Improving PixelCNN

Stacking layers of masked convolution creates a blindspot

Solution: use two stacks of convolution, convolution creates a blindspot a vertical stack and a horizontal stack
Improving PixelCNN

There is a problem with this form of masked convolution.

Stacking layers of masked convolution creates a blindspot.
Improving PixelCNN II

Use more expressive nonlinearity: $h_{k+1} = \tanh(W_{k,f} \ast h_k) \odot \sigma(W_{k,g} \ast h_k)$

This information flow (between vertical and horizontal stacks) preserves the correct pixel dependencies.
Samples from PixelCNN

**Topics:** CIFAR-10

- Samples from a class-conditioned PixelCNN
Samples from PixelCNN

**Topics:** CIFAR-10

- Samples from a class-conditioned PixelCNN

Sorrel horse
Samples from PixelCNN

**Topics:** CIFAR-10

- Samples from a class-conditioned PixelCNN

![Sandbar samples](image-url)
Convolutional Long Short-Term Memory

Row LSTM

Stollenga et al, 2015
Oord, Kalchbrenner, Kavukcuoglu, 2016
Pixel RNN
Multiple layers of convolutional LSTM
Samples from PixelRNN

<table>
<thead>
<tr>
<th>occlusion</th>
<th>completions</th>
<th>original</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Samples from PixelRNN
Samples from PixelRNN
Architecture for 1D sequences (Bytenet / Wavenet)

- Stack of dilated, masked 1-D convolutions in the decoder
- The architecture is parallelizable along the time dimension (during training or scoring)
- Easy access to many states from the past
Video Pixel Net

Masked convolution
Video Pixel Net

PixelCNN Decoders

Resolution Preserving CNN Encoders
VPN Samples for Moving MNIST

No frame dependencies

VPN

Videos on nal.ai/vpn
VPN Samples for Robotic Pushing

No frame dependencies

VPN

Videos on nal.ai/vpn
VPN Samples for Robotic Pushing
Variational Autoencoders
Variational Auto-Encoders in General

Variational Auto-encoder (VAE)
Amortised variational inference for latent variable models

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

**Design choices**

- **Prior on the latent variable**
  - Continuous, Discrete, Gaussian, Bernoulli, Mixture

- **Likelihood function**
  - iid (static), sequential, temporal, spatial

- **Approximating posterior**
  - distribution, sequential, spatial

**For scalability and ease of implementation**

- Stochastic gradient descent (and variants),
- Stochastic gradient estimation
Variational Autoencoders (VAEs)

• The VAE defines a generative process in terms of ancestral sampling through a cascade of hidden stochastic layers:

\[ p(x|\theta) = \sum_{h^1, \ldots, h^L} p(h^L|\theta)p(h^{L-1}|h^L, \theta) \cdots p(x|h^1, \theta) \]

![Diagram of VAE](image)
Variational Autoencoders (VAEs)

- The VAE defines a generative process in terms of ancestral sampling through a cascade of hidden stochastic layers:

\[ p(x|\theta) = \sum_{h^1, \ldots, h^L} p(h^L|\theta)p(h^{L-1}|h^L, \theta) \cdots p(x|h^1, \theta) \]

\[ P(h^3) \]
\[ h^3 \]
\[ W^3 \]
\[ P(h^2|h^3) \]
\[ h^2 \]
\[ W^2 \]
\[ P(h^1|h^2) \]
\[ h^1 \]
\[ W^1 \]
\[ P(x|h^1) \]
\[ x \]

Generative Process

Input data
Variational Autoencoders (VAEs)

- The VAE defines a generative process in terms of ancestral sampling through a cascade of hidden stochastic layers:

\[ p(x|\theta) = \sum_{h^1, \ldots, h^L} p(h^L|\theta)p(h^{L-1}|h^L, \theta) \cdots p(x|h^1, \theta) \]

- \( \theta \) denotes parameters of VAE.
- \( L \) is the number of stochastic layers.
- Sampling and probability evaluation is tractable for each \( p(h^\ell|h^{\ell+1}) \)
Variational Autoencoders (VAEs)

• The VAE defines a generative process in terms of ancestral sampling through a cascade of hidden stochastic layers:

\[ p(x|\theta) = \sum_{h^1,\ldots,h^L} p(h^L|\theta)p(h^{L-1}|h^L, \theta) \cdots p(x|h^1, \theta) \]

Each term may denote a complicated nonlinear relationship.

- \( \theta \) denotes parameters of VAE.
- \( L \) is the number of stochastic layers.
- Sampling and probability evaluation is tractable for each \( p(h^\ell|h^{\ell+1}) \)
Variational Autoencoders (VAEs)

- The VAE defines a generative process in terms of ancestral sampling through a cascade of hidden stochastic layers:

\[ p(x|\theta) = \sum_{h^1, h^2} p(h^2|\theta)p(h^1|h^2, \theta)p(x|h^1, \theta) \]

- This term denotes a one-layer neural net.

