Autoregressive Models

Stefano Ermon, Aditya Grover

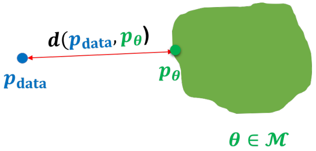
Stanford University

Lecture 3

Learning a generative model

- We are given a training set of examples, e.g., images of dogs

$\mathbf{x}^{(j)} \sim p_{\text{data}}$
$j = 1, 2, \ldots, |\mathcal{D}|$



Model family

- We want to learn a probability distribution $p(x)$ over images x such that
  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*)
- First question: how to represent $p(x)$. Second question: how to learn it.

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

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

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

- Neural Models

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

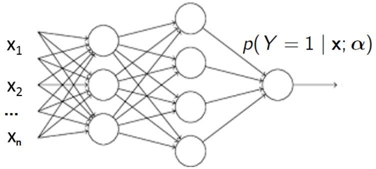
Assumes specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.

Neural Models for classification

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

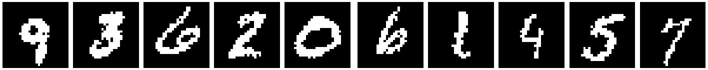
$$p(Y = 1 \mid \mathbf{x}; \boldsymbol{\alpha}) = f(\mathbf{x}, \boldsymbol{\alpha})$$

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



Motivating Example: MNIST

- Suppose we have 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).
- We want to learn a probability distribution $p(v) = p(v_1, \cdots, v_{784})$ over $v \in \{0,1\}^{784}$ such that when $v \sim p(v)$, v looks like a digit
- Idea: define a model family $\{p_\theta(v), \theta \in \Theta\}$, then pick a good one based on training data $\mathcal{D}$ (more on that later)
- How to parameterize $p_\theta(v)$?

Fully Visible Sigmoid Belief Network

- We can pick an ordering, i.e., order variables (pixels) from top-left (X_1) to bottom-right ($X_{n=784}$)
- Use chain rule factorization without loss of generality:

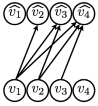
$$p(v_1, \cdots, v_{784}) = p(v_1)p(v_2 \mid v_1)p(v_3 \mid v_1, v_2) \cdots p(v_n \mid v_1, \cdots, v_{n-1})$$

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

$$p(v_1, \cdots, v_{784}) = p_{\mathrm{CPT}}(v_1; \alpha^1) p_{\mathrm{logit}}(v_2 \mid v_1; \boldsymbol{\alpha}^2) p_{\mathrm{logit}}(v_3 \mid v_1, v_2; \boldsymbol{\alpha}^3) \cdots p_{\mathrm{logit}}(v_n \mid v_1, \cdots, v_{n-1}; \boldsymbol{\alpha}^n)$$

- More explicitly:
  - $p_{\mathrm{CPT}}(V_1 = 1; \alpha^1) = \alpha^1$, $p(V_1 = 0) = 1 - \alpha^1$
  - $p_{\mathrm{logit}}(V_2 = 1 \mid v_1; \boldsymbol{\alpha}^2) = \sigma(\boldsymbol{\alpha}_0^2 + \boldsymbol{\alpha}_1^2 v_1)$
  - $p_{\mathrm{logit}}(V_3 = 1 \mid v_1, v_2; \boldsymbol{\alpha}^3) = \sigma(\boldsymbol{\alpha}_0^3 + \boldsymbol{\alpha}_1^3 v_1 + \boldsymbol{\alpha}_2^3 v_2)$
- Note: This is a **modeling assumption**. We are using a logistic regression to predict next pixel based on the previous ones. Called **autoregressive**.

