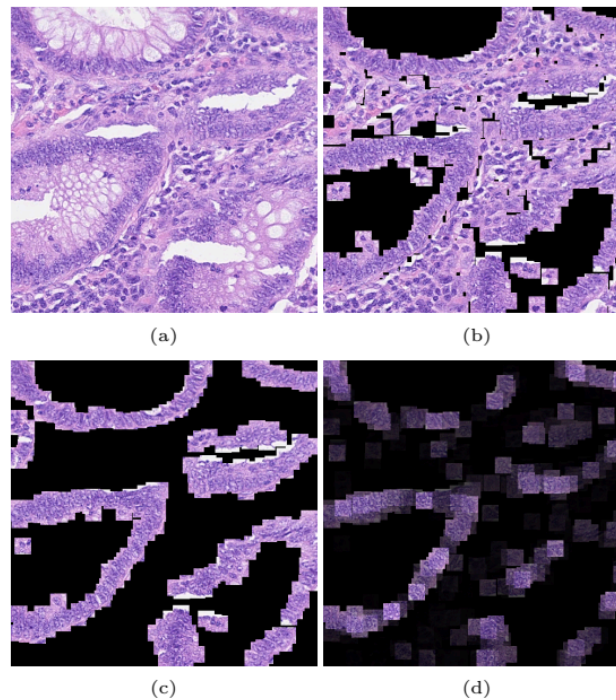


Figure 5. (a) H&E stained histology image. (b) 27×27 patches centered around all marked nuclei. (c) Ground truth: Patches that belong to the class epithelial. (d) Heatmap: Every patch from (b) multiplied by its corresponding attention weight, we rescaled the attention weights using $a'_k = (a_k - \min(\mathbf{a})) / (\max(\mathbf{a}) - \min(\mathbf{a}))$.

M. Ilse et al., [2018](#)

Explicit Density Estimation with Normalizing Flows

- In VAE and GAN we can learn to sample from the distribution...
- Is there a way to learn the explicit density $p(x)$?

$$\int f(g(x)) \frac{\partial g(x)}{dx} dx = \int f(u) du \quad \text{where } u = g(x)$$

Multivariate:

$$\int f(g(\mathbf{x})) \left| \det \frac{\partial g(\mathbf{x})}{d\mathbf{x}} \right| d\mathbf{x} = \int f(\mathbf{u}) d\mathbf{u} \quad \text{where } \mathbf{u} = g(\mathbf{x})$$



Determinant of Jacobian
of the transformation

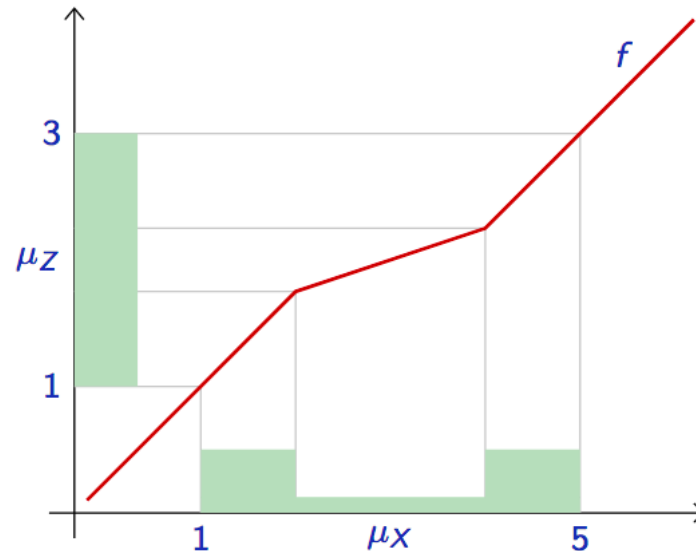
→ Change of volume

Change of Variables in Probability

11
1

- If f is continuous, invertible, differentiable, and $\mathbf{x} = f^{-1}(\mathbf{z}) \equiv \phi(\mathbf{z})$ then

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z}) \left| \det \left(\frac{\partial \phi(\mathbf{z})}{d\mathbf{z}} \right)^{-1} \right| \quad \text{where } \mathbf{x} = \phi(\mathbf{z})$$



The term $\left| \det \left(\frac{\partial \phi(\mathbf{z})}{d\mathbf{z}} \right)^{-1} \right|$ accounts for the local stretching of space