- \( \theta \) denotes parameters of VAE.

- \( L \) is the number of stochastic layers.

- Sampling and probability evaluation is tractable for each \( p(h^\ell|h^{\ell+1}) \).
Variational Bound

• The VAE is trained to maximize the variational lower bound:

$$\log p(x) = \log \mathbb{E}_{q(h|x)} \left[ \frac{p(x, h)}{q(h|x)} \right] \geq \mathbb{E}_{q(h|x)} \left[ \log \frac{p(x, h)}{q(h|x)} \right] = \mathcal{L}(x)$$
Variational Bound

• The VAE is trained to maximize the variational lower bound:

\[ \log p(x) = \log \mathbb{E}_{q(h|x)} \left[ \frac{p(x, h)}{q(h|x)} \right] \geq \mathbb{E}_{q(h|x)} \left[ \log \frac{p(x, h)}{q(h|x)} \right] = \mathcal{L}(x) \]

\[ \mathcal{L}(x) = \log p(x) - D_{KL} (q(h|x) || p(h|x)) \]

• Trading off the data log-likelihood and the KL divergence from the true posterior.
Variational Bound

• The VAE is trained to maximize the variational lower bound:

\[
\log p(x) = \log \mathbb{E}_{q(h|x)} \left[ \frac{p(x, h)}{q(h|x)} \right] \geq \mathbb{E}_{q(h|x)} \left[ \log \frac{p(x, h)}{q(h|x)} \right] = \mathcal{L}(x)
\]

\[
\mathcal{L}(x) = \log p(x) - D_{KL} (q(h|x)) || p(h|x))
\]

• Trading off the data log-likelihood and the KL divergence from the true posterior.

• Hard to optimize the variational bound with respect to the recognition network (high-variance).

• Key idea of Kingma and Welling is to use reparameterization trick.
Reparameterization Trick

• Assume that the recognition distribution is Gaussian:

\[ q(h^\ell | h^{\ell-1}, \theta) = \mathcal{N} (\mu(h^{\ell-1}, \theta), \Sigma(h^{\ell-1}, \theta)) \]

with mean and covariance computed from the state of the hidden units at the previous layer.
Reparameterization Trick

• Assume that the recognition distribution is Gaussian:

\[ q(h^\ell | h^{\ell-1}, \theta) = \mathcal{N}(\mu(h^{\ell-1}, \theta), \Sigma(h^{\ell-1}, \theta)) \]

with mean and covariance computed from the state of the hidden units at the previous layer.

• Alternatively, we can express this in term of auxiliary variable:

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

\[ h^\ell (\epsilon^\ell, h^{\ell-1}, \theta) = \Sigma(h^{\ell-1}, \theta)^{1/2} \epsilon^\ell + \mu(h^{\ell-1}, \theta) \]
Reparameterization Trick

• Assume that the recognition distribution is Gaussian:

\[ q(h^\ell | h^{\ell-1}, \theta) = \mathcal{N}(\mu(h^{\ell-1}, \theta), \Sigma(h^{\ell-1}, \theta)) \]

• Or \( \epsilon^\ell \sim \mathcal{N}(0, I) \)

\[ h^\ell (\epsilon^\ell, h^{\ell-1}, \theta) = \Sigma(h^{\ell-1}, \theta)^{1/2} \epsilon^\ell + \mu(h^{\ell-1}, \theta) \]

• The recognition distribution \( q(h^\ell | h^{\ell-1}, \theta) \) can be expressed in terms of a deterministic mapping:

\[ h(\epsilon, x, \theta), \quad \text{with} \quad \epsilon = (\epsilon^1, \ldots, \epsilon^L) \]

Deterministic Encoder \quad Distribution of \( \epsilon \) does not depend on \( \theta \)
Reparameterization Trick

\[ \mathbb{KL}[\mathcal{N}(\mu(X), \Sigma(X)) || \mathcal{N}(0, I)] \]

\text{Encoder (Q)}

Sample \( z \) from \( \mathcal{N}(\mu(X), \Sigma(X)) \)

\text{Decoder (P)}

\( ||X - f(z)||^2 \)

\text{Decoder (P)}

\( ||X - f(z)||^2 \)

\text{Encoder (Q)}

Sample \( c \) from \( \mathcal{N}(0, I) \)

\text{without reparameterization trick}

\text{with reparameterization trick}

Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where \( P(X|z) \) is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

The objective we want to optimize is:

\[
\mathbb{E}_{X \sim D} \left[ \log P(X) \right] + \mathbb{E}_{Z \sim Q(Z|X)} \left[ \sum_{k=1}^{K} \log P(Z_k) \right]
\]

If we take the gradient of this equation, the gradient symbol can be moved into the expectations. Therefore, we can sample a single value of \( X \) and a single value of \( z \) from the distribution \( Q(z|X) \), and compute the gradient of:

\[
\log P(X|z)
\]

\[
\mathbb{E}_{Z \sim Q(Z|X)} \log P(X|z)
\]

We can then average the gradient of this function over arbitrarily many samples of \( X \) and \( z \), and the result converges to the gradient of Equation 8.

