Deep Generative Architectures

Oliver Schulte

School of Computing Science

Simon Fraser University

Introduction to Deep Learning

Generative Probabilistic Models

- Supervised learning models $P(\mathbf{y} | \mathbf{x})$: the probability of one or more target variables $\mathbf{y}$ given input variables $\mathbf{x}$
- Generative model $P(\mathbf{x})$: just model the distribution of the inputs

Example: synthetic outputs

- See tutorial
- And face generator

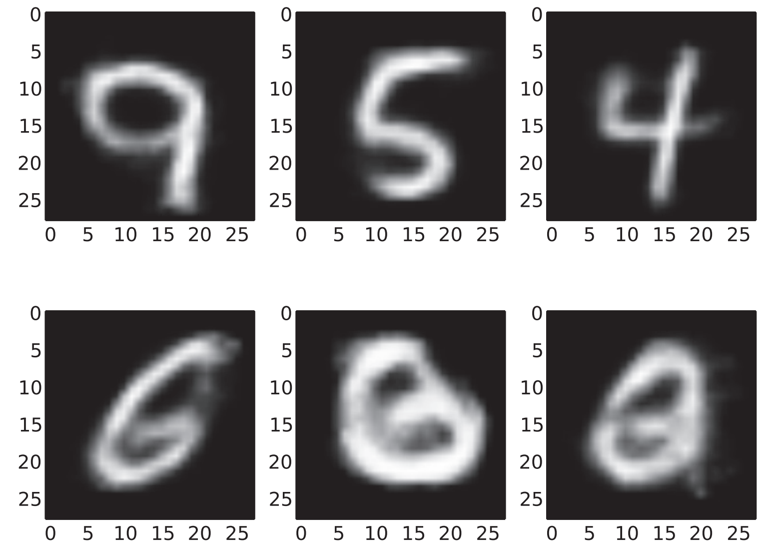
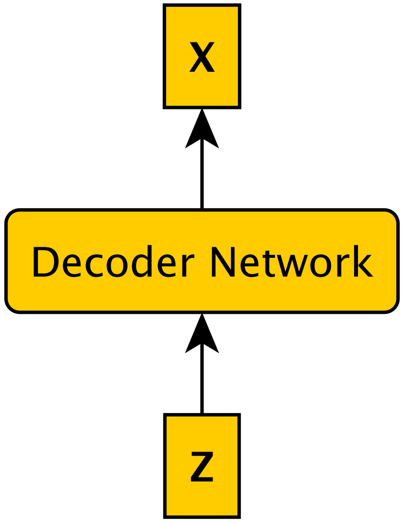


Figure 7.10: VAE Mnist digits generated from scratch

How can we get an NN to generate output? Doesn't it just map inputs to outputs?

General Neural Network Approach

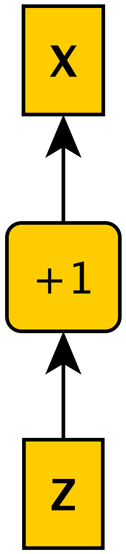
- Input random vector **z** to neural net
  - Typically from a Gaussian bell curve distribution
- Network maps **z** to output vector **x**
- Intuitively, the output should be like the observations **x**



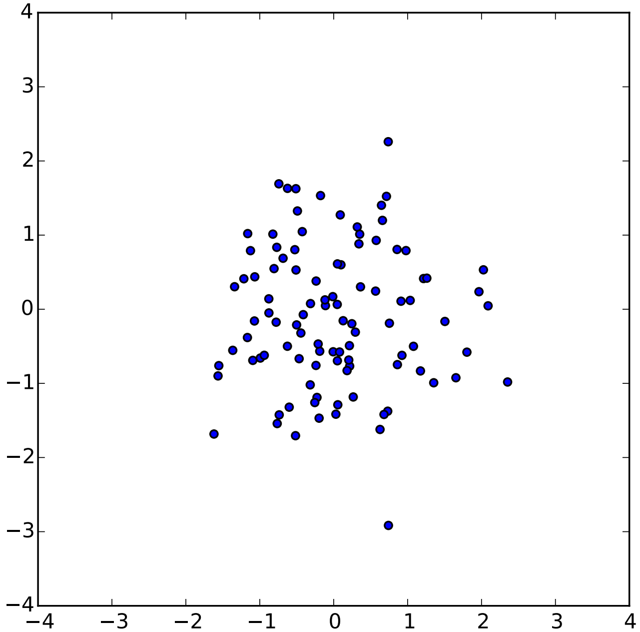
p(Z) = Gaussian bell curve

Simple Example Mappings

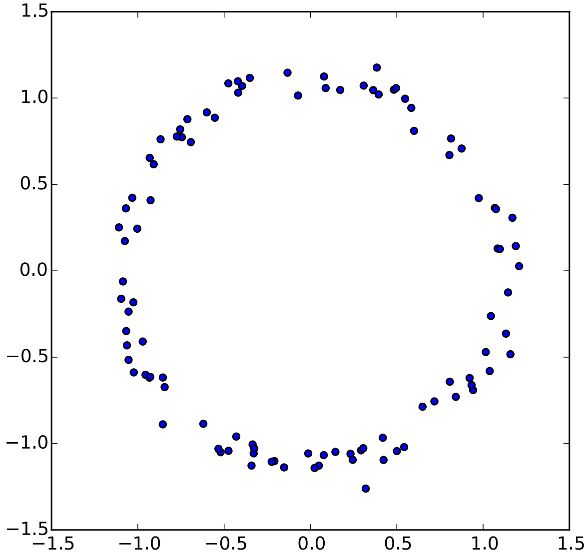
Latent distribution	+1 distribution
Uniform over [0,1]	Uniform over [1,2]
Gaussian with mean 0 and standard deviation 1	Gaussian with mean 1 and standard deviation 1



Generate Ring from Gaussian



f(z) =
z/10+z/||z||



Samples from
Gaussian distribution of 2D **Z**

Samples from output distribution

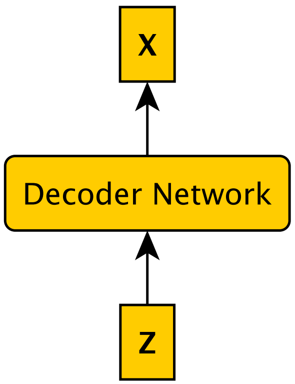
How to Train Deep Generative Models?

