Visual AI

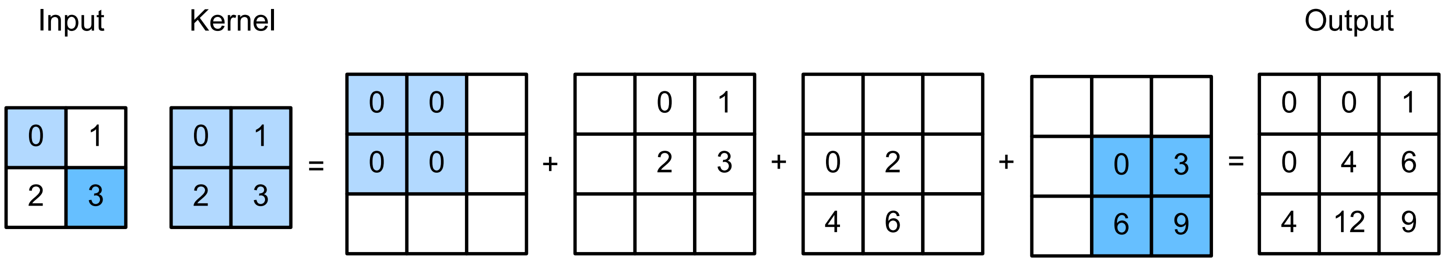
CPSC 532R/533R – 2019/2020 Term 2

Lecture 8. Representation learning II (probabilistic models)

Helge Rhodin

Recap: Transposed convolution

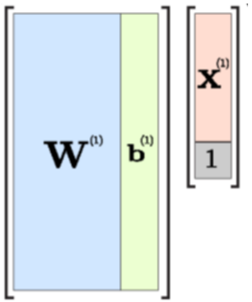
Example: 2D transposed convolution with 3x3 kernel



[https://d2l.ai/chapter_computer-vision/tranposed-conv.html]

Transposed what?

- *Express classical convolution as linear matrix and transpose it*
  - *special case of a linear/fully-connected layer*



Convolution as matrix multiplication

$$\begin{bmatrix} 0, & 1, & 0, & 2, & 3, & 0, & 0, & 0, & 0 \\ 0, & 0, & 1, & 0, & 2, & 3, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1, & 0, & 2, & 3, & 0 \\ 0, & 0, & 0, & 0, & 0, & 1, & 0, & 2, & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 0\cdot 1 + 1\cdot 2 + 2\cdot 4 + 3\cdot 5 \\ 0\cdot 2 + 1\cdot 3 + 2\cdot 5 + 3\cdot 6 \\ 0\cdot 4 + 1\cdot 5 + 2\cdot 7 + 3\cdot 8 \\ 0\cdot 5 + 1\cdot 6 + 2\cdot 8 + 3\cdot 9 \end{bmatrix} = \begin{bmatrix} 25 \\ 31 \\ 43 \\ 49 \end{bmatrix}$$

Convolution with 2x2 kernel as a linear mapping

Flattened 3x3 input image

$$\underbrace{\begin{bmatrix} 1, & 2, & 3 \\ 4, & 5, & 6 \\ 7, & 8, & 9 \end{bmatrix}}_{\text{Input}} * \underbrace{\begin{bmatrix} 0, & 1 \\ 2, & 3 \end{bmatrix}}_{\text{Kernel}} = \underbrace{\begin{bmatrix} 25, & 31 \\ 43, & 49 \end{bmatrix}}_{\text{Output}}$$

Convolution operator

Convolution as matrix multiplication (details)

What about horizontal striding?
Skip every second row

Larger kernel?
Add non-zeroes here
(5 values for 3x3 from 2x2)

Padding?
Insert a row of zeroes

Larger input height?
Insert more rows here.

$$\begin{bmatrix} 0, 1, 0, 2, 3, 0, 0, 0, 0 \\ 0, 0, 1, 0, 2, 3, 0, 0, 0 \\ 0, 0, 0, 0, 1, 0, 2, 3, 0 \\ 0, 0, 0, 0, 0, 1, 0, 2, 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix}$$

What about vertical striding?
Skip block of rows

Larger input width?
insert zeroes here.

$$\begin{bmatrix} 1, 2, 3 \\ 4, 5, 6 \\ 7, 8, 9 \end{bmatrix}$$

Input

$$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$$

Kernel

Transposed convolution as matrix multiplication

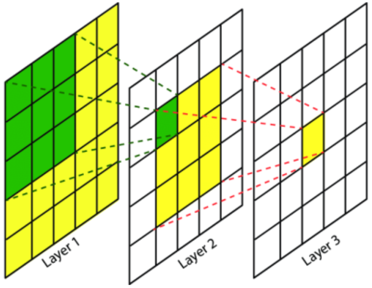
$$
\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 3 \end{bmatrix}
\begin{bmatrix} 25 \\ 31 \\ 43 \\ 49 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 1\cdot 25 \\ 1\cdot 31 \\ 2\cdot 25 \\ 3\cdot 25+2\cdot 31+1\cdot 43 \\ 3\cdot 31+1\cdot 49 \\ 2\cdot 43 \\ 3\cdot 43+2\cdot 49 \\ 3\cdot 49 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 25 \\ 31 \\ 50 \\ 180 \\ 142 \\ 86 \\ 227 \\ 147 \end{bmatrix}
$$

Transposed weight matrix

Transposed convolution

$$
\underbrace{\begin{bmatrix} 25, & 31 \\ 43, & 49 \end{bmatrix}}_{\text{Input}} *^{T} \underbrace{\begin{bmatrix} 0, & 1 \\ 2, & 3 \end{bmatrix}}_{\text{Kernel}}
= \underbrace{\begin{bmatrix} 0, & 25, & 31 \\ 50, & 180, & 142 \\ 86, & 227, & 147 \end{bmatrix}}_{\text{Output}}
$$

Recap: Feature map size after convolutional kernels



Transformation of input and output by convolutions

- output size = (input size + 2*padding – kernel size + stride)/stride
  - e.g., a 3x3 kernel that preserves size: W + 2*1 – 3 + 1 = W
  - e.g., a 4x4 kernel that reduces size by factor two: (W + 2*1 – 4 + 2)/2 = W/2
- holds per dimension, i.e., 1D, 2D and 3D convolutions

Transformation of input and output by **transposed** convolutions (aka. deconvolution)

- output size = input size * stride - stride + kernel size - 2*padding
  - it has exactly the opposite effect of convolution
  - e.g., a 3x3 kernel that preserves size: W - 1 + 3 - 2*1 = W
  - e.g., a 4x4 kernel that **increases** size by **factor** two: W*2 + 2*1 – 4 + 2 = W*2
  - e.g., a 3x3 kernel that **increases** size by two elements: W - 1 + 3 - 2*0 = W + 2

Assignment 3

Task I will be published tonight.