- If f is continuous, invertible, differentiable, and $x = f^{-1}(z) \equiv \phi(z)$ then

$$p_x(\mathbf{x}) = p_z(\mathbf{z}) \left| \det \left(\frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \right)^{-1} \right| \quad \text{where } \mathbf{x} = \phi(\mathbf{z})$$

- x = data we want to model, z = known noise
- $\phi_\theta(z)$ will be a neural network with parameters θ
 - Must be continuous, invertible, differentiable
- Output of ϕ is a potential sample x
 - **Learn the right ϕ :** adjust weights θ to maximize data probability (formula above)

- If f is continuous, invertible, differentiable, and $x = f^{-1}(z) \equiv \phi(z)$ then

$$p_x(\mathbf{x}) = p_z(\mathbf{z}) \left| \det \left(\frac{\partial \phi(\mathbf{z})}{\partial \mathbf{z}} \right)^{-1} \right| \quad \text{where } \mathbf{x} = \phi(\mathbf{z})$$

- x = data we want to model, z = known noise

$\phi(\mathbf{z})$ neural network

- Input = a sample of noise
- Output = a sample of X

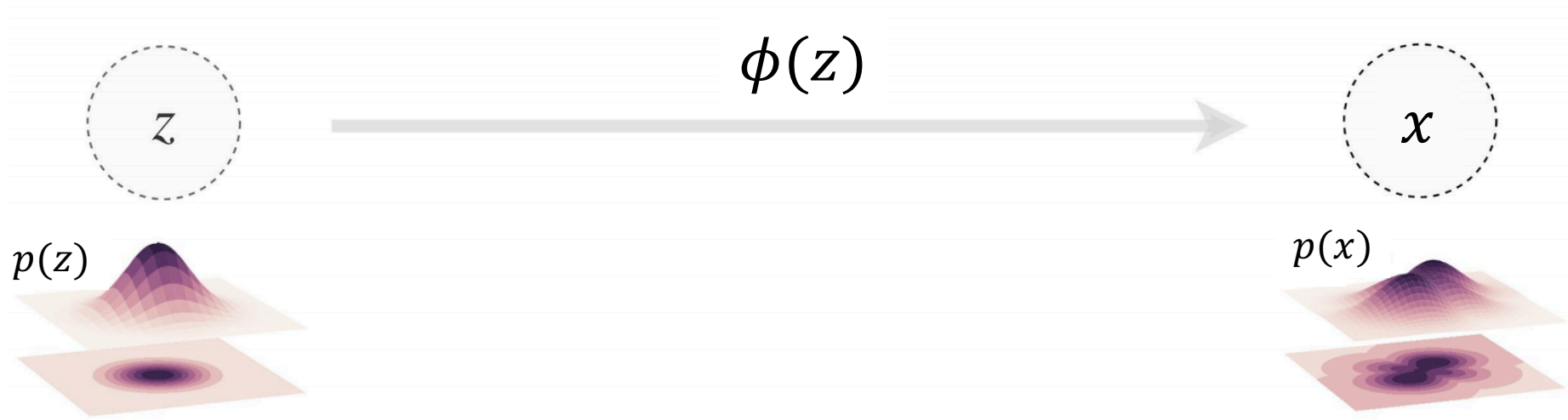


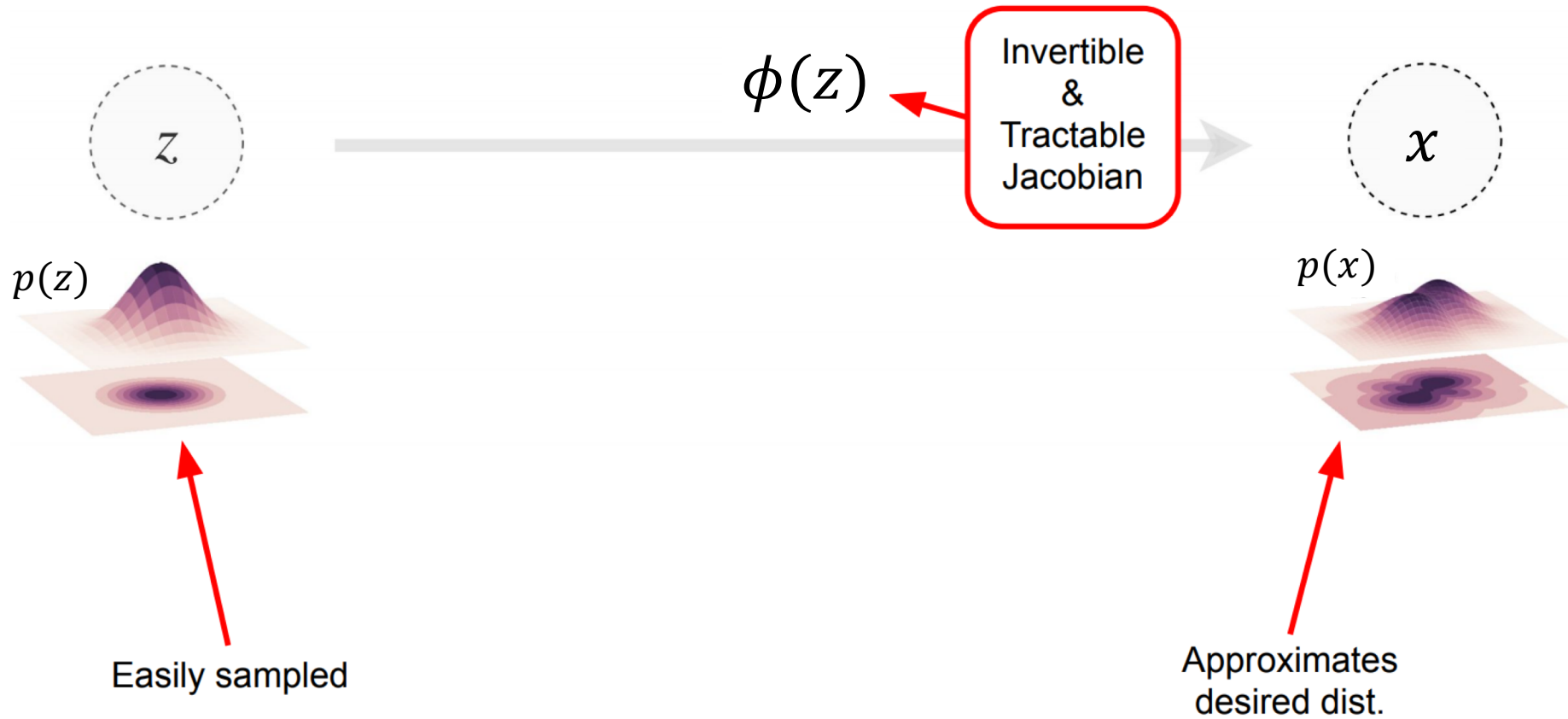
$\phi^{-1}(\mathbf{x})$ inverse

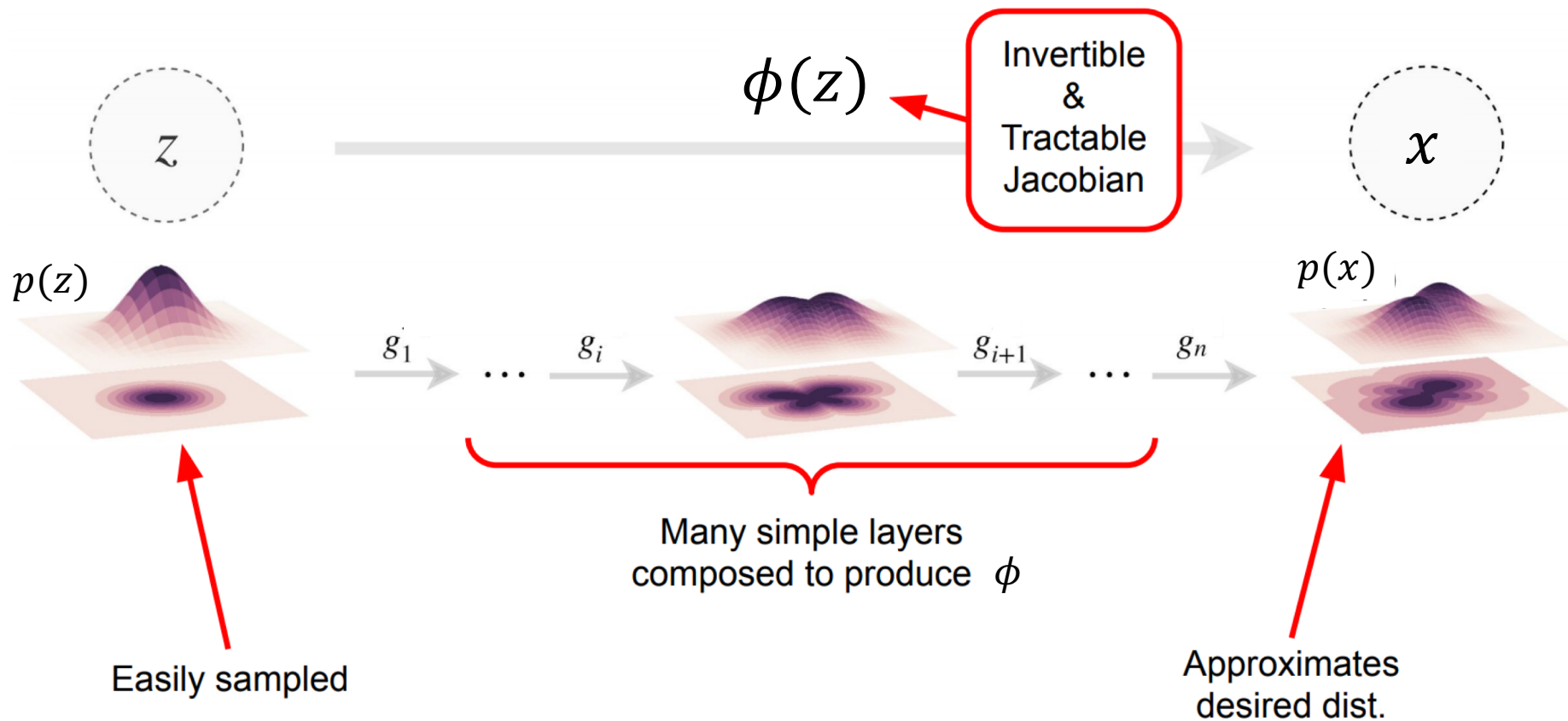
- Input = a sample X
- Output = a sample of noise

- Calculate the probability of a sample using the formula above

$$p_x(\mathbf{x}) = p_z(\mathbf{z}) \left| \det \left(\frac{\partial \phi(\mathbf{z})}{d\mathbf{z}} \right)^{-1} \right|$$

