Deep Generative Models

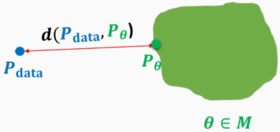
Lecture 3: Autoregressive Models

Aditya Grover

UCLA

- **Given:** a training set of examples, e.g., images of dogs

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



Model family

- **Goal:** learn a probability distribution $p(x)$ over images x
  But why?
  1. **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like a dog (*sampling*)
  2. **Density estimation:** $p(x)$ should be high if x looks like a dog, and low otherwise (*anomaly detection*)
  3. **Unsupervised representation learning:** We should be able to learn what these images have in common, e.g., ears, tail, etc. (*features*)

Recap: Bayesian networks vs neural models

- Using Chain Rule

  $$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)p(x_4 \mid x_1, x_2, x_3)$$

  Fully General, no assumptions needed (exponential size, no free lunch)

- Bayes Net

  $$p(x_1, x_2, x_3, x_4) \approx p_{\mathrm{CPT}}(x_1)p_{\mathrm{CPT}}(x_2 \mid x_1)p_{\mathrm{CPT}}(x_3 \mid \cancel{x_1}, x_2)p_{\mathrm{CPT}}(x_4 \mid x_1, \cancel{x_2}, \cancel{x}$$

  Assumes conditional independencies; tabular representations via conditional probability tables (CPT)

- Neural Models

  $$p(x_1, x_2, x_3, x_4) \approx p(x_1)p(x_2 \mid x_1)p_{\mathrm{Neural}}(x_3 \mid x_1, x_2)p_{\mathrm{Neural}}(x_4 \mid x_1, x_2, x_3)$$

  Assumes specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.

Neural Models for classification

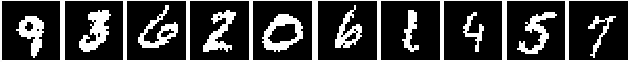
- Setting: binary classification of $Y \in \{0, 1\}$ given input features $X \in \{0, 1\}^n$
- For classification, we care about $p(Y \mid \mathbf{x})$, and assume that

$$p(Y = 1 \mid \mathbf{x}; \boldsymbol{\alpha}) = f(\mathbf{x}, \boldsymbol{\alpha})$$

- **Logistic regression**: let $z(\boldsymbol{\alpha}, \mathbf{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i$.
  $p_{\text{logit}}(Y = 1 \mid \mathbf{x}; \boldsymbol{\alpha}) = \sigma(z(\boldsymbol{\alpha}, \mathbf{x}))$, where $\sigma(z) = 1/(1 + e^{-z})$
- **Non-linear** dependence: let $\mathbf{h}(A, \mathbf{b}, \mathbf{x})$ be a non-linear transformation of the input features.
  $p_{\text{Neural}}(Y = 1 \mid \mathbf{x}; \boldsymbol{\alpha}, A, \mathbf{b}) = \sigma(\alpha_0 + \sum_{i=1}^{h} \alpha_i \mathbf{h}_i)$
    - More flexible
    - More parameters: $A, \mathbf{b}, \boldsymbol{\alpha}$
    - Repeat multiple times to get a multilayer perceptron (neural network)

Motivating Example: MNIST

- **Given:** a dataset $\mathcal{D}$ of handwritten digits (binarized MNIST)



- Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1).
- **Goal:** Learn a probability distribution $p(x) = p(x_1, \cdots, x_{784})$ over $x \in \{0,1\}^{784}$ such that when $x \sim p(x)$, x looks like a digit
- Two step process:
  1. Parameterize a model family $\{p_\theta(x), \theta \in \Theta\}$ [This lecture]
  2. Search for model parameters θ based on training data $\mathcal{D}$ [Next lecture]

Autoregressive Models

- We can pick an ordering of all the random variables, i.e., raster scan ordering of pixels from top-left (X_1) to bottom-right ($X_{n=784}$)
- Without loss of generality, we can use chain rule for factorization