- Neural rendering

The other ones are delayed due to unforeseen difficulties.

Paper assignment finished (on Piazza)

The current assignment:

W8	25-Feb	Conditional content generation	
		Park et al., Semantic Image Synthesis with Spatially-Adaptive Normalization	Daniele Reda
		Li et al., Putting Humans in a Scene: Learning Affordance in 3D Indoor Environments	Shih-Han Chou
	27-Feb	Motion transfer	
		Chan et al, Everybody Dance Now	Zikun Chen
		Gao et al., Automatic Unpaired Shape Deformation Transfer	Willis Peng
W9	3-Mar	Character animation	
		Rhodin et al., Interactive Motion Mapping for Real-time Character Control	Michela Minerva
		Holden et al., Phase-Functioned Neural Networks for Character Control	Dingqing Yang
	5-Mar	Self-supervised learning	
		Vondrick et al., Tracking Emerges by Colorizing Videos	Dave Pagurek
		Doersch et al., Unsupervised visual representation learning by context prediction	Zicong Fan
W10	10-Mar	Novel view synthesis	
		Hinton et al., Transforming Auto-encoders	Arda Ege Unlu
		Rhodin et al., Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation	Shane Sims
	12-Mar	Differentiable rendering	
		Rhodin et al., A Versatile Scene Model with Differentiable Visibility Applied to Generative Pose Estimation	Lawrence Li
		Liu et al., Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning	Jerry Yin
W11	17-Mar	Learning person models	
		Lorenz et al., Unsupervised Part-Based Disentangling of Object Shape and Appearance	Tim Straubinger
		Rhodin et al., Neural Scene Decomposition for Human Motion Capture	Farnoosh Javadi
	19-Mar	Object parts and physics	
		Li et al., GRASS: Generative Recursive Autoencoders for Shape Structures	Peyman Bateni
		Xie et al., tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow	Michelle Appel
W12	24-Mar	Objective functions and log-likelihood	
		Christopher Bishop, Mixture Density Networks	Shenyi Pan
		Jonathan T. Barron, A General and Adaptive Robust Loss Function	Tianxin Tao
	26-Mar	Self-supervised object detection	
		Crawford et al., Spatially invariant unsupervised object detection with convolutional neural networks	Shuxian Fan
		Bielski and Paolo Favaro, Emergence of Object Segmentation in Perturbed Generative Models	Mona Fadaviardakani
W13	31-Mar	Mesh processing	
		Bagautdinov et al., Modeling Facial Geometry using Compositional VAEs	Matheus Stolet
		Verma et al., Feastnet: Feature-steered graph convolutions for 3d shape analysis	Matthew Wilson
	2-Apr	Neural rendering	
		Sitzmann et al., DeepVoxels: Learning Persistent 3D Feature Embeddings	Weidong Yin
		Saito et al., PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization	Peiyuan (Gary) Zhu

Recap: Auto Encoder (AE)

General case

$$\mathbf{h} = \mathrm{encoder}_\theta(\mathbf{x})$$
$$\mathbf{x}' = \mathrm{decoder}_\theta(\mathbf{h})$$

Simple non-linear case

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{x}' = \sigma(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

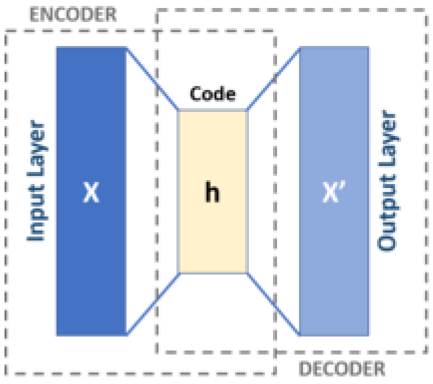
Linear case (similar to PCA)

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{x}' = \mathbf{W}'\mathbf{h} + \mathbf{b}'$$

General reconstruction objective

$$\mathrm{loss}(\mathbf{x}, \mathbf{x}')$$

- e.g., MSE loss



https://en.wikipedia.org/wiki/Autoencoder

[From Principal Subspaces to Principal Components with Linear Autoencoders]

Recap: Autoencoder variants

Bottleneck autoencoder:

- hidden dimension smaller than input dimension
  - leads to compressed representations
    - like dimensionality reduction with PCA

Sparse autoencoder:

- hidden dimension larger than input dimension
- hidden activation enforced to be sparse (=few activations

Denoising autoencoder:

- corrupt the input values, e.g. by additive noise

$$\mathbf{h} = \text{encoder}_\theta(\text{noise}(\mathbf{x}))$$
$$\mathbf{x}' = \text{decoder}_\theta(\mathbf{h})$$

Variational Auto Encoder (VAE)

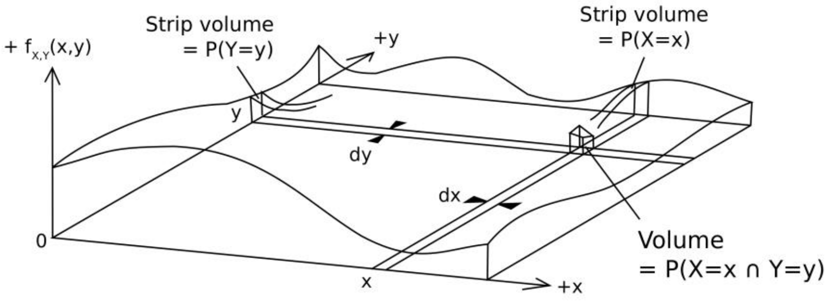
- a probabilistic model
  - *'adding noise on the hidden variables'*
  - More on this topic today!