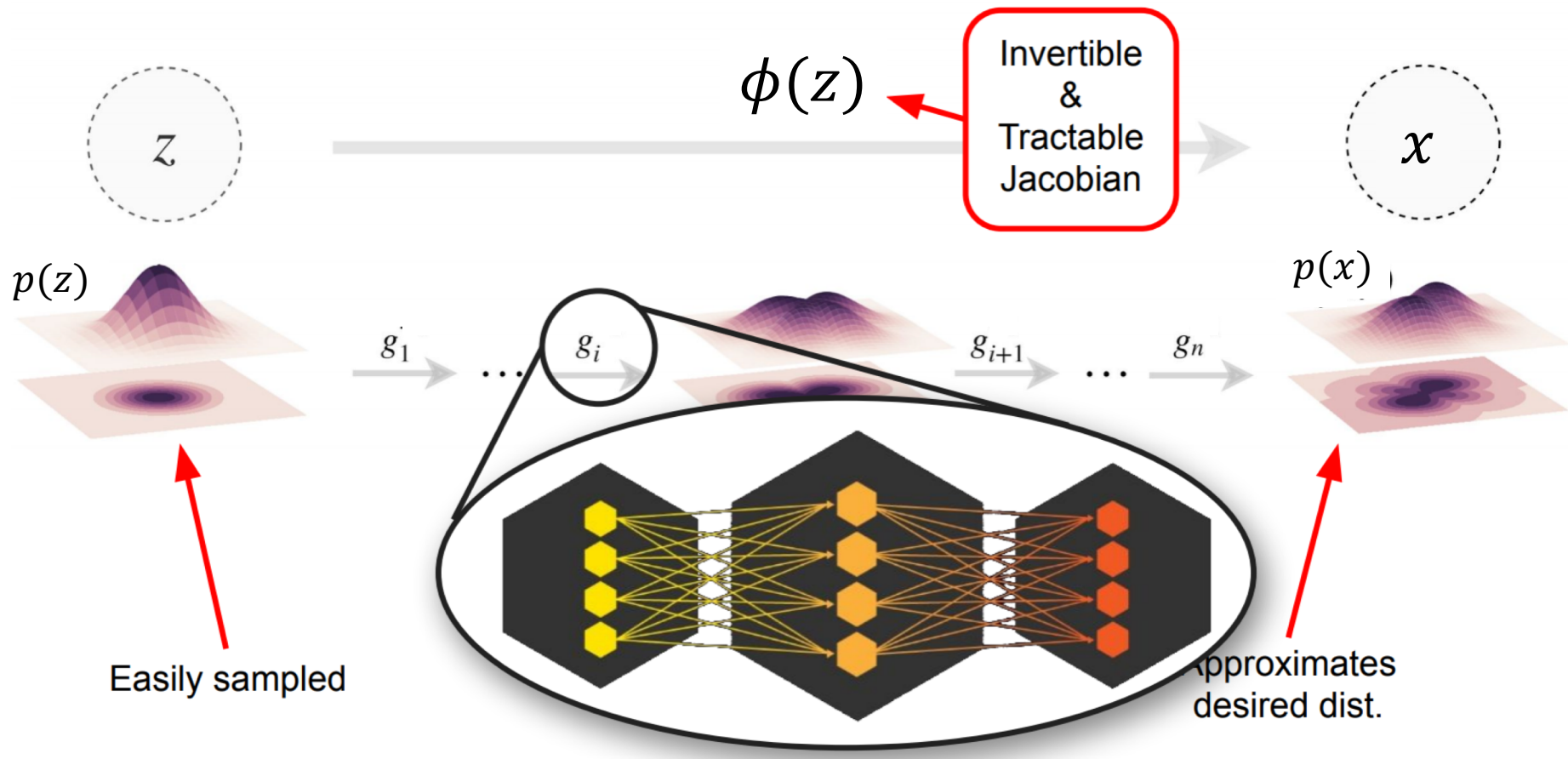


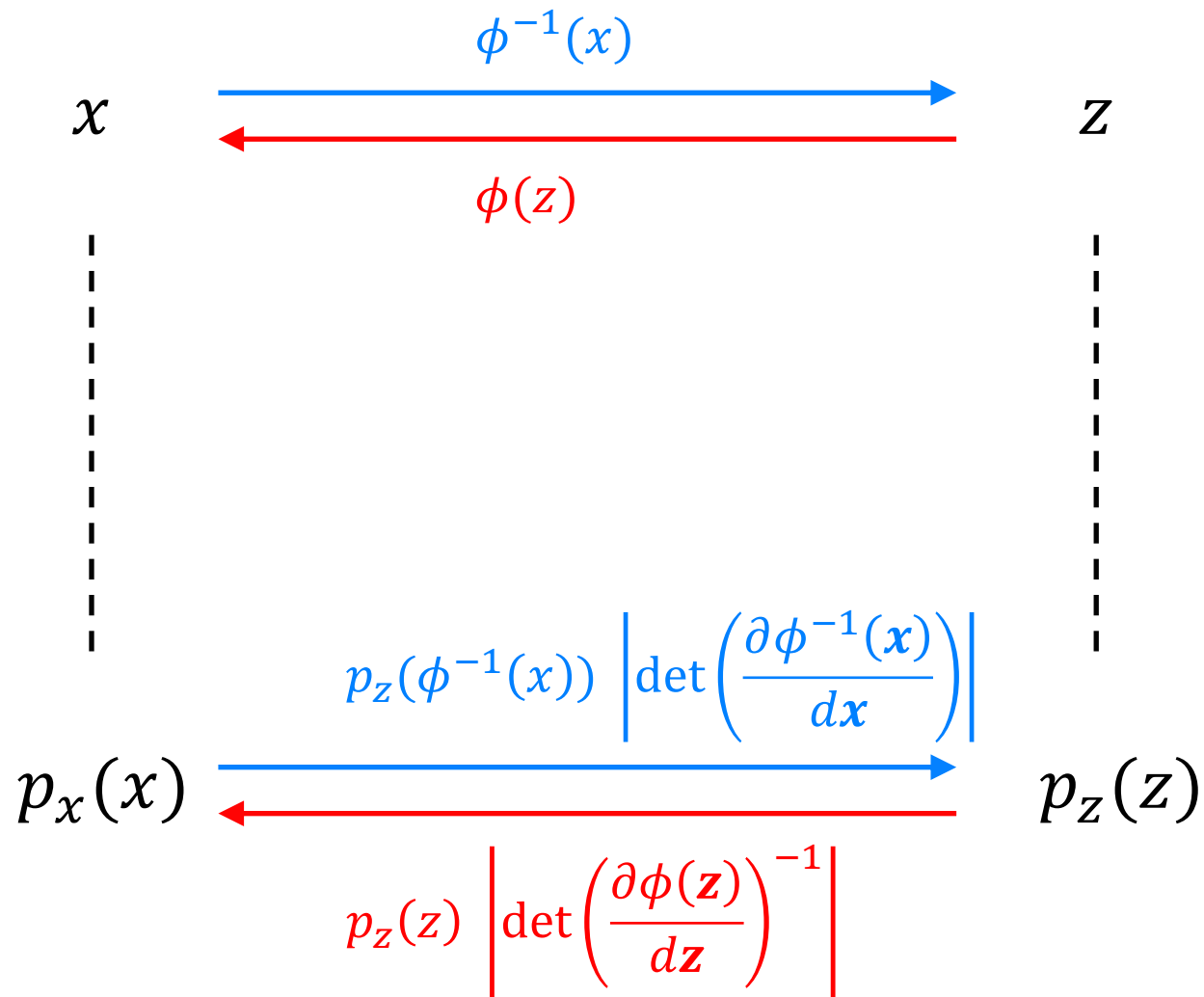




Normalizing Flows

11
7





- **Learn θ** with maximum likelihood

$$\max_{\theta} p(x) = \max_{\theta} p_z(\phi_{\theta}^{-1}(x)) \left| \det \left(\frac{\partial \phi_{\theta}^{-1}(x)}{dx} \right) \right|$$

- Gradient descent on θ
- Find transformation s.t. data is most likely
- **Benefits** once trained
 - Can evaluate $p(x)$ for any point X
 - Can generate “new” data points
 - Sample noise: $z \sim p(z)$
 - Transform: $\phi(z) = x$

Example Normalizing Flow: Real NVP

12
0

- Data vector $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- Transformation

Functions $f()$ and $g()$
are neural networks

