A Hybrid: Deep Learning and Graphical Models

Kayhan Batmanghelich
Outlines

• Joint deep feature extraction and learning

• Variational Auto Encoder (VAE)

• Generative Adversarial Networks (GANs)
Motivation:
Hybrid Models

**Graphical models** let you encode domain knowledge

**Neural nets** are really good at fitting the data discriminatively to make good predictions

Could we define a neural net that incorporates domain knowledge?
Motivation: Hybrid Models

**Key idea:** Use a NN to learn features for a GM, then train the entire model by backprop.
HYBRID:
NEURAL NETWORK + HMM
Markov Random Field (MRF)

Joint distribution over tags $Y_i$ and words $X_i$

The individual factors aren’t necessarily probabilities.

\[ p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots) \]
Hidden Markov Model

But sometimes we choose to make them probabilities. Constrain each row of a factor to sum to one. Now $Z = 1$.

\[
p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{2} (.3 \times .8 \times .2 \times .5 \times ...)
\]
Hybrid: NN + HMM

Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

Continuous HMM emission: $Y_t \in \mathcal{R}^K$

HMM: $p(Y, S) = \prod_{t=1}^{T} p(Y_t|S_t)p(S_t|S_{t-1})$

Gaussian emission:

$$p(Y_t|S_t = i) = b_{i,t} = \sum_k \frac{Z_k}{(2\pi)^{n/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(Y_t - \mu_k)^T \Sigma_k^{-1} (Y_t - \mu_k)\right)$$

What if we jointly learn the features?
Hybrid: NN + HMM

Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /q/\}$

Continuous HMM emission: $Y_t \in \mathcal{R}^K$

HMM: $p(Y, S) = \prod_{t=1}^{T} p(Y_t | S_t)p(S_t | S_{t-1})$

Lots of oddities to this picture:
• **Clashing visual notations** (graphical model vs. neural net)
• HMM generates data **top-down**, NN generates **bottom-up** and they meet in the middle.
• The “observations” of the HMM are **not actually observed** (i.e. x’s appear in NN only)

So what are we missing?

(Bengio et al., 1992)
Hybrid: NN + HMM
$a_{i,j} = p(S_t = i | S_{t-1} = j)$

$\beta_{i,t} = p(Y_{t+1} | S_t = i \text{ and model}) = \sum_j a_{ij} \beta_{j,t+1}$

$\gamma_{i,t} = p(S_t = i | Y_1^t \text{ and model}) = \alpha_{i,t} \beta_{i,t}$

**Forward-backward algorithm:** A “feed-forward” algorithm for computing alpha-beta probabilities.

**Log-likelihood:** A “feed-forward” objective function.

$$\log p(S, Y) = \alpha_{\text{END}, T}$$
1. Given training data: \( \{ x_i, y_i \}_{i=1}^{N} \)

2. Choose each of these:
   - Decision function
   - Loss function

\[ \hat{y} = f_\theta(x_i) \]

\[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:

   - Train with SGD: (take small steps opposite the gradient)
   - Log-likelihood: a “feed-forward” objective function.

   \[ \log p(S, Y) = \alpha_{END,T} \]

   \[ -\eta_t \nabla \ell(f_\theta(x_i), y_i) \]


   \[ \alpha_{i,t} = P(Y_1^t \text{ and } S_t = i | \text{model}) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1} \]

   \[ \beta_{i,t} = P(Y_{t+1}^T | S_t = i \text{ and model}) = \sum_j a_{ij} b_{j,t+1} \beta_{j,t+1} \]

   \[ \gamma_{i,t} = P(S_t = i | Y_1^t \text{ and model}) = \alpha_{i,t} \beta_{i,t} \]
Backpropagation is just repeated application of the chain rule from Calculus 101.

The topologies presented in this section are very simple. However, we will later (Chap-

2.2.2 Backpropagation

How to compute these partial derivatives?

Chain Rule:

\[
\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \forall i, k
\]

Recall...
2.2. NEURAL NETWORKS AND BACKPROPAGATION

The material presented here acts as a supplement to later uses of backpropagation such as in Chapter 4 for training of a hybrid graphical model / neural network, and in Chapter 5 and Chapter 6 for approximation-aware training.