Fully Visible Sigmoid Belief Network



FVSBN

- The conditional variables $V_i \mid V_1, \cdots, V_{i-1}$ are Bernoulli with parameters

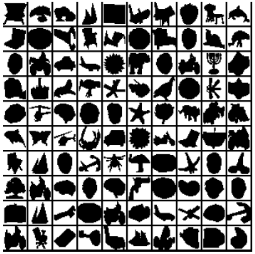
$$\hat{v}_i = p(V_i = 1 | v_1, \cdots, v_{i-1}; \boldsymbol{\alpha}^i) = p(V_i = 1 | v_{<i}; \boldsymbol{\alpha}^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i v_j)$$

- How to evaluate $p(v_1, \cdots, v_{784})$? Multiply all the conditionals (factors)
  - In the above example:

$$p(V_1 = 0, V_2 = 1, V_3 = 1, V_4 = 0) = (1 - \hat{v}_1) \times \hat{v}_2 \times \hat{v}_3 \times (1 - \hat{v}_4)$$
$$= (1 - \hat{v}_1) \times \hat{v}_2(V_1 = 0) \times \hat{v}_3(V_1 = 0, V_2 = 1) \times (1 - \hat{v}_4(V_1 = 0, V_2 = 1, V_3 = 1))$$

- How to sample from $p(v_1, \cdots, v_{784})$?
  1. Sample $\overline{v}_1 \sim p(v_1)$ (`np.random.choice([1,0],p=[`$\hat{v}_1, 1 - \hat{v}_1$`]))`)
  2. Sample $\overline{v}_2 \sim p(v_2 \mid v_1 = \overline{v}_1)$
  3. Sample $\overline{v}_3 \sim p(v_3 \mid v_1 = \overline{v}_1, v_2 = \overline{v}_2) \cdots$
- How many parameters? $1 + 2 + 3 \cdots + n \approx n^2/2$

FVSBN Results



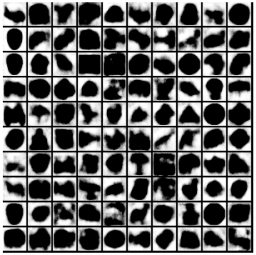
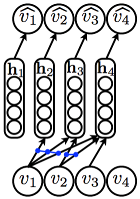


Figure from *Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015*. Training data on the left (*Caltech 101 Silhouettes*). Samples from the model on the right. Best performing model they tested on this dataset in 2015 (more on evaluation later).

NADE: Neural Autoregressive Density Estimation



$\widehat{v_1}$ $\widehat{v_2}$ $\widehat{v_3}$ $\widehat{v_4}$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$
v_1 v_2 v_3 v_4

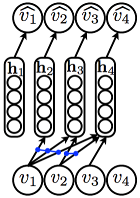
NADE

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

$$\mathbf{h}_i = \sigma(A_i \mathbf{v}_{<i} + \mathbf{c}_i)$$
$$\hat{v}_i = p(v_i | v_1, \cdots, v_{i-1}; \underbrace{A_i, \mathbf{c}_i, \boldsymbol{\alpha}_i, b_i}_{\text{parameters}}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

- For example $\mathbf{h}_2 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{A_2} v_1 + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{c_2} \right)$ $\mathbf{h}_3 = \sigma \left(\underbrace{\begin{pmatrix} \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} v_1 & v_2 \end{pmatrix} + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{c_3} \right)$

NADE: Neural Autoregressive Density Estimation



$\widehat{v_1}$ $\widehat{v_2}$ $\widehat{v_3}$ $\widehat{v_4}$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$
v_1 v_2 v_3 v_4

NADE

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

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

$$\hat{v}_i = p(v_i|v_1, \cdots, v_{i-1}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

- For example $\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ \mathbf{w}_1 \\ \vdots \end{pmatrix}}_{A_2} v_1\right)$ $\mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} v_1 & v_2 \end{pmatrix}\right)$ $\mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}\right)$

- How many parameters? Linear in n: $W \in \mathbb{R}^{H\times n}$, and n logistic regression coefficient vectors $\boldsymbol{\alpha}_i, b_i \in \mathbb{R}^{H+1}$. Probability is evaluated in $O(nH)$.

