Chapter 20: Deep Generative Models

Deep Learning Book by I. Goodfellow, Y. Bengio and A. Courville
OIST Deep Learning Reading Group

Makoto Otsuka

2016-08-26 Fri

Outline

20.6 Convolutional Boltzmann Machines

- High dimensional inputs (e.g., images) requires computational, memory, and statistical resources
- Approach: Use convolution
  - Works well also with RBM (Desjardins and Bengio, 2008)
  - Difficult to generalize pooling operation in energy-based model
    - $2^9 = 512$ energy function evaluations per 3×3 pooling units

Probabilistic max pooling (Lee et al, 2009)

- At most one unit may be active at a time
  - $n + 1$ possible states
  - $+1$ for the states with all units off (E = 0)
- No overlapoing pooling regions :(

20.6 Convolutional Boltzmann Machines

Convolutional DBM with probabilistic max pooling (Lee et al, 2009)

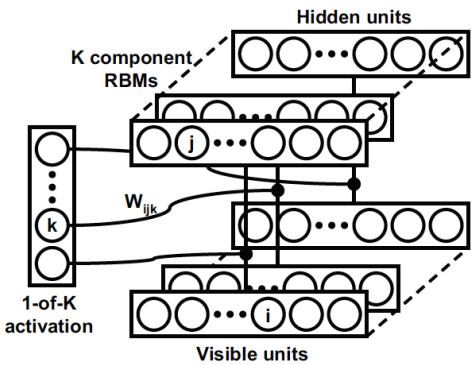
- Pros
  - Possible to fill in the missing portions of its input
- Cons
  - Challenging to make it work in practice
  - CNNs with supervised training objective usually perfoms better
  - Cannot change the size of pooling region to accept images in different sizes
    - Partition function changes as the size of the input changes
    - Variable sized inputs can be accepted if output feature maps can be scaled automatically
  - Difficult to handle image boundary

20.7 Boltzmann Machine for Structured or Sequential Outputs

- Tools for modeling structured ouput can be used for modeling sequences
- Conditional RBM (Taylor et al., 2007)
- Conditional RBM + spectral hashing (Mnih et al., 2011)
- Factored conditional RBM for modeling motion style (Taylor and Hinton, 2009)
- RTRBM (Sutskever, Hinton, Taylor, 2009)
- RNN-RBM for learning musical notes (Boulanger-Lewandowski et al., 2012)

20.8 Other Boltzmann Machines

- Discriminative RBMs (Larochelle and Bengio, 2008)
  - maximize $\log p(y|\mathbf{v})$
- Higher-order BMs
  - Higher-order BMs (Sejnowski, 1987)
  - Mixture of RBMs (Nair and Hinton, 2009)
  - Two groups of hidden units (Luo et al., 2011)
  - Point-wise gated BMs (PGBMs) (Sohn et al. 2013)



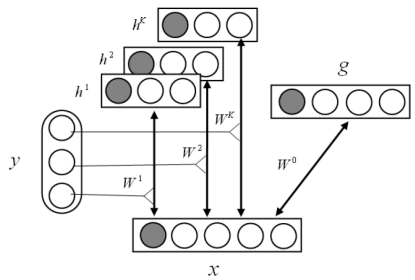
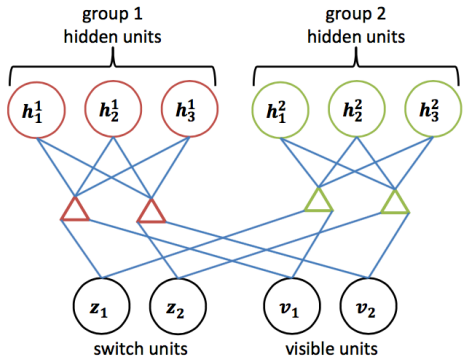


Figure 1: Third-order Restricted Boltzmann Machine with class-irrelevant units. The black circles are for implementing biases.



(a) PGBM

20.9 Back-Propagation through Random Operations

- $\mathbf{x}$: input variable
- $\mathbf{z}$: simple distribution (e.g., uniform, Gaussian)
- $f(\mathbf{x}, \mathbf{z})$ appears stochastic
- If f is continuous and differentiable, we can apply BP as usual

Example: Drawing samples from a Gaussian distribution

- It seems counter intuitive to take a derivative of the sampling process y with respect to μ or σ

$$y \sim \mathcal{N}(\mu, \sigma^2)$$

- But it make sense if we rewrite this sampling process as follows:

$$y = \mu + \sigma z, \quad z \sim \mathcal{N}(0, 1)$$

- Note that z does not depend on μ nor σ

$$\mu = g_1(\mathbf{x}; \theta_1), \quad \sigma = g_2(\mathbf{x}; \theta_2)$$

Reparametrization trick

- Reparameterization trick (stochastic back-propagation, perturbation analysis)
- ω: a variable containing both parameters θ, and if applicable, the inputs $\mathbf{x}$
- We can rewrite

$$y \sim p(y|\omega)$$

as

$$y = f(z;\omega) = \mu_{\theta_1} + \sigma_{\theta_2} z$$

- ω is not the function of $\mathbf{z}$
- $\mathbf{z}$ is not the function of ω

20.10 Directed Generative Nets

- Since 2013, directed generative nets became popular
- 20.10.1 Sigmoid Belief Nets
- 20.10.2 Differentiable Generator Nets
- 20.10.3 Variational Autoencoders (VAEs)
- 20.10.4 Generative Adversarial Networks (GANs)
- 20.10.5 Generative Moment Matching Networks
- 20.10.6 Convolutional Generative Networks
- 20.10.7 Auto-Regressive Networks
- 20.10.8 Linear Auto-Regressive Networks
- 20.10.9 Neural Auto-Regressive Networks
- 20.10.10 Neural autoregressive density estimator (NADE)

