[course site]

#DLUPC

Deep Generative Models I

Santiago Pascual de la Puente
santi.pascual@upc.edu

PhD Candidate
Universitat Politecnica de Catalunya
Technical University of Catalonia

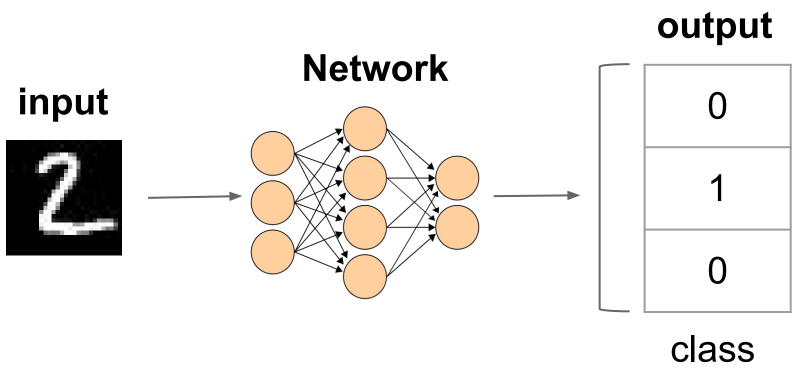
Outline

- Introduction
- Taxonomy
- PixelCNN & Wavenet
- Variational Auto-Encoders (VAEs)
- Generative Adversarial Networks (GANs)
- Application examples
- Model comparison

Introduction

What we are used to with Neural Nets

Discriminative model: P(Y | X)



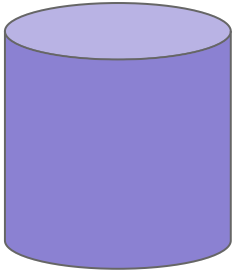
$$P(Y = [0,1,0] \mid X = [pixel_1, pixel_2, \ldots, pixel_{784}])$$

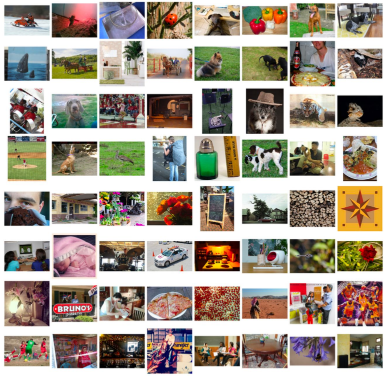
Figure credit: Javier Ruiz

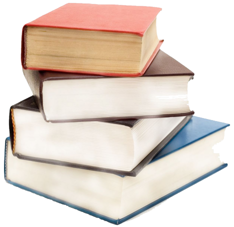
What is a generative model?

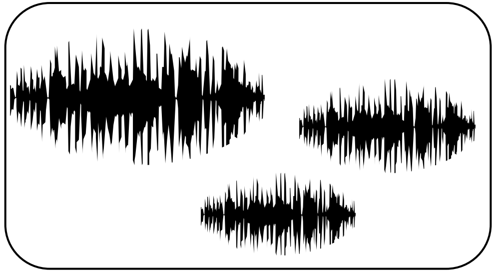
We have generic unlabeled datapoints:

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$





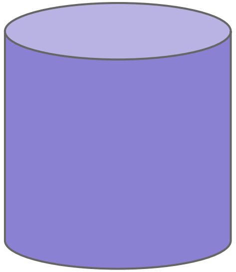


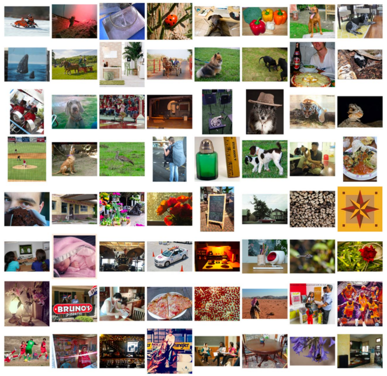


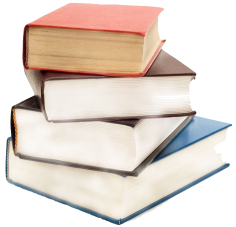
What is a generative model?

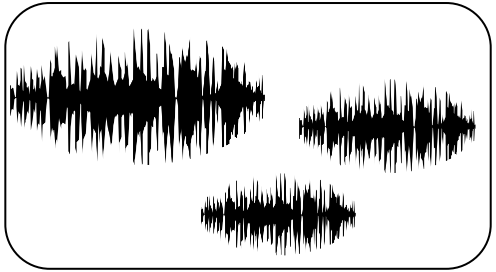
We have generic unlabeled datapoints:

$$X = \{x_1, x_2, \ldots, x_N\}$$







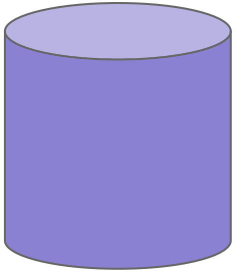


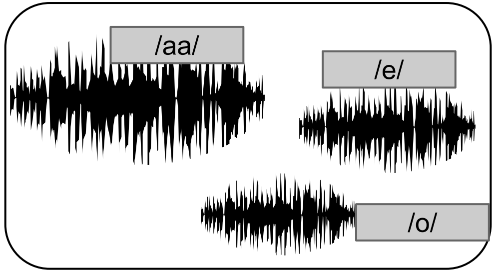
$X = \{x_1, x_2, \ldots, x_N\}$ follows pdf $P(X)$ → Model it!

What is a generative model?

We have generic labeled datapoints:

$$X = \{x_1, x_2, \dots, x_N\}$$
$$Y = \{x_1, x_2, \dots, x_N\}$$



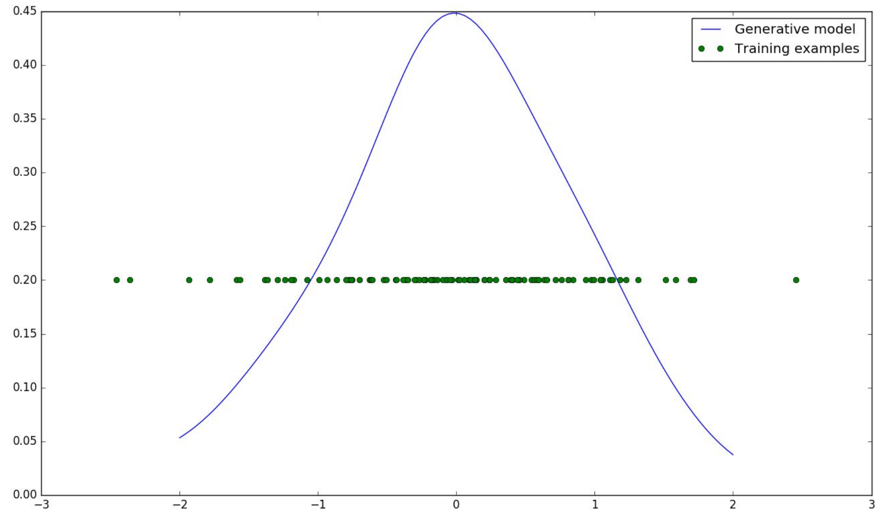


We can also model joint P(X, Y) !

What is a generative model?

We want our model with parameters $\boldsymbol{\theta}$ to output samples distributed *Pmodel,* matching the distribution of our training data *Pdata* or P(**X**).

Example) y = f(**x**), where y is scalar, make *Pmodel* similar to *Pdata* by training the parameters $\boldsymbol{\theta}$ to maximize their similarity.



What is a generative model?

Key Idea: our model cares about what distribution generated the input data points, and we want to mimic it with our probabilistic model. **Our learned model should be able to make up new samples from the distribution**, **not just copy and paste existing samples**!

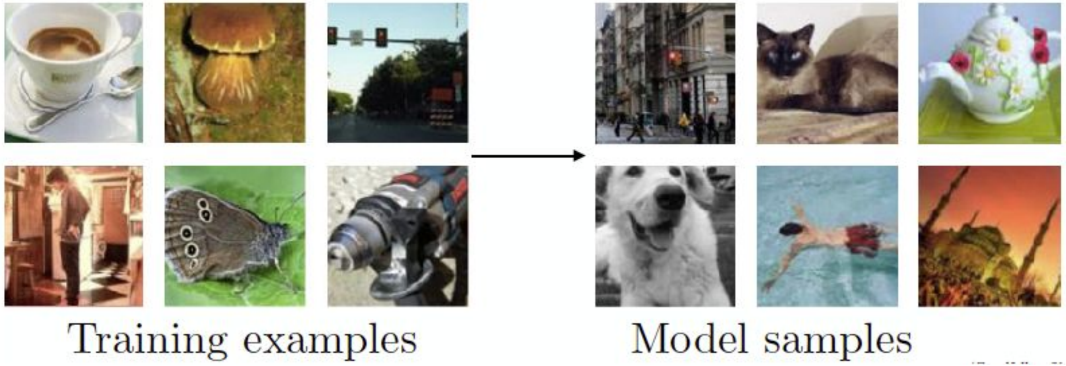


Figure from [NIPS 2016 Tutorial: Generative Adversarial Networks (I. Goodfellow)]

Why Generative Models?