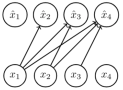
  $$p(x_1, \cdots, x_{784}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$$

- Some conditionals are too complex to be stored in tabular form. Instead, we **assume**

  $$p(x_1, \cdots, x_{784}) = p_{\mathrm{CPT}}(x_1; \alpha^1)p_{\mathrm{logit}}(x_2 \mid x_1; \boldsymbol{\alpha}^2)p_{\mathrm{logit}}(x_3 \mid x_1, x_2; \boldsymbol{\alpha}^3) \cdots$$

- More explicitly
  - $p_{\mathrm{CPT}}(X_1 = 1; \alpha^1) = \alpha^1$, $p(X_1 = 0) = 1 - \alpha^1$
  - $p_{\mathrm{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\alpha}^2) = \sigma(\boldsymbol{\alpha}_0^2 + \boldsymbol{\alpha}_1^2 x_1)$
  - $p_{\mathrm{logit}}(X_3 = 1 \mid x_1, x_2; \boldsymbol{\alpha}^3) = \sigma(\boldsymbol{\alpha}_0^3 + \boldsymbol{\alpha}_1^3 x_1 + \boldsymbol{\alpha}_2^3 x_2)$
- Note: This is a **modeling assumption**. We are using parameterized functions (e.g., logistic regression above) to predict next pixel given all the previous ones. Called **autoregressive** model.

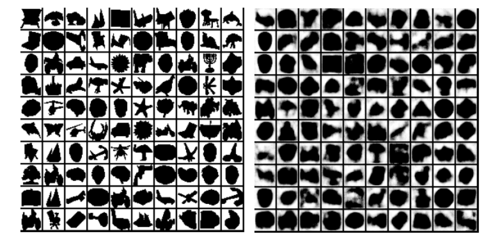
Fully Visible Sigmoid Belief Network (FVSBN)



FVSBN

- The conditional variables $X_i \mid X_1, \cdots, X_{i-1}$ are Bernoulli with parameters
$$\hat{x}_i = p(X_i = 1 | x_1, \cdots, x_{i-1}; \boldsymbol{\alpha}^i) = p(X_i = 1 | x_{<i}; \boldsymbol{\alpha}^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$
- How to evaluate $p(x_1, \cdots, x_{784})$? Multiply all the conditionals
$$p(X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0) = (1 - \hat{x}_1) \times \hat{x}_2 \times \hat{x}_3 \times (1 - \hat{x}_4)$$
$$= (1 - \hat{x}_1) \times \hat{x}_2(X_1 = 0) \times \hat{x}_3(X_1 = 0, X_2 = 1) \times (1 - \hat{x}_4(X_1 = 0, X_2 = 1, X_3 = 1))$$
- How to sample from $p(x_1, \cdots, x_{784})$?
  1. Sample $\overline{x}_1 \sim p(x_1)$ (`np.random.choice([1,0],p=[`$\hat{x}_1, 1 - \hat{x}_1$`])`)
  2. Sample $\overline{x}_2 \sim p(x_2 \mid x_1 = \overline{x}_1)$
  3. Sample $\overline{x}_3 \sim p(x_3 \mid x_1 = \overline{x}_1, x_2 = \overline{x}_2) \cdots$
- How many parameters? $1 + 2 + 3 + \cdots + n \approx n^2/2$

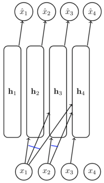
FVSBN Results



Training data on the left (*Caltech 101 Silhouettes*). Samples from the model on the right.

Figure from *Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015*.

