Autoregressive Generative Models

Aaron Courville
Université de Montréal

Slides are drawn from Hugo Larochelle, Vincent Dumoulin and Aaron Courville
WHY GENERATIVE MODELS

• Useful learning signal for semi-supervised learning
  ‣ expect a good model to distinguish between real and fake data

Why is one a character and not the other?

real image

Why is one a character and not the other?

random image
WHY GENERATIVE MODELS

• Perhaps that’s what the brain does?
  ▪ sparse coding neurons vs. neurons in V1

Data

Learned representation
WHY GENERATIVE MODELS

• To synthesize new observations
  ‣ useful for planning in a visual environment

Action-Conditional Video Prediction using Deep Networks in Atari Games
Oh, Guo, Lee, Singh, Lewis. NIPS 2015
WHY GENERATIVE MODELS

- As a prior over real observations
  - useful for denoising or super-resolution

Amortised MAP Inference for Image Super-resolution
Sønderby, Caballero, Theis, Shi, Huszár: arXiv 2016
FAMILY OF GENERATIVE MODELS

• Directed graphical models
  ‣ define prior over top-most latent representation
  ‣ define conditionals from top latent representation to observation

\[ p(x, h^{(1)}, h^{(2)}, h^{(3)}) = p(x|h^{(1)})p(h^{(1)}|h^{(2)})p(h^{(2)}|h^{(3)})p(h^{(3)}) \]

› examples: variational autoencoders (VAE), generative adversarial networks (GAN), sparse coding, helmholtz machines

• Properties
  ‣ pros: easy to sample from (ancestral sampling)
  ‣ cons: \( p(x) \) is intractable, so hard to train
FAMILY OF GENERATIVE MODELS

• **Undirected graphical models**
  
  ‣ define a joint energy function

  \[
  E(x, h^{(1)}, h^{(2)}, h^{(3)}) = -xW^{(1)}h^{(1)} - h^{(2)}W^{(2)}h^{(3)} - h^{(3)}W^{(3)}h^{(4)}
  \]

  ‣ exponentiate and normalize

  \[
  p(x, h^{(1)}, h^{(2)}, h^{(3)}) = \exp \left( -E(x, h^{(1)}, h^{(2)}, h^{(3)}) \right) / Z
  \]

  ‣ examples: deep Boltzmann machines (DBM), deep energy models

• **Properties**

  ‣ **pros**: can compute \( p(x) \) up to a multiplicative factor

  ‣ **cons**: hard to sample from (MCMC), \( p(x) \) is intractable, so hard to train
Autoregressive generative models

- choose an ordering of the dimensions in $\mathbf{x}$
- define the conditionals in the product rule expression of $p(\mathbf{x})$

$$p(\mathbf{x}) = \prod_{k=1}^{D} p(x_k | \mathbf{x}_{<k})$$

- examples: masked autoencoder distribution estimator (MADE), pixelCNN, neural autoregressive distribution estimator (NADE), spatial LSTM, pixelRNN

Properties

- **pros**: $p(\mathbf{x})$ is tractable, so easy to train, easy to sample (though slower)
- **cons**: doesn’t have a natural latent representation
• Autoregressive generative models
  ‣ autoregressive models are well known for sequence data (language modeling, time series, etc.)
  ‣ less obviously applicable to arbitrary (non-sequential) observations

• Little history
• Autoregressive generative models
  ‣ autoregressive models are well known for sequence data (language modeling, time series, etc.)
  ‣ less obviously applicable to arbitrary (non-sequential) observations

• Little history
logistic regression for the conditionals (Frey et al., 1996)
Autoregressive generative models

- autoregressive models are well known for sequence data (language modeling, time series, etc.)
- less obviously applicable to arbitrary (non-sequential) observations

Little history

logistic regression for the conditionals (Frey et al., 1996)

neural networks for the conditionals (Bengio and Bengio, 2000)
• Autoregressive generative models
  ‣ autoregressive models are well known for sequence data
    (language modeling, time series, etc.)
  ‣ less obviously applicable to arbitrary (non-sequential) observations