- What training objective?
- 2 main ideas
  - Variational Auto-Encoder
  - Generative Adversarial Network

Variational Auto-Encoder (VAE)

VAE Objective Approximates Log-Likelihood

- Loss function for input $\mathbf{x}$ = - $\ln(P(\mathbf{x}))$
  - The negative log-likelihood is the standard loss for generative models
- In the decoder architecture
  $P(\mathbf{x}) \approx \int_{\mathbf{z}} 1(f(\mathbf{z}) = \mathbf{x})\, p(\mathbf{z})\, d\mathbf{z}$
  - $1(f(\mathbf{z}) = \mathbf{x})$ returns 1 if the decoder maps random to $\mathbf{x}$, 0 o.w.
- Integral is intractable
- VAE architecture is designed so that <u>training loss approximates integral negative log-likelihood</u>

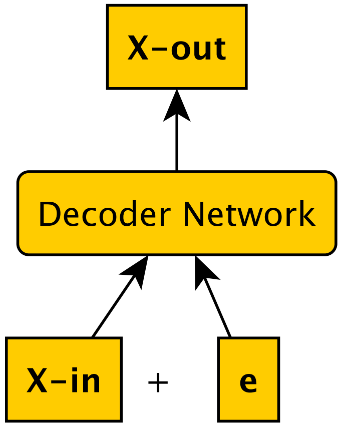


p(Z) = Gaussian bell curve

VAE Architecture

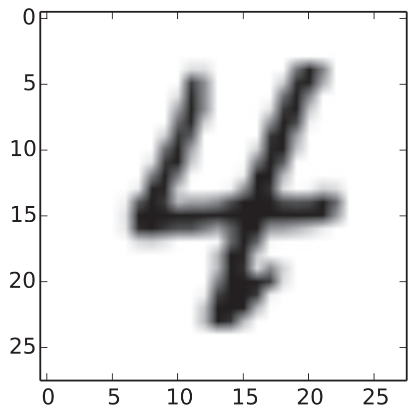
- Combines Auto-Encoding with Generative Modeling
- High-level idea

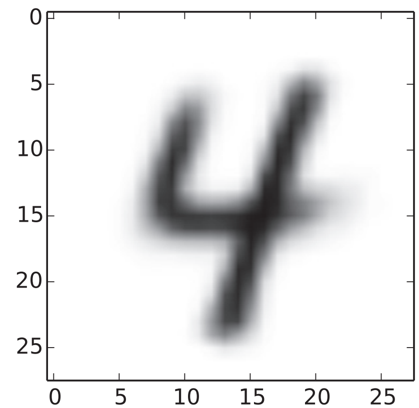
1. Training: Design a non-deterministic version of an auto-encoder
    - Can produce multiple outputs **x**-out for the same input **x**-in
2. Testing: Generate <u>random input</u> **z**, produce (non-deterministic) output