- Model very complex and high-dimensional distributions.
- Be able to generate realistic synthetic samples
  - possibly perform data augmentation
  - simulate possible futures for learning algorithms
- Fill the blanks in the data
- Manipulate real samples with the assistance of the generative model
  - Example: edit pictures with guidance

Motivating Applications

Image inpainting

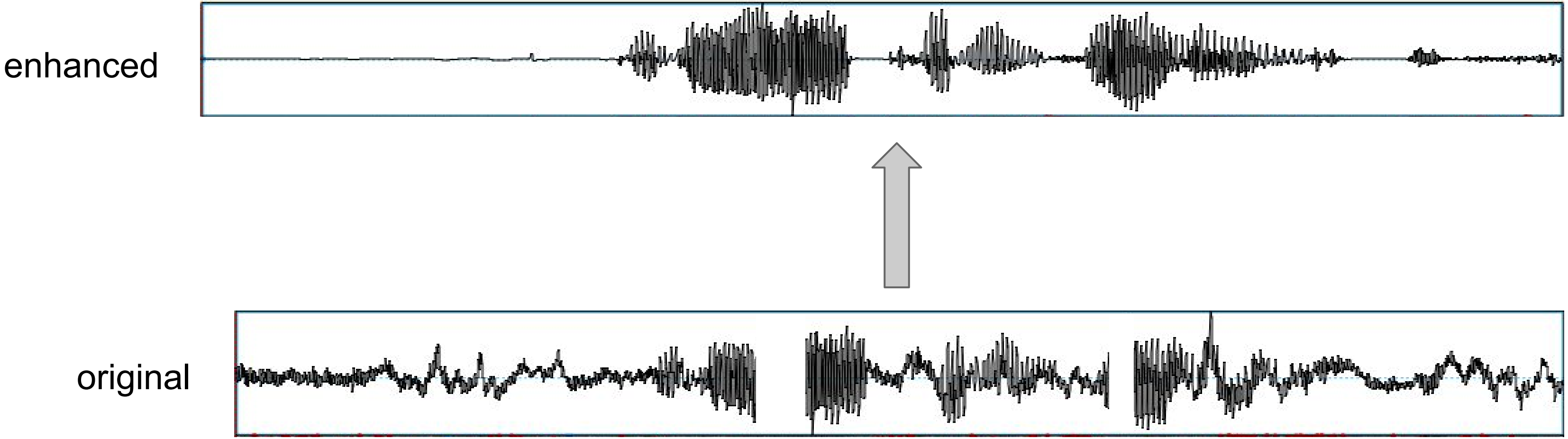
Recover lost information/add enhancing details by learning the natural distribution of pixels.

original

enhanced

Speech Enhancement

Recover lost information/add enhancing details by learning the natural distribution of audio samples.



Speech Synthesis

Generate spontaneously new speech by learning its natural distribution along time.

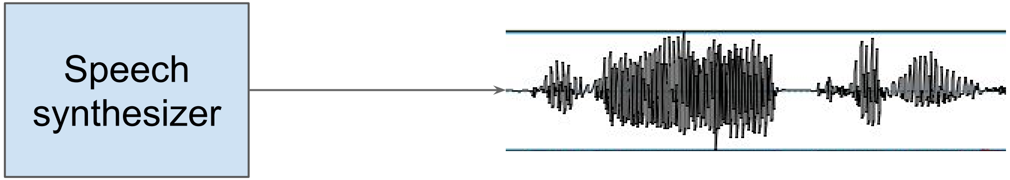


Image Generation

Generate spontaneously new images by learning their spatial distribution.

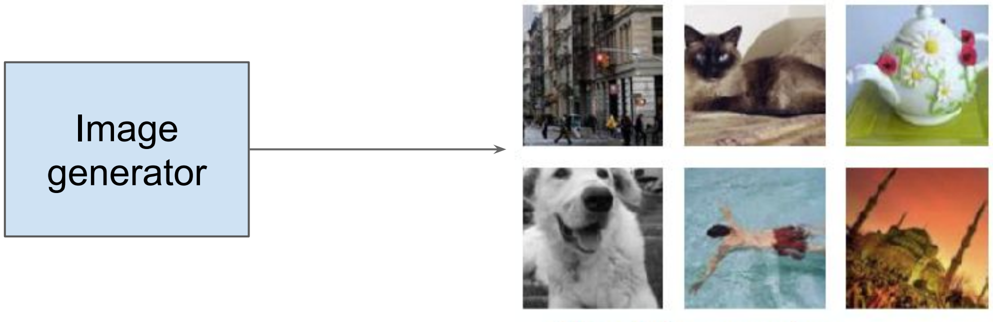
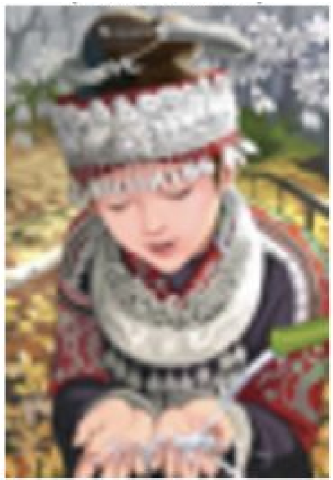


Figure credit: I. Goodfellow

Super-resolution

Generate spontaneously new images by learning their spatial distribution.



Augment resolution introducing plausible details

(Ledig et al. 2016)

Generative Models Taxonomy

Taxonomy

Model the probability density function:

- Explicitly
  - With tractable density → PixelRNN, PixelCNN and Wavenet
  - With approximate density → Variational Auto-Encoders
- Implicitly
  - Generative Adversarial Networks

PixelRNN, PixelCNN & Wavenet

Factorizing the joint distribution

- Model **explicitly** the **joint probability distribution** of data streams **x** as a product of element-wise conditional distributions for each element x_i in the stream.
  - **Example:** An image **x** of size (n, n) is decomposed scanning pixels in raster mode (Row by row and pixel by pixel within every row)
  - Apply **probability chain rule**: x_i is the i-*th* pixel in the image.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.
  - **Q: How can we deal with this (which model)?**

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

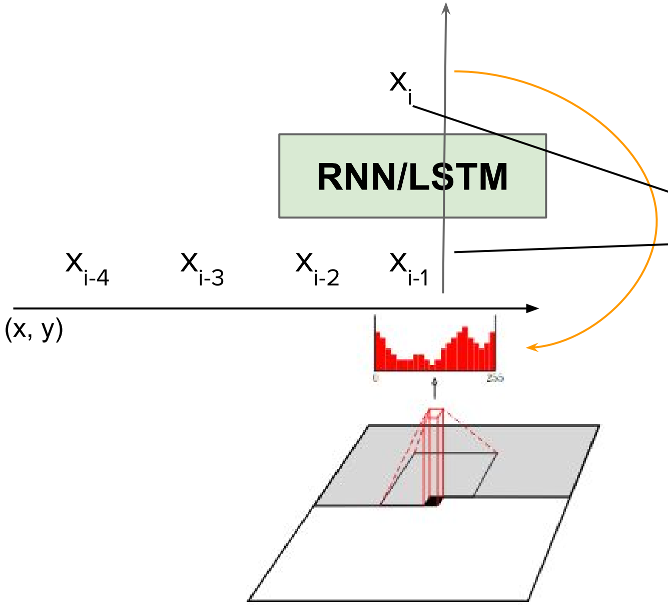
Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.
    - **Q: How can we deal with this (which model)?**
        - A: Recurrent Neural Network.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

PixelRNN

An RNN predicts the probability of each sample x_i with a categorical output distribution: Softmax



Recall RNN formulation:

$$\boxed{h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)}$$

At every pixel, in a greyscale image: 256 classes (8 bits/pixel).

Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.

  - **Q: How can we deal with this (which model)?**
    - A: Recurrent Neural Network.
    - B: Convolutional Neural Network.

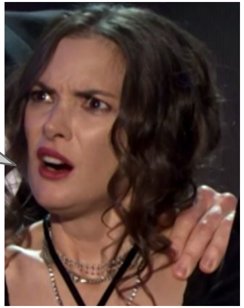
$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.

    - **Q: How can we deal with this (which model)?**
        - A: Recurrent Neural Network.
        - B: Convolutional Neural Network.

A CNN?
Whut??