• Little history

  logistic regression for the conditionals (Frey et al., 1996)

  neural networks for the conditionals (Bengio and Bengio, 2000)

  idem, with new weight sharing (NADE) (Larochelle and Murray, Gregor and Lecun, 2011)
• **Autoregressive generative models**
  - autoregressive models are well known for sequence data
    (language modeling, time series, etc.)
  - less obviously applicable to arbitrary (non-sequential) observations

• **Little history**

  logistic regression for the conditionals (Frey et al., 1996)

  neural networks for the conditionals (Bengio and Bengio, 2000)

  idem, with new weight sharing (NADE) (Larochelle and Murray, Gregor and Lecun, 2011)

  Deep NADE, Spatial LSTM, PixelRNN, PixelCNN, WaveNet, Video Pixel Network, etc.
On the menu:

- NADE: Neural Autoregressive Density Estimator
- MADE: Masked Autoencoder for Density Estimator
- PixelCNN Autoregressive CNN
- PixelRNN: Autoregressive RNN for images
- WaveNet: Audio synthesis model, autoregressive dialated CNN
- PixelVAE: VAE with PixelCNN decoder
- Bonus models
NADE (Larochelle and Murray, AISTATS 2011)

• NADE: connectivity is the same as for the original neural auto-regressive network of Bengio and Bengio (NIPS 2000)

• NADE introduces an additional parameter sharing scheme:

  ‣ weights $W'_{j,k,i}$ from the $i$-th input $x_i$ to the $k$-th element of the $j$-th group of hidden unit $h_k^{(j)}$ ($j \geq i$) are shared among the groups:

  $$W'_{j,k,i} = W_{k,i}$$
Neural autoregressive distribution estimator (NADE)

some ordering of $\mathbf{x}$

$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_{783}$ $\mathbf{h}_{784}$ $\mathbf{x}_O$ $\mathbf{\hat{x}}_O$

$500$ units $784$ units $784$ units

$\mathbf{p}(x_{o1} = 1 \mid x_{o<1})$
$\mathbf{p}(x_{o2} = 1 \mid x_{o<2})$
$\mathbf{p}(x_{o3} = 1 \mid x_{o<3})$
$\mathbf{p}(x_{o_{783}} = 1 \mid x_{o<783})$
$\mathbf{p}(x_{o_{784}} = 1 \mid x_{o<784})$

$\mathbf{p}(x_{od} = 1 \mid x_{o<d}) = \sigma(\mathbf{V}_{od} : \mathbf{h}_d + b_{od})$
$\mathbf{h}_d = \sigma(\mathbf{W}_{i,od} \mathbf{x}_{od} + \mathbf{c})$

Neural autoregressive distribution estimator (NADE)

\[
\mathcal{L}(\mathbf{x}) = - \log p(\mathbf{x}) = - \sum_{i=1}^{\lvert \mathbf{x} \rvert} \log p(x_{o_i} \mid x_{o_{<i}})
\]

\[
p(x_{o_d} = 1 \mid x_{o_{<d}}) = \sigma(V_{o_d},:h_d + b_{od})
\]

\[
h_d = \sigma(W_{:,o_d}x_{od} + c)
\]

NADE training is done using "teacher forcing"

- Ground truth values of the pixels are used for conditioning when predicting subsequent values.

\[
\mathcal{L}(x) = - \log p(x) = - \sum_{i=1}^{\left| x \right|} \log p(x_{o_i} | x_{o_{<i}})
\]

\[
p(x_{o_d} = 1 | x_{o_{<d}}) = \sigma(V_{o_d} h_d + b_{o_d})
\]

\[
h_d = \sigma(W_{:o_d} x_{o_d} + c)
\]
NADE Generation

NADE generation (at “test time”) is done by conditioning on values previously sampled from the model.
NADE (Larochelle and Murray, AISTATS 2011)

- NADE for images: must choose an arbitrary ordering over pixels for conditional sampling.

Figure 2: (Left): samples from NADE trained on a binary version of MNIST. (Middle): probabilities from which each pixel was sampled. (Right): visualization of some of the rows of W. This figure is better seen on a computer screen.
Neural autoregressive distribution estimator (NADE)

Extensions

• Real-valued NADE (RNADE): conditionals are modelled by a mixture of gaussians.