NADE: Neural Autoregressive Density Estimation



NADE

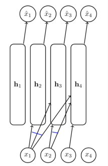
- To improve model: use one layer neural network instead of logistic regression

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\hat{x}_i = p(x_i | x_1, \cdots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \boldsymbol{\alpha}_i, b_i}_{\text{parameters}}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

- For example $\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{A_2} x_1 + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{c_2}\right)$ $\mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{c_3}\right)$

NADE: Neural Autoregressive Density Estimation



NADE

- Tie weights to *reduce the number of parameters* and *speed up computation* (see blue dots in the figure):

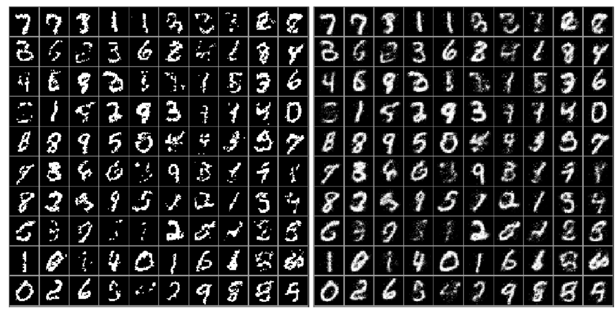
$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{x}_i = p(x_i|x_1, \cdots, x_{i-1}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

$$\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ \boldsymbol{w}_1 \\ \vdots \end{pmatrix}}_{A_2} x_1\right) \quad \mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ \boldsymbol{w}_1 & \boldsymbol{w}_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) \quad \mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ \boldsymbol{w}_1 & \boldsymbol{w}_2 & \boldsymbol{w}_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$

- If $\mathbf{h}_i \in \mathbb{R}^d$, how many total parameters? Linear in n: weights $W \in \mathbb{R}^{d \times n}$, biases $c \in \mathbb{R}^d$, and n logistic regression coefficient vectors $\boldsymbol{\alpha}_i, b_i \in \mathbb{R}^{d+1}$. Probability is evaluated in $O(nd)$.

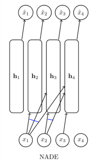
NADE results



Samples on the left. Conditional probabilities $\hat{x}_i$ on the right.

Figure from *The Neural Autoregressive Distribution Estimator, 2011*.

General discrete distributions



How to model non-binary discrete random variables $X_i \in \{1, \cdots, K\}$?

E.g., pixel intensities varying from 0 to 255

One solution: Let $\hat{\boldsymbol{x}}_i$ parameterize a categorical distribution

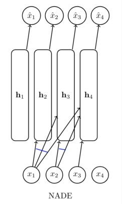
$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$
$$p(x_i|x_1, \cdots, x_{i-1}) = Cat(p_i^1, \cdots, p_i^K)$$
$$\hat{\mathbf{x}}_i = (p_i^1, \cdots, p_i^K) = softmax(A_i\mathbf{h}_i + \mathbf{b}_i)$$

Softmax generalizes the sigmoid/logistic function $\sigma(\cdot)$ and transforms a vector of K numbers into K *probabilities* (non-negative, sum=1).

$$softmax(\mathbf{a}) = softmax(a^1, \cdots, a^K) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)}, \cdots, \frac{\exp(a^K)}{\sum_i \exp(a^i)} \right)$$

In numpy: `np.exp(a)/np.sum(np.exp(a))`

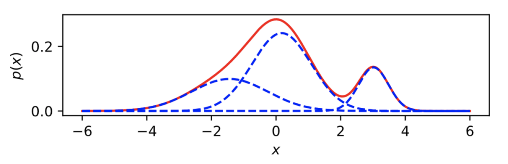
RNADE



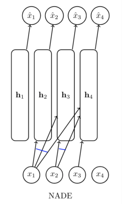
How to model continuous random variables $X_i \in \mathbb{R}$? E.g., speech signals

Solution: let $\hat{\boldsymbol{x}}_i$ parameterize a continuous distribution

E.g., uniform mixture of K Gaussians



RNADE



How to model continuous random variables $X_i \in \mathbb{R}$? E.g., speech signals