$$\phi(\mathbf{z}): \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \phi_1(\mathbf{z}) \\ \phi_2(\mathbf{z}) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 * f(z_1) + g(z_1) \end{pmatrix}$$

$$\phi^{-1}(\mathbf{x}): \quad \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \phi_1^{-1}(\mathbf{x}) \\ \phi_2^{-1}(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} x_1 \\ (x_2 - g(x_1))/f(x_1) \end{pmatrix}$$

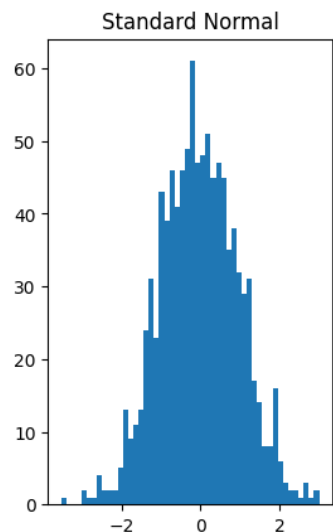
- Determinant:

$$\det\left(\frac{\partial \phi(\mathbf{z})}{d\mathbf{z}}\right) = \det\left(\begin{pmatrix} 1 & 0 \\ \frac{\partial \phi_2(\mathbf{z})}{dz_1} & f(z_1) \end{pmatrix}\right) = f(z_2)$$

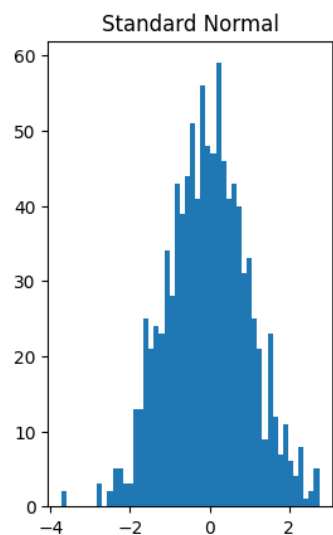
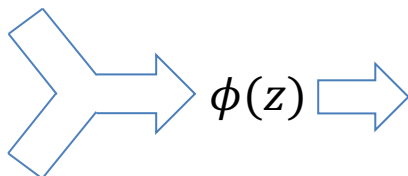
Jacobian is
lower triangular