2.2.1 Topologies

A feed-forward neural network (Rumelhart et al., 1986) defines a decision function $y = h(x)$ where $x$ is termed the input layer and $y$ the output layer. A feed-forward neural network has a statically defined topology. Figure 2.1 shows a simple 2-layer neural network consisting of an input layer $x$, a hidden layer $z$, and an output layer $y$. In this example, the output layer is of length 1 (i.e. just a single scalar $y$). The model parameters of the neural network are a matrix $\beta$ and a vector $\alpha$.

The feed-forward computation proceeds as follows: we are given $x$ as input (Fig. 2.1(A)). Next, we compute an intermediate vector $a$, each entry of which is a linear combination of the input (Fig. 2.1(B)). We then apply the sigmoid function $a(z) = \frac{1}{1+\exp(b)}$ element-wise to obtain $z$ (Fig. 2.1(C)). The output layer is computed in a similar fashion, first taking a linear combination of the hidden layer to compute $b$ (Fig. 2.1(D)) then applying the sigmoid function to obtain the output $y$ (Fig. 2.1(E)). Finally we compute the loss $J$ (Fig. 2.1(F)) as the squared distance to the true value $y^{(d)}$ from the training data.

We refer to this topology as an arithmetic circuit. It defines both a function mapping...
2.2. NEURAL NETWORKS AND BACKPROPAGATION

The backward pass computes $\frac{dJ}{d✓j}$.

Forward Backward

$J = y^* \log q + (1 - y^*) \log(1 - q)$

$q = \frac{1}{1 + \exp(-b)}$

$b = \sum_{j=0}^{D} β_j z_j$

$z_j = \frac{1}{1 + \exp(-a_j)}$

$a_j = \sum_{i=0}^{M} α_{ji} x_i$

$\frac{dJ}{dq} = \frac{y^*}{q} + \frac{(1 - y^*)}{q - 1}$

$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db} \frac{db}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2}$

$\frac{dJ}{dβ_j} = \frac{dJ}{db} \frac{db}{dβ_j} \frac{db}{db} = z_j$

$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j} \frac{db}{dz_j} = β_j$

$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j} \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2}$

$\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i} \frac{da_j}{dx_i} = \sum_{j=0}^{D} α_{ji}$

Recall…

Notice that this application of backpropagation computes both the derivatives with respect to each model parameter $\frac{dJ}{d✓ji}$ and $\frac{dJ}{d✓j}$, but also the partial derivatives with respect to each intermediate quantity $\frac{dJ}{da_j}$, $\frac{dJ}{dz_j}$, $\frac{dJ}{db}$, $\frac{dJ}{dy}$ and the input $\frac{dJ}{dx_i}$. 

Backpropagation
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

Forward computation

$$\log p(S, Y) = \alpha_{END, T}$$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

$a_{i,j} = \ldots$ (transitions)

$b_{i,t} = \ldots$ (emissions)

$$y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$
# Hybrid: NN + HMM

**Computing the Gradient:** \( \nabla \ell(f_\theta(x_i), y_i) \)

**Forward computation**

\[
J = \log p(S, Y) = \alpha_{END,T}
\]

- \( \alpha_{i,t} = \ldots \) (forward prob)
- \( \beta_{i,t} = \ldots \) (backward prop)
- \( \gamma_{i,t} = \ldots \) (marginals)
- \( a_{i,j} = \ldots \) (transitions)
- \( b_{i,t} = \ldots \) (emissions)

\[
Y_{tk} = \frac{1}{1 + \exp(-b)}
\]

\[
b = \sum_{j=0}^{D} \beta_{j}z_{j}
\]

\[
z_{j} = \frac{1}{1 + \exp(-a_{j})}
\]

\[
a_{j} = \sum_{i=0}^{M} \alpha_{ji}x_{i}
\]
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

**Forward computation**

$$J = \log p(S, Y) = \alpha_{END,T}$$

$\alpha_{i,t} = \ldots$ (forward prob)
$\beta_{i,t} = \ldots$ (backward prop)
$\gamma_{i,t} = \ldots$ (marginals)
$a_{i,j} = \ldots$ (transitions)
$b_{i,t} = \ldots$ (emissions)

$$Y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

**Backward computation**

$$\frac{dJ}{db_{i,t}} = \frac{\partial \alpha_{model,T}}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} = (\sum_j \frac{\partial \alpha_{j,t+1}}{\partial \alpha_{i,t}} \frac{\partial L_{model}}{\partial \alpha_{j,t+1}})(\sum_j a_{ji} \alpha_{j,t-1})$$

$$= (\sum_j b_{j,t+1} a_{ji} \frac{\partial \alpha_{model,T}}{\partial \alpha_{j,t+1}})(\sum_j a_{ji} \alpha_{j,t-1}) = \beta_{i,t} \alpha_{i,t} b_{i,t} = \gamma_{i,t}$$
Hybrid: NN + HMM