Probability preliminaries

Bayes' theorem

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}$$

- Links the degree of belief in a proposition before and after accounting for evidence



$$P(Y=y|X=x) = \frac{P(X=x \cap Y=y)}{P(X=x)} \qquad P(X=x|Y=y) = \frac{P(X=x \cap Y=y)}{P(Y=y)}$$

Prior distribution $P(Y)$

- belief in a proposition before accounting for evidence

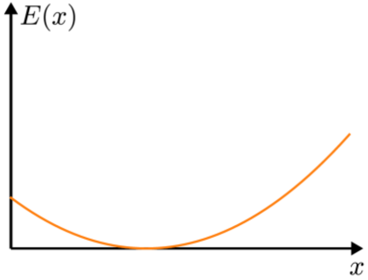
Posterior distribution $P(Y \mid X)$

- belief in a proposition after accounting for evidence
  - here without knowing event B
- a conditional probability
  - here conditioned on B

From lecture 3: **Regression revisited**

Many loss functions are –log of probability distributions

Error functions
$E(x)$



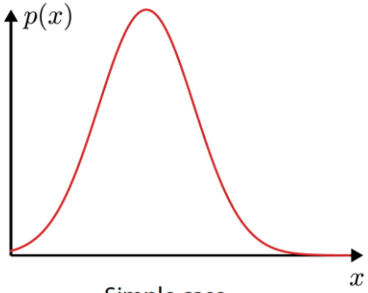
$$x^2$$

Mean squared error (MSE)

$$|x|$$

Mean absolute error (MAE)

Distributions
$p(x) = \exp(-E(x))$



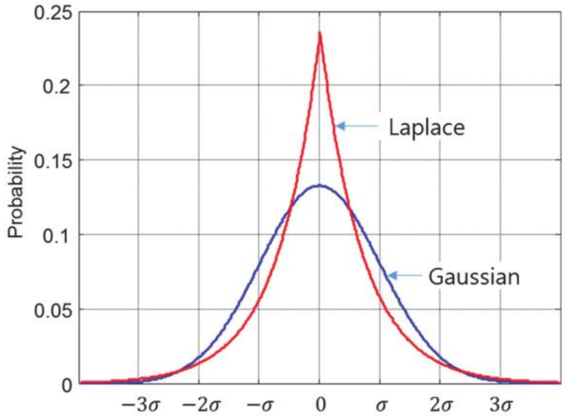
Simple case

$$\exp(-x^2)$$

Gaussian distribution

$$\exp(-|x|)$$

Laplace distribution



The prior is a regularizer

For instance, an l2 loss on the neural network weights

Regularizer / prior term

$$O(X,Y) = L(f_\theta(X),Y) + \lambda(\theta - 0)^2$$

corresponds to a prior on the weights

Data term / log likelihood

$$P(\theta|X,Y) = P(f_\theta(X,Y)|\theta) * P(\theta)$$

$$\propto \frac{P(f_\theta(X,Y)|\theta) * P(\theta)}{P(X,Y)}$$

We usually don't know the prior probability of X,Y, but we know that it is constant

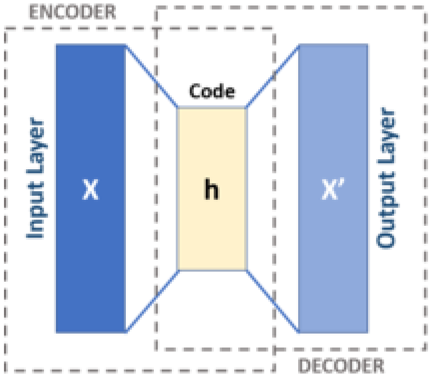
When optimizing a network…

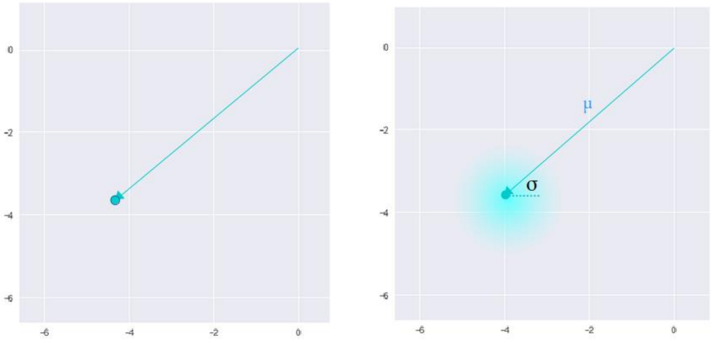
- without a prior, we infer the maximum likelihood (ML) estimate
- with a prior term, we infer the maximum a posteriori (MAP) estimate
- while considering the distribution of weights, we infer the posterior distribution of networks

  Bayesian networks, usually via Variational Inference of parametric distributions

Variational Autoencoder (VAE) concept

- mapping to a latent variable distribution
    - a parametric distribution
        - usually a Gaussian
            - with variable mean and std parameters
    - impose a prior distribution on the latent variables
        - usually a Gaussian
            - with fixed mean=0 and std=1
- Enables the generation of new samples
    - draw a random sample from the prior
        - pass it through the decoder
    - or draw a sample from the posterior
        - pass it through the decoder





Standard Autoencoder
(direct encoding coordinates)

Variational Autoencoder
(μ and σ initialize a probability distribution)

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

VAE examples

Generating unseen faces

https://github.com/yzwxx/vae-celebA

Generating music

[Roberts et al., Hierarchical Variational Autoencoders for Music]

The Variational Autoencoder (VAE)

VAE Objective (general)

Regularizer / prior term

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = -\mathbf{E}_{\mathbf{h}\sim q_\phi(\mathbf{h}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}|\mathbf{h})\right) + D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x}) \| p(\mathbf{h}))$$

Expectation over q

Data term / log likelihood

Kullback–Leibler divergence (relative entropy)

- a dissimilarity measure between distributions
  - not symmetric, KL(p,q) != KL(q,p)
- Definition for continuous distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

probability density of Q

Common parametrization

- Normal distributions

$$p(\mathbf{h}) = \mathcal{N}(0, 1)$$
$$q_\phi(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\boldsymbol{e}(\mathbf{x}), \boldsymbol{\omega}(\mathbf{x})\mathbf{I})$$
$$p_\theta(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\boldsymbol{d}(\mathbf{h}), \boldsymbol{\sigma}\mathbf{I})$$

- parametrized by neural networks
  - encoder e
  - decoder d

The Variational Autoencoder (VAE), simplified I

VAE Objective (general)

