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

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} + \begin{pmatrix} 0 & 2 \\ 4 & 6 \end{pmatrix} + \begin{pmatrix} 0 & 3 \\ 6 & 9 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 4 & 6 \\ 4 & 12 & 9 \end{pmatrix}$$

[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

Convolution operator

Input

Kernel

Output
Convolution as matrix multiplication (details)

What about horizontal striding?  
Skip every second row

What about vertical striding?  
Skip block of rows

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

Larger input width?  
insert zeroes here.

Larger input height?  
Insert more rows here.

Padding?  
Insert a row of zeroes

Input

Kernel
### Transposed convolution as matrix multiplication

Given:

- **Input** matrix $\begin{bmatrix} 25 \\ 31 \\ 43 \\ 49 \end{bmatrix}$
- **Kernel** matrix $\begin{bmatrix} 0 & 1 \\ 1.25 & 1.31 \\ 2.25 & \end{bmatrix}$

The transposed convolution can be represented as:

$$\begin{bmatrix} 25 \\ 31 \\ 43 \\ 49 \end{bmatrix} \times^T \begin{bmatrix} 0 & 1 \\ 1.25 & 1.31 \\ 2.25 & \end{bmatrix} = \begin{bmatrix} 0 & 25 \\ 25 & 31 \\ 50 & 180 \\ 142 & 86 \\ 227 & 227 \\ 147 & \end{bmatrix}$$

The output is as follows:

- **Output** matrix $\begin{bmatrix} 0, 25, 31 \\ 50, 180, 142 \\ 86, 227, 147 \end{bmatrix}$

This demonstrates how transposed convolution can be seen as a matrix multiplication operation.
Recap: Feature map size after convolutional kernels

Transformation of input and output by convolutions

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

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

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

Task I will be published tonight.
- Neural rendering

The other ones are delayed due to unforeseen difficulties.
# Paper assignment finished (on Piazza)

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

General case
\[ h = \text{encoder}_\theta(x) \]
\[ x' = \text{decoder}_\theta(h) \]

Simple non-linear case
\[ h = \sigma(Wx + b) \]
\[ x' = \sigma(W'h + b') \]

Linear case (similar to PCA)
\[ h = Wx + b \]
\[ x' = W'h + b' \]

General reconstruction objective
\[ \text{loss}(x, x') \]
- e.g., MSE loss

https://en.wikipedia.org/wiki/Autoencoder
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
  \[
  h = \text{encoder}_\theta(\text{noise}(x))
  \]
  \[
  x' = \text{decoder}_\theta(h)
  \]

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

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

$$E(x)$$

Error functions

$$p(x)$$

Distributions

$x^2$

Mean squared error (MSE)

$|x|$

Mean absolute error (MAE)

$\exp(-x^2)$

Gaussian distribution

$\exp(-|x|)$

Laplace distribution
The prior is a regularizer

For instance, an l2 loss on the neural network weights

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

corresponds to a prior on the weights

\[ P(\theta|X, Y) = P(f_\theta(X, Y)|\theta) \times P(\theta) \]

\[ \propto \frac{P(f_\theta(X, Y)|\theta) \times P(\theta)}{P(X, Y)} \]

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

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

Generating unseen faces

Generating music

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

https://github.com/yzwxx/vae-celebA
The Variational Autoencoder (VAE)

VAE Objective (general)
\[
\mathcal{L}(\phi, \theta, x) = -\mathbb{E}_{h \sim q_\phi(h|x)} \left( \log p_\theta(x|h) \right) + D_{KL}(q_\phi(h|x) \| p(h))
\]

Common parametrization
- Normal distributions
  \[
  p(h) = \mathcal{N}(0, 1) \\
  q_\phi(h|x) = \mathcal{N}(\mu(x), \omega(x)I) \\
  p_\theta(x|h) = \mathcal{N}(\mu(h), \sigma I)
  \]
  - parametrized by neural networks
    - encoder \( e \)
    - decoder \( d \)

Kullback–Leibler divergence (relative entropy)
- a dissimilarity measure between distributions
  - not symmetric, \( KL(p,q) \neq KL(q,p) \)
- Definition for continuous distributions
  \[
  D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left( \frac{p(x)}{q(x)} \right) \, dx
  \]
  probability density of \( Q \)
The Variational Autoencoder (VAE), simplified I