Computing the Gradient: \( \nabla \ell(f_\theta(x_i), y_i) \)

**Forward computation**

\[
J = \log p(S, Y) = \alpha_{\text{END}, T} \\
\alpha_{i,t} = \ldots \text{(forward prob)} \\
\beta_{i,t} = \ldots \text{(backward prop)} \\
\gamma_{i,t} = \ldots \text{(marginals)} \\
a_{i,j} = \ldots \text{(transitions)} \\
b_{i,t} = \ldots \text{(emissions)} \\
Y_{t,k} = \frac{1}{1 + \exp(-b)} \\
b = \sum_{j=0}^{D} \beta_j z_j \\
z_j = \frac{1}{1 + \exp(-a_j)} \\
a_j = \sum_{i=0}^{M} \alpha_{ji} x_i
\]

**Backward computation**

\[
\frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}} \\
\frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}} \\
\frac{\partial b_{i,t}}{\partial Y_{j,t}} = \sum_{k} \frac{Z_k}{((2\pi)^n |\Sigma_k|)^{1/2}} \left( \sum_{i} d_{k,ij}(\mu_{kl} - Y_{it}) \right) \exp(-\frac{1}{2}(Y_{t} - \mu_k)\Sigma_k^{-1}(Y_{t} - \mu_k)^T) \\
\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}' \frac{dy}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2} \\
\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}' \frac{db}{d\beta_j} = z_j \\
\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}' \frac{db}{dz_j} = \beta_j \\
\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}' \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2} \\
\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}' \frac{da_j}{d\alpha_{ji}} = x_i
\]
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

**Forward computation**

$J = \log p(S, Y) = \alpha_{END,T}$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

The derivative of the log-likelihood with respect to the neural network parameters!

$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$

**Backward computation**

$\frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$

$\frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}}$

$\frac{\partial b_{i,t}}{\partial y_{j,t}} = \sum_{k} \frac{Z_k}{((2\pi)^n |\Sigma_k|)^{1/2}}(\sum_i d_{k,ij}(\mu_{kl} - Y_{it})) \exp\left(-\frac{1}{2}(Y_t - \mu_k)^{-1}(Y_t - \mu_k)^T\right)$

$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}$

$\frac{dJ}{dy} = \frac{dJ}{db} \frac{db}{dy}$

$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}$

$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}$

$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}$
Hybrid: NN + HMM

Experimental Setup:

- **Task:** Phoneme Recognition (aka. speaker independent recognition of plosive sounds)
- **Eight output labels:**
  - /p/, /t/, /k/, /b/, /d/, /g/, /dx/, /all other phonemes/
  - These are the HMM hidden states
- **Metric:** Accuracy
- **3 Models:**
  1. NN only
  2. NN + HMM (trained independently)
  3. NN + HMM (jointly trained)

(Bengio et al., 1992)
HYBRID: CNN + CRF
Markov Random Field (MRF)

Joint distribution over tags $Y_i$ and words $X_i$

\[
p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z} \left( 4 \times 8 \times 5 \times 3 \times \ldots \right)
\]
Conditional Random Field (CRF)

Conditional distribution over tags $Y_i$ given words $x_i$.
The factors and $Z$ are now specific to the sentence $x$.

\[
p(n, v, p, d, n \mid \text{time, flies, like, an, arrow}) \quad \frac{1}{Z} \quad (4 \times 8 \times 5 \times 3 \times ...)
\]
In a standard CRF, each of the factor cells is a parameter (e.g. transition or emission)
In the hybrid model, these values are computed by a neural network with its own parameters
Hybrid: Neural Net + CRF

Forward computation
Hybrid: CNN + MRF

Experimental Setup:
- **Task:** pose estimation
- **Model:** Deep CNN + MRF

Thompson et al., “Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation” 2014
VAE: VARIATIONAL AUTO-ENCODER
Recap: Interpreting the Lower Bound (ELBO)

\[ \mathcal{F}(y, q) = \mathbb{E}_{q(z)} \left[ \log p(y|z) \right] - KL \left[ q(z) \| p(z) \right] \]

Data  
Approximate Posterior

Reconstruction

Penalty

**Approximate posterior distribution** \( q(z) \): Best match to true posterior \( p(z|y) \), one of the unknown inferential quantities of interest to us.