20.10.1 Sigmoid Belief Nets

- Proposed by Neal (1990)
- Universal approximator of binary visible units (Sutskever and Hinton, 2008)
- Inferece is hard
  - Mean field inference is intractable
  - Approximate lowerbound (Saul et al., 1996) is only applicable for small networks
  - SBNs combined with an inference net need to rely on unreliable BP through discrete sampling processes
- Recent approaches enable fast training
  - Importance sampling + reweighted wake-sleep (Bornschein and Bengio, 2015)
  - Bidirectional Helmnoltz machines (Bornschein et al, 2015)

20.10.2 Differentiable Generator Nets

- The model transforms samples of latent varialbes $\mathbf{z}$ to
  - samples $\mathbf{x}$ directly (Approach 1) or
  - distributions over samples $\mathbf{x}$ (Approach 2)

  using differentiable function $g(\mathbf{z}; \theta^{(g)})$ (usually NN)
- Parameterized computational procedures for generating samples
- Examples
  - Variational autoencoders (VAE)
    - generator net + inference net
  - Generative adversarial networks (GAN)
    - generator net + discriminator net
  - Generative moment matching networks

Approach 1: Direct Sampling (1/2)

- Generating samples from a multivariate Gaussian distribution
  - $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, I)$
  - $g(\mathbf{z}; \theta^{(g)})$ is given by

$$\mathbf{x} = g(\mathbf{z}; \theta^{(g)}) = \mu + \mathbf{L}\mathbf{z} \tag{20.71}$$

$$\Sigma = \mathbf{L}\mathbf{L}^{\top} \quad \cdots \text{Cholesky decomposition}$$

- Inverse transform sampling (Devroye, 2013)
  - $z \sim U(0, 1)$
  - $g(z)$ is given by the inverse of the cdf $F(x) = \int_{-\infty}^{x} p(v)dv$
- $g()$ is transforming the distribution over $\mathbf{z}$ into the desired distribution over $\mathbf{x}$

Approach 1: Direct Sampling (2/2)

- For invertible, differentiable, continuous g,

$$p_x(\mathbf{x})d\mathbf{x} = p_z(\mathbf{z})d\mathbf{z}$$

$$p_x(g(\mathbf{z}))\left|\det\left(\frac{\partial g}{\partial \mathbf{z}}\right)\right| = p_z(\mathbf{z}) \tag{20.72}$$

$$p_x(\mathbf{x}) = \frac{p_z(g^{-1}(\mathbf{x}))}{\left|\det\left(\frac{\partial g}{\partial \mathbf{z}}\right)\right|} \tag{20.73}$$

where $\mathbf{x} = g(\mathbf{z})$

- Usually difficult to evaluate $\log p(\mathbf{x})$ directly
- Often use indirect means of learning $g(\cdot)$

Approach 2: Defining conditional distribution

- Defining conditional distribution

$$p(x_i = 1|\mathbf{z}) = g(\mathbf{z})_i \tag{20.74}$$

- Marginalizing it out to define $p_g(\mathbf{x})$

$$p(\mathbf{x}) = \mathbf{E}_{\mathbf{z}} p(\mathbf{x}|\mathbf{z}) \tag{20.75}$$

Direct sampling v.s. defining conditional distribution

- Approach 1: Emitting samples directly from $p_g(\mathbf{x})$
  - Pros
    - No longer forced to use carefully-designed conditional distributions
  - Cons
    - Capable of generating only continuous data
- Approach 2: Emitting parameters of $p_g(\mathbf{x})$
  - Pros
    - Capable of generating discrete data as well as continuous data
  - Cons
    - Need to use carefully-designed conditional distributions

Properties of Differentiable Generator Nets

- Diffrentiable generator nets were motivated by the success of BP applied to SL
- But unsupervised generative modeling is more difficult than SL
- Differentiable generator nets need to solve two objectives
  - How to arrange $\mathbf{z}$ space in a useful way
  - How to map from $\mathbf{z}$ to $\mathbf{x}$
- Chair generation experiment (Dosovitskiy et al., 2015)
  - Mapping between $\mathbf{z}$ and $\mathbf{x}$ is given beforehand (mapping is deterministic)
  - Result implies DGNs have sufficient model capacity and it is optimizable
  - But, don't know what happends when mapping between $\mathbf{z}$ and $\mathbf{x}$ is non-deterministic

20.10.3 Variational Autoencoders (VAEs)

- Proposed by Kingma (2013) and Rezende et al. (2014)
- Directed model that use learned approximate inference
- Can be trained purely with gradient-based methods
- Great blog posts about VAEs: (link 1, link 2)

Building blocks of VAE

- $p_{\text{model}}(\mathbf{z})$ Code distribution (prior)
- $g(\mathbf{z}; \theta^{(g)})$ Differentiable generator net (decoder)
- $p_{\text{model}}(\mathbf{x}|\mathbf{z}) = p_{\text{model}}(\mathbf{x}; g(\mathbf{z}; \theta^{(g)}))$ Generative model
- $f(\mathbf{x}; \theta^{(f)})$ Approximate inference net (encoder)
- $q(\mathbf{z}|\mathbf{x}) = q(\mathbf{z}; f(\mathbf{x}; \theta^{(f)}))$ Recognition model

Training VAEs

- Instead of directly maximizing log likelihood

$$\log p_{\text{model}}(\mathbf{x}) = \mathcal{L}(q) + D_{\text{KL}}(q(\mathbf{z}|\mathbf{x})\|p_{\text{model}}(\mathbf{z}|\mathbf{x}))$$

