

CSC321 Lecture 20: Reversible and Autoregressive Models

Roger Grosse

Overview

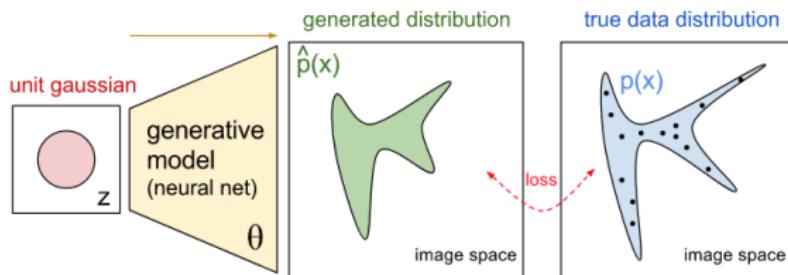
Four modern approaches to generative modeling:

- Generative adversarial networks (last lecture)
- **Reversible architectures** (today)
- Autoregressive models (today)
- Variational autoencoders (CSC412)

All four approaches have different pros and cons.

Overview

- Remember that the GAN generator network represents a distribution by sampling from a simple distribution $p_Z(z)$ over code vectors z .
 - I'll use p_Z here to emphasize that it's a distribution on z .



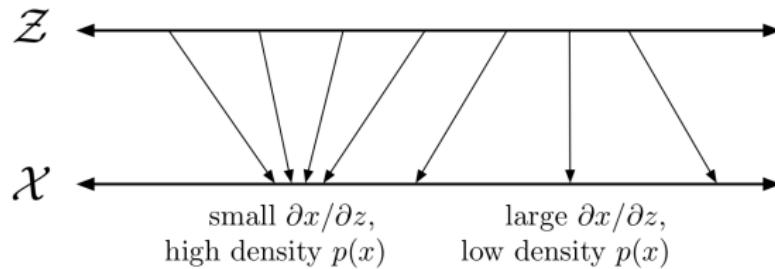
- A GAN was an implicit generative model, since we could only generate samples, not evaluate the log-likelihood.
 - Can't tell if it's missing modes, memorizing the training data, etc.
- Reversible architectures are an elegant kind of generator network with tractable log-likelihood.

Change of Variables Formula

- Let f denote a differentiable, **bijective** mapping from space \mathcal{Z} to space \mathcal{X} . (I.e., it must be 1-to-1 and cover all of \mathcal{X} .)
- Since f defines a one-to-one correspondence between values $\mathbf{z} \in \mathcal{Z}$ and $\mathbf{x} \in \mathcal{X}$, we can think of it as a change-of-variables transformation.
- Change-of-Variables Formula** from probability theory: if $\mathbf{x} = f(\mathbf{z})$, then

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(z) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right|^{-1}$$

- Intuition for the Jacobian term:



Change of Variables Formula

- Suppose we have a generator network which computes the function f . It's tempting to apply the change-of-variables formula in order to compute the density $p_X(\mathbf{x})$.
- I.e., compute $\mathbf{z} = f^{-1}(\mathbf{x})$

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right|^{-1}$$

- Problems?
 - The mapping f needs to be invertible, with an easy-to-compute inverse.
 - It needs to be differentiable, so that the Jacobian $\partial \mathbf{x} / \partial \mathbf{z}$ is defined.
 - We need to be able to compute the (log) determinant.
- The GAN generator may be differentiable, but it doesn't satisfy the other two properties.

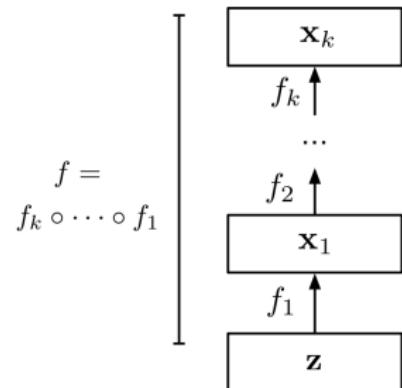
Reversible Blocks

- Now let's define a **reversible block** which is invertible and has a tractable determinant.

- Such blocks can be composed.