**Reconstruction Cost**: The expected log-likelihood measure how well samples from \( q(z) \) are able to explain the data \( y \).

**Penalty**: Ensures the explanation of the data \( q(z) \) doesn’t deviate too far from your beliefs \( p(z) \). A mechanism for realising Okham’s razor.
Recap: Decoder Encoder View

\[
\max_{\phi, \theta} \mathcal{F}(x, q_{\phi}) = \mathbb{E}_{q_{\phi}(z)} \left[ \log p_{\theta}(x | z) \right] - KL [q_{\phi}(z) \| p(z)]
\]

**Encoder:** variational distribution \( q_{\phi}(z | y) \)

**Decoder:** likelihood \( p_{\theta}(y | z) \)

The goal: \( q_{\phi}(z | x) \approx p_{\theta}(z | x) \)

Data code-length

Hypothesis code

Stochastic encoder

\( x \sim p_{\theta}(x | z) \)

Data x
\[
\max_{\phi, \theta} \mathcal{F}(x, q_\phi) = \mathbb{E}_{q_\phi(z)} \left[ \log p_{\theta}(x|z) \right] - KL[q_\phi(z) \| p(z)]
\]

How to implement it?
What is q exactly?
Variational Auto Encoder (VAE)

\[
\max_{\phi, \theta} F(x, q_\phi) = \mathbb{E}_{q_\phi(z)} [\log p_\theta(x | z)] - KL [q_\phi(z) || p(z)]
\]

Stochastic encoder

Data code-length

Hypothesis code

\[
q_\phi(z | x) = q_\phi(z_1, ..., z_M | x) = \prod_{j=1}^{M} q_\phi(z_j | Pa(z_j), x)
\]

Amortization (sharing parameters)

Example:

\[
(\mu, \log \sigma) = \text{EncoderNeuralNet}_\phi(x)
\]

\[
q_\phi(z | x) = \mathcal{N}(z; \mu, \text{diag}(\sigma))
\]
Mapping the Distributions

- VAE learns a mapping that transforms a simple distribution, $q(z)$, to a complex distribution, $q(z|x)$ so that the empirical distribution (in the $x$-space) are matched.

Fig Credit: Kingma’s thesis
Learning

\[ \mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_{\phi}(z|x)} \left[ \log p_{\theta}(x, z) - \log q_{\phi}(z|x) \right] \]

**Alternative optimization** for the variational parameters and then model parameters (VEM).

\[ \nabla_{\theta} \mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_{\phi}(z|x)} \left[ \nabla_{\theta} \log p_{\theta}(x, z) \right] \]

\[ \nabla_{\phi} \mathcal{L}_{\theta,\phi}(x) = \nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} \left[ \log p_{\theta}(x, z) - \log q_{\phi}(z|x) \right] \]
Reparametrization Trick

Changes of variables:

\[ z = g(\epsilon, \phi, x) \quad \epsilon \sim p(\epsilon) \]

Examples (white board):
Reparametrization Trick

Changes of variables:

\[ z = g(\epsilon, \phi, x) \]

\[ \epsilon \sim p(\epsilon) \]

What is it good for?

- \( z \) is deterministic
- \( \epsilon \) is not a function of \( \phi \)
- MC estimation with a single sample
Reparametrization Trick

Changes of variables:

\[ z = g(\epsilon, \phi, x) \]
\[ \epsilon \sim p(\epsilon) \]

Original form

\[ z \sim q_\phi(z|x) \]

Reparameterized form

\[ \nabla_\phi f \]
\[ \nabla_z f \]
\[ z = g(\phi, x, \epsilon) \]
\[ \epsilon \sim p(\epsilon) \]