p(e) = Gaussian bell curve

Example: Non-deterministic output





original

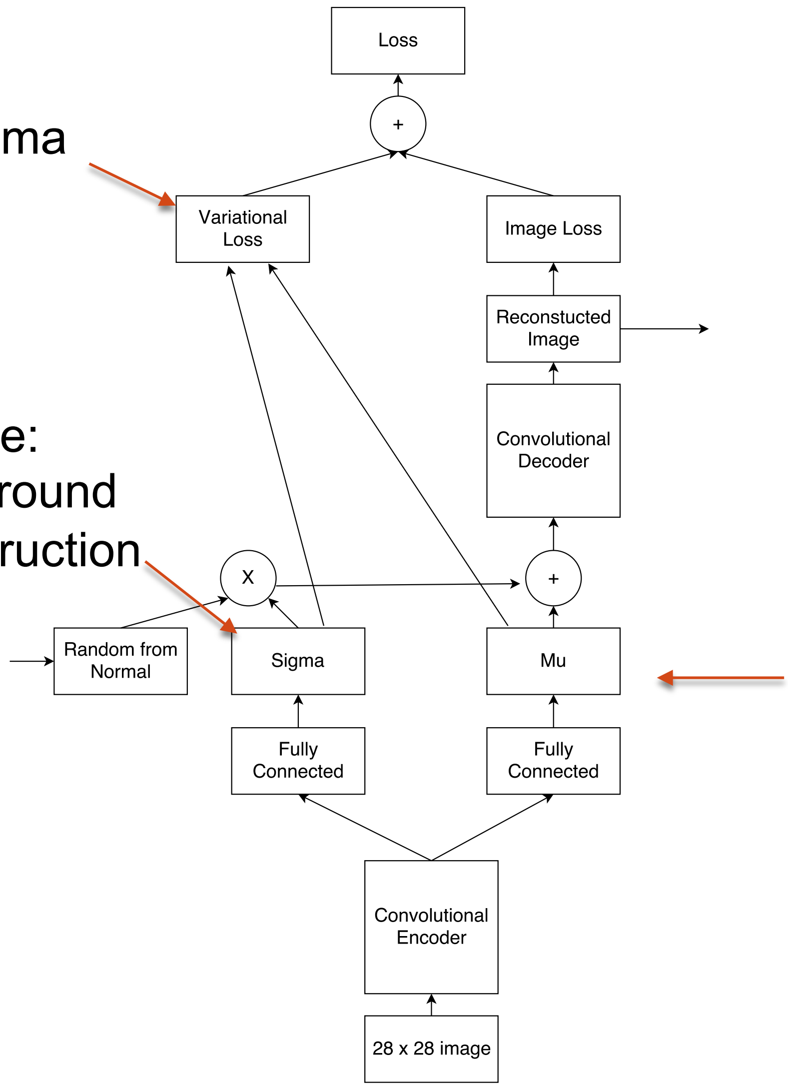
reconstruction

VAE Training Architecture

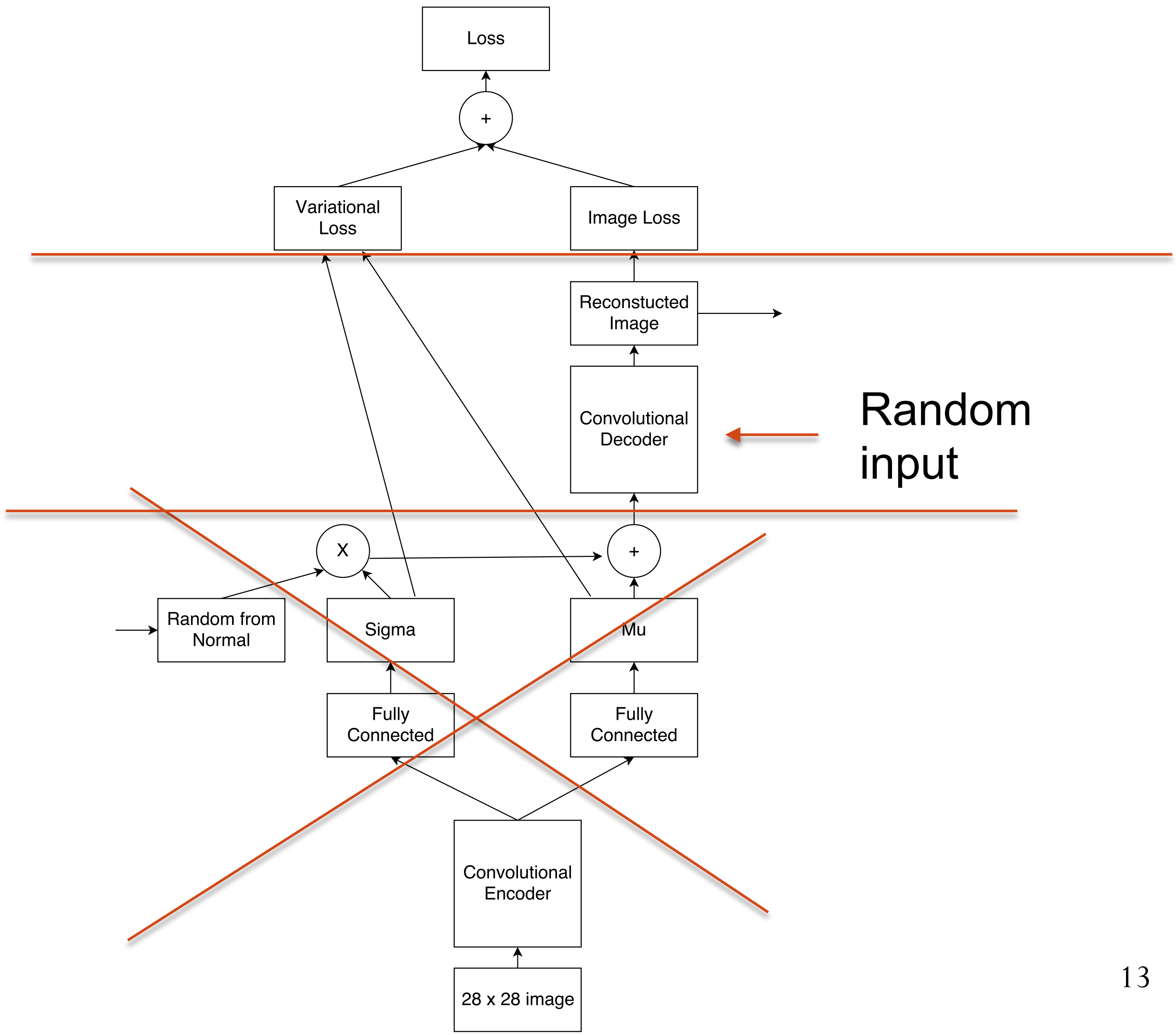
Regularizer for Mu, Sigma

Variance:
noise around
reconstruction

reconstruction/
encoding



VAE Test Architecture



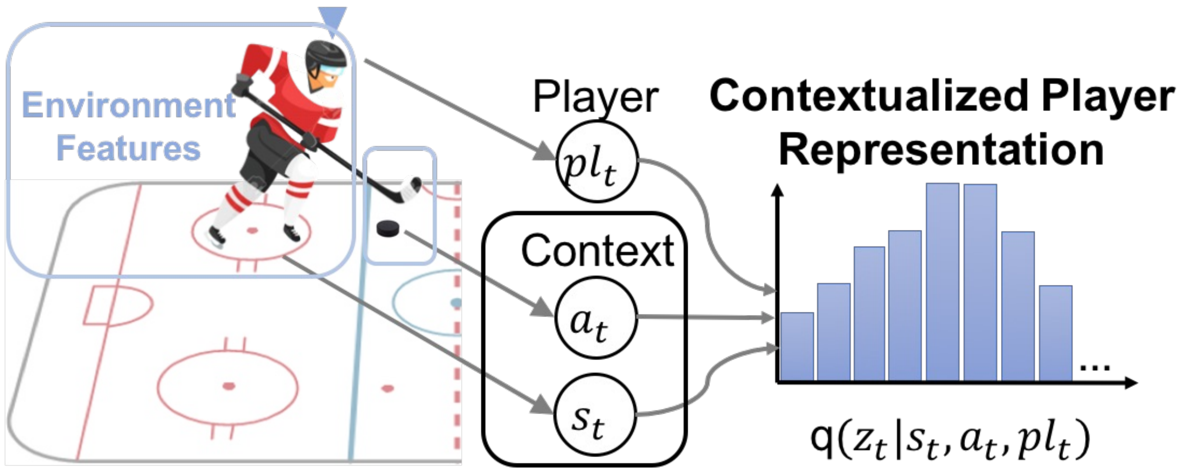
Random input

Example Application

Embeddings for Hockey Players

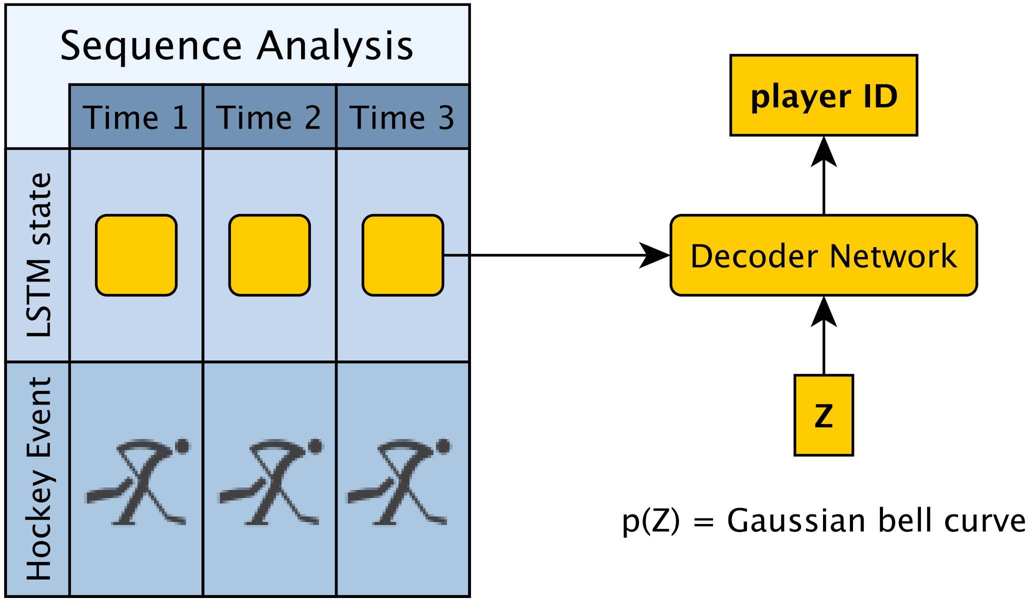
Contextualized Player Embedding

- The set-up: we see a hockey game sequence.
- Using a VAE, compute a contextualized player embedding as a function of the game sequence.
- The VAE generates a distribution P(player$_t$ = i | sequence up to time *t*)



Combining LSTM with VAE

- Sequence is processed using LSTM
- VAE generates probability distribution over players



Accuracy in Player Recognition

Given a match sequence, can you predict which player is likely to have the puck now?

Method	Accuracy	Log-likelihood
LSTM	12.41%	-3.131
LSTM + VAE	48%	-2.228

- The player embeddings help with applications like predicting
  - Whether a shot will lead to a goal
  - The final game outcome

