Deep Generative Models
(Unsupervised Learning)

CEng 783 – Deep Learning
Fall 2017

Emre Akbaş
Reminders

• Next week: project progress demos in class
  – Describe your problem/goal
  – What you have done so far
  – What you will do

• Each project will have 4 minutes for presentation. Then, a few minutes of feedback.
Today

• Boltzmann Machines
• Restricted Boltzmann Machines
  – Contrastive Divergence
• Deep Belief Networks
• Generative Adversarial Networks
• Autoencoders
• Variational Autoencoders
What is a generative model?

A generative model is a model using which you can generate a data value.

\[ p_{\theta}(x) \]

Some generative models:
- Gaussian Mixture Models or other mixture models (e.g. k-means)
- Hidden Markov Model
- Naive Bayes
- Latent Dirichlet Allocation
- Restricted Boltzman Machines
- Generative Adversarial Networks (GANs)
- Variational Autoencoders
- ...

CEng 783 - Deep Learning - E.A.
What is a generative model?

A generative model is a model using which you can generate a data value.

\[ p_\theta(x) \]

Some generative models:
- Gaussian Mixture Models or other mixture models (e.g. k-means)
- Hidden Markov Model
- Naive Bayes
- Latent Dirichlet Allocation
- Restricted Boltzman Machines
- Generative Adversarial Networks (GANs)
- Variational Autoencoders
- ...

Today
Generative vs Discriminative

Given a dataset \( D = \{(x_i, y_i)\}_{i=1}^{N} \)

a generative model models \( p(x, y) \)

a discriminative model models \( p(y|x) \)

Toy example: Let \( D = \{(3,+1), (3,+1), (1,+1), (1,-1)\} \)

What are \( p(x,y) \) and \( p(y|x) \) ?

Discriminative models tend to do better in classification
[Ng and Jordan (2002)]
Example generative model: GMM

\[ p(\theta) = \sum_{i=1}^{K} \phi_i N(\mu_i, \Sigma_i) \]

We can efficiently draw samples from this model.
Boltzmann Machine

\[ P(x) = \frac{\exp(-E(x))}{Z}, \quad x \in \{0, 1\}^d \]

\[ E(x) = -x^\top U x - b^\top x, \]

- Energy based model
- \( Z \): partition function that ensures \( P(x) \) is a proper probability distribution.
- Fully connected

(See Section 20.1 [of the book, Goodfellow et al. (2016)] for references.)
Boltzmann Machine

• Becomes more powerful when some of $x$ are hidden variables.
  (a universal approximator of probability mass functions over discrete variables [Le Roux and Bengio (2008)])

• Decompose $x$ into $v$ and $h$. $v$: visible units, $h$: hidden units.
  Energy function becomes:

$$E(v, h) = -v^\top Rv - v^\top Wh - h^\top Sh - b^\top v - c^\top h.$$
Boltzmann Machine

- Training is based on maximum-likelihood.
- But, the partition function $Z$ is intractable.
  - The maximum-likelihood gradient must be approximated (see Section 18 for methods).
- Can use simulated annealing to train.
- Boltzmann machines (BM) with unconstrained connectivity did not prove useful in practical problems.
  - Restricted connectivity $\rightarrow$ Restricted Boltzmann Machines
Restricted Boltzmann Machines

Compare:

BM

RBM

[Smolensky (1986)]

Blue: visible units, white: hidden units.
Restricted Boltzmann Machines

Energy based model like BM:

\[
P(v = v, h = h) = \frac{1}{Z} \exp \left( -E(v, h) \right)
\]

\[
E(v, h) = -b^T v - c^T h - v^T Wh
\]

\(v\): visible units, \(h\): hidden units

Partition function:

\[
Z = \sum_v \sum_h \exp \left\{ -E(v, h) \right\}
\]
Restricted Boltzmann Machines

Energy based model like BM:

\[
P(v = v, h = h) = \frac{1}{Z} \exp \left( -E(v, h) \right)
\]

\[
E(v, h) = -b^\top v - c^\top h - v^\top W h
\]