Example Normalizing flow

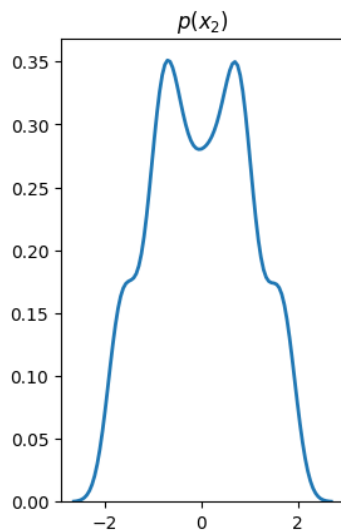
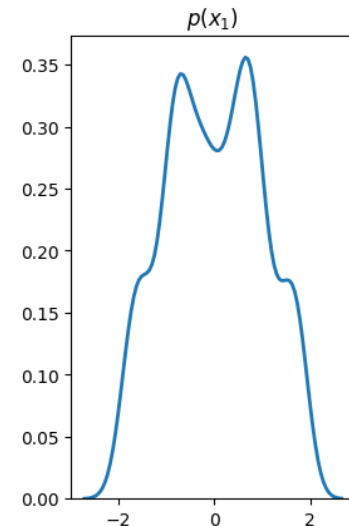
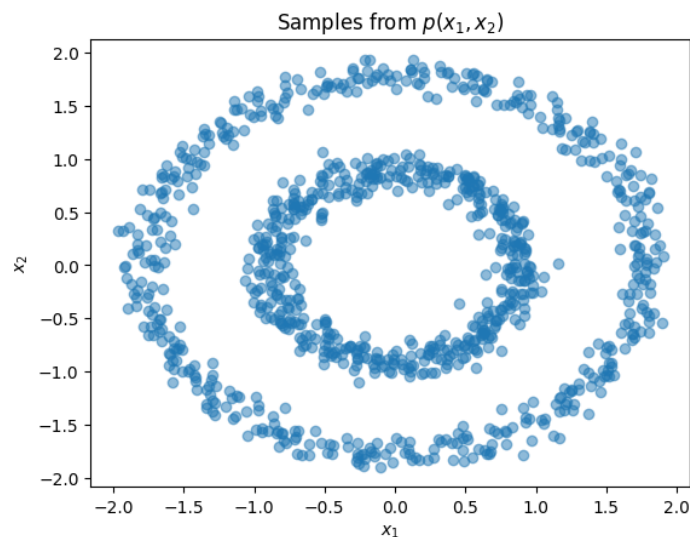
12
1



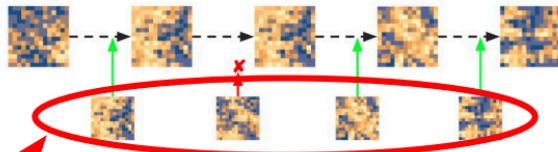
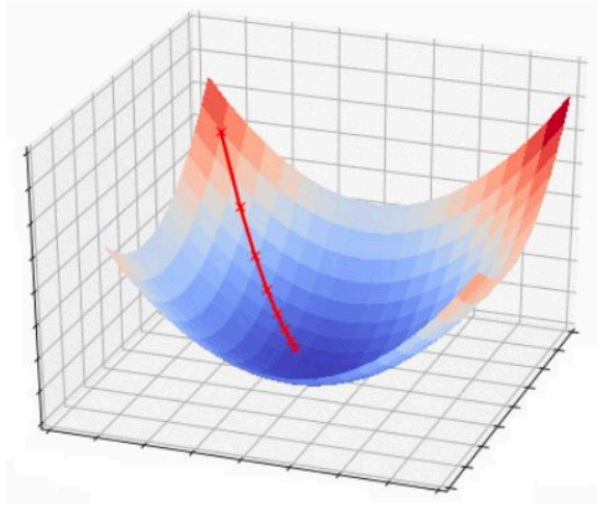
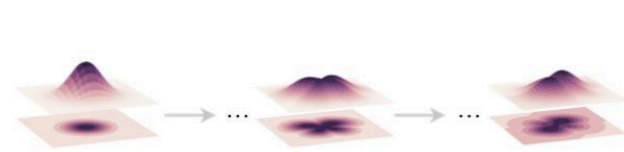
z_1



z_2



Applications: Sampling in Lattice QCD



generating samples is
"embarrassingly parallel"