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = -\mathbf{E}_{\mathbf{h}\sim q_\phi(\mathbf{h}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}|\mathbf{h})\right) + D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x})\|p(\mathbf{h}))$$

$$\mathcal{N}(\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$\log\left(\mathcal{N}(\mu, \sigma)\right) = \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}(x-\mu)^2$$

$$p_\theta(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\boldsymbol{d}(\mathbf{h}), \boldsymbol{\sigma}\mathbf{I})$$

$$\Leftrightarrow \mathcal{L}(\phi, \theta, \mathbf{x}) = \mathbf{E}_{\mathbf{h}\sim q_\phi(\mathbf{h}|\mathbf{x})}\left(\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{d}(h))^2\right) + D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x})\|p(\mathbf{h})) + C$$

$$\Leftrightarrow \mathcal{L}(\phi, \theta, \mathbf{x}) = \lambda\mathbf{E}_{\mathbf{h}\sim q_\phi(\mathbf{h}|\mathbf{x})}(\mathbf{x} - \mathbf{d}(h))^2 + D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x})\|p(\mathbf{h})) + C$$

A simple autoencoder reconstruction loss,
the squared difference between input and output

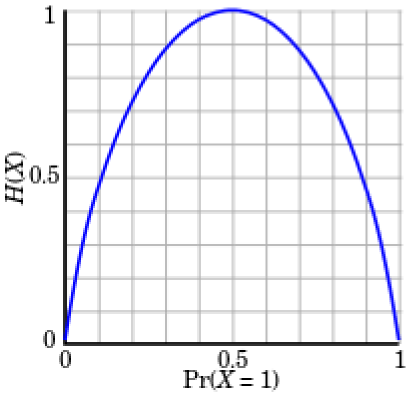
Kullback–Leibler divergence and entropy

Definition

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

Interpretation

- information gain achieved if Q is used instead of P
- relative entropy
  - Entropy: $H(p) = -\sum_{i=1}^{n} p(x_i) \log p(x_i).$
- the expected number of extra bits required to code samples from P using a code optimized for Q rather than the code optimized for P



KL divergence between Normal distributions (univariate case)

The KL divergence can be split in two parts

$$D_{\mathrm{KL}}(P \parallel Q) = \boxed{\int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right)} dx = \int_{-\infty}^{\infty} p(x) \log\left(p(x)\right)\, dx - \int_{-\infty}^{\infty} p(x) \log\left(q(x)\right)\, dx$$

For Gaussians $p(x) = N(\mu_1, \sigma_1)$ and $q(x) = N(\mu_2, \sigma_2)$ it holds

$$\int p(x) \log q(x) dx = -\frac{1}{2}\log(2\pi\sigma_2^2) - \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}$$

Hence

$$KL(p,q) = -\frac{1}{2}\log(2\pi\sigma_1^2) - \frac{1}{2} + \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}$$

$$= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

$$\boxed{= -\log\sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} - \frac{1}{2}}$$

Using that m2=0 and s2=1

https://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians

The Variational Autoencoder (VAE), simplified II

Starting point

$$\Leftrightarrow \mathcal{L}(\phi, \theta, \mathbf{x}) = \lambda \mathbf{E}_{\mathbf{h} \sim q_\phi(\mathbf{h}|\mathbf{x})} \left(\mathbf{x} - \mathbf{d}(\mathbf{h})\right)^2 + D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x}) \| p(\mathbf{h})) + C$$

Simplification (for Gaussian prior p and Gaussian q with $\mu_1 = \boldsymbol{e}(\mathbf{x})$ and $\sigma_1 = \boldsymbol{\omega}(\mathbf{x})$)

Data term / log likelihood

$$\Leftrightarrow \mathcal{L}(\phi, \theta, \mathbf{x}) = \lambda \mathbf{E}_{\mathbf{h} \sim q_\phi(\mathbf{h}|\mathbf{x})} \left(\mathbf{x} - \mathbf{d}(\mathbf{h})\right)^2 + - \log \sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} + C'$$

Sampling (here a single sample)

$$\approx \lambda \left(\mathbf{x} - \mathbf{d}(h)\right)^2 + - \log \sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} + C' \text{ with } h \sim q_\phi$$

reconstruct the image

‘keep sigma > 0’

‘keep sigma and mu small’

Expected value

$$\mathrm{E}_{x \sim q} f(x) = \int q(x) f(x) \, dx$$
$$= \sum_{i=1}^{k} f(x_i) \text{ with } x_i \sim q$$

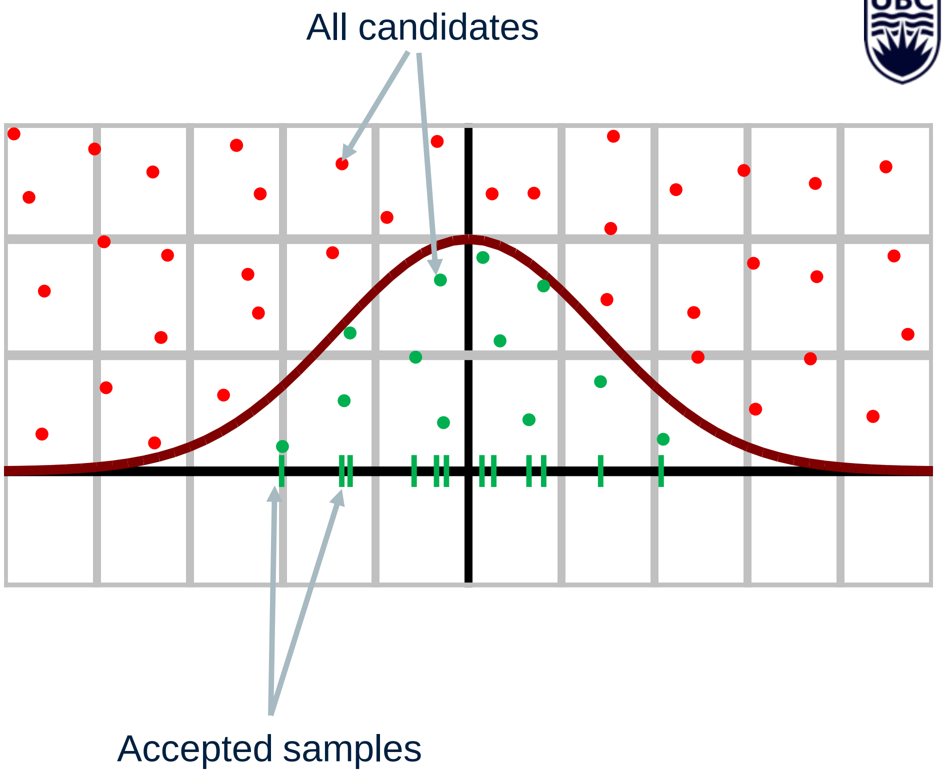
Sampling from a Gaussian

Rejection sampling from a uniform distribution

- intuitive approach
- ignores the tails of the distribution

Better alternative:

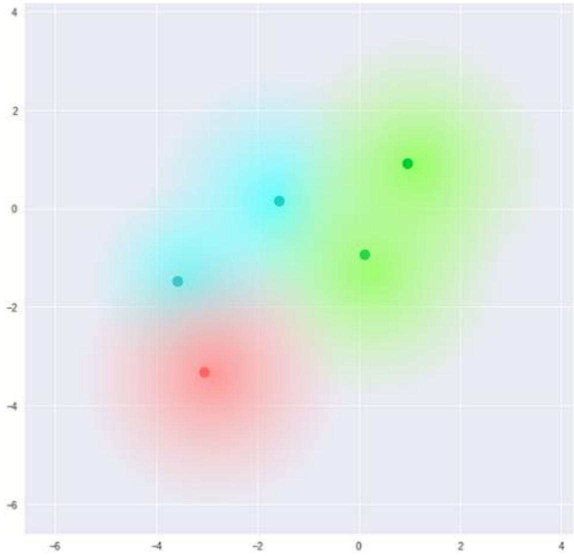
- Box-Muller Transform
  - requires only two uniform samples
  - mathematically correct (not an approximation)
  - efficient to compute



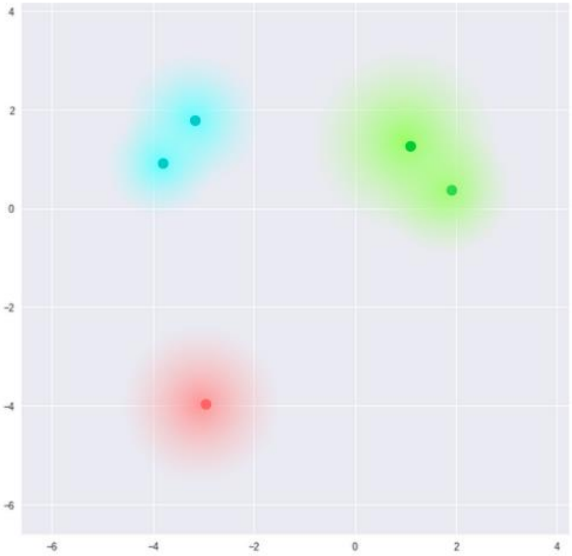
The effect of the prior

Create a dense and smooth latent space

- without holes
  - all samples will make sense
    - e.g., will reconstruct to plausible images



with prior



without prior

Deriving the posterior via Bayes

Goal: Compute the posterior

$$p(h \mid x) = \frac{p(x \mid h)p(h)}{p(x)}$$

Good hidden code h, given x

The evidence, intractable to compute (marginalization)

$$p(x) = \int p(x|h)p(h)dh$$

It requires integration over all possible latent values h

Attempt: Approximate the posterior with a NN (encoder)

$$\mathbf{KL}(q_\phi(h|x)||p(h|x)) = \mathbf{E}_\mathbf{q}[\log_{q_\phi}(h|x)] - \mathbf{E}_q[\log p(x,h)] + \log p(x)$$

- still intractable due to p(x) in the divergence

Evidence Lower BOund

Consider the term

$$ELBO(\phi) = E_q[\log p(x,z)] - E_q[\log q_\phi(z|x)]$$

$$= -\mathbf{E}_{\mathbf{h}\sim q_\phi(\mathbf{h}|\mathbf{x})}\big(\log p_\theta(\mathbf{x}|\mathbf{h})\big) + D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x})\|p(\mathbf{h}))$$

(equal to what we had before)

Together with the KL divergence from before, we get log p(x) as

$$\log p(x) = ELBO(\phi) + \mathbf{KL}(q_\phi(h|x)||p(h|x))$$

- the Kullback-Leibler divergence is always greater than or equal to zero
  - minimizing the Kullback-Leibler divergence is equivalent to maximizing the ELBO (making one bigger must reduce the other one)

Differentiation and sampling

Problem: How to differentiate through the sampling step?

- it's a random process, only statistically dependent on the mean and standard deviation of the sampling distribution

Solutions:

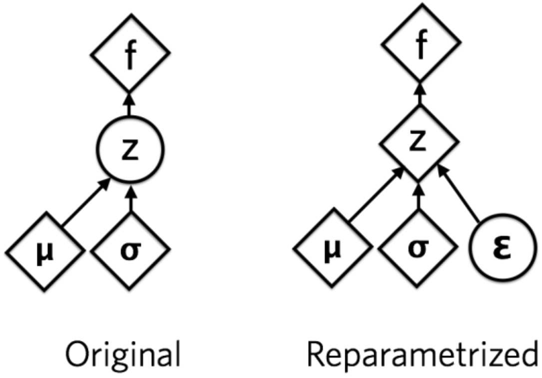
1. The reparameterization trick: Use

$$h = \mu + \sigma \odot \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0,1)$$

instead of

$$h \sim \mathcal{N}(\mu, \sigma)$$

2. Monte-Carlo solution
   - related to reinforcement learning and importance sampling
   - works for discrete and continuous variables
   - we will cover it next week

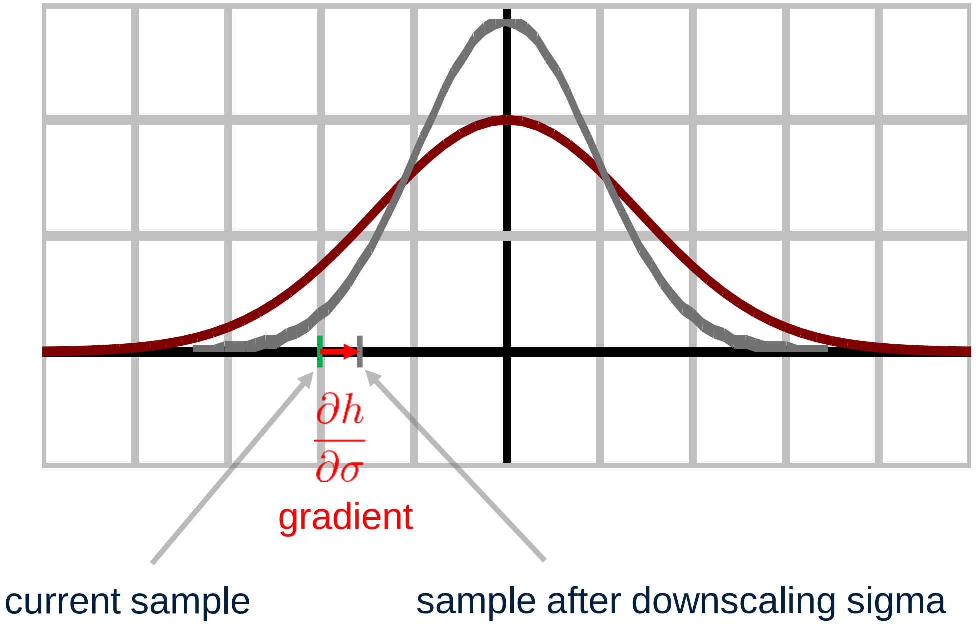


Reparametrization trick, visually and mathematically

Equation: $h = \mu + \sigma \odot \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0,1)$