• Orderless and deep NADE (DeepNADE): a single deep neural network is trained to assign a conditional distribution to any variable given any subset of the others.

• Convolutional NADE (ConvNADE)

We then recover the mean-field updates of Equations 7 and 8.

\[ \mu_j = \frac{1}{Z} \sum_{i \neq j} \exp \left( \sum_k \theta_{kj} \phi_k \right) \]

Similarly, we set the derivative with respect to \( j < i \)

\[ \frac{\partial c_i}{\partial x_j} = \begin{cases} 1 & \text{if } x_j = 1 \\ 0 & \text{if } x_j = 0 \end{cases} \]

Figure 2:

- Binarized MNIST samples (NADE)
- Binarized MNIST samples (DeepNADE)
- Binarized MNIST samples (ConvNADE)

References:

• **MADE**: Masked Autoencoder for Density Estimator

• **Question**: How do you construct an autoregressive autoencoder?

  ➡ **Specifically**: How to modify the autoencoder so as to satisfy the autoregressive property: where prediction of $x_d$ depends only on the preceding inputs $x_{<d}$, *relative to some (arbitrary) ordering*.

  ➡ I.e. there must be no computational path between output unit $x_d$ and any of the input units $x_d, \ldots, x_D$, *again relative to some ordering*.

  ➡ I.e. For each of these paths, at least one connection in the weight matrix must be 0.
• **Question**: How do you construct an autoregressive autoencoder?

• Convenient way of zeroing connections is to elementwise-multiply each matrix by a binary mask matrix $M$, whose entries that are set to 0 correspond to the connections we wish to remove.

• For a single hidden layer autoencoder:

\[
\begin{align*}
    h(x) &= g(b + (W \odot M^W)x) \\
    \hat{x} &= \text{sigmoid}(c + (V \odot M^V)h(x))
\end{align*}
\]
Topics: MADE (Germain et al. 2015)

• Idea: constrain output so can be used for the conditionals \( p(x_k | x_{<k}) \)

• Generalization of the work from Bengio and Bengio (2000)
Topics: MADE (Germain et al. 2015)

• Idea: constrain output so can be used for the conditionals \( p(x_k | x_{<k}) \)

• Generalization of the work from Bengio and Bengio (2000)
**Topics:** MADE (Germain et al. 2015)

- Idea: constrain output so can be used for the conditionals \( p(x_k | x_{<k}) \)

- Generalization of the work from Bengio and Bengio (2000)
Topics: MADE (Germain et al. 2015)

- Idea: constrain output so can be used for the conditionals $p(x_k | x_{<k})$

- Generalization of the work from Bengio and Bengio (2000)
Topics: MADE (Germain et al. 2015)

• Idea: constrain output so can be used for the conditionals \( p(x_k | x_{<k}) \)

• Generalization of the work from Bengio and Bengio (2000)
Topics: MADE (Germain et al. 2015)

- Idea: constrain output so can be used for the conditionals $p(x_k | x_{<k})$

- Generalization of the work from Bengio and Bengio (2000)
Topics: MADE (Germain et al. 2015)

- Idea: constrain output so can be used for the conditionals \( p(x_k|x_{<k}) \)
- Generalization of the work from Bengio and Bengio (2000)
**Topics:** MADE (Germain et al. 2015)

- Idea: constrain output so can be used for the conditionals

\[ p(x_k | \mathbf{x}_{<k}) \]

\[ M_{k',k}^W = 1_{m^l(k') \geq m^{l-1}(k)} \]

\[ M_{d,k}^V = 1_{d > m^L(k)} \]

- Generalization of the work from Bengio and Bengio (2000)
Topics: MADE (Germain et al. 2015)

• Training has the same complexity as regular autoencoders

• Computing $p(\mathbf{x})$ is just a matter of performing a forward pass

• Sampling however requires $D$ forward passes

• In practice, very large hidden layers may be required
  ‣ not all hidden units can contribute to each conditional
Masked Autoencoder for Distribution Estimation (MADE)

\[
\hat{x} = \text{decode}(\text{encode}(x))
\]

\[
\mathcal{L}(x) = -\sum_{i=1}^{\left| x \right|} \left( x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i) \right)
\]