Compare this with BM's energy function:

\[
E(v, h) = -v^\top R v - v^\top W h - h^\top S h - b^\top v - c^\top h.
\]
Restricted Boltzmann Machines

Energy based model like BM:

\[ P(v = v, h = h) = \frac{1}{Z} \exp (-E(v, h)) \]

\[ E(v, h) = -b^\top v - c^\top h - v^\top W h \]

\( v \): visible units, \( h \): hidden units

Partition function:

\[ Z = \sum_v \sum_h \exp \{-E(v, h)\} \]

Long and Servedio (2010) formally proved that \( Z \) is intractable. The intractable \( Z \) implies that the normalized joint probability distribution \( P(v) \) is also intractable to evaluate.
Restricted Boltzmann Machines

Although $p(v)$ is intractable, conditional distributions are not:

$$P(h \mid v) = \frac{1}{Z'} \prod_{j=1}^{n_h} \exp \left\{ c_j h_j + v^\top W_{:,j} h_j \right\}$$

[derivation on board]

$$P(h_j = 1 \mid v) = \sigma \left( c_j + v^\top W_{:,j} \right) \quad \sigma \text{ is the sigmoid function.}$$

$$P(v_i = 1 \mid h) \text{ is similar.}$$
\[ p(h_j = 1 \mid v) = ? \]

\[ p(h \mid v) = \frac{p(v, h)}{p(v)} = \frac{1}{Z \cdot p(v)} \cdot \exp(-E(v, h)) \]

\[ = \frac{1}{Z \cdot p(v)} \cdot \exp(b^T u + c^T h + v^T W h) \]

\[ = \frac{\exp(b^T u)}{Z \cdot p(v)} \cdot \exp\left(\sum_{i=1}^{n_h} c_i h_i + \sum_{i=1}^{n_h} h_i (v^T W)_i\right) \]

[Equation]
\[ p(h_i = 1 | v) = \frac{p(h_i = 1 | v)}{p(h_i = 1 | v) + p(h_i = 0 | v)} \]

\[ = \frac{\exp(c_i + (v^T W)_i)}{\exp(c_i + (v^T W)_i) + \exp(\varnothing)} \left( \frac{\exp(-(c_i + (v^T W)_i))}{\exp(-(c_i + (v^T W)_i))} \right) \]

\[ = \frac{1}{1 + \exp(-(c_i + (v^T W)_i))} \]

\[ = \sigma(c_i + (v^T W)_i) \text{ (sigmoid)} \]
Restricted Boltzmann Machines

Given a set of observations \( \{ v_n \}_{n=1}^{N} \) and for

\[
E(v, h; \theta) = -v^T W h - b^T v - a^T h
\]

Derivatives of the log-likelihood function are:

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}} [v_i b_j] - \mathbb{E}_{P_{\text{model}}} [v_i b_j],
\]

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial a_j} = \mathbb{E}_{P_{\text{data}}} [b_j] - \mathbb{E}_{P_{\text{model}}} [b_j],
\]

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial b_i} = \mathbb{E}_{P_{\text{data}}} [v_i] - \mathbb{E}_{P_{\text{model}}} [v_i],
\]

(Equations from Salakhutdinov (2015))
Given $D = \{ v_n \}_{n=1}^N$

Likelihood function? $P(D | \text{model}) = \prod_{i=1}^N p(v_i | \text{model})$

Log-likelihood $\Rightarrow \sum_{i=1}^N \log p(v_i | \text{model})$

$\frac{\partial \log p(v)}{\partial \mathbf{W}} = ?$

we have $p(v, h) = \frac{1}{Z} \exp(-E(v, h))$

$Z = \sum_{v, h} \exp(-E(v, h))$
$E(v, h) = -v^T \mathbf{W} h - b^T v - \alpha^T \mathbf{h}$