VAE Objective (general)

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

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

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

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

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

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

A simple autoencoder reconstruction loss, the squared difference between input and output
Kullback–Leibler divergence and entropy

Definition

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

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

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

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

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

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

\[
= -\log \sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} - \frac{1}{2}
\]

Using that \( m_2=0 \) and \( s_2=1 \)

https://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians
The Variational Autoencoder (VAE), simplified II

Starting point

\[ \mathcal{L}(\phi, \theta, x) = \lambda E_{h \sim q(\theta \mid x)} (x - d(h))^2 + D_{KL}(q(\theta \mid x) \| p(\theta)) + C \]

Simplification (for Gaussian prior p and Gaussian q with \( \mu_1 = e(x) \) and \( \sigma_1 = \omega(x) \))

Data term / log likelihood

\[ \mathcal{L}(\phi, \theta, x) = \lambda E_{h \sim q(\theta \mid x)} (x - d(h))^2 - \log \sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} + C' \]

Sampling (here a single sample)

\[ \approx \lambda (x - d(h))^2 - \log \sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} + C' \text{ with } h \sim q(\theta) \]

- **reconstruct the image**
- **'keep sigma > 0'**
- **'keep sigma and mu small'**

Expected value

\[ E_{x \sim q} f(x) = \int q(x) f(x) \, dx \]

\[ = \sum_{i=1}^{k} f(x_i) \text{ with } x_i \sim q \]
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
Deriving the posterior via Bayes

Goal: Compute the posterior

\[
p(h \mid x) = \frac{p(x \mid h)p(h)}{p(x)}
\]

The evidence, \textit{intractable} to compute (marginalization)

\[
p(x) = \int p(x|h)p(h)dh
\]

It requires integration over all possible latent values \( h \)

Good hidden code \( h \), given \( x \)

Attempt: Approximate the posterior with a NN (encoder)

\[
KL(q_\phi(h|x) || p(h|x)) = E_q[\log q_\phi(h|x)] - E_q[\log p(x, h)] + \log p(x)
\]

• still intractable due to \( p(x) \) in the divergence
Evidence Lower BOund

Consider the term

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

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

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

$$= -E_{h\sim q_\phi(h|x)} \left( \log p_\theta(x|h) \right) + D_{KL}(q_\phi(h|x)||p(h))$$

(equal to what we had before)

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

Influence

- changing mu
  - increase \( \rightarrow \) moves sample right
  - decrease \( \rightarrow \) moves sample left

- changing sigma
  - increase \( \rightarrow \) moves away from center
  - decrease \( \rightarrow \) 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

A min max game

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

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

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

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

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 $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
      $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left( x^{(i)} \right) + \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \right].$$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
    $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).$$
end for
```

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

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

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

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

[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.

```plaintext
for number of training iterations do
  for $k$ steps do
    ● Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    ● Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    ● Update the discriminator by ascending its stochastic gradient:
      $$\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$
  end for
  ● Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  ● Update the generator by descending its stochastic gradient:
    $$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$
end for
```

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

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

**Require:** $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size. $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.

**Require:** $w_0$, initial critic parameters. $\theta_0$, initial generator’s parameters.

1: while $\theta$ has not converged do
2:   for $t = 0, ..., n_{\text{critic}}$ do
3:     Sample $\{x^{(t)}\}_{i=1}^{m} \sim \mathcal{P}$, a batch from the real data.
4:     Sample $\{z^{(t)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
5:     $g_w \leftarrow \nabla_{w} \left[ \frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)})) \right]$
6:     $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
7:     $w \leftarrow \text{clip}(w, -c, c)$
8:   end for
9:   Sample $\{z^{(t)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
10:  $g_\theta \leftarrow -\nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
11:  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
DCGAN

Convolutional generator architecture
PatchGAN

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

[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

1. Hidden rule of hidden loci. Is there a hidden rule for hiding on a哲学 paper?

2. What is a hidden variable? Can we identify a hidden variable?

3. What if there is a hidden variable that is not supported by evidence, known?