The Evidence Lower Bound

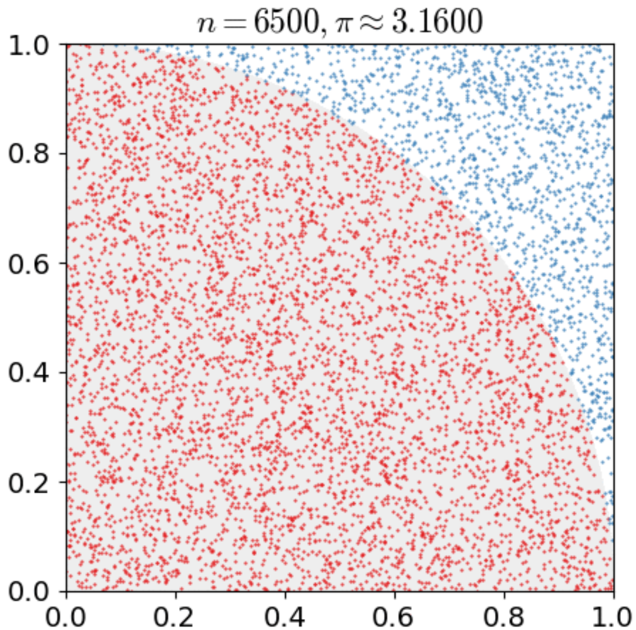
The Mathematics Behind VAEs

Background

- Monte Carlo Sampling
- The KL Divergence

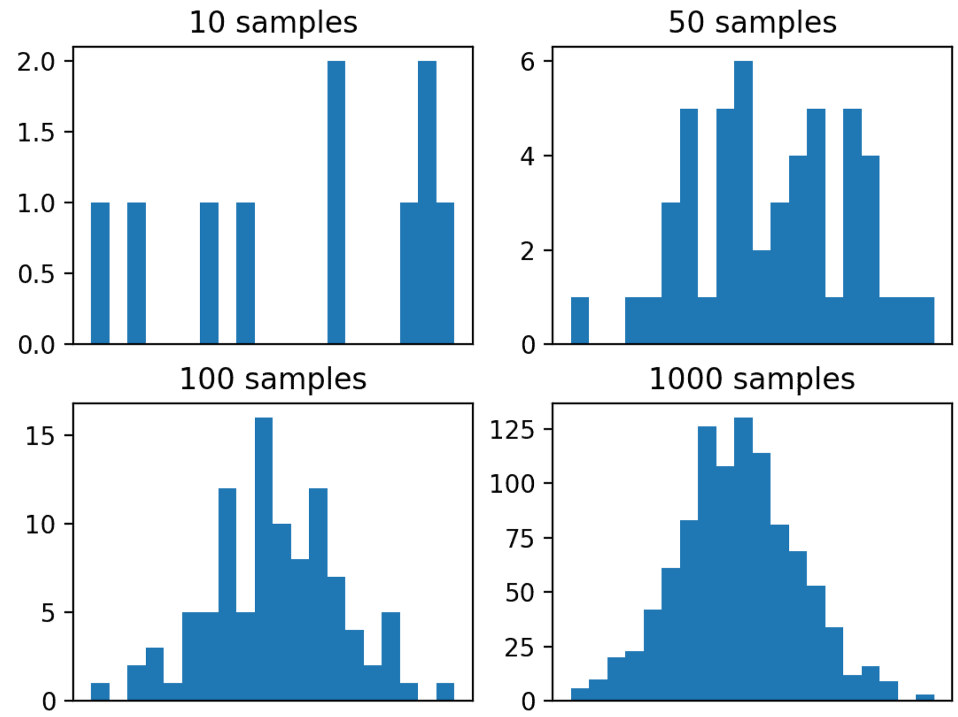
Sampling Example

- Area of unit circle = π
- Estimate π by randomly sampling points in square, checking if they lie within circle. Demo



Sample Complexity

- Basic Problem: Sampling can require <u>many data points</u> for accurate estimate



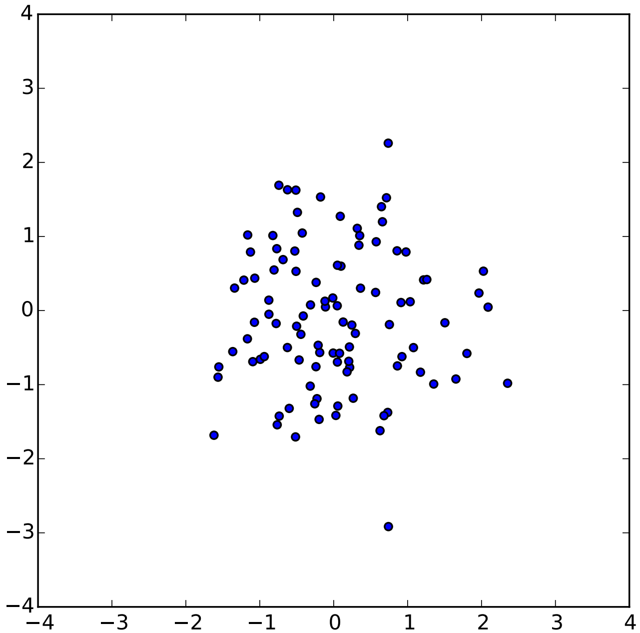
Kulback-Leibler Divergence

- KLD: $D(q||p) = \sum_j q(\mathbf{x}_j) \log(q(\mathbf{x}_j)/p(\mathbf{x}_j))$
  $= E_q[\log(q(\mathbf{x})/p(\mathbf{x}))]$
- Entropy: $H(q) = - \sum_j q(\mathbf{x}_j) \log(q(\mathbf{x}_j))$
  $= - E_q[\log(q(\mathbf{x})]$
- Cross-Entropy: $H(q,p) = - \sum_j q(\mathbf{x}_j) \log(p(\mathbf{x}_j))$
  $= - E_q[\log(p(\mathbf{x})]$
- Exercise: Prove that $D(p||q) = H(q,p) – H(q)$
- Exercise: Recall the "cross-entropy" of Assignment
  $\sum_{j=1}^{N} y_j ln(p_j^+) + (1 - y_j) ln(1 - p_j^+)$
  What is the relationship between this formula and H(q,p)?

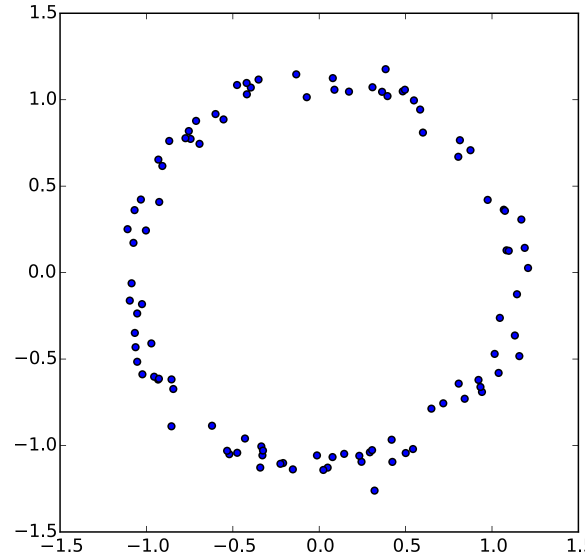
Likelihood: The True Objective

- Recall that the standard objective for generative modelling is the data likelihood
  $\Pi_j \, P(\mathbf{x}_j; f) = \int_{\mathbf{z}} 1(f(\mathbf{z}) = \mathbf{x}_j) \, p(\mathbf{z}) \, d\mathbf{z}$
  for observed data points $\mathbf{x}_1, \ldots, \mathbf{x}_j$
- We could estimate this integral for each data point using Monte Carlo estimation.
  1. Sample k $\mathbf{z}$ points,
  2. apply f
  3. Calculate the mean estimate: average $f(\mathbf{z}_1), \ldots, f(\mathbf{z}_k)$

Sampling With Functions



f(z) =
z/10+z/||z||



Samples from Gaussian distribution of 2D P(**Z**)

Samples from output distribution P(**X**)

- If f is invertible, then for each output **x**, there is a unique $f^{-1}(z)$ that generates it.
- Even if f is not invertible, typically for each output **x**, few samples will generate **x**.