- Inversion: $f^{-1} = f_1^{-1} \circ \dots \circ f_k^{-1}$

- Determinants: $\left| \frac{\partial \mathbf{x}_k}{\partial \mathbf{z}} \right| = \left| \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-1}} \right| \dots \left| \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \right| \left| \frac{\partial \mathbf{x}_1}{\partial \mathbf{z}} \right|$



Reversible Blocks

- Recall the residual blocks:

$$\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x})$$

- Reversible blocks are a variant of residual blocks. Divide the units into two groups, \mathbf{x}_1 and \mathbf{x}_2 .

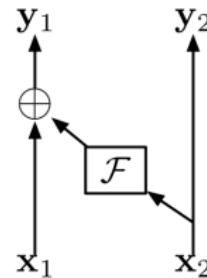
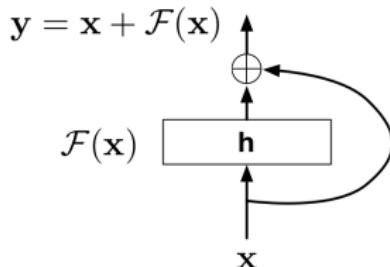
$$\mathbf{y}_1 = \mathbf{x}_1 + \mathcal{F}(\mathbf{x}_2)$$

$$\mathbf{y}_2 = \mathbf{x}_2$$

- Inverting a reversible block:

$$\mathbf{x}_2 = \mathbf{y}_2$$

$$\mathbf{x}_1 = \mathbf{y}_1 - \mathcal{F}(\mathbf{x}_2)$$



Reversible Blocks

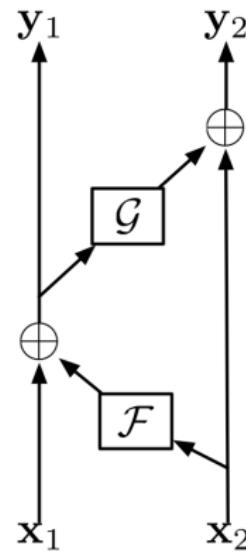
Composition of two reversible blocks, but with \mathbf{x}_1 and \mathbf{x}_2 swapped:

- Forward:

$$\begin{aligned}\mathbf{y}_1 &= \mathbf{x}_1 + \mathcal{F}(\mathbf{x}_2) \\ \mathbf{y}_2 &= \mathbf{x}_2 + \mathcal{G}(\mathbf{y}_1)\end{aligned}$$

- Backward:

$$\begin{aligned}\mathbf{x}_2 &= \mathbf{y}_2 - \mathcal{G}(\mathbf{y}_1) \\ \mathbf{x}_1 &= \mathbf{y}_1 - \mathcal{F}(\mathbf{x}_2)\end{aligned}$$



Volume Preservation

- It remains to compute the log determinant of the Jacobian.
- The Jacobian of the reversible block:

$$\begin{aligned}\mathbf{y}_1 &= \mathbf{x}_1 + \mathcal{F}(\mathbf{x}_2) \\ \mathbf{y}_2 &= \mathbf{x}_2\end{aligned}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \mathbf{I} & \frac{\partial \mathcal{F}}{\partial \mathbf{x}_2} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$



- This is an upper triangular matrix. The determinant of an upper triangular matrix is the product of the diagonal entries, or in this case, 1.
- Since the determinant is 1, the mapping is said to be **volume preserving**.

Nonlinear Independent Components Estimation

- We've just defined the reversible block.
 - Easy to invert by subtracting rather than adding the residual function.
 - The determinant of the Jacobian is 1.
- Nonlinear Independent Components Estimation (NICE) trains a generator network which is a composition of lots of reversible blocks.
- We can compute the likelihood function using the change-of-variables formula:

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right|^{-1} = p_Z(\mathbf{z})$$

- We can train this model using maximum likelihood. I.e., given a dataset $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$, we maximize the likelihood

$$\prod_{i=1}^N p_X(\mathbf{x}^{(i)}) = \prod_{i=1}^N p_Z(f^{-1}(\mathbf{x}^{(i)}))$$