- Train VAE by maximizing variational lower bound $\mathcal{L}(q)$

$$\begin{aligned}
\mathcal{L}(q) &= \mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{z}, \mathbf{x}) + \mathcal{H}(q(\mathbf{z}|\mathbf{x})) && (20.76)\\
&= \mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{x}|\mathbf{z}) + \mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{z}) - \mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log q(\mathbf{z}|\mathbf{x})\\
&= \mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{x}|\mathbf{z}) - D_{\text{KL}}(q(\mathbf{z}|\mathbf{x})\|p_{\text{model}}(\mathbf{z})) && (20.77)\\
&\leq \log p_{\text{model}}(\mathbf{x}) && (20.78)
\end{aligned}$$

- Interpretation
  - $\mathcal{H}(q(\mathbf{z}|\mathbf{x}))$ encourages higher variance
  - $D_{\text{KL}}(q(\mathbf{z}|\mathbf{x})\|p_{\text{model}}(\mathbf{z}))$ encourages smaller distance between approximate posterior and prior
  - $\mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_{\text{model}}(\mathbf{x}|\mathbf{z})$ encourages lower reconstruction error

Traditional approach to variational inference and learning

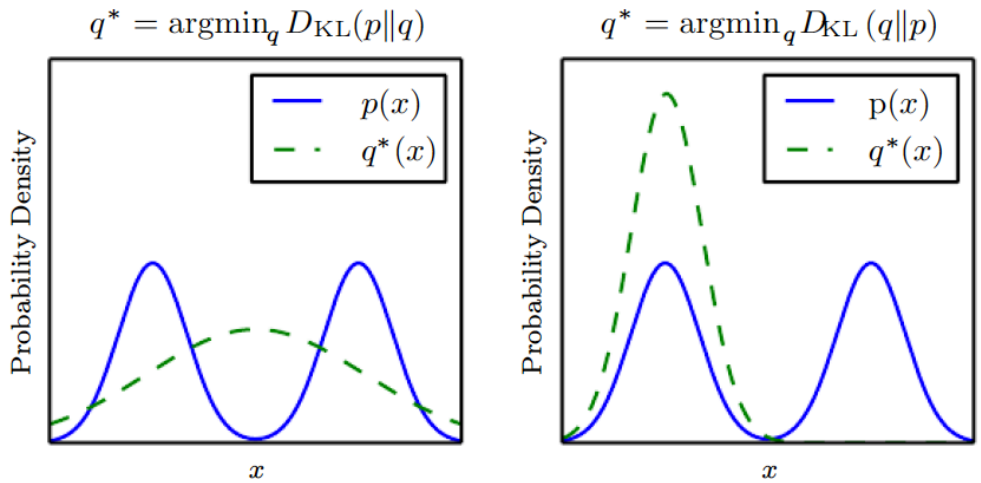
- Infer $q(\cdot)$ via optimization algorithm (e.g., iterated fixed point equations)
- Iterative scheme is slow
- Often require the ability to compute $\mathbf{E}_{\mathbf{z}\sim q} \log p_{\text{model}}(\mathbf{z}, \mathbf{x})$ in closed form

Main idea behind VAE

- Train a parameteric encoder $f(\mathbf{x}; \theta^{(f)})$ for $q(\mathbf{z}|\mathbf{x}) = q(\mathbf{z}; f(\mathbf{x}; \theta^{(f)}))$
- If $\mathbf{z}$ is continuous, we can use BP
- All the expectation in $\mathcal{L}$ may be approximated by MC sampling

Shortcomings of VAE

1. Generated images are blurry probably due to
   - ML estimate
     - Minimizing $D_{\mathrm{KL}}(p\|q)$ instead of $D_{\mathrm{KL}}(q\|p)$
     - Tendency to ignore small changes in images (Theis et al., 2015; Huszar 2015)
   - Use of Gaussian model $p_{\mathrm{model}}(\mathbf{x}; g(\mathbf{z}))$



1. Tend to use only small subset of $\mathbf{z}$

Extension of VAE (1/2)