Invertible Function Case

- We can replace
  $P(\mathbf{x}; f) = \int_{\mathbf{z}} 1(f(\mathbf{z}) = \mathbf{x})\, p(\mathbf{z})\, d\mathbf{z} = \int_{\mathbf{z}} p(f^{-1}(\mathbf{x}))\, dz$
- I.e., just consider probability of the random input that generates the observed output $\mathbf{x}$
- How to compute $f^{-1}(\mathbf{x})$?
  - Learn another neural network for f^{-1}!

$$\boxed{\mathbf{x}} \xrightarrow{f^{-1}} \boxed{\mathbf{z}} \xrightarrow{f} \boxed{\mathbf{x}}$$

Probabilistic Version

- Even if the generating function *f* is not invertible, few samples **z** have a significant likelihood of generating an observed data point **x**.
- We can measure this with the *posterior probability* $p(\mathbf{z} \mid \mathbf{x})$: given that **x** was observed, what is the probability that **z** generated **x**.
- The posterior probability is the inverse of a *conditional probability* model $p(\mathbf{x} \mid \mathbf{z})$.
- $P(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z})\, \underline{p(\mathbf{z})}\, d\mathbf{z} \approx \int_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z})\, \underline{p(\mathbf{z} \mid \mathbf{x})}\, d(\mathbf{z})$

$$\boxed{\mathbf{x}} \xrightarrow{p(z|x)} \boxed{\mathbf{z}} \xrightarrow{p(x|z)} \boxed{\mathbf{x}}$$

Approximate Posterior

- Given the approximation
  $P(\mathbf{x}) \approx \int_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z} \mid \mathbf{x})\, d(\mathbf{z})$
- How can we compute $p(\mathbf{z} \mid \mathbf{x})$?
- Learn a neural network to output (the parameters of) $p(\mathbf{z} \mid \mathbf{x})$!
- Hard to get this exactly so we try to learn an *approximate posterior* $q(\mathbf{z} \mid \mathbf{x}) \approx p(\mathbf{z} \mid \mathbf{x})$

$$\boxed{\mathbf{x}} \xrightarrow{\text{Encoder}} \boxed{q(\mathbf{z}|\mathbf{x})} \xrightarrow{\text{Sampling}} \boxed{\mathbf{z}} \xrightarrow{\text{decoder}} \boxed{p(\mathbf{x}|\mathbf{z})}$$

Monte Carlo Estimation of Data Likelihood

- $P(\mathbf{x}) \approx \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})\, p(\mathbf{z}|\mathbf{x})\, d(\mathbf{z})$

  $\approx E_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\, p(\mathbf{x}|\mathbf{z})$
- For each data point $\mathbf{x}$
  1. Apply the encoder to find an approximate posterior distribution $q(\mathbf{z}|\mathbf{x}_j)$
  2. Sample $\mathbf{z}$ from $q(\mathbf{z}|\mathbf{x}_j)$
  3. Apply the decoder to compute $p(\mathbf{x}|\mathbf{z})$
  4. Average the estimates from the $\mathbf{z}$ samples to get an estimated $P(\mathbf{x})$
- See VAE tutorial
- What is the relationship between the encoder-decoder approximations and the true data likelihood $P(\mathbf{x})$?

The Evidence Lower Bound

- The following relationship holds for *any* approximate posterior $q(\mathbf{z}|\mathbf{x})$, prior $p(\mathbf{z})$, conditional probability $p(\mathbf{x}|\mathbf{z})$, unconditional probability $p(\mathbf{x})$

  $$\log(p(\mathbf{x})) \geq E_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log(p(\mathbf{x}|\mathbf{z}))] - \mathrm{KLD}(q(\mathbf{z}|\mathbf{x})\,||\,p(\mathbf{z}))$$

True log-prob

Sample z from approximate posterior (encoder)

Compute log-probability of **x** given **z** (decoder)

Penalize differences between app. Posterior and true prior

- The bigger the RHS, the better our approximation.
- So we try to maximize it:

  $$\operatorname{argmax}_{p,q} \sum_j E_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x}_j)}[\log(p(\mathbf{x}_j|\mathbf{z}))] - \mathrm{KLD}(q(\mathbf{z}|\mathbf{x}_j)\,||\,p(\mathbf{z}))$$

Implementing the ELBO objective

- The ELBO is a very general result, known for decades.
- The VAE is a neural architecture proposed by Kingma and Welling 2014 that fills in q and p as follows.
- $q(\mathbf{z}|\mathbf{x}) = \text{Normal}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}^2(\mathbf{x}))$
  - $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are vectors of the same dimension as $\mathbf{z}$
  - They are computed by a neural network (encoder)
- $p(\mathbf{x}|\mathbf{z}) = \text{Normal}(f(\mathbf{z}), \boldsymbol{\sigma}^2 * I)$
  - $f(\mathbf{z})$ is a vector of the same dimension as $\mathbf{x}$
  - $\boldsymbol{\sigma}^2$ is a hyper-parameter
  - $f(\mathbf{z})$ is computed by a neural network (decoder)
- Visualization