Influence

- changing mu
  - increase -> moves sample right
  - decrease -> moves sample left
- changing sigma
  - increase -> moves away from center
  - decrease -> moves to the center



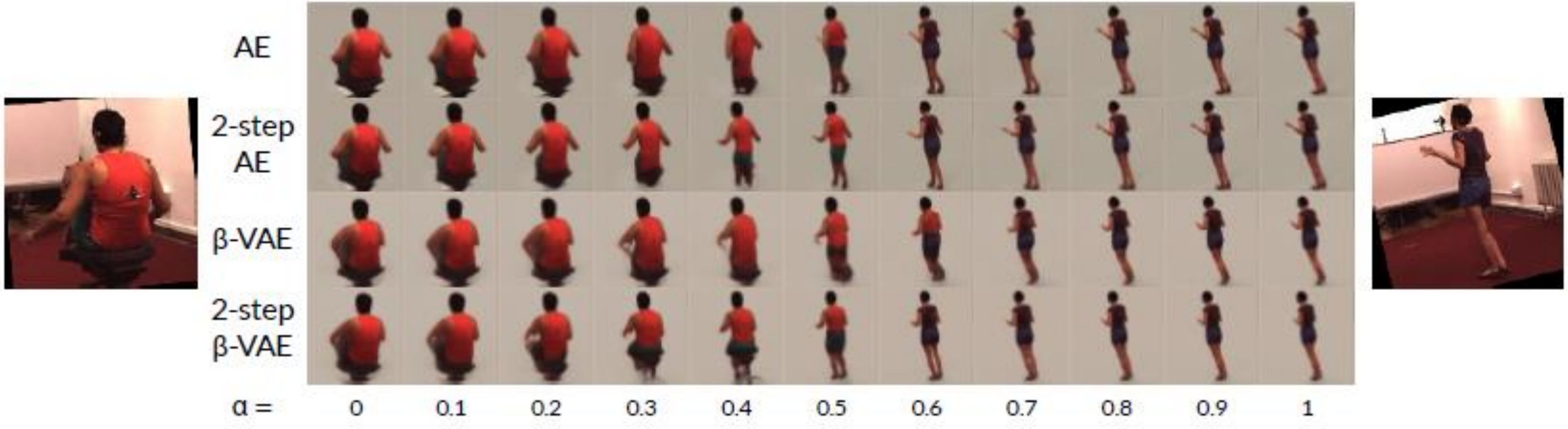
Gradient

$$\frac{\partial h}{\partial \sigma} = \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0,1)$$

$$\frac{\partial h}{\partial \mu} = 1$$

VAE results

Mixed appearance generation



Interpolation

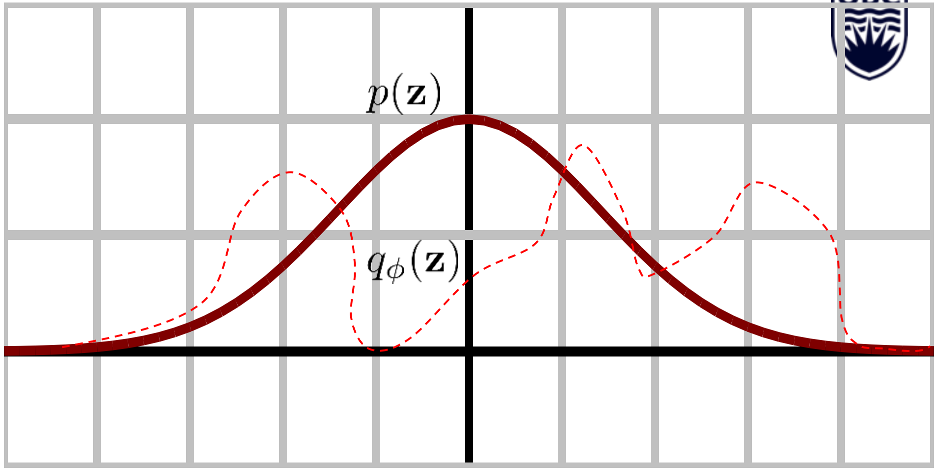
VAE limitations

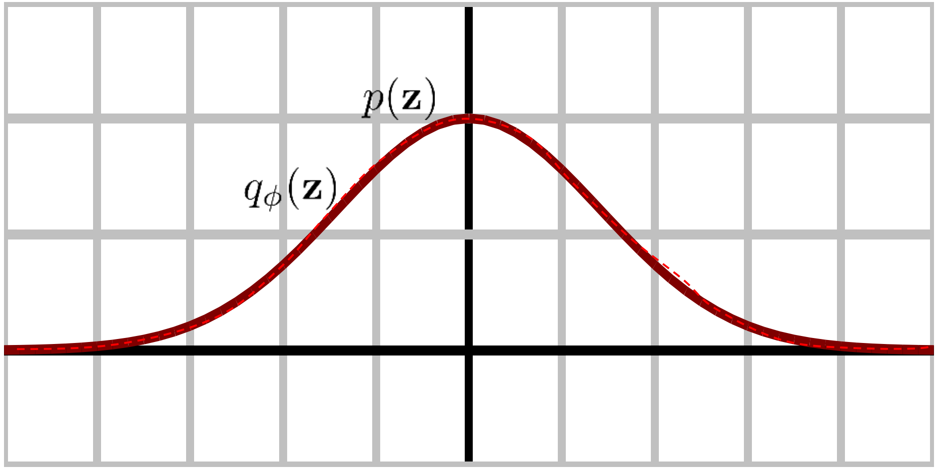
Generating human pose and appearance

VAE Limitations II

Tradeoff between data and prior term

- high weight on data term (big lambda):
  - crisp reconstruction of training data
  - but latent code is not Gaussian
    - the reconstruction of latent code samples from a Gaussian will be incorrect
- high weight on prior term (small lambda):
  - blurry reconstruction
  - but latent code follows a Gaussian distribution
    - sampling leads to expected outcomes (as good as training samples)