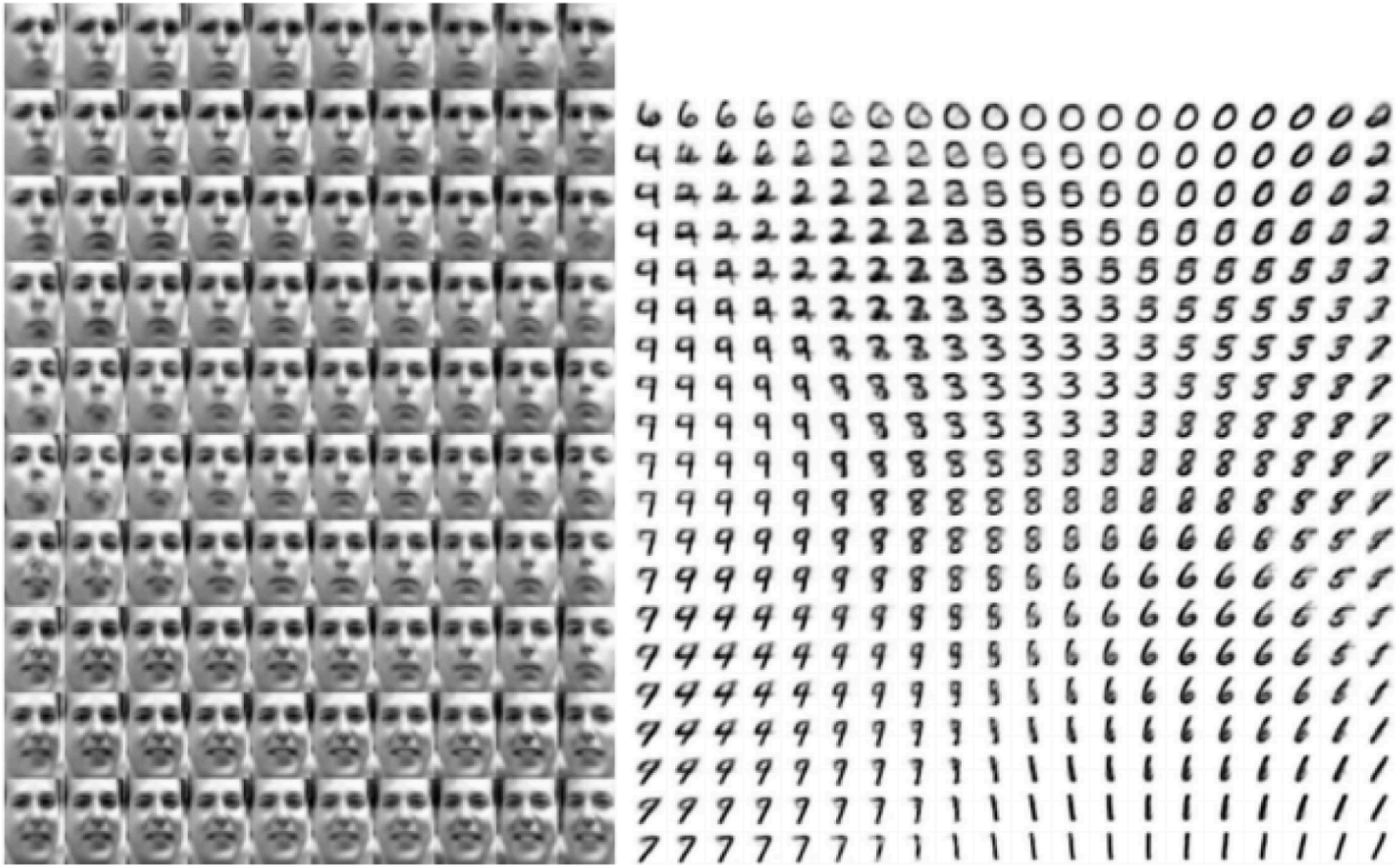
- VAE is much easier to extend than Boltzmann machines
- Deep recurrent attention writer (DRAW) model (Gregor et al., 2015)
  - Recurrent encoder + recurrent decoder + attention
- Variational RNNs (Chung et al, 2015b)
  - Recurrent encoder and decoder
  - Unlike traditional RNN, it also has variability in latent space
- Importance weighted autoencoder or IWAE (Burda et al., 2015) objective
  - Equivalent to the traditional lower bound when $k = 1$
  - Tighter bound for $\log p_{\text{model}}(\mathbf{x})$ when k increases

$$\mathcal{L}_k(\mathbf{x}, q) = \mathbf{E}_{\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(k)} \sim q(\mathbf{z}|\mathbf{x})} \left[\log \frac{1}{k} \sum_{i=1}^{k} \frac{p_{\text{model}(\mathbf{x},\mathbf{z}^{(i)})}}{q(\mathbf{z}^{(i)}|\mathbf{x})} \right]$$

Extension of VAE (2/2)

- Some interesting connections to the multi-prediction DBM (MP-DBM) in Fig. 20.5 and other approaches that involve back-propagation through the approximate inference graph (Goodfellow et al., 2013b; Stoyanov et al., 2011; Brakel et al., 2013).
- VAE is defined for arbitrary computational graphs
  - No need to restrict the choice of models to those with tractable mean field fixed point equations
- One disadvantage of the variational autoencoder is that it learns an inference network for only one problem, inferring $\mathbf{z}$ given $\mathbf{x}$.

Manifold learning with VAE



20.10.4 Generative Adversarial Networks (GANs)

- Proposed by Goodfellow et al. (2014c)
- Example of differentiable generator networks
- Generator net: $g(\mathbf{z}; \theta^{(g)})$
  - Directly produces samples: $\mathbf{x} = g(\mathbf{z}; \theta^{(g)})$
  - Payoff is $-v(\theta^{(g)}, \theta^{(d)})$
  - Attempts to fool the classifier into believing its samples are real
- Discriminator net: $d(\mathbf{x}; \theta^{(d)})$ (probability of $\mathbf{x}$ being real)
  - Payoff is $v(\theta^{(g)}, \theta^{(d)})$
  - Attempts to learn to correctly classify samples as real or fake

$$g* = \arg\min_{g} \max_{d} v(g, d)$$

$$v(\theta^{(g)}, \theta^{(d)}) = \mathbf{E}_{\mathbf{x} \sim p_{\text{data}}} \log d(\mathbf{x}) + \mathbf{E}_{\mathbf{x} \sim p_{\text{model}}} \log(1 - d(\mathbf{x}))$$

Difficulties in GANs

- Learning is difficult in GAN
  - E.g.) $v(a, b) = ab$
  - Note that the equilibria for a minimax game are not local minima of v
  - Instead, they are points that are simultaneously minima for both players' costs.
  - This means that they are saddle points of v that are local minima with respect to the first player's parameters and local maxima with respect to the second player's parameters.
- Alternative formulation of payoffs (Goodfellow et al., 2014c)

Deep convolutional GAN (DCGAN)

- Proposed by Radford et al. (2015)