KLD for Gaussians

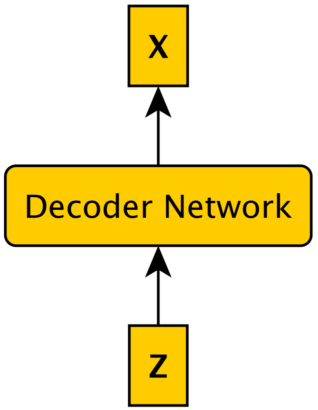
- The ELBO requires us to evaluate KLD($q(\mathbf{z}|\mathbf{x}_j)||p(\mathbf{z})$)
- For two multi-variate Gaussians $N_Q(m_Q, \Sigma_Q)$ and $N_p(m_p, \Sigma_p)$, the KLD divergence is given by the formula
  $D(N_Q||N_p) = ½$
  $\{\mathrm{tr}\ (\Sigma^{-1}{}_p\ \Sigma_q) + \ln|\ \Sigma_p\ | - \ln|\ \Sigma_Q\ | + (m_p - m_Q)^T\ \Sigma^{-1}{}_p\ (m_p - m_Q)^T\ \} - d/2$
  where
- d is the latent embedding dimension
- tr is the matrix trace (sum of diagonal elements)
- In a VAE, we have $N_p=(\mathbf{0}, I)$ and Σ_q is diagonal with entries $\boldsymbol{\sigma}_i^2$ for $i = 1,..,d$
- Exercise: Show that
  $D(N_Q||N_p) = ½\{\sum_i \boldsymbol{\sigma}_i^2 - \ln(\boldsymbol{\sigma}_i^2) + m_i^2\} - d/2$ where $m = m_Q$

Generative Adversarial Models

GANs

GAN Architecture

- Recall the basic generative NN architecture
- How to train?
- VAE: approximate log-likelihood of the observations
- GAN: train the generator so that **synthetic generated examples cannot be distinguished from actual observations**



p(Z) = Gaussian bell curve

Intuition: Taste Test

- Coke is the real thing
- Pepsi wants to imitate coke
- How does Pepsi know they have succeeded?
- When a blind taste test cannot tell the difference!

Chatbots and the Turing Test

Alan Turing (1950) "Computing Machines and Intelligence"

- How can we know if a machine is intelligent?

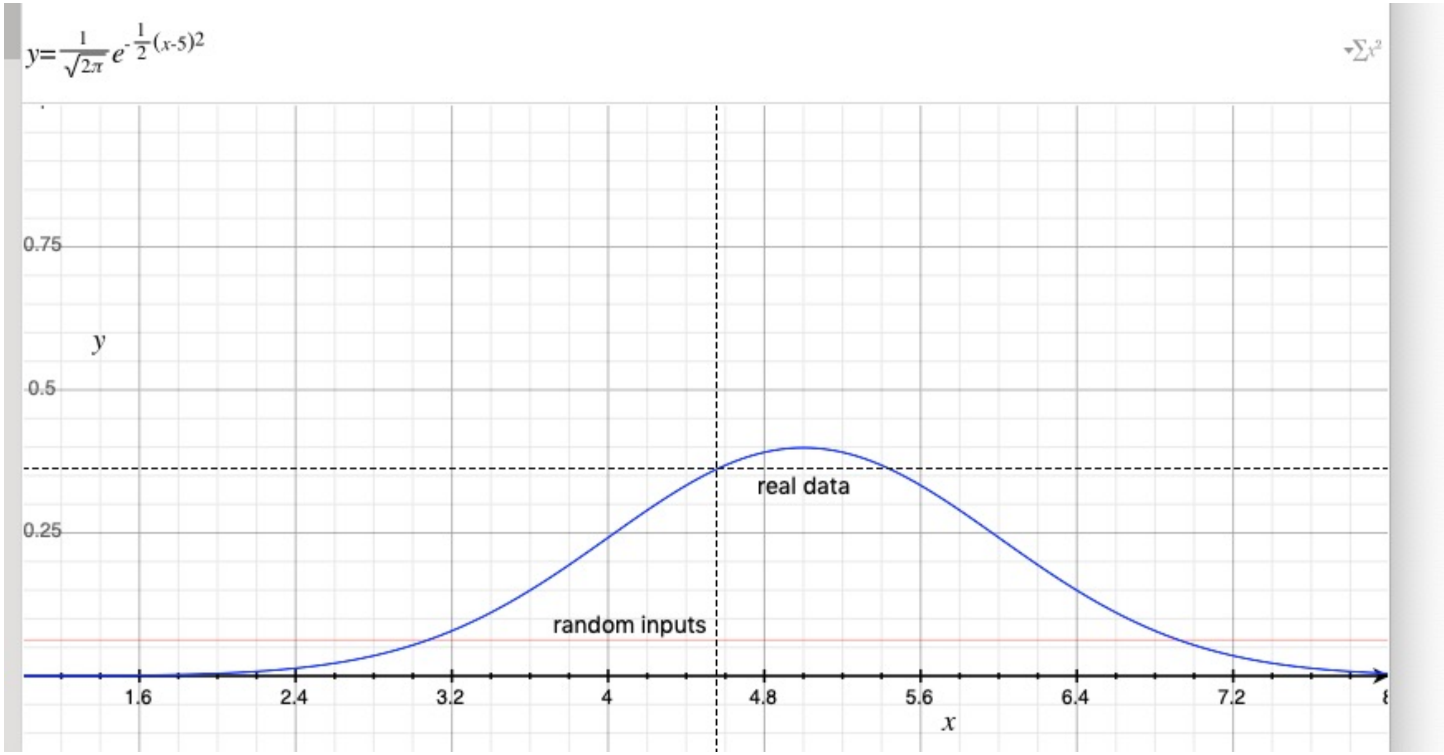
➢ Test if it can fool a human!