Backprop
Let's revisit the change of variables:

\[ \log q_\phi(z|x) = \log p(\epsilon) - \log \left| \det \left( \frac{\partial z}{\partial \epsilon} \right) \right| \]
An Example of the Encoder

\[ \epsilon \sim \mathcal{N}(0, I) \]

\[ (\mu, \log \sigma) = \text{EncoderNeuralNet}_\phi(x) \]

\[ z = \mu + \sigma \odot \epsilon \]

Remember the change of the variable:

\[ \log q_\phi(z|x) = \log p(\epsilon) - \log \left| \text{det} \left( \frac{\partial z}{\partial \epsilon} \right) \right| \]

\[ \sum_i \log \sigma_i \]

Constant
Some Results

http://www.dpkingma.com/sgvb_mnist_demo/demo.html
Variations

Fig Credit: https://arxiv.org/pdf/1607.07539.pdf
Fig Credit: Learning Deconvolution Network for Semantic Segmentation
Variations

Generating Sentences from a Continuous Space. S. R. Bowman, et al.c
How to use this technique in a Graphical Model?

Example: Mixture Model

\[
\begin{align*}
\pi & \sim \text{Dir}(\alpha), \\
(\mu_k, \Sigma_k) & \sim \text{NIW}(\lambda), \\
z_n \mid \pi & \sim \pi \\
y_n \mid z_n, \{(\mu_k, \Sigma_k)\}_{k=1}^K & \sim \mathcal{N}(\mu_{z_n}, \Sigma_{z_n}).
\end{align*}
\]

(b) GMM

\[
\begin{align*}
\pi & \sim \text{Dir}(\alpha), \\
(\mu_k, \Sigma_k) & \sim \text{NIW}(\lambda), \\
z_n \mid \pi & \sim \pi \\
x_n & \sim \mathcal{N}(\mu(z_n), \Sigma(z_n)), \\
y_n \mid x_n, \gamma & \sim \mathcal{N}(\mu(x_n; \gamma), \Sigma(x_n; \gamma)).
\end{align*}
\]

(d) GMM SVAE

https://www.youtube.com/watch?v=btr1poCYIzw&t=60s
Limitations of VAE

• When applied to image data, it results into blurry images.
• For images, it is sensitive to irrelevant variance, e.g., translations.
• Not applicable to discrete latent variables.
• Limited flexibility: converting the data distribution to fixed, single-mode prior distribution
GAN: GENERATIVE ADVERSARIAL NETWORK
GANs

• Introduced by Goodfellow et al., 2014
• Assumes implicit generative model
• Asymptotically consistent (unlike variational methods)
• No Markov chains needed
• Often regarded as producing the best samples (but, ... no good way to quantify it)
Generator Network

- Maps noise variable to the data space
- Must be differentiable but no invertibility is required

\[ z \sim \mathcal{N}(0, I) \]

\[ x = G(z; \theta^{(G)}) \]
1-D Example

Figure: Roger Grosse
Learning

• Train $D$ to discriminate between training examples and generated samples
• Train $G$ to fool the discriminator

Figure courtesy: Kim’s slides
Minimax Game

\[
\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

\[
\int_x p_{\text{data}}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \, dx
\]

Fixing \( G \)

Taking derivative of 
\( a \log(y) + b \log(1 - y) \)

Optimal \( D \):

\[
D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}
\]
Discriminator Strategy

\[ D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)} \]
Minimax Game

$$\min_G \max_D \mathbb{E}_x \sim p_{\text{data}}(x) \left[ \log D(x) \right] + \mathbb{E}_{z \sim p_z(z)} \left[ \log (1 - D(G(z))) \right]$$

Substituting

$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$

$$\mathbb{E}_x \sim p_{\text{data}} \left[ \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \right] + \mathbb{E}_x \sim p_g \left[ \log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \right]$$

$$KL \left( p_{\text{data}} \parallel \frac{p_{\text{data}} + p_g}{2} \right)$$

$$KL \left( p_g \parallel \frac{p_{\text{data}} + p_g}{2} \right)$$

GANs minimizes the Jensen–Shannon divergence between two distributions.
More Stable Version

• DCGAN: Most “deconvs” are batch normalized
DCGANs for LSUN Bedrooms

(Radford et al 2015)

Redford et al., “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”
Practical Issues

Strong intra-batch correlation
Practical Issues

Mode Collapse

(Metz et al 2016)
Practical Issues