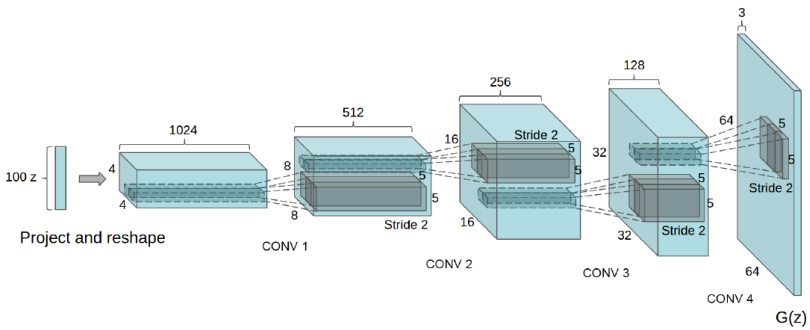
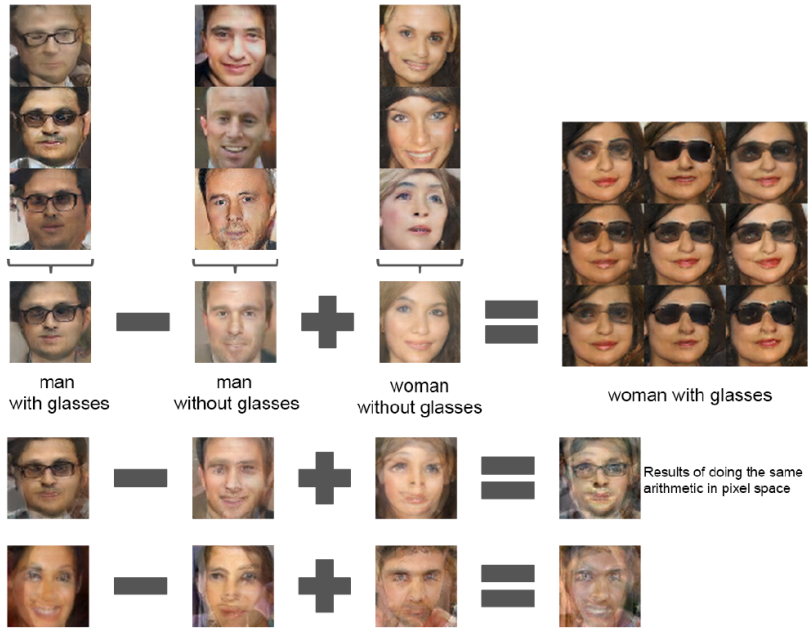
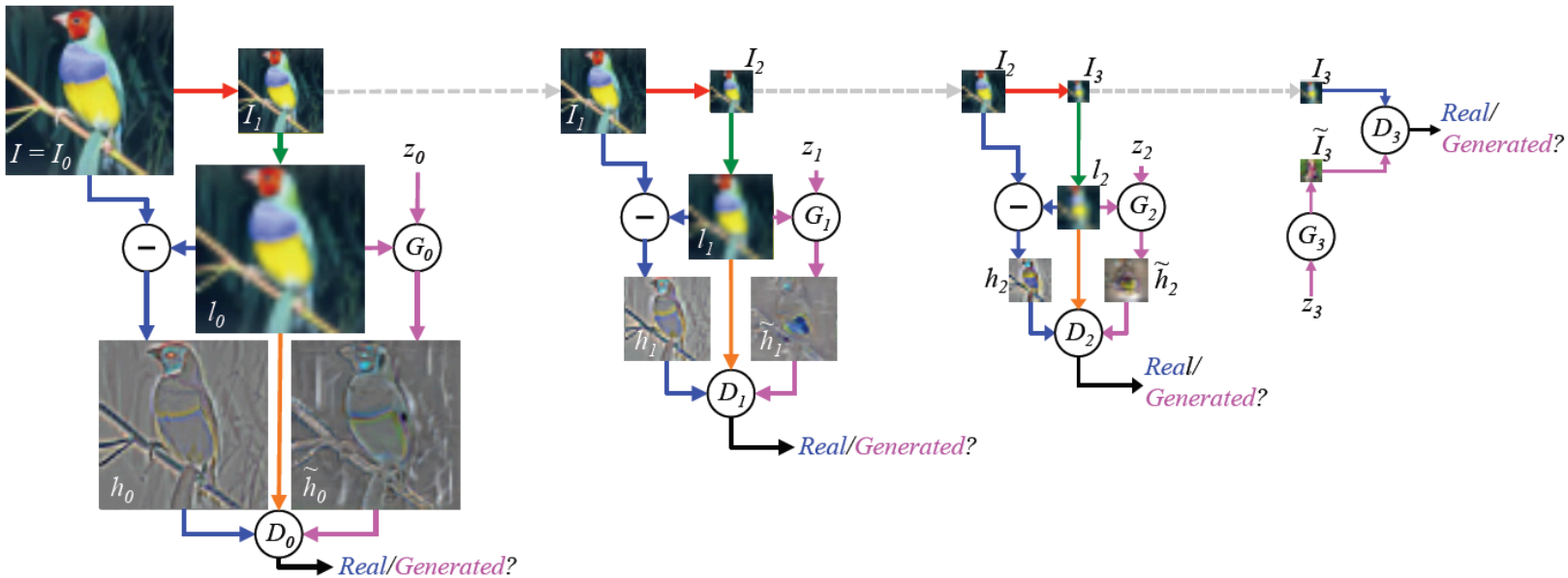


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.



Conditional GANs and LAPGAN

- Conditional GANs (Mirza and Osindero, 2014)
  - Learn to sample from $p(\mathbf{x}|\mathbf{y})$ rather than $p(\mathbf{x})$
- LAPGAN (Denton et al. 2015) uses series of conditional GANs with Laplacian pyramid



Generated Images: DCGAN and LAPGAN

Figure 20.7: Images generated by GANs trained on the LSUN dataset. *(Left)* Images of bedrooms generated by a DCGAN model, reproduced with permission from Radford *et al.* (2015). *(Right)* Images of churches generated by a LAPGAN model, reproduced with permission from Denton *et al.* (2015).

Properties of GAN

- It can fit distribution that assign zero probability to the training points.
- Somehow tracing out manifold that is spanned by training data
- Add Gaussian noise to all of the generated values to ensure non-zero probabilities to all points
- Dropout seems to be important in the discriminator network
  - In particular, units should be stochastically dropped while computing the gradient for the generator network to follow.