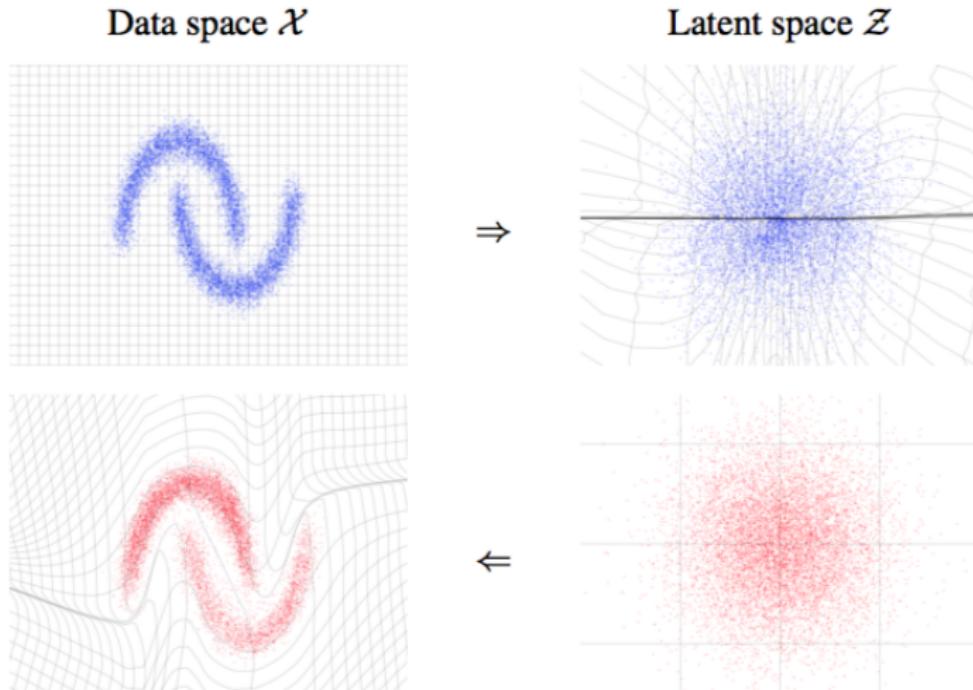
Nonlinear Independent Components Estimation

- Likelihood:

$$p_{\mathbf{x}}(\mathbf{x}) = p_Z(\mathbf{z}) = p_Z(f^{-1}(\mathbf{x}))$$

- Remember, p_Z is a simple, fixed distribution (e.g. independent Gaussians)
- Intuition: train the network such that f^{-1} maps each data point to a high-density region of the code vector space \mathcal{Z} .
 - Without constraints on f , it could map everything to $\mathbf{0}$, and this likelihood objective would make no sense.
 - But it can't do this because it's volume preserving.

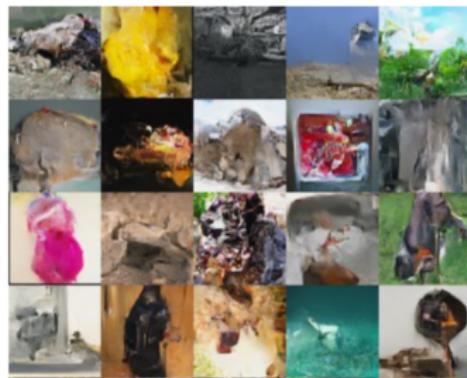
Nonlinear Independent Components Estimation



Dinh et al., 2016. Density estimation using RealNVP. ↗

Nonlinear Independent Components Estimation

Samples produced by RealNVP, a model based on NICE.



ImageNet



celebrities



bedrooms

Dinh et al., 2016. Density estimation using RealNVP.

RevNets (optional)

- A side benefit of reversible blocks: you don't need to store the activations in memory to do backprop, since you can reverse the computation.
 - I.e., compute the activations as you need them, moving backwards through the computation graph.
- Notice that reversible blocks look a lot like residual blocks.
- We recently designed a reversible residual network (RevNet) architecture which is like a ResNet, but with reversible blocks instead of residual blocks.
 - Matches state-of-the-art performance on ImageNet, but without the memory cost of activations!
 - Gomez et al., NIPS 2017. "The reversible residual network: backprop without storing activations".

Overview

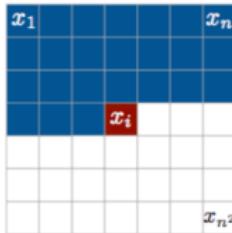
Four modern approaches to generative modeling:

- Generative adversarial networks (last lecture)
- Reversible architectures (today)
- **Autoregressive models** (today)
- Variational autoencoders (CSC412)

All four approaches have different pros and cons.