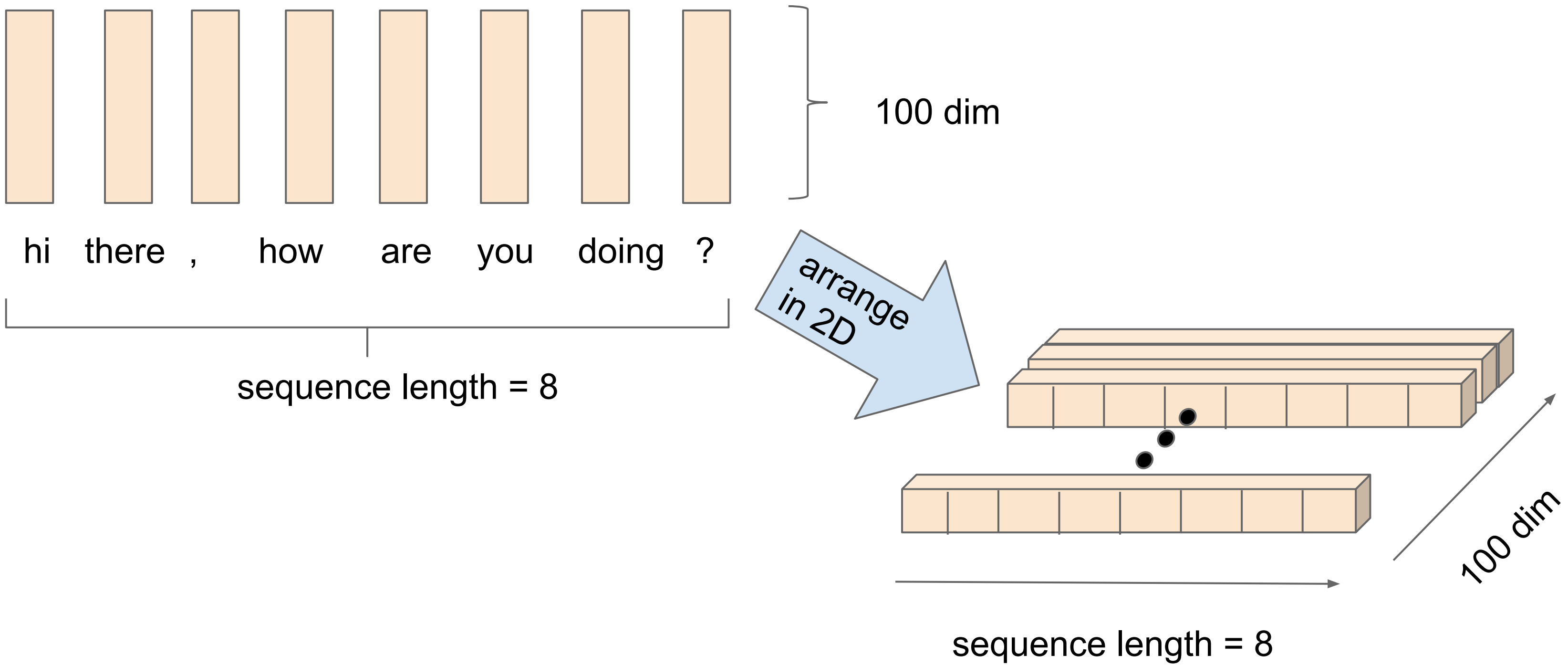


$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

My CNN is going causal...

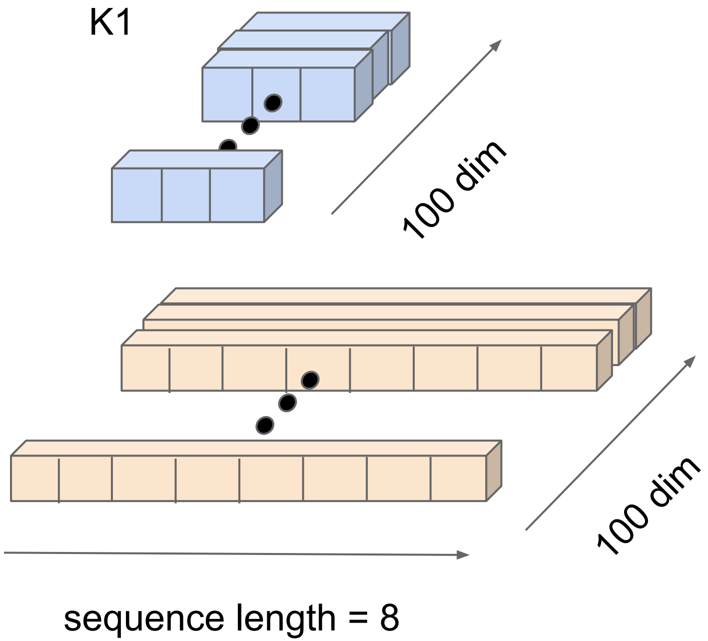
Focus in 1D to exemplify causalitaion

Let’s say we have a sequence of 100 dimensional vectors describing words.



My CNN is going causal...

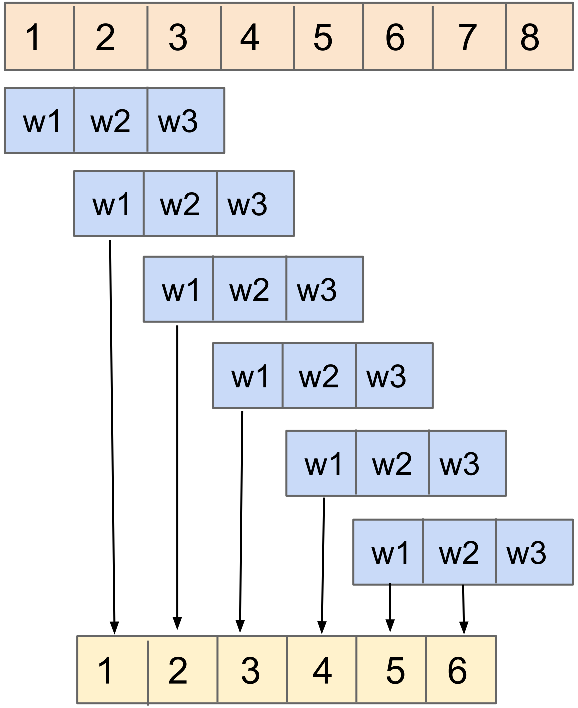
We can apply a 1D convolutional activation over the 2D matrix: for an arbitrari kernel of width=3



Each 1D convolutional kernel is a 2D matrix of size (3, 100)

My CNN is going causal...

Keep in mind we are working with 100 dimensions although here we depict just one for simplicity

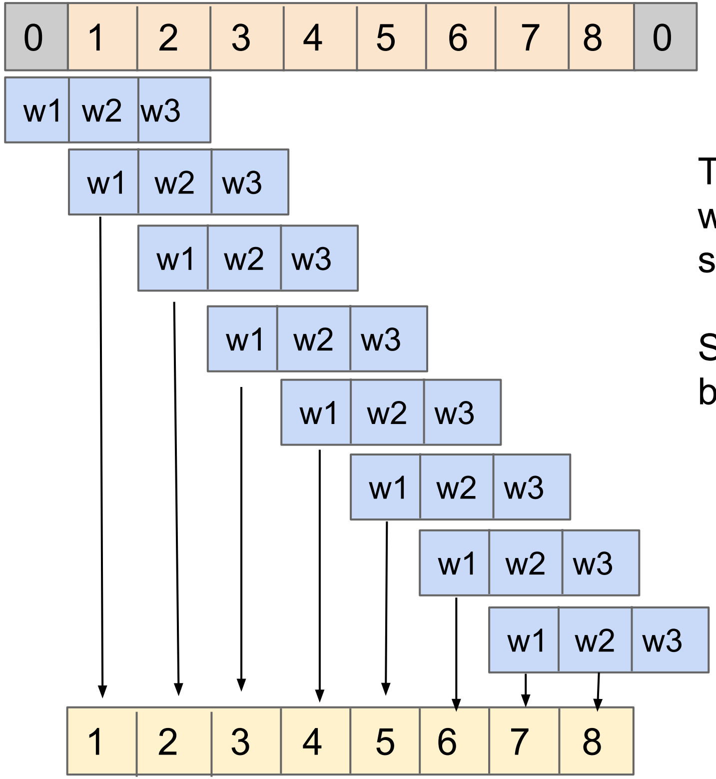


The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 8 - 3 + 1 = 6

So the output matrix will be (6, 100) because there was no padding

My CNN is going causal...

When we add zero padding, we normally do so on both sides of the sequence (as in image padding)