Models similar to GAN

- GAN is designed for differentiable generator net.
- Similar principles can be used to train other kind of models
- Self-supervised boosting can be used to train RBM generator to fool a logistic regression discriminator (Welling et al., 2002)

20.10.5 Generative Moment Matching Networks

- Proposed by Li et al., 2015; Dziugaite et al., 2015
- Example of differentiable generator networks
- Unlike VAEs and GANs, no need to pair with other network
- Trained with moment matching

$$\mathbf{E}_{\mathbf{x}} \prod_i x_i^{n_i} \tag{20.82}$$

$$\mathbf{n} = [n_1, \ldots, n_d]^\top$$

- Matching all moment for all dimensions is infeasible
- Instead, minimize the maximum mean discrepancy, MMD (Scholkopf and Smola, 2002; Gretton et al., 2012)
  - Measures the error in the first moments in an infinite-dimensional space implicitly defined by kernel
  - MMD cost is 0 if and only if the two distributions being compared are equal

Generative Moment Matching Networks

- Samples from GMMN is not visually pleasing
- Visual can be improved if generator is combined with an autoencoder
- Need large batch size to obtain reliable estimate of the moment

20.10.6 Convolutional Generative Networks

- When generating images, it is useful to include convoluitonal structure (Goodfellow et al., 2014c; Dosovitskiy et al., 2015)
- Pooling function is not invertible
- "Un-pooling" (Dosovitskiy et al., 2015)
  - Inverse of max-pooling
  - Upper-left corner takes maximum value and other cells are set to 0
  - Samples generated by the model are visually pleasing

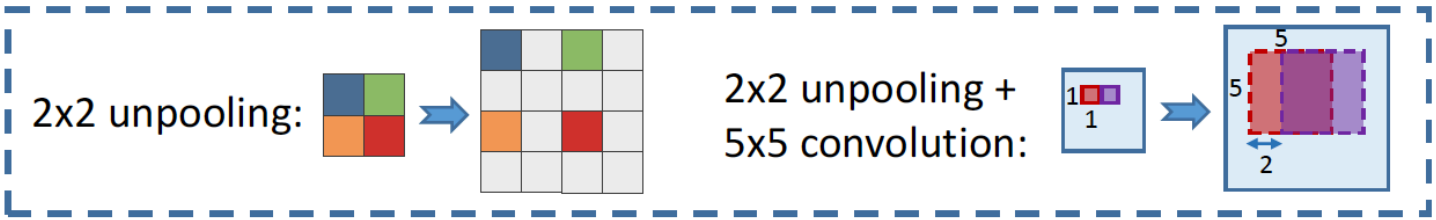


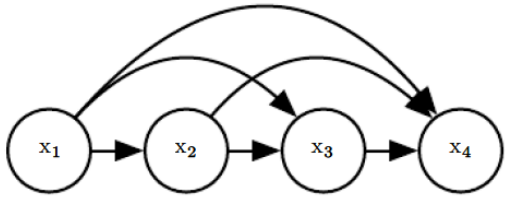
Figure 3. Illustration of unpooling (left) and unpooling+convolution (right) as used in the generative network.

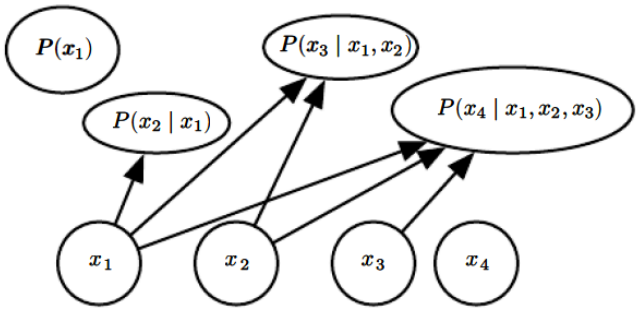
20.10.7 Auto-Regressive Networks

- Directed probabilistic models with no latent random variables
- Fully-visible Bayes networks (FVBNs)
- NADE (Larochelle and Murray, 2011)
  - One type of auto-regressive network
- Reuse of features
  - Statistical advantages (fewer unique parameters)
  - Computational advantages (less computation)

20.10.8 Linear Auto-Regressive Networksv

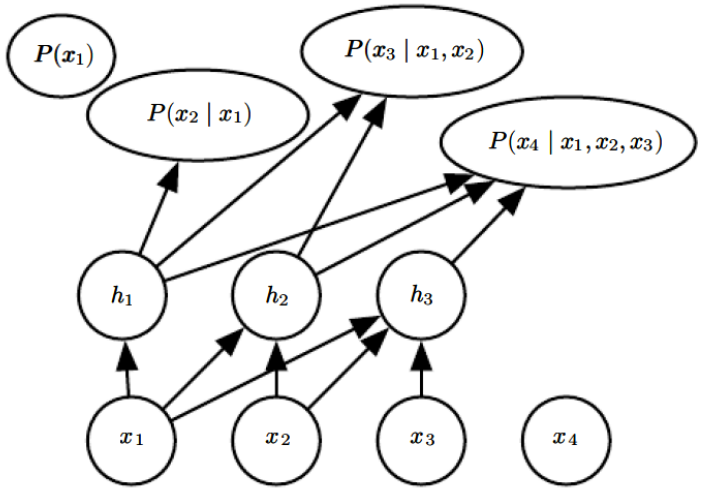
- Simplest form of auto-regressive network
  - No hidden units
  - No shared parameters or features
- Examples
  - Logistic auto-regressive network (binary)
- Introduced by Frey (1998)
- $O(d^2)$ parameters (d variables)