Autoregressive Models

- We've already looked at autoregressive models in this course:
 - Neural language models
 - RNN language models (and decoders)
- We can push this further, and generate very long sequences.



Treat an image as a
very long sequence
using raster scan order



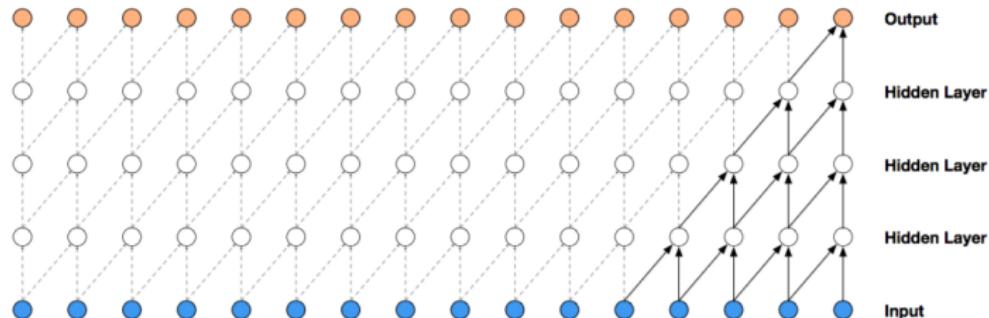
A speech signal can be
represented as a waveform, with at least
16,000 samples per second.

- Problem: training an RNN to generate these sequences requires a for loop over $> 10,000$ time steps.

Causal Convolution

Idea 1: causal convolution

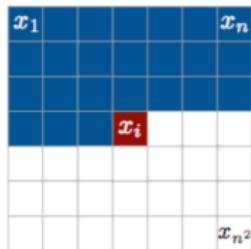
- For RNN language models, we used the training sequence as both the inputs and the outputs to the RNN.
 - We made sure the model was **causal**: each prediction depended only on inputs earlier in the sequence.
- We can do the same thing using a convolutional architecture.



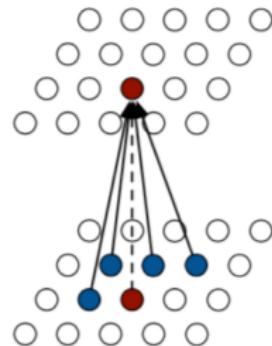
- No for loops! Processing each input sequence just requires a series of convolution operations.

Causal Convolution

Causal convolution for images:

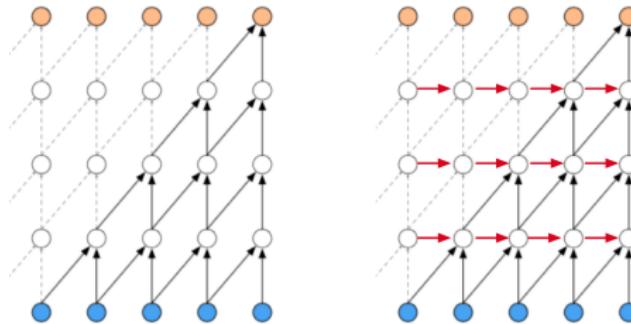


The image is treated as a very long sequence of pixels using raster scan order.



We can restrict the connectivity pattern in each layer to make it causal. This can be implemented by clamping some weights to zero.

CNN vs. RNN



- We can turn a causal CNN into an RNN by adding recurrent connections. Is this a good idea?
 - The RNN has a memory, so it can use information from all past time steps. The CNN has a limited context.
 - But training the RNN is very expensive since it requires a for loop over time steps. The CNN only requires a series of convolutions.
 - Generating from both models is very expensive, since it requires a for loop. (Whereas generating from a GAN or a reversible model is very fast.)