Solution: let $\hat{\mathbf{x}}_i$ parameterize a continuous distribution

E.g., In a mixture of K Gaussians,

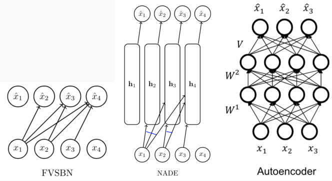
$$p(x_i|x_1, \cdots, x_{i-1}) = \sum_{j=1}^{K} \frac{1}{K} \mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = (\mu_i^1, \cdots, \mu_i^K, \sigma_i^1, \cdots, \sigma_i^K) = f(\mathbf{h}_i)$$

$\hat{\mathbf{x}}_i$ defines the mean and stddev of each Gaussian (μ_i^j, σ_i^j). Can use exponential $\exp(\cdot)$ to ensure stddev is positive

Autoregressive models vs. autoencoders



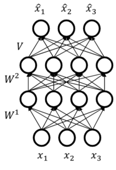
- FVSBN and NADE look similar to an **autoencoder**:
  - an **encoder** $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1x + b^1) + b^2)$
  - a **decoder** such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.
- Loss function for dataset $\mathcal{D}$

$$\text{Binary r.v.:} \quad \min_{W^1,W^2,b^1,b^2,V,c} \sum_{x\in\mathcal{D}} \sum_i -x_i \log \hat{x}_i - (1 - x_i)\log(1 - \hat{x}_i)$$

$$\text{Continuous r.v.:} \quad \min_{W^1,W^2,b^1,b^2,V,c} \sum_{x\in\mathcal{D}} \sum_i (x_i - \hat{x}_i)^2$$

- e and d are constrained so that we don't learn identity mappings. Hope that $e(x)$ is a meaningful, compressed representation of x (feature learning)
- A vanilla autoencoder is *not* a generative model: it does not define a distribution over x we can sample from to generate new data points.

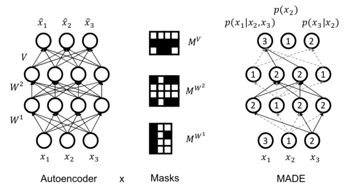
Autoregressive autoencoders



Autoencoder

- FVSBN and NADE look similar to an **autoencoder**. Can we get a generative model from an autoencoder?
- We need to make sure it corresponds to a valid BayesNet (DAG structure), i.e., we need an *ordering*. If the ordering is $1, 2, 3$, then:
  - $\hat{x}_1$ cannot depend on any input x. Then at generation time we don't need any input to get started
  - $\hat{x}_2$ can only depend on x_1
  - $\cdots$
- **Bonus**: we can use a single neural network (with n outputs) to produce all the parameters. In contrast, NADE requires n passes. Much more efficient on modern hardware.

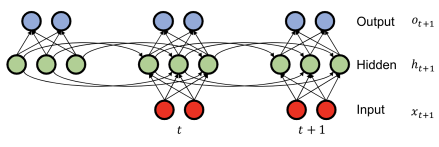
MADE: Masked Autoencoder for Distribution Estimation



1. **Challenge**: An autoencoder that is autoregressive (DAG structure)
2. **Solution**: use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1.
   2.1 The unit producing the parameters for $p(x_2)$ is not allowed to depend on any input. Unit for $p(x_3|x_2)$ only on x_2. And so on...
   2.2 For each unit in a hidden layer, pick a random integer i in $[1, n-1]$. That unit is allowed to depend only on the first i inputs (according to the chosen ordering).
   2.3 Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly $<$ in final layer)

RNN: Recurrent Neural Nets

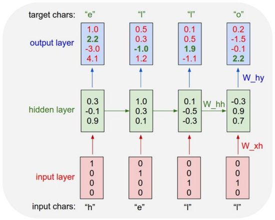
Challenge: model $p(x_t|x_{1:t-1};\boldsymbol{\alpha}^t)$. "History" $x_{1:t-1}$ keeps getting longer. **Idea**: keep a summary and recursively update it



$$
\begin{aligned}
\text{Summary update rule:} \quad h_{t+1} &= tanh(W_{hh}h_t + W_{xh}x_{t+1}) \\
\text{Prediction:} \quad o_{t+1} &= W_{hy}h_{t+1} \\
\text{Summary initalization:} \quad h_0 &= \boldsymbol{b}_0
\end{aligned}
$$

1. Hidden layer h_t is a summary of the inputs seen till time t
2. Output layer o_{t-1} specifies parameters for conditional $p(x_t \mid x_{1:t-1})$
3. Parameterized by $\boldsymbol{b}_0$ (initialization), and matrices W_{hh}, W_{xh}, W_{hy}. Constant number of parameters w.r.t n!