20.10.9 Neural Auto-Regressive Networks

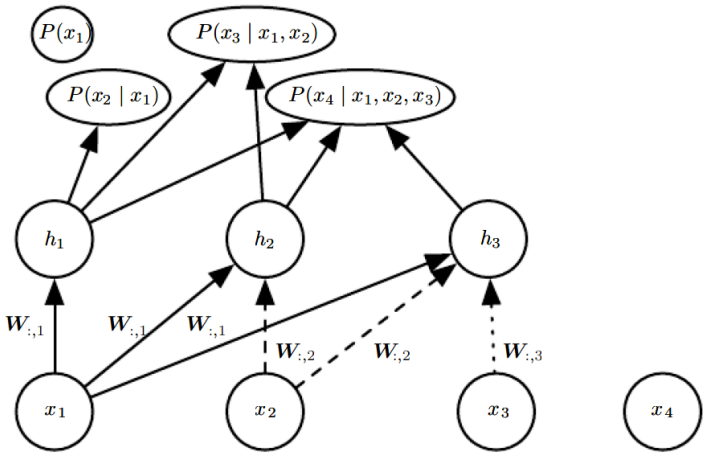
- Introduced by Bengio and Bengio (2000a, b)
- Model capacity can be increased
- Generalization can be improved by parameter and feature sharing
- Two advantages of using NN
  1. Can capture nonlinear dependency with limited number of parameters
  2. Left-to-right architecture allows one to merge all the NN into one



20.10.10 Neural autoregressive density estimator (NADE)

- Proposed by Larochelle and Murray (2011)
- Successful recent form of neural auto-regressive network (NAN)
- NAN with parameter sharing

$$W'_{j,k,i} = W_{k,i} \qquad (20.83)$$
$$W'_{j,k,i} = 0 \text{ if } j < i$$



- i -th input node
- j -th hidden node
- k -th element of j -th hidden node

Variants of NADE

- NADE-k (Raiko et al, 2014)
  - k-step mean field recurrent inference
- RNADE (Uria et al., 2013)
  - Processing continuous-valued data using Gaussian mixture
  - Use pseudo-gradient to reduce unstable gradient calculation caused by coupling of μ_i and σ_i^2
- Ensemble of NADE models handling reordered inputs o (Murray and Larochelle, 2014)
  - Better generalization than a fixed-order model

$$p_{\text{ensemble}}(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} p(\mathbf{x}|o^{(i)})$$

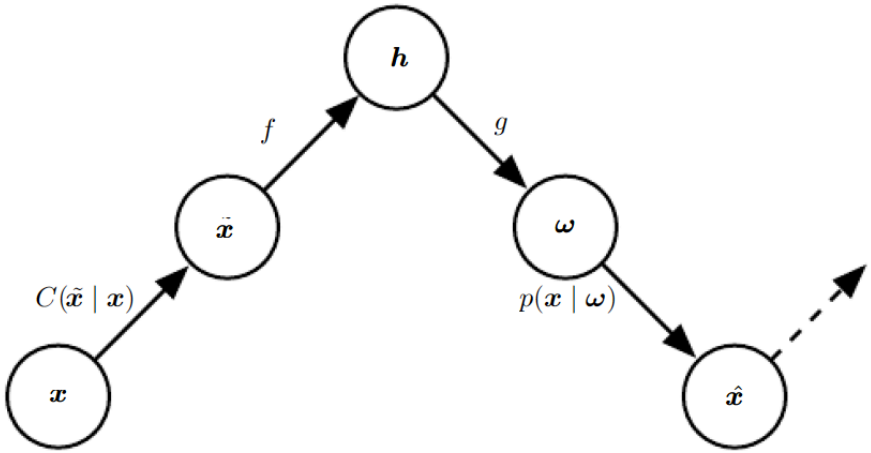
- Deep NADE is computationally expensive and not so much gain

20.11 Drawing Samples from Autoencoders

- Underlying connections between score matching, denoising autoencoders, and contractive autoencoders
- They are learning data distributions, but how can we sample data?
- VAE - ancestral sampling
- CAE
  - Repeated encoding and decoding with injected noise will induce a random walk along the surface of the manifold (Rifai et al., 2012; Mesnil et al., 2012)
  - Manifold diffusion technique (kind of MC)
- DAE
  - More general MC

20.11.1 Markov Chain Associated with any DAE

- What noise to inject and where?
- How to construct such a MC for generalized DAE



1. Starting from the previous state $\boldsymbol{x}$, inject corruption noise, sampling $\tilde{\boldsymbol{x}}$ from $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$.
2. Encode $\tilde{\boldsymbol{x}}$ into $\boldsymbol{h} = f(\tilde{\boldsymbol{x}})$.
3. Decode $\boldsymbol{h}$ to obtain the parameters $\boldsymbol{\omega} = g(\boldsymbol{h})$ of $p(\mathbf{x} \mid \boldsymbol{\omega} = g(\boldsymbol{h})) = p(\mathbf{x} \mid \tilde{\boldsymbol{x}})$.
4. Sample the next state $\boldsymbol{x}$ from $p(\mathbf{x} \mid \boldsymbol{\omega} = g(\boldsymbol{h})) = p(\mathbf{x} \mid \tilde{\boldsymbol{x}})$.

20.11.3 Walk-Back Training Procedure

- Proposed by Bengio et al. (2013c) to accerelate the convergence rate of generative training of DAE
- Includes alternative multiple stochastic encode-decode steps

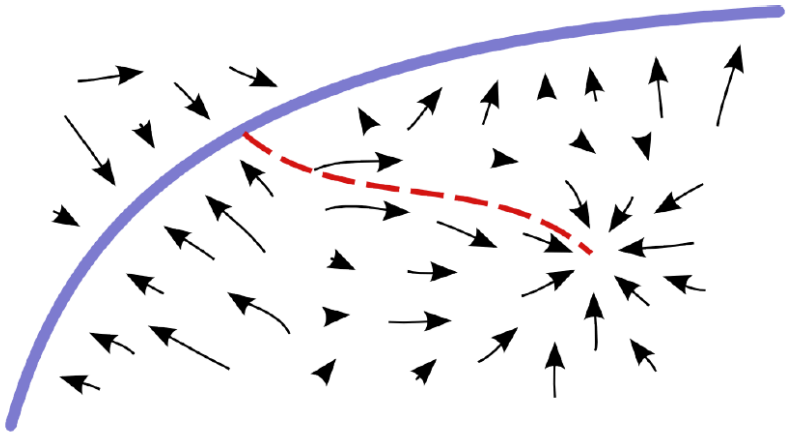


Figure 2: Walkback samples get attracted by spurious modes and contribute to removing them. Segment of data manifold in violet and example walkback path in red dotted line, starting on the manifold and going towards a spurious attractor. The vector field represents expected moves of the chain, for a unimodal $P(X|\tilde{X})$, with arrows from $\tilde{X}$ to X.