model parameters are $\mathbf{W}, \mathbf{b}, \alpha$

$p(v) = \sum_h p(v, h) = \frac{1}{Z} \sum_h \exp(-E(v, h))$

$\log p(v) = -\log(Z) + \log \left( \sum_h \exp(-E(v, h)) \right)$
\[ \frac{\partial \log p(v)}{\partial w} = \frac{-1}{z} \frac{\partial z}{\partial w} + \frac{1}{\sum_h \exp(-E(v,h))} \sum_h \exp(-E(v,h)) \cdot v^h \]

\[ = \sum_h v^h \frac{\exp(-E(v,h))}{\sum_k \exp(-E(v,k))} - \frac{1}{z} \sum_{v,h} \exp(-E(v,h)) (v^h) \]

\[ p(v|h) = \frac{p(v,h)}{\rho(v)} \]

\[ = E_{p(h|v)} [v^h] - E_{p(v,h)} [v^h] \]
Restricted Boltzmann Machines

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}} [v_i b_j] - \mathbb{E}_{P_{\text{model}}} [v_i b_j],
\]

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial a_j} = \mathbb{E}_{P_{\text{data}}} [b_j] - \mathbb{E}_{P_{\text{model}}} [b_j],
\]

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial b_i} = \mathbb{E}_{P_{\text{data}}} [v_i] - \mathbb{E}_{P_{\text{model}}} [v_i],
\]

\[
\mathbb{E}_{\text{data}} [x] \equiv \mathbb{E}_{p(h|v)} [x] \quad \quad \mathbb{E}_{\text{model}} [x] \equiv \mathbb{E}_{p(v,h)} [x]
\]

(Equations from Salakhutdinov (2015))
Restricted Boltzmann Machines

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}}[v_i b_j] - \mathbb{E}_{P_{\text{model}}}[v_i b_j],
\]

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial a_j} = \mathbb{E}_{P_{\text{data}}}[b_j] - \mathbb{E}_{P_{\text{model}}}[b_j],
\]

\[
\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(v_n; \theta)}{\partial b_i} = \mathbb{E}_{P_{\text{data}}}[v_i] - \mathbb{E}_{P_{\text{model}}}[v_i],
\]

\[E_{\text{data}}[x] \equiv \mathbb{E}_{P(h|v)}[x]\]

\[E_{\text{model}}[x] \equiv \mathbb{E}_{P(v,h)}[x]\]

Easy to compute (see our previous derivations)

Intractable (still has the Z in it)

(Equations from Salakhutdinov (2015))
Contrastive Divergence

Algorithm to train RBMs. By Hinton (2002).

\[ E_{\text{model}}[x] \equiv E_{p(v,h)}[x] \]

is approximated using Gibbs sampling with finite steps starting from the current data \((x)\).

The gradient is

\[ \Delta W = \alpha (E_{p_{\text{data}}}[vh^T] - E_{p_T}[vh^T]), \]

where \(\alpha\) is the learning rate and \(P_T\) represents a distribution defined by running a Gibbs chain initialized at the data for \(T\) full steps.
Gibbs sampling

The goal is to draw a sample from a multivariate probability distribution.

Idea: sample one variable at a time using the conditional distribution.

\[
p(x_j | x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) = \frac{p(x_1, \ldots, x_n)}{p(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)} \propto p(x_1, \ldots, x_n)
\]

Remember for RBMs conditionals are tractable (they are sigmoids).
So far we have seen binary valued RBMs

How do we model real-valued vectors, such as pixel intensities in an image?

**Answer: Gaussian-Bernoulli RBM** [Hinton & Salakhutdinov (2006)]

When modeling real-valued vectors, such as pixel intensities of image patches, one can easily extend RBMs to the Gaussian–Bernoulli variant (Hinton & Salakhutdinov 2006). In particular, consider modeling visible real-valued variables $v \in \mathbb{R}^D$, and let $h \in \{0, 1\}^F$ be stochastic binary hidden variables. The energy of the joint state $\{v, h\}$ of the Gaussian RBM is defined as follows:

$$E(v, h; \theta) = \sum_{i=1}^{D} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{D} \sum_{j=1}^{F} W_{ij} b_j \frac{v_i}{\sigma_i} - \sum_{j=1}^{F} a_j b_j,$$

where $\theta = \{W, a, b, \sigma^2\}$ are the model parameters.