NLL criterion for a binary \( x \)

Masked Autoencoder for Distribution Estimation (MADE)

Binarized MNIST samples

PixelCNN

Idea: use masked convolutions to enforce the autoregressive relationship

\[ p(x_i \mid x_{<i}) \]

PixelCNN

\[ p(x_i \mid x_{<i}) = p(x_{i,R} \mid x_{<i})p(x_{i,G} \mid x_{i,R}, x_{<i})p(x_{i,B} \mid x_{i,R}, x_{i,G}, x_{<i}) \]

autoregressive over color channels

Mask A

Mask B

8-bits pixel values (multinoulli distribution)
How can convolutions make this raster scan faster?

Use a stack of masked convolutions

Training can be parallelized, though generation is still a sequential operation over pixels.

PixelCNN

- only depends on pixel above and to the left
- masked convolution
- composing multiple layers increases the context size

There is a problem with this form of masked convolution.

Stacking layers of masked convolution creates a blind spot

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

Topics: CIFAR-10

- Performance measured in bits/dim

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL Test (Train)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Distribution: [30]</td>
<td>8.00</td>
</tr>
<tr>
<td>Multivariate Gaussian: [30]</td>
<td>4.70</td>
</tr>
<tr>
<td>NICE: [4]</td>
<td>4.48</td>
</tr>
<tr>
<td>Deep Diffusion: [24]</td>
<td>4.20</td>
</tr>
<tr>
<td>DRAW: [9]</td>
<td>4.13</td>
</tr>
<tr>
<td>Deep GMMs: [31, 29]</td>
<td>4.00</td>
</tr>
<tr>
<td>Conv DRAW: [8]</td>
<td>3.58 (3.57)</td>
</tr>
<tr>
<td>RIDE: [26, 30]</td>
<td>3.47</td>
</tr>
<tr>
<td>PixelCNN: [30]</td>
<td>3.14 (3.08)</td>
</tr>
<tr>
<td>PixelRNN: [30]</td>
<td>3.00 (2.93)</td>
</tr>
<tr>
<td><strong>Gated PixelCNN:</strong></td>
<td><strong>3.03 (2.90)</strong></td>
</tr>
</tbody>
</table>
**EXPERIMENTAL RESULTS**

**Topics:** CIFAR-10

- Samples from a class-conditional PixelCNN

---

*Conditional Image Generation with PixelCNN Decoders*
van den Oord, Kalchbrenner, Vinyals, Espeholt, Graves, Kavukcuoglu, NIPS 2016

---

Coral Reef
EXPERIMENTAL RESULTS

Topics: CIFAR-10

• Samples from a class-conditional PixelCNN

![Sample images from a class-conditional PixelCNN](image)

*Conditional Image Generation with PixelCNN Decoders*
van den Oord, Kalchbrenner, Vinyals, Espeholt, Graves, Kavukcuoglu, NIPS 2016

Sorrel horse
EXPERIMENTAL RESULTS

Topics: CIFAR-10

- Samples from a class-conditional PixelCNN

*Conditional Image Generation with PixelCNN Decoders*
van den Oord, Kalchbrenner, Vinyals, Espeholt, Graves, Kavukcuoglu, NIPS 2016

![Sample images from CIFAR-10](image)

- Sandbar
Topics: CIFAR-10

• Samples from a class-conditional PixelCNN

Conditional Image Generation with PixelCNN Decoders
van den Oord, Kalchbrenner, Vinyals, Espeholt, Graves, Kavukcuoglu, NIPS 2016

Lhasa Apso (dog)
PixelRNN

PixelRNN

Diagonal BiLSTM

In the Diagonal BiLSTM, to allow for parallelization along the diagonals, the input map is skewed by offsetting each row by one position with respect to the previous row. When the spatial layer is computed left to right and column by column, the output map is shifted back into the original size. The convolution uses a kernel of size $2 \times 1$.