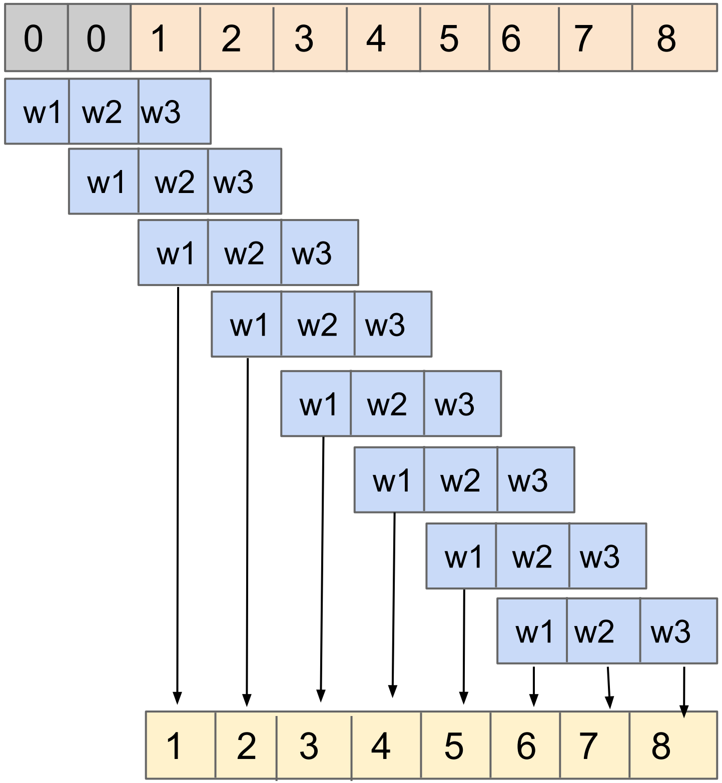


The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 10 - 3 + 1 = 8

So the output matrix will be (8, 100) because we had padding

My CNN went causal

Add the zero padding just on the left side of the sequence, not symmetrically



The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 10 - 3 + 1 = 8

So the output matrix will be (8, 100) because we had padding

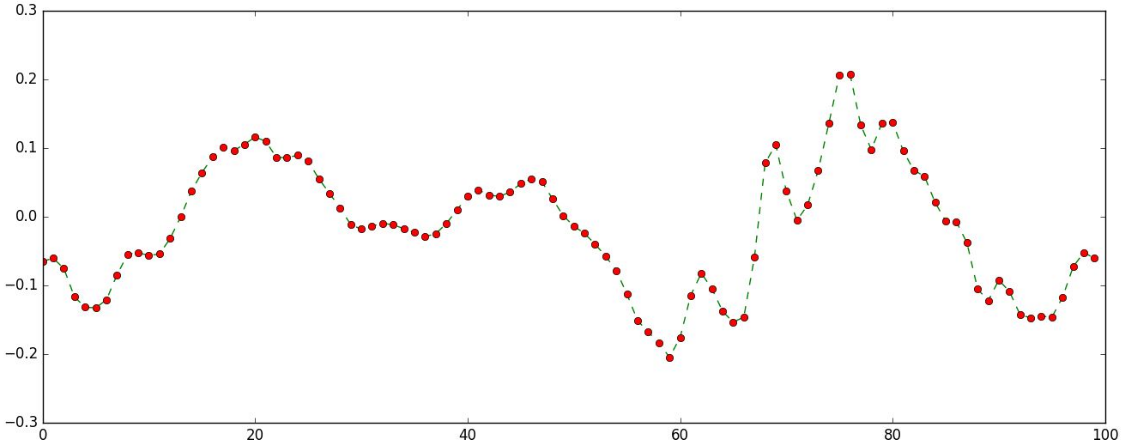
HOWEVER: now every time-step t depends on the two previous inputs as well as the current time-step → every output **is causal**

Roughly: We make a causal convolution by padding left the sequence with (kwidth - 1) zeros

PixelCNN/Wavenet

We can simply exemplify how causal convolutions help us with the SoTA model for audio generation Wavenet. We have a one dimensional stream of samples of length T:

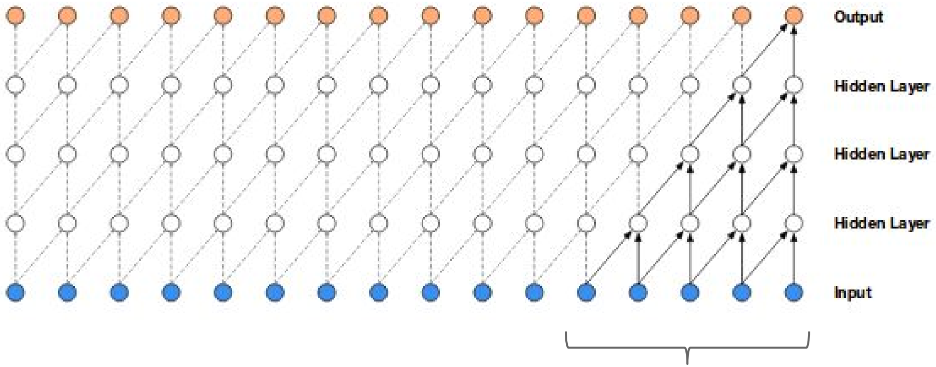
$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



PixelCNN/Wavenet

We can simply exemplify how causal convolutions help us with the SoTA model for audio generation Wavenet. We have a one dimensional stream of samples of length T (e.g. audio):

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

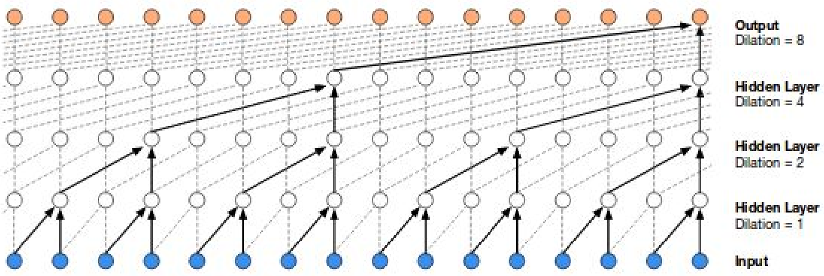


receptive field hast to be T

PixelCNN/Wavenet

We can simply exemplify how causal convolutions help us with the SoTA model for audio generation Wavenet. We have a one dimensional stream of samples of length T:

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



receptive field hast to be T

Dilated convolutions help us reach a receptive field sufficiently large to emulate an RNN!

Conditional PixelCNN/Wavenet

- Q: Can we make the generative model learn the natural distribution of **X** and perform a specific task conditioned on it?
  - A: Yes! we can condition each sample to an embedding/feature vector **h**, which can represent encoded text, or speaker identity, generation speed, etc.

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

Conditional PixelCNN/Wavenet

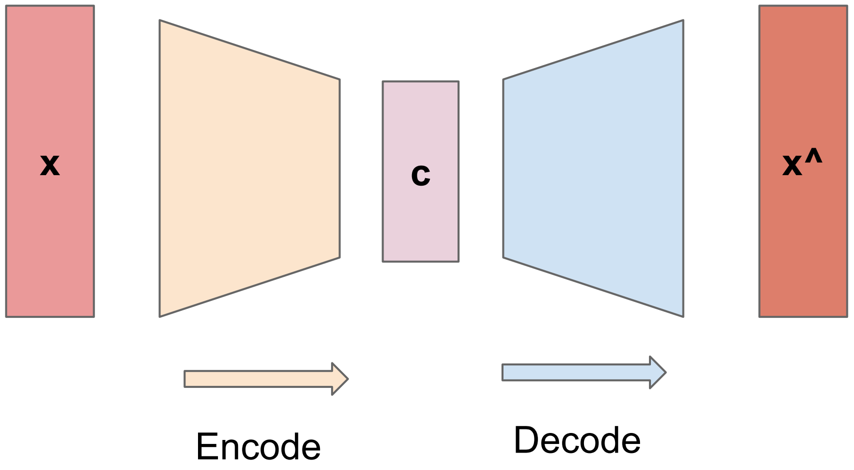
- PixelCNN produces very sharp and realistic samples, but...
- Q: Is this a cheap generative model computationally?

Conditional PixelCNN/Wavenet

- PixelCNN produces very sharp and realistic samples, but...
- Q: Is this a cheap generative model computationally?
  - A: Nope, take the example of wavenet: Forward 16.000 times in autorregressive way to predict 1 second of audio at standard 16kHz sampling.

Variational Auto-Encoders

Auto-Encoder Neural Network

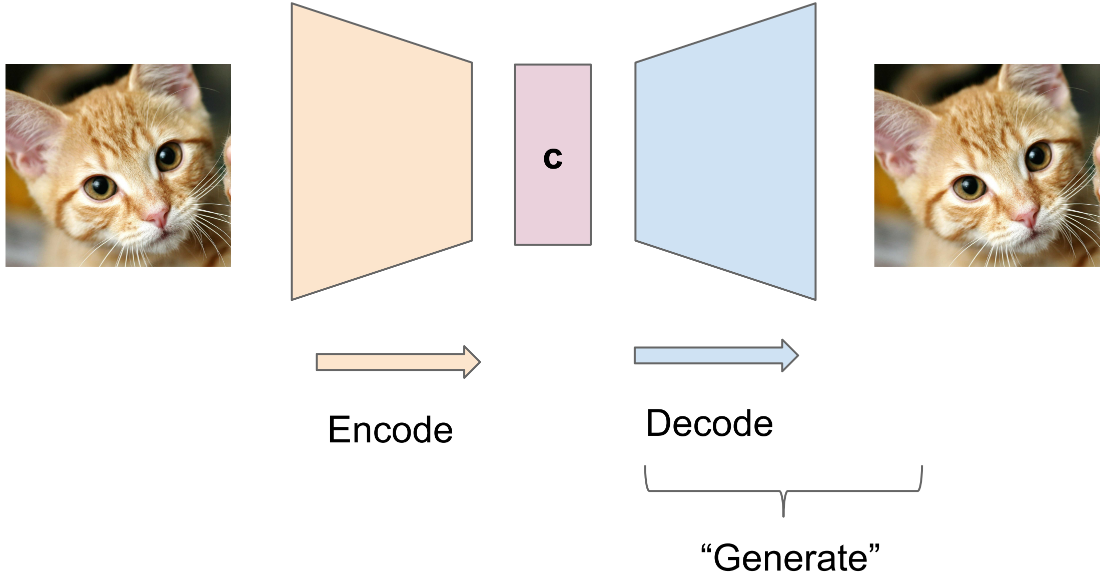


Autoencoders:

- Predict at the output the same input data.
- Do not need labels

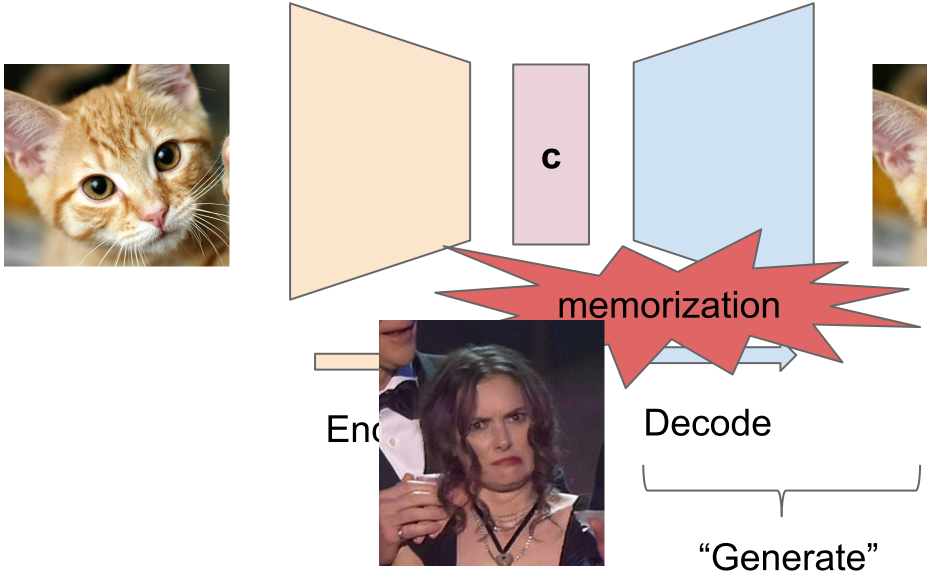
Auto-Encoder Neural Network

- Q: What generative thing can we do with an AE? How can we make it generate data?



Auto-Encoder Neural Network

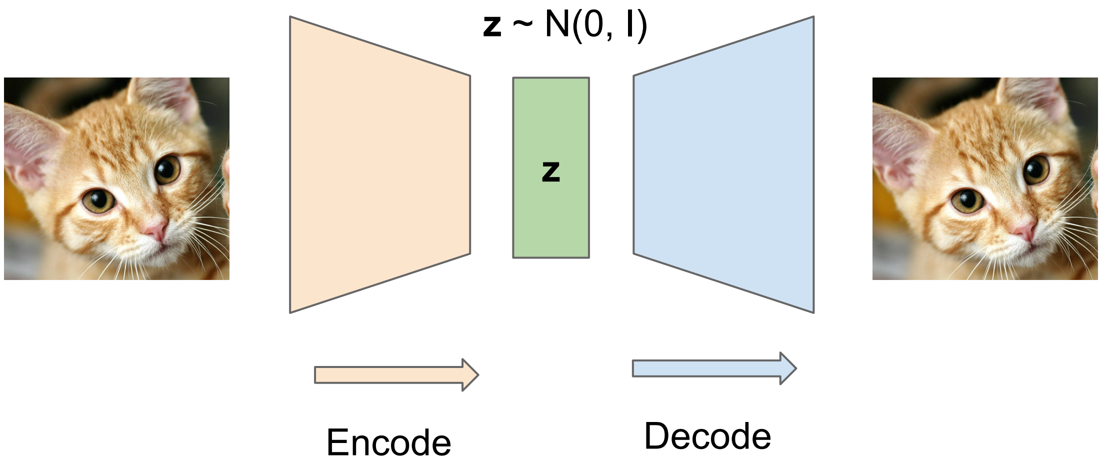
- Q: What generative thing can we do with an AE? How can we make it generate data?
  - A: This “just” memorizes codes **C** from our training samples!



Variational Auto-Encoder

VAE intuitively:

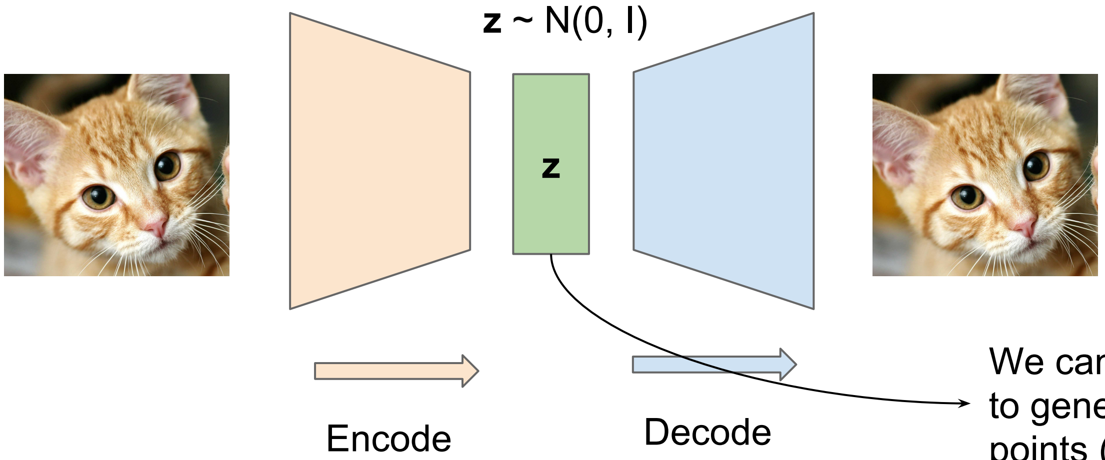
- Introduce a restriction in **z**, such that our data points **x** (e.g. images) are distributed in a latent space (manifold) following a specified probability density function **Z** (normally N(0, I)).



Variational Auto-Encoder

VAE intuitively:

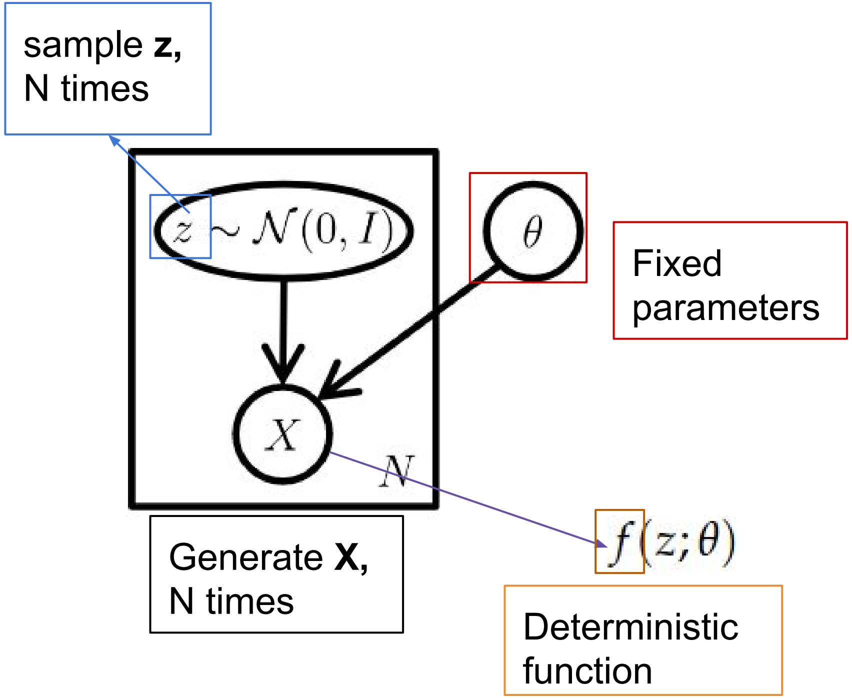
- Introduce a restriction in **z**, such that our data points **x** (e.g. images) are distributed in a latent space (manifold) following a specified probability density function **Z** (normally N(0, I)).



We can then sample **z** to generate new **x** data points (e.g. images).

Variational Auto-Encoder

- VAE, aka. where Bayesian theory and deep learning collide, in depth:

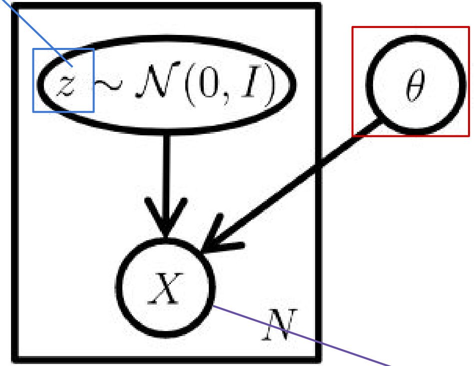


Credit:

Variational Auto-Encoder

- VAE, aka. where Bayesian theory and deep learning collide, in depth:

sample **z**, N times



Fixed parameters

Generate **X**, N times

$f(z;\theta)$

Deterministic function

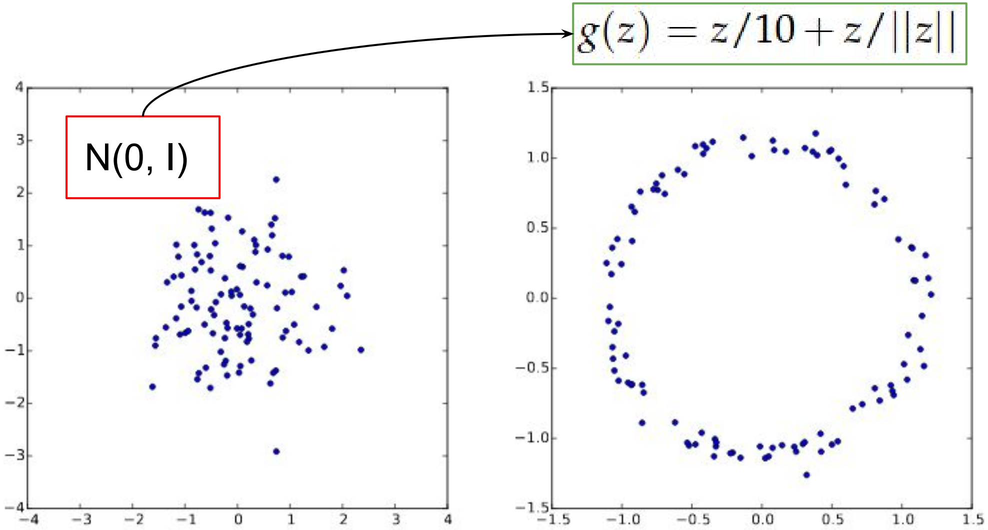
Maximize the probability of each **X** under the generative process.

$$P(X) = \int \boxed{P(X|z;\theta)} P(z) dz.$$

Maximum likelihood framework

Variational Auto-Encoder

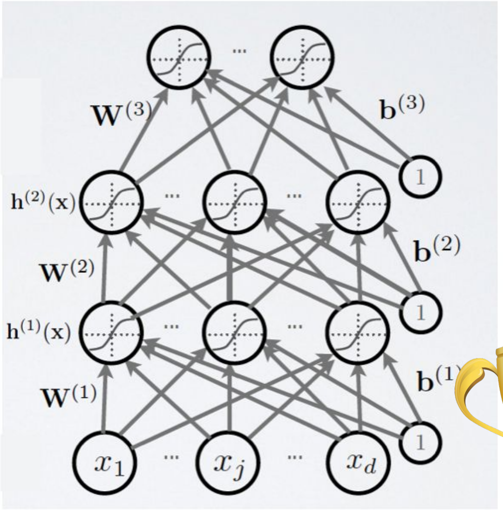
Intuition behind normally distributed **z** vectors: any output distribution can be achieved from the simple N(0, I) with powerful non-linear mappings.



Variational Auto-Encoder

Intuition behind normally distributed **z** vectors: any output distribution can be achieved from the simple N(0, I) with powerful non-linear mappings.

Who's the strongest non-linear mapper in the universe?

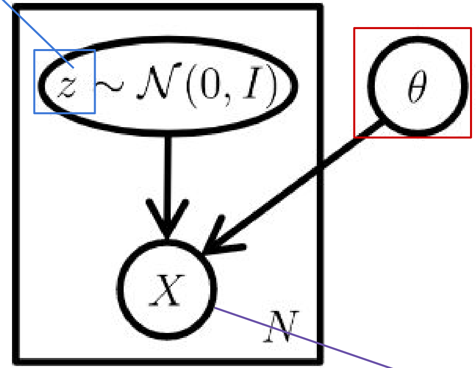


Slide Credit: Hugo Laroche NN course

Variational Auto-Encoder

- VAE, aka. where Bayesian theory and deep learning collide, in depth:

sample **z,** N times



Fixed parameters

Generate **X,** N times

$f(z;\theta)$

Deterministic function

Maximize the probability of each **X** under the generative process.

$$P(X) = \int \boxed{P(X|z;\theta)} P(z)dz.$$

Maximum likelihood framework

Variational Auto-Encoder

Now to solve the maximum likelihood problem… We'd like to know $P(X|z)$ and $P(z)$. We introduce $P(z|X)$ as a key piece → sample values **z** likely to produce **X**, not just the whole $P(z)$ possibilities.

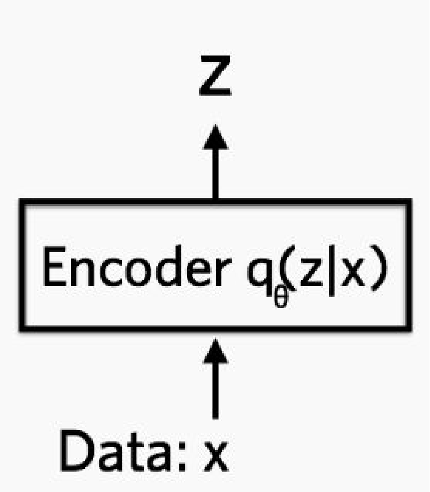
But $P(z|X)$ is unkown too! **Variational** Inference comes in to play its role: approximate $P(z|X)$ with $Q(z|X)$.

Key Idea behind the variational inference application: find an approximation function that is good enough to represent the real one → optimization problem.

Variational Auto-Encoder

Neural network prespective

The approximated function starts to shape up as a neural encoder, going from training datapoints **x** to the likely **z** points following $Q(z|X)$, which in turn is similar to the real $P(z|X)$.

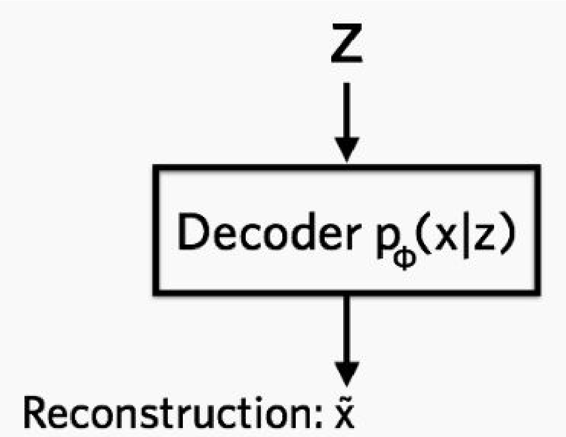


Credit: Altosaar

Variational Auto-Encoder

Neural network prespective

The (latent→ data) mapping starts to shape up as a neural decoder, where we go from our sampled **z** to the reconstruction, which can have a very complex distribution.



Credit: Altosaar

Variational Auto-Encoder

Continuing with the encoder approximation $Q(z|X)$, we compute the KL divergence with the true distribution:

$$D_{KL}[Q(z|X) \| P(z|X)] = \boxed{\sum_z Q(z|X) \log \frac{Q(z|X)}{P(z|X)}}$$

KL divergence

$$= E\left[\log \frac{Q(z|X)}{P(z|X)}\right]$$

$$= E[\log Q(z|X) - \log P(z|X)]$$

Credit: Kristiadis

Variational Auto-Encoder

Continuing with the encoder approximation $Q(z|X)$, we compute the KL divergence with the true distribution:

$$D_{KL}[Q(z|X)\|P(z|X)] = \sum_z Q(z|X) \log \frac{Q(z|X)}{P(z|X)}$$

$$= E\left[\log \frac{Q(z|X)}{P(z|X)}\right]$$

$$= E[\log Q(z|X) - \log \boxed{P(z|X)}]$$

Bayes rule

$$\boxed{\frac{P(X|z)P(z)}{P(X)}}$$

Credit: Kristiadis

Variational Auto-Encoder

Continuing with the encoder approximation $Q(z|X)$, we compute the KL divergence with the true distribution:

$$D_{KL}[Q(z|X)\|P(z|X)] = E[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X)$$

$$D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

$$= E[\log Q(z|X) - (\log P(X|z) + \log P(z) - \log P(X))]$$

$$= E[\log Q(z|X) - \log P(X|z) - \log P(z) + \boxed{\log P(X)}]$$

Gets out of expectation for no dependency over **z**.

Credit: Kristiadis

Variational Auto-Encoder

Continuing with the encoder approximation $Q(z|X)$, we compute the KL divergence with the true distribution:

$$D_{KL}[Q(z|X)\|P(z|X)] = E[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X)$$

$$D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

Credit: Kristiadis

Credit: Kristiadis

Variational Auto-Encoder

Continuing with the encoder approximation $Q(z|X)$, we compute the KL divergence with the true distribution:

$$D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

$$\begin{aligned}\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] &= E[\log P(X|z) - (\log Q(z|X) - \log P(z))] \\ &= E[\log P(X|z)] - E[\log Q(z|X) - \log P(z)] \\ &= E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]\end{aligned}$$

A bit more rearranging with sign and grouping leads us to a new KL term between $Q(z|X)$ and $P(z)$, thus the encoder approximate distribution and our prior.

Variational Auto-Encoder

We finally reach the Variational AutoEncoder objective function.

$$\log P(X) - D_{KL}[Q(z|X) \| P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X) \| P(z)]$$

Credit: Kristiadis

Variational Auto-Encoder

We finally reach the Variational AutoEncoder objective function.

$$\boxed{\log P(X)} - D_{KL}[Q(z|X) \| P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X) \| P(z)]$$

log likelihood of our data

Credit: Kristiadis

Variational Auto-Encoder

We finally reach the Variational AutoEncoder objective function.

Not computable and non-negative (KL) approximation error.

$$\boxed{\log P(X)} - \boxed{D_{KL}[Q(z|X) \| P(z|X)]} = E[\log P(X|z)] - D_{KL}[Q(z|X) \| P(z)]$$

log likelihood of our data

Credit: Kristiadis

Variational Auto-Encoder

We finally reach the Variational AutoEncoder objective function.