GANs

D should be **high** for fake examples
(from perspective of G)

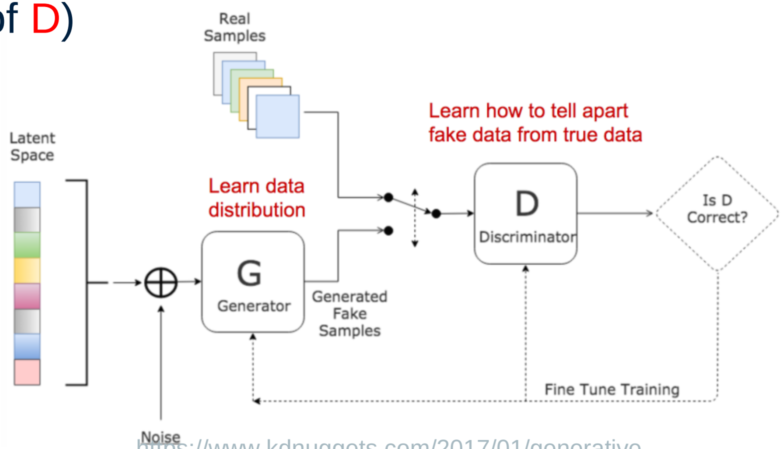
A min max game

$$\min_G \max_D V(D,G) = \min_G \max_D [E\sim p_{\text{data}}[\log D(x)] + Ez \sim p_z[\log(1 - D(G(z)))]]$$

D should be **high** for real examples
(not influenced by G)

D should be **low** for fake examples
(from perspective of D)

- Effects:
  - learning a loss function
  - like a VAE, we sample from a Gaussian distribution (some form of a prior assumption)



https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html

GAN training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

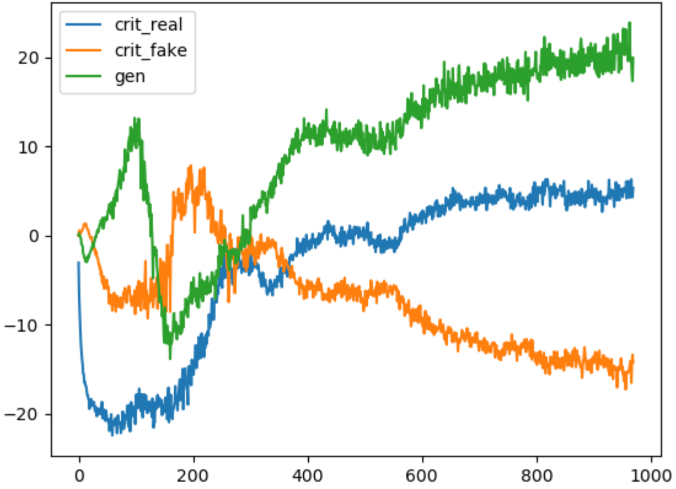
  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



Chaotic GAN loss behavior
(e.g., generator loss going up not down)

[Goodfellow et al., Generative Adversarial Networks. 2014]

Wasserstein GAN

Diverse measures exist to compare probability distributions

- The *Total Variation* (TV) distance

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)| \ .$$

- The *Kullback-Leibler* (KL) divergence

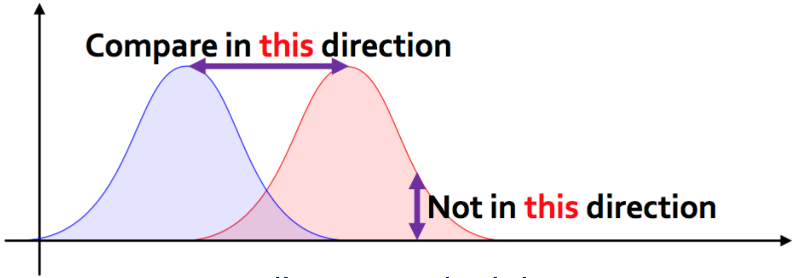
$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d\mu(x) \ ,$$

- The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

- The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\, \|x - y\| \,\big] \ ,$$



EM distance principle

[Arjovsky et al., Wasserstein GAN. 2017]

GAN vs. WGAN

Wasserstein distance is even simpler!

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
- **for** k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

- **end for**
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

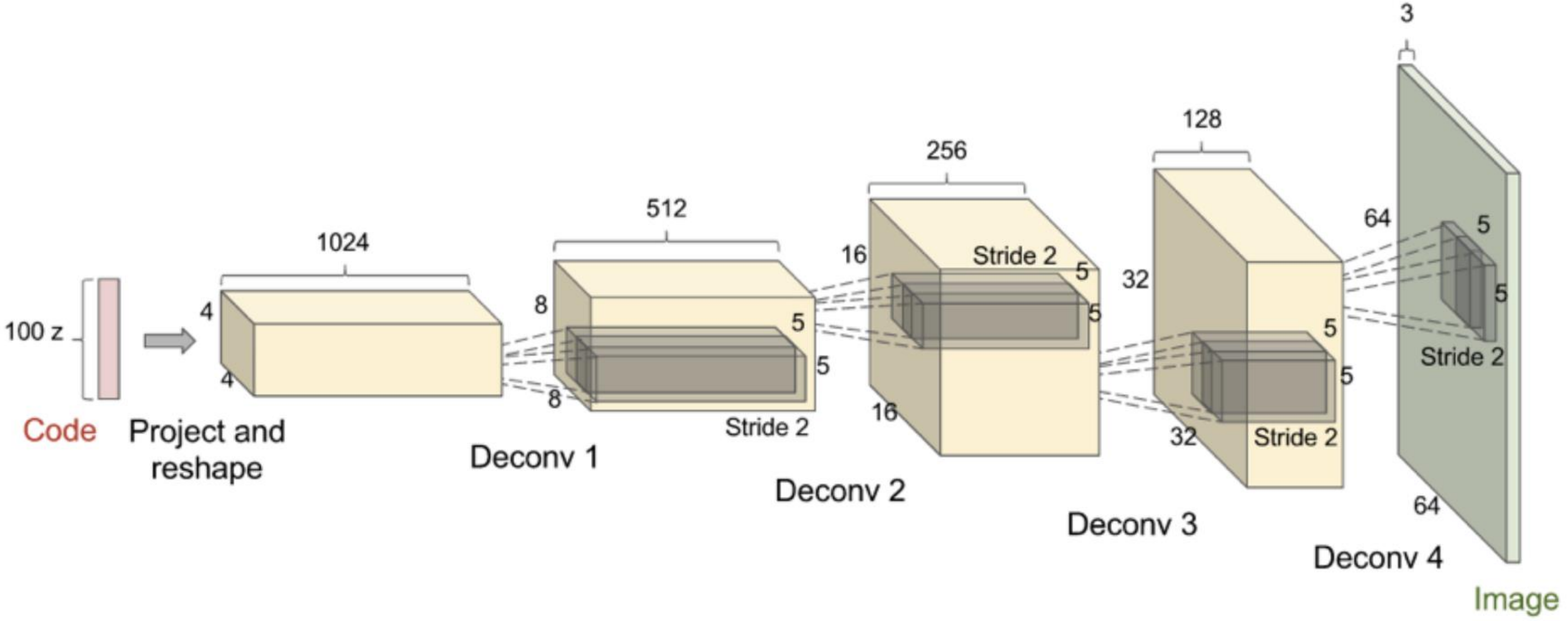
while θ has not converged **do**
- **for** $t = 0, \dots, n_{\text{critic}}$ **do**
  - Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
  - Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  - $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
  - $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
  - $w \leftarrow \text{clip}(w, -c, c)$
- **end for**
- Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
- $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
- $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$