PixelRNN


<table>
<thead>
<tr>
<th>Model</th>
<th>NLL Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBM 2hl [1]:</td>
<td>≈ 84.62</td>
</tr>
<tr>
<td>DBN 2hl [2]:</td>
<td>≈ 84.55</td>
</tr>
<tr>
<td>NADE [3]:</td>
<td>88.33</td>
</tr>
<tr>
<td>EoNADE 2hl (128 orderings) [3]:</td>
<td>85.10</td>
</tr>
<tr>
<td>EoNADE-5 2hl (128 orderings) [4]:</td>
<td>84.68</td>
</tr>
<tr>
<td>DLGM [5]:</td>
<td>≈ 86.60</td>
</tr>
<tr>
<td>DLGM 8 leapfrog steps [6]:</td>
<td>≈ 85.51</td>
</tr>
<tr>
<td>DARN 1hl [7]:</td>
<td>84.13</td>
</tr>
<tr>
<td>MADE 2hl (32 masks) [8]:</td>
<td>86.64</td>
</tr>
<tr>
<td>DRAW [9]:</td>
<td>≤ 80.97</td>
</tr>
<tr>
<td>PixelCNN:</td>
<td>81.30</td>
</tr>
<tr>
<td>Row LSTM:</td>
<td>80.54</td>
</tr>
<tr>
<td>Diagonal BiLSTM (1 layer, $h = 32$):</td>
<td>80.75</td>
</tr>
<tr>
<td>Diagonal BiLSTM (7 layers, $h = 16$):</td>
<td>79.20</td>
</tr>
</tbody>
</table>

Downsampled ImageNet samples
PixelRNN

Theano implementation: https://github.com/igul222/pixel_rnn

TensorFlow implementation: https://github.com/carpedm20/pixel-rnn-tensorflow

WaveNet

Audio: much larger dimensionality than images (at least 16,000 samples per second)

Idea: adapt PixelCNN to allow very large temporal dependencies

WaveNet

Addressing large-scale temporal dependencies

Regular convolutions

Dilated convolutions

Note: strided convolutions cannot be used because the output has to have the same dimensionality as the input.

WaveNet

Discrete conditional probabilities

\[ a_t \quad (16\text{-bit int}) \quad \rightarrow \quad x_t \in [-1, 1] \]

\[ \tilde{x}_t = \text{sign}(x_t) \frac{\ln(1 + 255|x_t|)}{\ln 256} \]

\[ \tilde{x}_t \in [-1, 1] \quad \rightarrow \quad \tilde{a}_t \quad (8\text{-bit int}) \]

\[ \tilde{x}_t \quad \rightarrow \quad \tilde{X}_t \quad \text{quantize back} \]

WaveNet

Complete architecture

WaveNet

Conditional generation

\[ z = \tanh(W_{k,f} \ast x + V_{k,f}^T h) \odot \sigma(W_{k,g} \ast x + V_{k,g}^T h) \]

Global conditioning (e.g., speaker ID)

\[ z = \tanh(W_{k,f} \ast x + V_{k,f} \ast h) \odot \sigma(W_{k,g} \ast x + V_{k,g} \ast h) \]

Local conditioning (e.g., text)

In Sect. 5 we show that the VPN achieves 87.6 nats/frame, a score that is near all the previous frames.

Here assumptions:

Probabilities, we can model it in a tractable manner and without introducing independence by applying the chain rule to factorize the video likelihood.

In this section we define the probabilistic model implemented by Video Pixel Networks. Let $x \in \mathbb{R}^{3 \times 288 \times 288}$ be a random variable that takes values from the RGB color intensities of the pixel.

$\mathbf{x}$ is a four-dimensional tensor of pixel values $\mathbf{x} = \{t, i, j, c\}$. By considering the video to be a sequence of frames $(t, i, j, c)$, the task is to predict the following 18 frames. We show that the VPN not only generalizes to new action sequences with objects seen during training, but also to new action sequences involving novel objects.