[Screenshot from Salakhutdinov (2015)]
RBM for real-valued input

Similar to the binary case, conditional distributions are easy to evaluate:

\[ p(b_j = 1|v) = g \left( b_j + \sum_i W_{ij} \frac{v_i}{\sigma_i} \right) \]

where \( g(\cdot) \) is the sigmoid function.

\[
\frac{\partial \log P(v; \theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}} \left[ \frac{1}{\sigma_i} v_i b_j \right] - \mathbb{E}_{P_{\text{model}}} \left[ \frac{1}{\sigma_i} v_i b_j \right]
\]
RBM for real-valued input

Hidden nodes act like localized cell receptive fields. See the visualizations:

**Figure 2**
A random subset of the training images along with the learned receptive fields. (a) The binary restricted Boltzmann machine (RBM) trained on the Handwritten Characters data set (resolution is $28 \times 28$). (b) The Gaussian–Bernoulli RBM trained on the CIFAR-100 data set (resolution is $32 \times 32$). Each square displays the incoming weights from all of the visible variables into one hidden unit.

(Figure from Salakhutdinov (2015))
There is another extension of RBM:

Replicated softmax model

Used for modeling **sparse count data**, such as word count vectors in a document.

(See Salakhutdinov (2015) for further references).
A RBM = a single layer of hidden units

Not the best way to capture non-linear dependencies in the data.

How do we increase the model capacity?
A RBM = a single layer of hidden units

Not the best way to capture non-linear dependencies in the data.

How do we increase the model capacity?

By going deeper.
Deep Belief Networks

Composed of stacking Restricted Boltzmann Machines on top of each other.

\textbf{Figure 4}

(a) A restricted Boltzmann machine (RBM). (b) A two-hidden-layer deep belief network (DBN) with tied weights $W^{(2)} = W^{(1)^T}$. The joint distribution $P(v, h^{(1)}; W^{(1)})$ defined by this DBN is identical to the joint distribution $P(v, h^{(1)}; W^{(1)})$ defined by an RBM.

(Figure from Salakhutdinov (2015))
Deep Belief Networks

DBNs are probabilistic generative models that contain many layers of hidden variables.

- Each layer captures high-order correlations between the activities of hidden features in the layer below.
- The top two layers of the DBN form an RBM model in which the lower layers form a directed sigmoid belief network.

(Figures in DBN slides are from Salakhutdinov (2015))
Deep Belief Networks

Hinton et al. (2006) introduced a fast unsupervised learning algorithm for training DBNs.

A key feature of this algorithm is its greedy layer-by-layer training, which can be repeated several times to learn a deep hierarchical model.
Deep Belief Networks

Greedy layer-by-layer training.

First, train a RBM (using Contrastive Divergence method)
Deep Belief Networks

Next, add a layer on top. This new layer has the same number of nodes as the visible units layer (v).

And, set the weights of this new layer to $W^{(1)}$. The reason for this: to ensure that the two-hidden-layer DBN is as good as the original RBM.
Deep Belief Networks

Initially, set $W^{(2)}$ to $W^{(1)^T}$

\[ P(v, h^{(1)}, h^{(2)}; \theta) = P(v|h^{(1)}; W^{(1)}) P(h^{(1)}, h^{(2)}; W^{(2)}), \]

\[ P(v|h^{(1)}; W^{(1)}) = \prod_i p(v_i|h^{(1)}; W^{(1)}), \quad p(v_i = 1|h^{(1)}; W^{(1)}) = g \left( \sum_j W^{(1)}_{ij} b^{(1)}_j \right) \]

\[ P(h^{(1)}, h^{(2)}; W^{(2)}) = \frac{1}{\mathcal{Z}(W^{(2)})} \exp(h^{(1)^T} W^{(2)} h^{(2)}), \]
Deep Belief Networks

When \( W^{(2)} = W^{(1)T} \)