20.12 Generative Stochastic Networks (GSNs) (1/2)

- Proposed by Bengio et al. (2014)
- Generalization of DAE
  - Include latent varialbe $\mathbf{h}$ in the generative Markov chain
- Generative process is parameterized by two distributions
  - $p(\mathbf{x}^{(k)}|\mathbf{h}^{(k)})$: reconstruction distribution
  - $p(\mathbf{h}^{(k)}|\mathbf{h}^{(k-1)}, \mathbf{x}^{(k-1)})$: state transition distribution

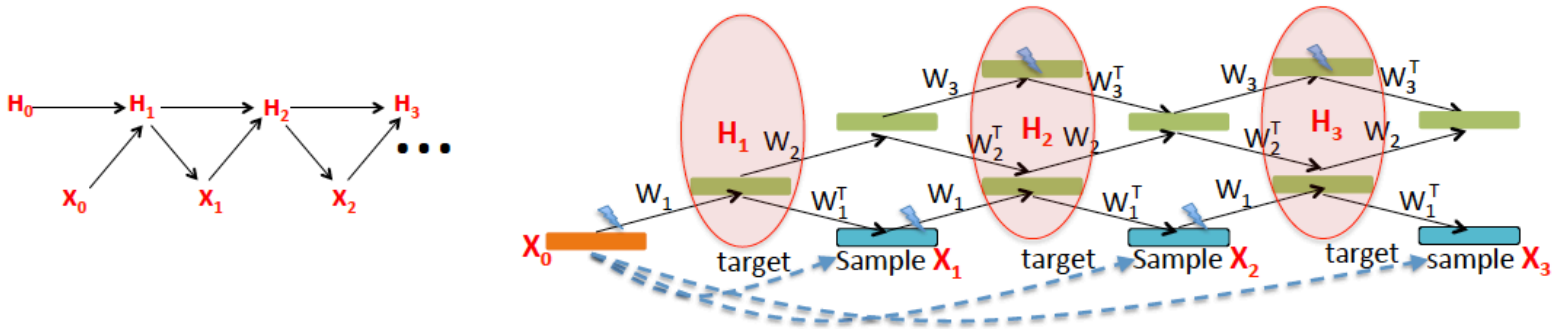


Figure 2. Left: Generic GSN Markov chain with state variables X_t and H_t. *Right:* GSN Markov chain inspired by the unfolded computational graph of the Deep Boltzmann Machine Gibbs sampling process, but with backprop-able stochastic units at each layer. The training example $X = x_0$ starts the chain. Either odd or even layers are stochastically updated at each step. All x_t's are corrupted by salt-and-pepper noise before entering the graph (lightning symbol). Each x_t for $t > 0$ is obtained by sampling from the reconstruction distribution for that step, $P_{\theta_2}(X_t|H_t)$. The walkback training objective is the sum over all steps of log-likelihoods of target $X = x_0$ under the reconstruction distribution. In the special case of a unimodal Gaussian reconstruction distribution, maximizing the likelihood is equivalent to minimizing reconstruction error; in general one trains to maximum likelihood, not simply minimum reconstruction error.

20.12 Generative Stochastic Networks (GSNs) (2/2)

- Properties of GSNs
  - Joint distribution is defined implicitly by MC if it exists
  - Maximize $\log p(\mathbf{x}^{(k)} = \mathbf{x}|\mathbf{h}^{(k)})$ where $\mathbf{x}^{(0)} = \mathbf{x}$
  - Walk-back training protocol was used to improve training convergence
- 20.12.1 Discriminant GSNs
  - Possible to model $p(\mathbf{y}|\mathbf{x})$ instead of $p(\mathbf{x})$ with GSNs

20.13 Other Generation Schemes

- So far, MCMC sampling, ancestral sampling, or some mixture of the two
- Diffusion inversion objective for learning a generative model based on non-equilibrium thermodynamics (Sohl-Dickstein et al., 2015)
  - Structured -> unstructured
  - Running this process backward
- Approximate Bayesian computation (ABC) framework (Rubin et al., 1984)
  - Samples are rejected or modified in order to make the moments of selected functions of the samples match those of the desired distribution.
  - This is not a moment matching because ABC changes the samples themselves

20.14 Evaluating Generative Models (1/2)

- Usually we cannot evaluate $\log p(\mathbf{x})$ directly
- Which is better?
  - Stochastic estimate of the log-likelihood for model A
  - Deterministic lower bound on the log-likelihood for model B
- High likelihood estimate $\log p(\mathbf{x}) = \log \tilde{p}(\mathbf{x}) - \log Z$ can be obtained due to
  - Good model, or
  - Bad AIS implimentation (underestimate Z)

20.14 Evaluating Generative Models (2/2)

- Changing preprocessing step is unacceptable when comparing different generative models
  - e.g., multiplying the input by 0.1 will artificially increase likelihood by a factor of 10
  - It is essential to compare real-valued MNIST models only to other real-valued models and binary-valued models only to other binary-valued models
  - Use exactly same binalization scheme for all compared models
- Visual inspection can be used but not a reliable measure
  - Poor model can produce good examples by overfitting
- Need to develop some other ways to evaluate generative models
  - You can get extremely high likelihood by asigning arbitrarily low variance to background pixels that never changes

Conclusions

- Deep generative models are important building blocks for AI systems