Not computable and non-negative (KL) approximation error.

$$\boxed{\log P(X)} - \boxed{D_{KL}[Q(z|X) \| P(z|X)]} = \boxed{E[\log P(X|z)]} - D_{KL}[Q(z|X) \| P(z)]$$

log likelihood of our data

Reconstruction loss of our data given latent space → NEURAL DECODER reconstruction loss!

Credit: Kristiadis

Variational Auto-Encoder

We finally reach the Variational AutoEncoder objective function.

Regularization of our latent representation → NEURAL ENCODER projects over prior.

Not computable and non-negative (KL) approximation error.

$$\boxed{\log P(X)} - \boxed{D_{KL}[Q(z|X) \| P(z|X)]} = \boxed{E[\log P(X|z)]} - \boxed{D_{KL}[Q(z|X) \| P(z)]}$$

log likelihood of our data

Reconstruction loss of our data given latent space → NEURAL DECODER reconstruction loss!

Credit: Kristiadis

Variational Auto-Encoder

Now, we have to define $Q(z|X)$ shape to compute its divergence against the prior (i.e. to properly condense **x** samples over the surface of **z**). Simplest way: distribute over normal distribution with moments: $\mu(X)$ and $\Sigma(X)$.

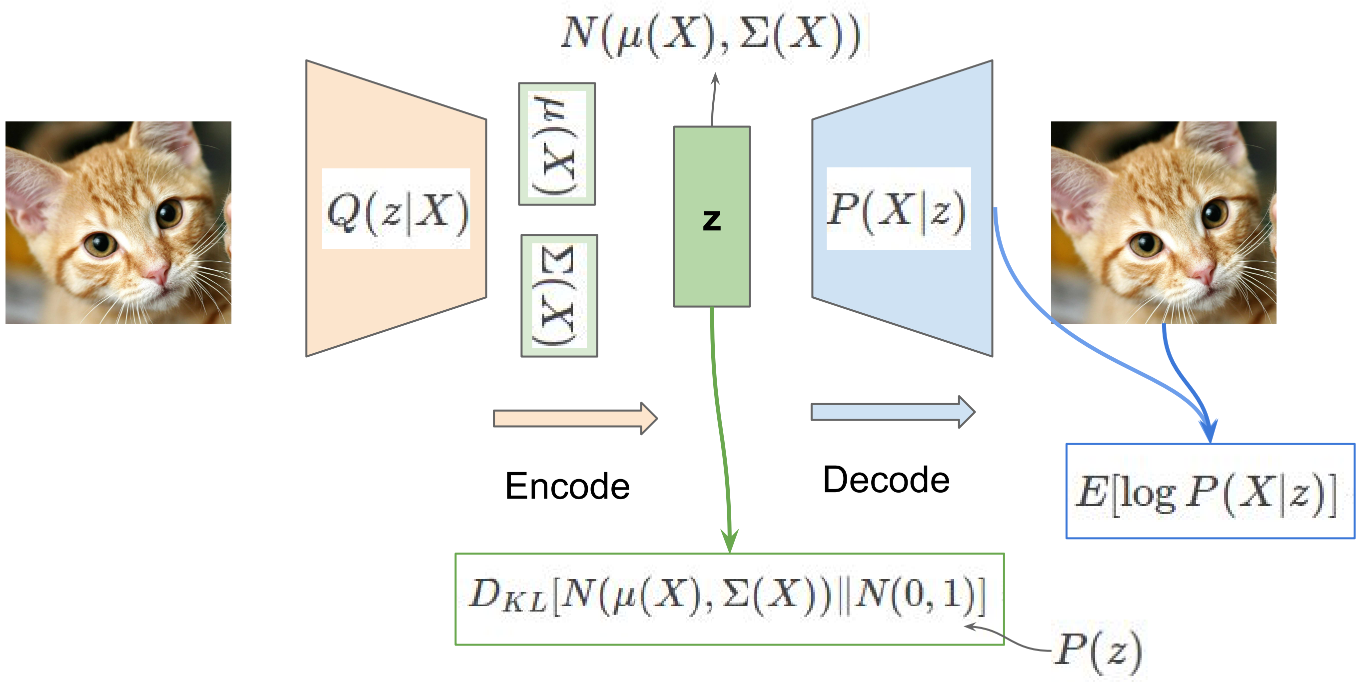
This allows us to compute the KL-div with $P(z)$ in a closed form :)

$$D_{KL}[\underbrace{N(\mu(X),\Sigma(X))}_{Q(z|X)}\|\underbrace{N(0,1)}_{P(z)}] = \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

Credit: Kristiadis

Variational Auto-Encoder

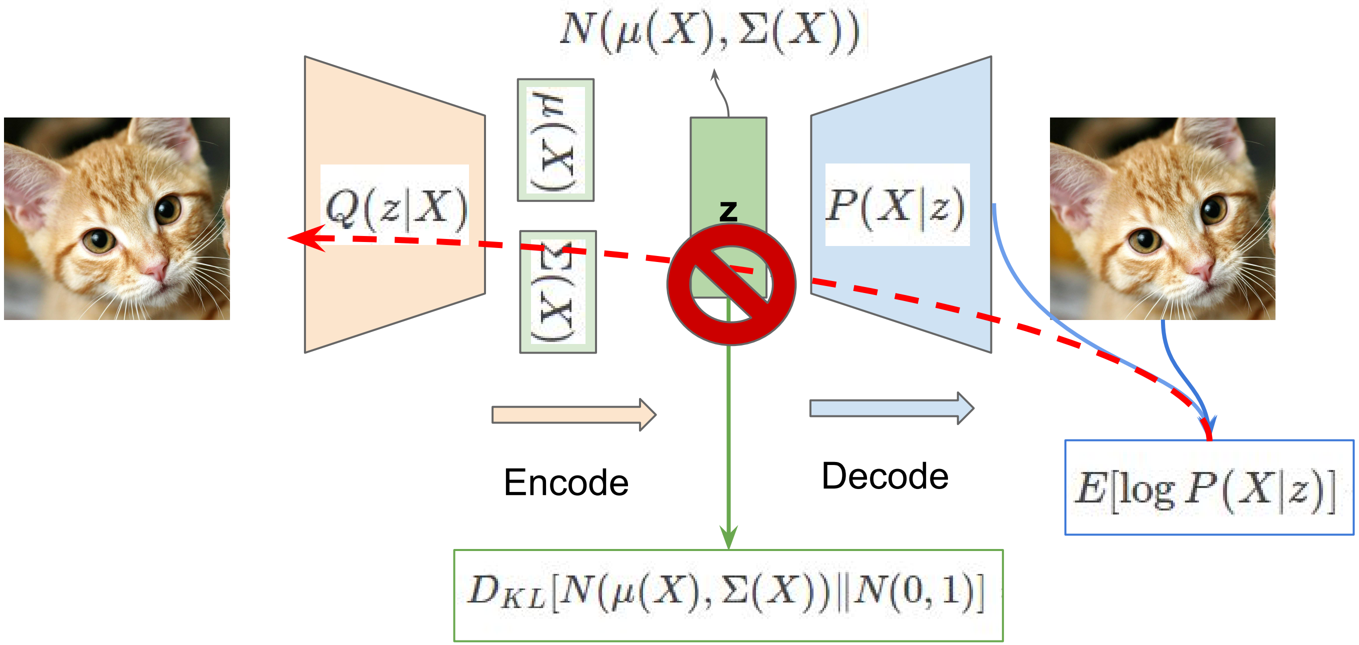
We can compose our encoder - decoder setup, and place our VAE losses to regularize and reconstruct.



Variational Auto-Encoder

<u>Reparameterization trick</u>

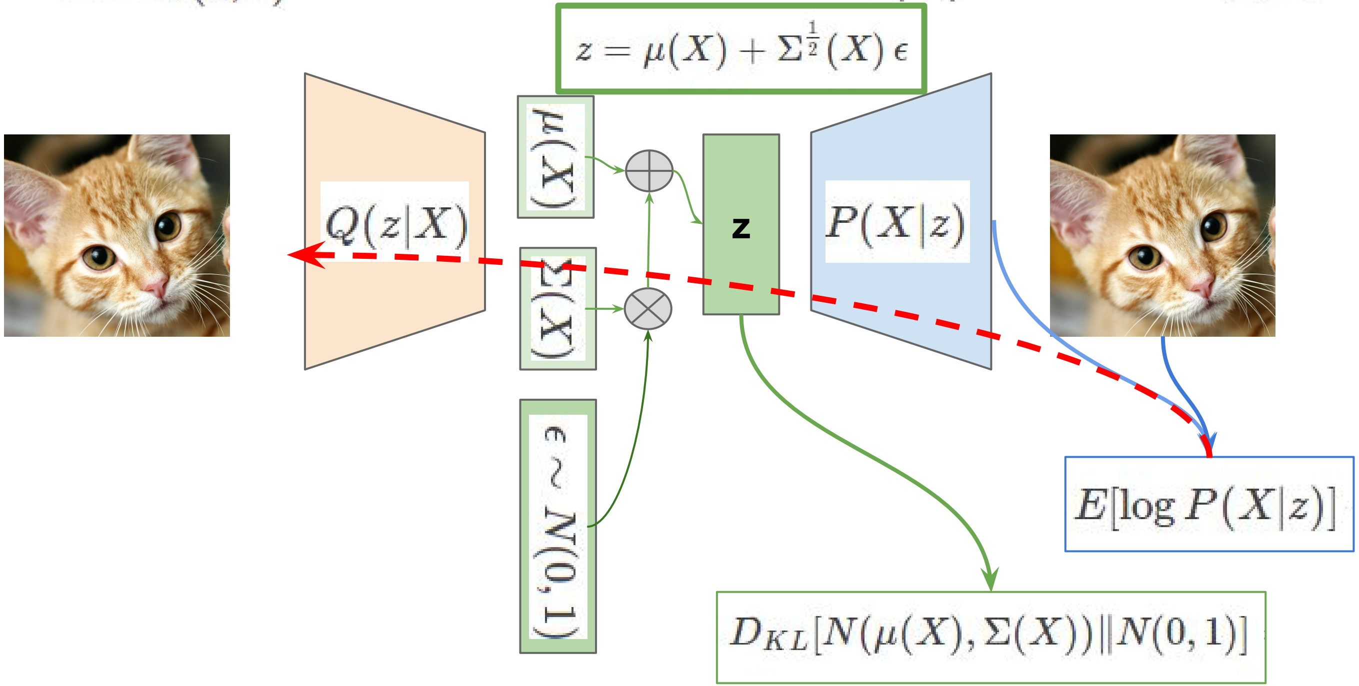
But WAIT, how can we backprop through sampling of $N(\mu(X), \Sigma(X))$? **Not differentiable!**