The joint distribution for this DBN

\[
P(v, h^{(1)}, h^{(2)}; \theta) = P(v|h^{(1)}; W^{(1)})P(h^{(1)}, h^{(2)}; W^{(2)}),
\]

Is equivalent to the joint distribution of the RBM:

\[
P(v, h^{(1)}; W^{(1)})
\]

[derivation on board]
\[ p(v, h^{(1)}; \theta) = \sum_{h^{(2)}} p(v, h^{(1)}, h^{(2)}; \theta) \]

\[ = \sum_{h^{(2)}} p(v/h^{(1)}; w^{(1)}) \sum_{h^{(2)}} p(h^{(1)}, h^{(2)}; w^{(1)}) \]

\[ = p(v/h^{(1)}; w^{(1)}) \sum_{h^{(2)}} p(h^{(1)}, h^{(2)}; w^{(1)}) \]

\[ = p(v/h^{(1)}; w^{(1)}) p(h^{(1)}; w^{(1)}) \]

\[ = p(v, h^{(1)}; w^{(1)}) \]
Deep Belief Networks

The greedy learning algorithm uses a stack of RBMs (on the right).

1) Train the bottom RBM with parameters $W^{(1)}$ as described before.
2) Initialize the second layer weights to $W^{(2)} = W^{(1)}$, to ensure that the two-hidden-layer DBN is at least as good as our original RBM.
3) We can now improve the fit of the DBN to the training data by untying and refitting parameters $W^{(2)}$. 
Deep Belief Networks

Algorithm 1 (Recursive greedy learning procedure for the deep belief network):

1: Fit the parameters $W^{(1)}$ of the first-layer RBM to data.
2: Fix the parameter vector $W^{(1)}$, and use samples $h^{(1)}$ from $Q(h^{(1)}|v) = P(h^{(1)}|v, W^{(1)})$ as the data for training the next layer of binary features with an RBM.
3: Fix the parameters $W^{(2)}$ that define the second layer of features, and use the samples $h^{(2)}$ from $Q(h^{(2)}|h^{(1)}) = P(h^{(2)}|h^{(1)}, W^{(2)})$ as the data for training the third layer of binary features.
4: Proceed recursively for the next layers.

(Screenshot from Salakhutdinov (2015))
DBNs and its variants (and neural networks in general) are able to learn distributed representations.
Sparse representation

Distributed representation
A previously unseen shape can be expressed in the distributed representation space:

\[ \approx \text{Vertical} + \text{Horizontal} + \text{Ellipse} = \bullet \bullet \bullet \circ \bullet \bullet \]
Generative Adversarial Networks
Generative Adversarial Networks

[Goodfellow et al. (2014)]

Training dataset

$x$

Discriminator Neural Network $D$

Decide if the input came from the training data.

Noise

Generator Neural Network $G$

$g$
Generative Adversarial Networks

[Goodfellow at el. (2014)]

The training procedure:
- For $D$ is to minimize the probability of a wrong decision,
- For $G$ is to maximize the probability of $D$ making a mistake.

A minimax two-player game

No need for any Markov chains or unrolled approximate inference networks.

[Diagram of Generative Adversarial Networks]
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:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left(x^{(i)}\right) + \log \left(1 - D \left(G \left(z^{(i)}\right)\right)\right) \right].
      \n      \]
  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.

[From Goodfellow at el. (2014)]
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator $D$ is the parameter $k$. We used $k = 1$, the least expensive option, in our experiments.

for number of training steps do
  for $k$ steps do
    • Sample minibatch of $m$ data samples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • 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.