Video Pixel Networks
Kalchbrenner, van den Oord, Simonyan, Danihelka, Vinyals, Graves, Kavukcuoglu, NIPS 2016
**AUTOREGRESSIVE VIDEO MODELS**

**Topics:** Video Pixel Network

- Connect Pixel CNN to frame-wise convolutional networks and time-wise convolutional LSTMs

- Videos of robot manipulating
  - objects seen in the training set
  - new objects not seen in training set

*Video Pixel Networks*
Kalchbrenner, van den Oord, Simonyan, Danihelka, Vinyals, Graves, Kavukcuoglu, NIPS 2016
Can we speed up the generation time of PixelCNN?

• Yes, via multiscale generation:
Can we speed up the generation time of PixelCNN?

- Yes, via multiscale generation:

**Figure 2.** Example pixel grouping and ordering for a $4 \times 4$ image. The upper-left corners form group 1, the upper-right group 2, and so on. For clarity we only use arrows to indicate immediately-neighboring dependencies, but note that all pixels in preceding groups can be used to predict all pixels in a given group. For example all pixels in group 2 can be used to predict pixels in group 4. In our image experiments pixels in group 1 originate from a lower-resolution image. For video, they are generated given the previous frames.
Can we speed up the generation time of PixelCNN?

• Yes, via multiscale generation.
• Also seems to help to provide better global structure

Figure 1. Samples from our model at resolutions from $4 \times 4$ to $256 \times 256$, conditioned on text and bird part locations in the CUB data set. See Fig. 4 and the supplement for more examples.
• **Autoregressive generative models**
  
  ‣ choose an ordering of the dimensions in \( \mathbf{x} \)
  
  ‣ define the conditionals in the product rule expression of \( p(\mathbf{x}) \)

\[
p(\mathbf{x}) = \prod_{k=1}^{D} p(x_k | \mathbf{x}_{<k})
\]

  ‣ examples: masked autoencoder distribution estimator (MADE), pixelCNN, neural autoregressive distribution estimator (NADE), spatial LSTM, pixelRNN

• **Properties**

  ‣ *pros*: \( p(\mathbf{x}) \) is tractable, so easy to train, easy to sample (though slower)
  
  ‣ *cons*: doesn’t have a natural latent representation
• Uses a PixelCNN in the VAE decoder to help avoid the blurring caused by the standard VAE assumption of independent pixels.
PixelVAE samples

64x64 LSUN bedroom scenes

64x64 ImageNet
4.2.1 Features Modeled at Each Layer

To see which features are modeled by each of the multiple layers, we draw multiple samples while varying the sampling noise at only a specific layer (either at the pixel-wise output or one of the latent layers) and visually inspect the resulting images (Fig. 5). When we vary only the pixel-level sampling (holding $z_1$ and $z_2$ fixed), samples are almost indistinguishable and differ only in precise positioning and shading details, suggesting that the model uses the pixel-level autoregressive distribution to model only these features. Samples where only the noise in the middle-level ($8 \times 8$) latent variables is varied have different objects and colors, but appear to have similar basic room geometry and composition. Finally, samples with varied top-level latent variables have diverse room geometry.

Figure 5: We visually inspect the variation in image features captured by the different levels of stochasticity in our model. For the two-level latent variable model trained on $64 \times 64$ LSUN bedrooms, we vary only the top-level sampling noise (top) while holding the other levels constant, vary only the middle-level noise (middle), and vary only the bottom (pixel-level) noise (bottom).

It appears that the top-level latent variables learn to model room structure and overall geometry, the middle-level latents model color and texture features, and the pixel-level distribution models low-level image characteristics such as texture, alignment, shading.

4.3 ImageNet

The $64 \times 64$ ImageNet generative modeling task was introduced in (van den Oord et al., 2016a) and involves density estimation of a difficult, highly varied image distribution. We trained a hierarchical PixelVAE model (with a similar architecture to the model in section 4.2) of comparable size to the models in van den Oord et al. (2016a;b) on $64 \times 64$ ImageNet in 5 days on 3 NVIDIA GeForce GTX 1080 GPUs. We report validation set likelihood in Table 2. Our model achieves a slightly lower log-likelihood than PixelRNN (van den Oord et al., 2016a), but a visual inspection of ImageNet samples from our model (Fig. 6) reveals them to be significantly more globally coherent than samples from PixelRNN.