Variational Auto-Encoder

Reparameterization trick

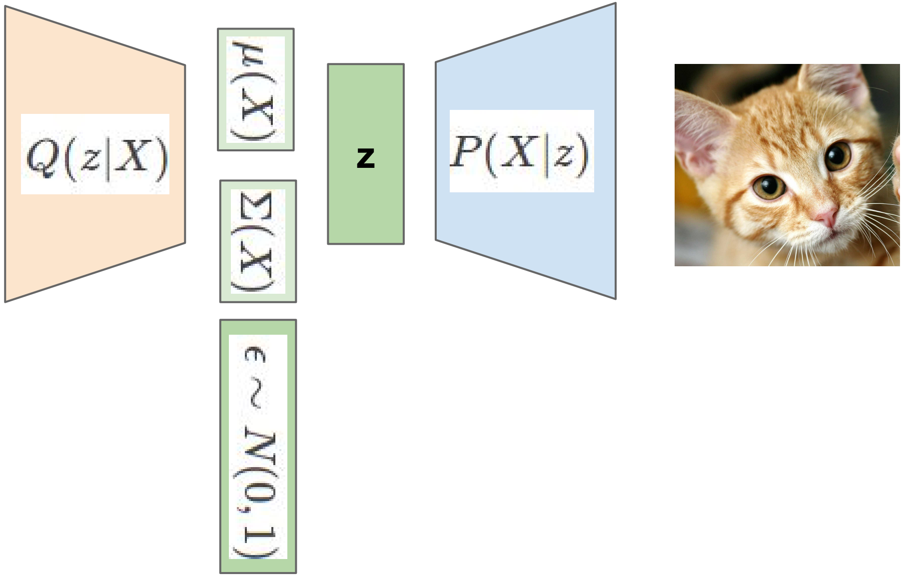
Sample $\epsilon \sim N(0,1)$ and operate with it, multiplying by $\Sigma(X)$ and summing $\mu(X)$



Variational Auto-Encoder

Generative behavior

Q: How can we now generate new samples once the underlying generating distribution is learned?

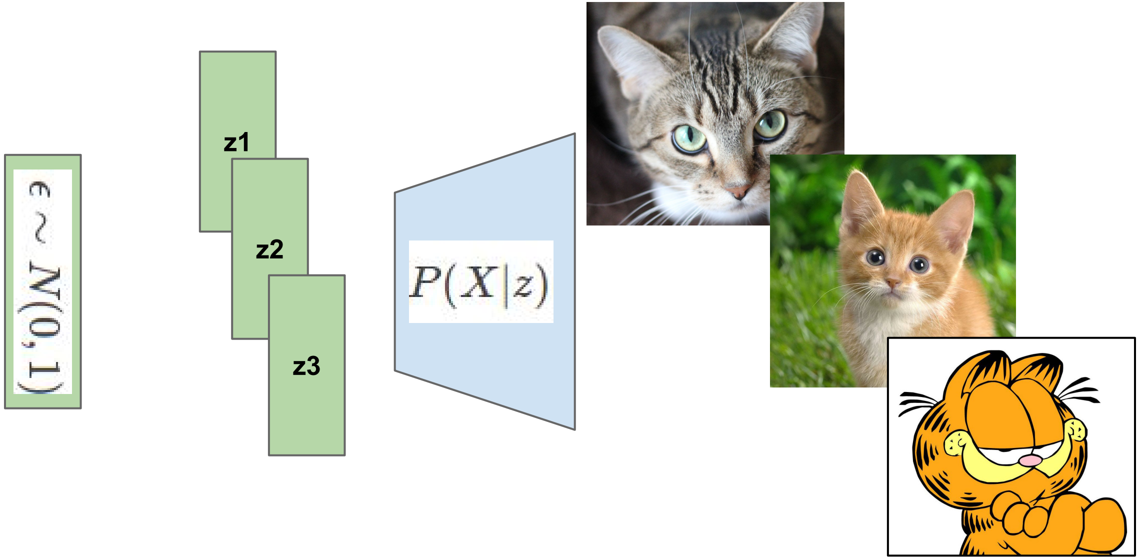


Variational Auto-Encoder

Generative behavior

Q: How can we now generate new samples once the underlying generating distribution is learned?

A: We can sample from our prior, for example, discarding the encoder path.



Variational Auto-Encoder

Walking around **z** manifold dimensions gives us spontaneous generation of samples with different shapes, poses, identitites, lightning, etc..

Examples:

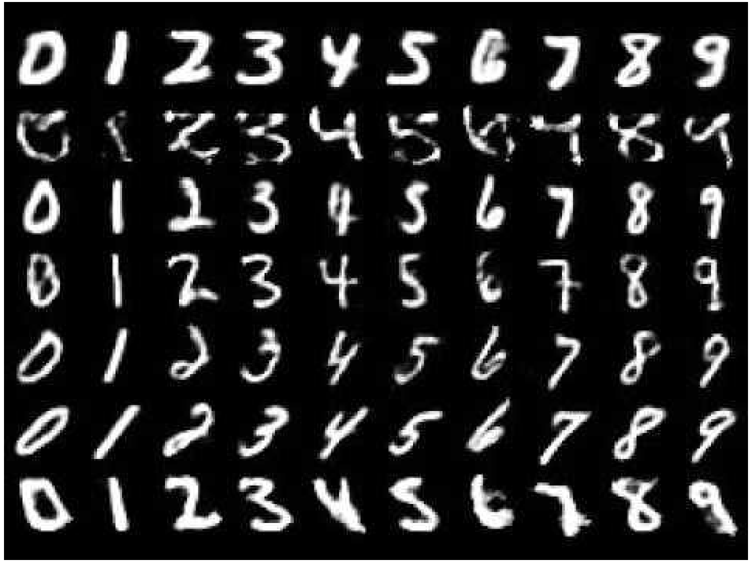
MNIST manifold: https://youtu.be/hgyB8RegAlQ

Face manifold: https://www.youtube.com/watch?v=XNZIN7Jh3Sg

Variational Auto-Encoder

Walking around **z** manifold dimensions gives us spontaneous generation of samples with different shapes, poses, identitites, lightning, etc..

Example with MNIST manifold



Variational Auto-Encoder

Walking around **z** manifold dimensions gives us spontaneous generation of samples with different shapes, poses, identitites, lightning, etc..

Example with Faces manifold

Variational Auto-Encoder

Code show with PyTorch on VAEs!

PYTORCH

https://github.com/pytorch/examples/tree/master/vae

Variational Auto-Encoder

Model

```python
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(784, 400)
        self.fc21 = nn.Linear(400, 20)
        self.fc22 = nn.Linear(400, 20)
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def encode(self, x):
        h1 = self.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, logvar):
        if self.training:
          std = logvar.mul(0.5).exp_()
          eps = Variable(std.data.new(std.size()).normal_())
          return eps.mul(std).add_(mu)
        else:
          return mu

    def decode(self, z):
        h3 = self.relu(self.fc3(z))
        return self.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar


model = VAE()
if args.cuda:
    model.cuda()
```

Loss

```python
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784))

    # see Appendix B from VAE paper:
    # Kingma and Welling. Auto-Encoding Variational Bayes. ICLR, 2014
    # https://arxiv.org/abs/1312.6114
    # 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    # Normalise by same number of elements as in reconstruction
    KLD /= args.batch_size * 784

    return BCE + KLD
```

Thanks! Questions?

References

- NIPS 2016 Tutorial: Generative Adversarial Networks (Goodfellow 2016)
- Pixel Recurrent Neural Networks (van den Oord et al. 2016)
- Conditional Image Generation with PixelCNN Decoders (van den Oord et al. 2016)
- Auto-Encoding Variational Bayes (Kingma & Welling 2013)
- https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/
- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- Tutorial on Variational Autoencoders (Doersch 2016)