[From Goodfellow at el. (2014)]
Training of D:
minimize the probability of a wrong decision

Training of G:
maximize the probability of D making a mistake
Generative Adversarial Networks

a)

b)

c)

d)
Generative Adversarial Networks
[Goodfellow et al. (2014)]

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and “deconvolutional” generator)
Improvements on GANs

- Deep Convolutional Generative Adversarial Networks (DCGANs) [Radford et al. (2015)]

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.
Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.
DCGAN results [Radford et al. (2015)]
GANs received a lot of attention and there has been a huge number of different GAN proposals for different problems. Only a few examples:
Figure 1. Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation

Yunjey Choi\textsuperscript{1,2} Minje Choi\textsuperscript{1,2} Munyoung Kim\textsuperscript{2,3} Jung-Woo Ha\textsuperscript{2} Sunghun Kim\textsuperscript{2,4} Jaegul Choo\textsuperscript{1,2}

\textsuperscript{1} Korea University \quad \textsuperscript{2} Clova AI Research, NAVER
\textsuperscript{3} The College of New Jersey \quad \textsuperscript{4} Hong Kong University of Science & Technology
Figure 1: Given any two unordered image collections $X$ and $Y$, our algorithm learns to automatically "translate" an image from one into the other and vice versa: (left) Monet paintings and landscape photos from Flickr; (center) zebras and horses from ImageNet; (right) summer and winter Yosemite photos from Flickr. Example application (bottom): using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

**Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks**

Jun-Yan Zhu*  Taesung Park*  Phillip Isola  Alexei A. Efros

Berkeley AI Research (BAIR) laboratory, UC Berkeley
Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros

(Submitted on 30 Mar 2017 (v1), last revised 24 Nov 2017 (this version, v3))

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. We present an approach for learning to translate an image from a source domain $X$ to a target domain $Y$ in the absence of paired examples. Our goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution $Y$ using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss to push $F(G(X)) \approx X$ (and vice versa). Qualitative results are presented on several tasks where paired training data does not exist, including collection style transfer, object transfiguration, season transfer, photo enhancement, etc. Quantitative comparisons against several prior methods demonstrate the superiority of our approach.
Unsupervised Machine Translation Using Monolingual Corpora Only

Guillaume Lample, Ludovic Denoyer, Marc'Aurelio Ranzato

(Submitted on 31 Oct 2017)

Machine translation has recently achieved impressive performance thanks to recent advances in deep learning and the availability of large-scale parallel corpora. There have been numerous attempts to extend these successes to low-resource language pairs, yet requiring tens of thousands of parallel sentences. In this work, we take this research direction to the extreme and investigate whether it is possible to learn to translate even without any parallel data. We propose a model that takes sentences from monolingual corpora in two different languages and maps them into the same latent space. By learning to reconstruct in both languages from this shared feature space, the model effectively learns to translate without using any labeled data. We demonstrate our model on two widely used datasets and two language pairs, reporting BLEU scores up to 32.8, without using even a single parallel sentence at training time.

This is not a GAN work but it is similar to the work in the previous slide.
And, finally, a mild warning: if you see a new GAN model, take it with a grain of salt.

Are GANs Created Equal? A Large-Scale Study

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, Olivier Bousquet

(Submitted on 28 Nov 2017 (v1), last revised 30 Nov 2017 (this version, v2))

Generative adversarial networks (GAN) are a powerful subclass of generative models. Despite a very rich research activity leading to numerous interesting GAN algorithms, it is still very hard to assess which algorithm(s) perform better than others. We conduct a neutral, multi-faceted large-scale empirical study on state-of-the-art models and evaluation measures. We find that most models can reach similar scores with enough hyperparameter optimization and random restarts. This suggests that improvements can arise from a higher computational budget and tuning more than fundamental algorithmic changes. To overcome some limitations of the current metrics, we also propose several data sets on which precision and recall can be computed. Our experimental results suggest that future GAN research should be based on more systematic and objective evaluation procedures. Finally, we did not find evidence that any of the tested algorithms consistently outperforms the original one.
Autoencoders
Autoencoders
Autoencoders

Encoder network

Input $x$

Representation $h$

Decoder network

$\hat{x}$
Autoencoders

Encoder network

Input $x$

Representation $h$

Decoder network

$\hat{x}$ : reconstructed $x$

Minimize $\|x - \hat{x}\|^2_2$
Autoencoders

Encoder network

Input $x$

Representation $h$

Decoder network

$\hat{x}$

Minimize $\|x - \hat{x}\|_2^2$
Autoencoders

- Typically used for feature extraction. (“h” are features)
- Encoder and decoders networks could be MLPs or CNNs with ReLUs.
- After training Decoder Network is thrown out.