- Loebner Prize
- ChatGPT

Distinguishing Uniform and Gaussians

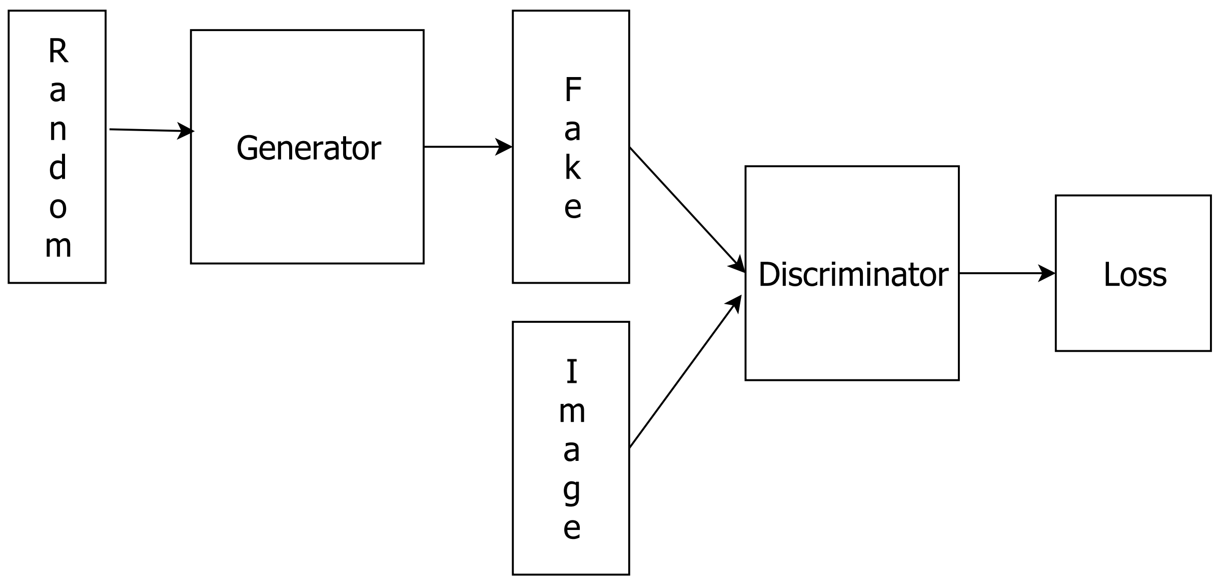
- Fake data: uniform [-8,8]

  Real data: Gaussian with mean 5, standard deviation 1



The Tester

- How can we test whether a network generates realistic outputs?
- Train another classifier network to distinguish synthetic from real examples!
- Discriminator D outputs $o_D(x) = P(\mathbf{x}$ is real example$)$
- Why can we not just backpropagate to train the generator?



The Loss Functions

- Discriminator objective function given real examples R and fake examples F
  $V(D,G) = mean_{\boldsymbol{r}}\, ln(\boldsymbol{o}_D(\boldsymbol{r})) + mean_{\boldsymbol{f}}\, ln(1\text{-}\boldsymbol{o}_D(\boldsymbol{r}))$
- The discriminator wants to maximize V, the generator to minimize V
- Minmax problem: GANs are tricky to train

	Discriminator			**Generator**	
x	Output	Objective	Loss	Objective	Loss
Real **r**	$\boldsymbol{o}_D(\boldsymbol{r})$	$\ln(\boldsymbol{o}_D(\boldsymbol{r}))$	$-\ln(\boldsymbol{o}_D(\boldsymbol{r}))$	$-\ln(\boldsymbol{o}_D(\boldsymbol{r}))$	$\ln(\boldsymbol{o}_D(\boldsymbol{r}))$
Fake **f**	$\boldsymbol{o}_D(\boldsymbol{f})$	$\ln(1\text{-}\boldsymbol{o}_D(\boldsymbol{f}))$	$-\ln(1\text{-}\boldsymbol{o}_D(\boldsymbol{f}))$	$-\ln(1\text{-}\boldsymbol{o}_D(\boldsymbol{f}))$	$\ln(1\text{-}\boldsymbol{o}_D(\boldsymbol{f}))$

Other Loss Functions

- Other loss functions can be considered to help with training, e.g. in the book:
- The generator wants $\boldsymbol{o}_D(\boldsymbol{f})$ to be big
  - So maximize ln $\boldsymbol{o}_D(\boldsymbol{f})$ or minimize $-\ln \boldsymbol{o}_D(\boldsymbol{f})$
- However it is more usual to ensure the zero-sum condition: loss(discriminator) = - loss(generator)

Book	Discriminator			Generator	
x	Output	Objective	Loss	Objective	Loss
Real **r**	$\boldsymbol{o}_D(\boldsymbol{r})$	$\ln(\boldsymbol{o}_D(\boldsymbol{r}))$	$-\ln(\boldsymbol{o}_D(\boldsymbol{r}))$		
Fake **f**	$\boldsymbol{o}_D(\boldsymbol{f})$	$\ln(1-\boldsymbol{o}_D(\boldsymbol{f}))$	$-\ln(1-\boldsymbol{o}_D(\boldsymbol{f}))$	$\ln(\boldsymbol{o}_D(\boldsymbol{f}))$	$-\ln(\boldsymbol{o}_D(\boldsymbol{f}))$

Examples and Demos

- Style GAN
- Deep Fakes
- GANlab
- nvidia-ai-playground

Conclusion

- Generative models generate outputs without an input
- Basic idea: map a random input to an output
- Without a target output, what is the training objective?
- Variational AE: approximate the log-likelihood of the observed data $\mathbf{x}$
  - Also produces an embedding of the input $\mathbf{x}$
- Generative Adversarial Network: generate synthetic outpus that cannot be distinguished from actual observations by an adversarial classifier network