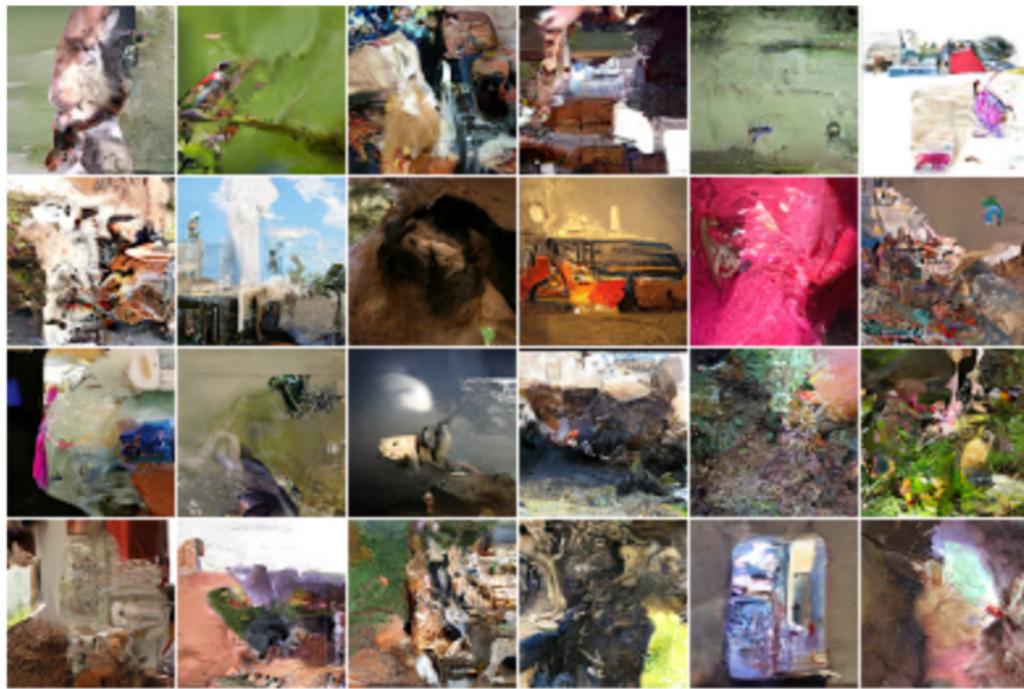
PixelCNN and PixelRNN

- Van den Oord et al., ICML 2016, “Pixel recurrent neural networks”
- This paper introduced two autoregressive models of images: the PixelRNN and the PixelCNN. Both generated amazingly good high-resolution images.
- The output is a softmax over 256 possible pixel intensities.
- Completing an image using an PixelCNN:



PixelCNN and PixelRNN

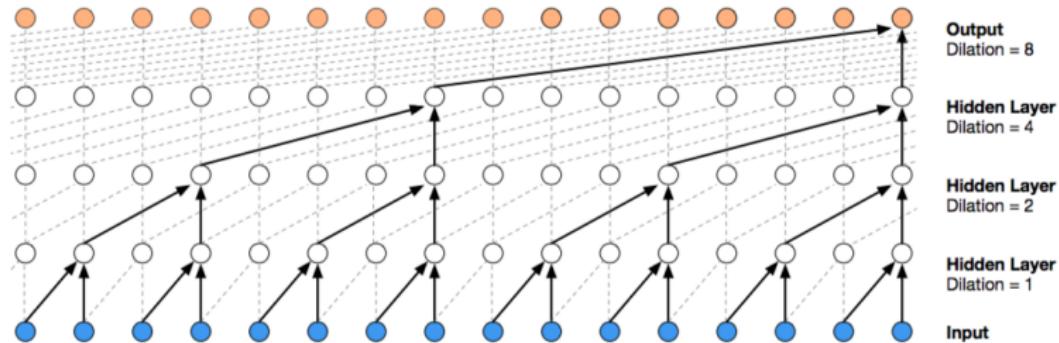
Samples from a PixelRNN trained on ImageNet:



Dilated Convolution

Idea 2: dilated convolution

- The advantage of RNNs over CNNs is that their memory lets them learn arbitrary long-distance dependencies.
 - But we can dramatically increase a CNN's receptive field using dilated convolution.
 - You did this in Programming Assignment 2.



WaveNet

- WaveNet is an autoregressive model for raw audio based on causal dilated convolutions.
 - van den Oord et al., 2016. “WaveNet: a generative model for raw audio”.
- Audio needs to be sampled at at least 16k frames per second for good quality. So the sequences are very long.
- WaveNet uses dilations of $1, 2, \dots, 512$, so each unit at the end of this block has a receptive field of length 1024, or 64 milliseconds.
- It stacks several of these blocks, so the total context length is about 300 milliseconds.
- <https://deepmind.com/blog/wavenet-generative-model-raw-audio/>