Minimize $\|x - \hat{x}\|^2_2$
Autoencoders

- Could be used to initialize a supervised model.
- Depending on the size of $x$ and $h \rightarrow$ autoencoder is “undercomplete” or “overcomplete”

Minimize $\|x - \hat{x}\|^2_2$
Autoencoders

• Undercomplete autoencoders capture the most salient features of training data.
• Overcomplete autoencoders could easily “memorize” the training data.
  – Therefore, they are usually regularized.
  – e.g. enforcing sparsity on $h \rightarrow$ Sparse Autoencoder
Denoising Autoencoder

Autoencoders minimize \( L(x, D(E(x))) \)

Denoising Autoencoders minimize \( L(x, D(E(\tilde{x}))) \)

where \( \tilde{x} \) is a copy of \( x \) that has been corrupted by some form of noise.

By corrupting \( x \), the model is prevented from learning \( E(D(\tilde{x})) \) as identity.
**Variational Autoencoder**

[Kingma and Welling (2013)]

**Classical Autoencoder:**

- **Encoder network**
- **Decoder network**
- **Representation** $h$
- **Input** $x$
- **Output** $\hat{x}$

**Variational Autoencoder:**

- **Encoder network**
- **Decoder network**
- **Draw sample** $z$
- **$p_\theta(z|x)$**
- **$p_\phi(x|z)$**
- **Input** $x$
- **Output** $\hat{x}$
Variational Autoencoder

[Kingma and Welling (2013)]

\[ p_{\theta}(z|x) \]  
\[ \text{Encoder network} \]

\[ p_{\phi}(x|z) \]  
\[ \text{Decoder network} \]

\[ x \]  
\[ \hat{x} \]  
\[ Z \]

Draw sample
Variational Autoencoder

[Kingma and Welling (2013)]

\[
p_\theta(z|x) = \frac{p_\phi(x|z) p_\phi(z)}{p_\theta(x)}
\]
**Variational Autoencoder**

[Kingma and Welling (2013)]

$p_{\theta}(z|x) \rightarrow \text{Decoder network} \rightarrow p_{\phi}(x|z) \rightarrow \hat{x}$

Learned by encoder

$p_{\theta}(z|x) = \frac{p_{\phi}(x|z)p_{\phi}(z)}{p_{\theta}(x)}$

Learned by decoder
Variational Autoencoder
[Kingma and Welling (2013)]

$\mathbb{R}^n \rightarrow \mathbb{R}^m \rightarrow \mathbb{R}^n$

**Encoder network**

$p_\theta(z|x)$

**Decoder network**

$p_\phi(x|z)$

**Learned by encoder**

$\mathbb{R}^m \rightarrow \mathbb{R}^n$

$p_\theta(x)$

**Learned by decoder**

$p_\phi(z)$

**Gaussian**

$p_\theta(z|x) = \frac{p_\phi(x|z)}{p_\phi(z)}$

**Intractable integral (not used)**

$x \rightarrow z \rightarrow \hat{x}$

Draw sample $z \sim p_\theta(z|x)$

Draw sample $\hat{x} \sim p_\phi(x|z)$
Variational Autoencoder

[Kingma and Welling (2013)]

(a) Learned Frey Face manifold

(b) Learned MNIST manifold
Variational Autoencoder

[Kingma and Welling (2013)]

Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables $z$. For each of these values $z$, we plotted the corresponding generative $p_{\theta}(x|z)$ with the learned parameters $\theta$. 
Figure 7: Samples from a VAE trained on MNIST.
Summary

- Boltzmann Machines
- Restricted Boltzmann Machines
  - Contrastive Divergence
- Deep Belief Networks
- Generative Adversarial Networks
- Autoencoders
- Variational Autoencoders
References


References