end while

WGAN

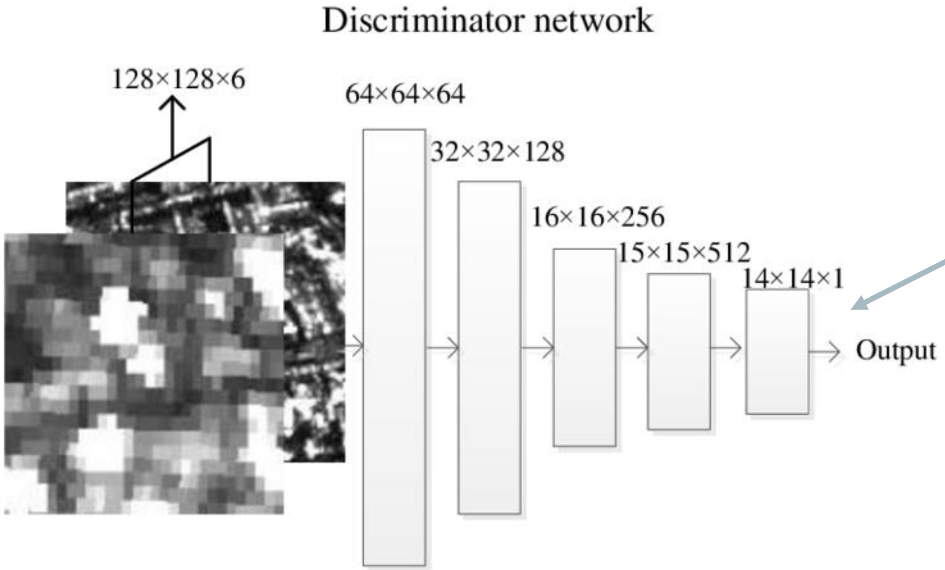
DCGAN

Convolutional generator architecture



PatchGAN

Patch-wise classification into real or fake (instead of globally)

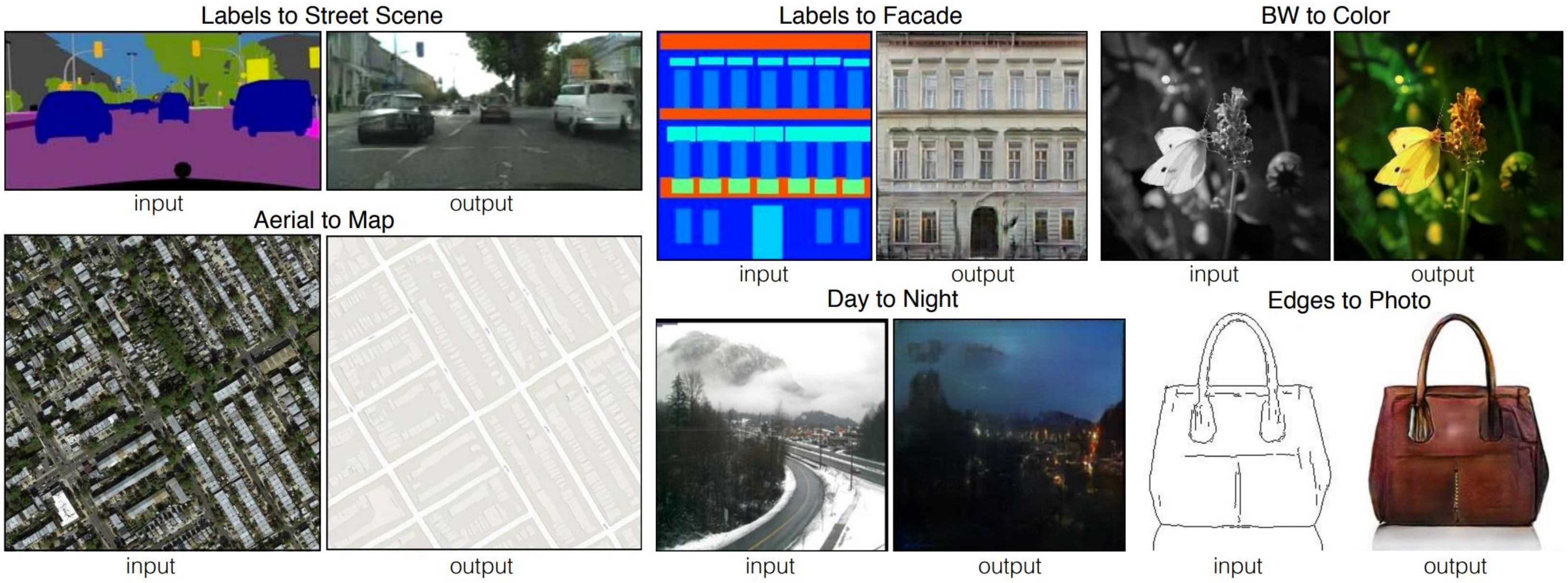


A feature map of fake/real probabilities

[Li and Wandt, Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks

Conditional Generative Adversarial Nets

First week of paper reading..



[Isola et al., Image-to-Image Translation with Conditional Adversarial Networks]

Hidden questions