There is, however, a significant problem with Equation 9. \( \mathbb{E}_{Z \sim Q(Z|X)} \log P(X|z) \) depends not just on the parameters of \( P \), but also on the parameters of \( Q \). However, in Equation 9, this dependency has disappeared! In order to make VAEs work, it's essential to drive \( Q \) to produce codes for \( X \) that \( P \) can reliably decode.
Computing the Gradients

• The gradient w.r.t the parameters: both recognition and generative:

\[ \nabla_{\theta} \mathbb{E}_{h \sim q(h|x, \theta)} \left[ \log \frac{p(x, h|\theta)}{q(h|x, \theta)} \right] \]

\[ = \nabla_{\theta} \mathbb{E}_{\epsilon^1, \ldots, \epsilon^L \sim \mathcal{N}(0, I)} \left[ \log \frac{p(x, h(\epsilon, x, \theta)|\theta)}{q(h(\epsilon, x, \theta)|x, \theta)} \right] \]

\[ = \mathbb{E}_{\epsilon^1, \ldots, \epsilon^L \sim \mathcal{N}(0, I)} \left[ \nabla_{\theta} \log \frac{p(x, h(\epsilon, x, \theta)|\theta)}{q(h(\epsilon, x, \theta)|x, \theta)} \right] \]

Gradients can be computed by backprop

The mapping \( h \) is a deterministic neural net for fixed \( \epsilon \)
Implementing a Variational Algorithm

Variational inference turns integration into optimization: **Automated Tools:**

- **Differentiation:** Theano, Torch7, TensorFlow, Stan.
- **Message passing:** infer.NET

- Stochastic gradient descent and other preconditioned optimization.
- Same code can run on both GPUs or on distributed clusters.
- Probabilistic models are modular, can easily be combined.

**Ideally want probabilistic programming using variational inference.**
Latent Gaussian VAE

$$p(z) = \mathcal{N}(0, I)$$

$$p(x | f^p_\theta (z))$$

$$p_\theta(x | z) = \mathcal{N}(\mu^p_\theta(z), \Sigma^p_\theta(z))$$

$$q_\phi(z | x) = \mathcal{N}(\mu^q_\phi(x), \Sigma^q_\phi(x))$$

$\mathcal{F}(x, q) = \mathbb{E}_{q(z)}[\log p(x | z)] - KL[q(z) || p(z)]$
Latent Gaussian VAE

Latent space disentangles the input data.

Latent space and likelihood bound gives a visualisation of importance.
Representations are useful for strategies such as episodic control.

Latent Gaussian VAE

- Require flexible approximations for the types of posteriors we are likely to see.
Latent Binary VAE


Samples from binarized Atari frames
Semi-supervised VAE

Visual Analogies
Sequential Latent Gaussian VAE

Prior
\[ p(z) \]

\[ \log p(z) \]

Inference
\[ q(z | x) \]

\[ H[q(z)] \]

Model
\[ p(x | z) \]

\[ \log p(x|z) \]

Data \( x \)

\[ p(z) = \prod_i p(z_i | z_{<i}) \]

\[ p(x | f_\theta^p(z)) \]

\[ p_\theta(x | z) = \mathcal{N}(\mu_\theta^p(z), \Sigma_\theta^p(z)) \]

\[ q_\phi(z) = \prod_i q_\phi(z_i | z_{<i}) \]

Gregor, Karol, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. ‘DRAW: A recurrent neural network for image generation.” ICML 2015
Sequential Latent Gaussian VAE

- **LSTM** or GRU networks for state modules
- **Spatial attention** in both the recognition and generation phase using spatial transformers.
- Can remove inference model RNN and share the generate model state.
- Can include additional canvas

Sequential Latent Gaussian VAE

"DRAW: A recurrent neural network for image generation." ICML 2015
Generating Images from Captions

- **Generative Model**: Stochastic Recurrent Network, chained sequence of Variational Autoencoders, with a single stochastic layer.

- **Recognition Model**: Deterministic Recurrent Network.

(Mansimov, Parisotto, Ba, Salakhutdinov, 2015)
Motivating Example

• Can we generate images from natural language descriptions?

A stop sign is flying in blue skies

A pale yellow school bus is flying in blue skies

A large commercial airplane is flying in blue skies

(Mansimov, Parisotto, Ba, Salakhutdinov, 2015)
Flipping Colors

A yellow school bus parked in the parking lot

A red school bus parked in the parking lot

A green school bus parked in the parking lot

A blue school bus parked in the parking lot
Flipping Backgrounds

A very large commercial plane flying in **clear skies**.

A herd of elephants walking across a **dry grass field**.

A very large commercial plane flying in **rainy skies**.

A herd of elephants walking across a **green grass field**.
Next lecture:
Generative Adversarial Networks