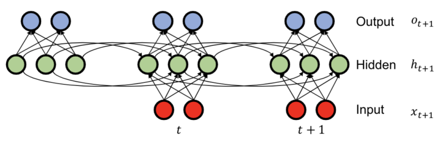
Example: Character RNN (from Andrej Karpathy)



1. Suppose $x_i \in \{h, e, l, o\}$. Use one-hot encoding: h encoded as $[1, 0, 0, 0]$, e encoded as $[0, 1, 0, 0]$, etc.
2. **Autoregressive**:
   $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 =$
3. $h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$

$$
\begin{aligned}
p(x_2 = e|x_1 = h) &= softmax(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.1)} \\
o_1 &= W_{hy} h_1 \\
h_1 &= tanh(W_{hh} h_0 + W_{xh} x_1)
\end{aligned}
$$

RNN: Recurrent Neural Nets



Pros:

1. Can be applied to sequences of arbitrary length.
2. Very general: For every computable function, there exists a finite RNN that can compute it

Cons:

1. Still requires an ordering
2. Sequential likelihood evaluation (very slow for training)
3. Sequential generation (unavoidable in an autoregressive model)

Example: Character RNN (from Andrej Karpathy)

Train 3-layer RNN with 512 hidden nodes on all the works of Shakespeare. Then sample from the model:

> KING LEAR: O, if you were a feeble sight, the courtesy of your law,
> Your sight and several breath, will wear the gods
> With his heads, and my hands are wonder'd at the deeds,
> So drop upon your lordship's head, and your opinion
> Shall be against your honour.

Note: generation happens **character by character**. Needs to learn valid words, grammar, punctuation, etc.

Example: Character RNN (from Andrej Karpathy)

Train on Wikipedia. Then sample from the model:

> Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25—21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict.

Note: correct Markdown syntax. Opening and closing of brackets

Example: Character RNN (from Andrej Karpathy)

Train on Wikipedia. Then sample from the model:

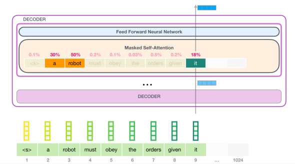
{ { cite journal — id=Cerling Nonforest Department—format=Newlymeslated—none } }
"www.e-complete".
"'See also"': [[List of ethical consent processing]]
== See also ==
*[[Iender dome of the ED]]
*[[Anti-autism]]
== External links==
* [http://www.biblegateway.nih.gov/entrepre/ Website of the World Festival. The labour of India-county defeats at the Ripper of California Road.]

Example: Character RNN (from Andrej Karpathy)

Train on data set of baby names. Then sample from the model:

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

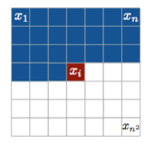
Generative Transformers



Current state of the art (GPTs): replace RNN with Transformer

- Attention mechanisms to adaptively focus only on relevant context
- Avoid recursive computation. Use only self-attention to enable parallelization
- Needs **masked** self-attention to preserve autoregressive structure
- Demo: `https://transformer.huggingface.co/doc/gpt2-large`

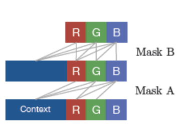
Pixel RNN (Oord et al., 2016)



1. Model images pixel by pixel using raster scan order
2. Each pixel conditional $p(x_t \mid x_{1:t-1})$ needs to specify 3 colors

   $$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$

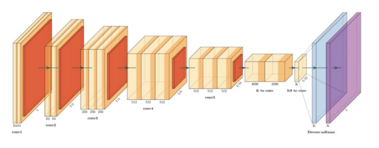
   and each conditional is a categorical random variable with 256 possible values
3. Conditionals modeled using LSTMs (an RNN variant) + masking (like MADE)



Pixel RNN

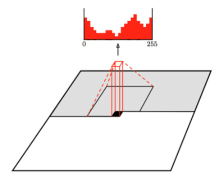
Results on downsampled ImageNet. Very slow: sequential likelihood evaluation.

Convolutional Architectures



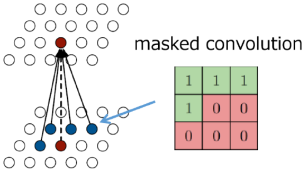
Convolutions are natural for image data and easy to parallelize on modern hardware.

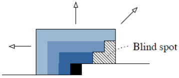
PixelCNN (Oord et al., 2016)



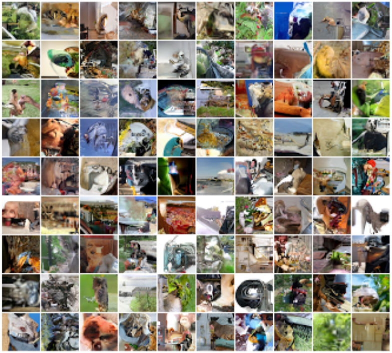
Idea: Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).

Challenge: Has to be autoregressive. Masked convolutions preserve raster scan order. Additional masking for colors order.





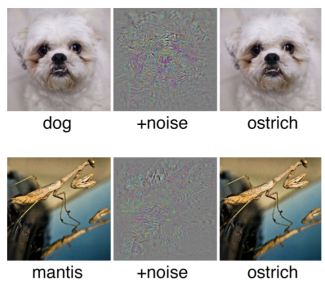
PixelCNN



Samples from the model trained on Imagenet (32×32 pixels).
Similar performance to PixelRNN, but much faster.

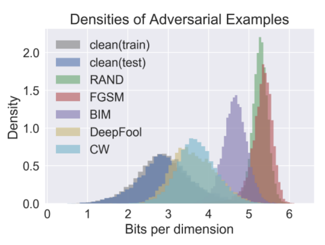
Application in Adversarial Attacks and Anomaly detection

Machine learning methods are vulnerable to adversarial examples



Can we detect them?

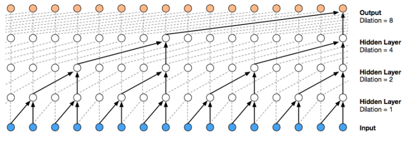
PixelDefend (Song et al., 2018)



- Train a generative model $p(x)$ on clean inputs (PixelCNN)
- Given a new input $\overline{x}$, evaluate $p(\overline{x})$
- Adversarial examples are significantly less likely under $p(x)$

WaveNet (Oord et al., 2016)

State of the art model for speech:



Dilated convolutions increase the receptive field: kernel only touches the signal at every 2^d entries.

Summary of Autoregressive Models

- Easy to sample from
    1. Sample $\overline{x}_0 \sim p(x_0)$
    2. Sample $\overline{x}_1 \sim p(x_1 \mid x_0 = \overline{x}_0)$
    3. $\cdots$
- Easy to compute probability $p(x = \overline{x})$
    1. Compute $p(x_0 = \overline{x}_0)$
    2. Compute $p(x_1 = \overline{x}_1 \mid x_0 = \overline{x}_0)$
    3. Multiply together (sum their logarithms)
    4. $\cdots$
    5. Ideally, can compute all these terms in parallel for fast training
- Easy to extend to continuous variables. For example, can choose Gaussian conditionals $p(x_t \mid x_{<t}) = \mathcal{N}(\mu_\theta(x_{<t}), \Sigma_\theta(x_{<t}))$ or mixture of logistics
- No natural way to get features, cluster points, do unsupervised learning
- Next: learning