NADE results

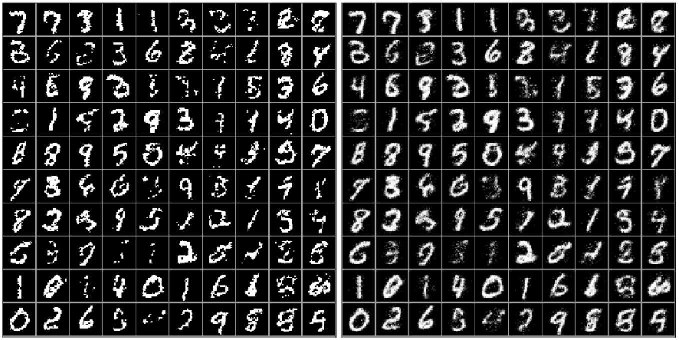
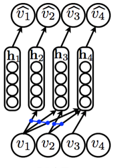


Figure from *The Neural Autoregressive Distribution Estimator, 2011*.
Samples on the left. Conditional probabilities $\hat{v}_i$ on the right.

General discrete distributions



NADE

How to model non-binary discrete random variables $V_i \in \{1, \cdots, K\}$ (e.g., color images)? Solution: let $\hat{\boldsymbol{v}}_i$ parameterize a categorical distribution

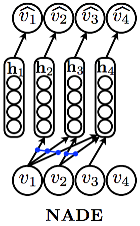
$$\begin{aligned} \mathbf{h}_i &= \sigma(W_{\cdot,<i}\mathbf{v}_{<i} + \mathbf{c}) \\ p(v_i|v_1, \cdots, v_{i-1}) &= Cat(p_i^1, \cdots, p_i^K) \\ \hat{\boldsymbol{v}}_i = (p_i^1, \cdots, p_i^K) &= softmax(V_i\mathbf{h}_i + \mathbf{b}_i) \end{aligned}$$

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

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

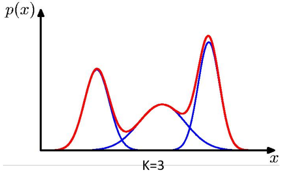
```
np.exp(a)/np.sum(np.exp(a))
```

RNADE



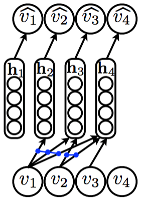
$\hat{v}_1$ $\hat{v}_2$ $\hat{v}_3$ $\hat{v}_4$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$
v_1 v_2 v_3 v_4
NADE

How to model continuous random variables $V_i \in \mathbb{R}$ (e.g., speech signals)? Solution: let $\hat{\boldsymbol{v}}_i$ parameterize a continuous distribution (e.g., mixture of K Gaussians)



$\hat{\boldsymbol{v}}_i$ needs to specify the mean and variance of each Gaussian

RNADE



$\widehat{v_1}$ $\widehat{v_2}$ $\widehat{v_3}$ $\widehat{v_4}$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$
v_1 v_2 v_3 v_4

NADE

How to model continuous random variables $V_i \in \mathbb{R}$ (e.g., speech signals)? Solution: let $\hat{\boldsymbol{v}}_i$ parameterize a continuous distribution (e.g., mixture of K Gaussians)

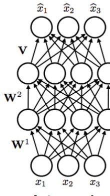
$$p(v_i | v_1, \cdots, v_{i-1}) = \sum_{j=1}^{K} \frac{1}{K} \mathcal{N}(v_i; \mu_i^j, \sigma_i^j)$$

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

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

$\hat{\boldsymbol{v}}_i$ defines the mean and variance of each Gaussian (μ^j, σ^j). Can use exponential $\exp(\cdot)$ to ensure variance is non-negative

Autoregressive models vs. autoencoders



Autoencoder

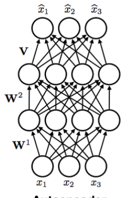
- On the surface, FVSBN and NADE look similar to an **autoencoder**:
- an **encoder** $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1x + b^1) + b^2)$
- a **decoder** such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.

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

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

- 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.

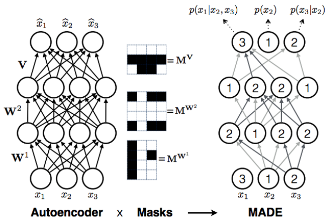
Autoregressive vs. autoencoders



Autoencoder

- On the surface, 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 Bayesian Network (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 Autoregressive density estimator

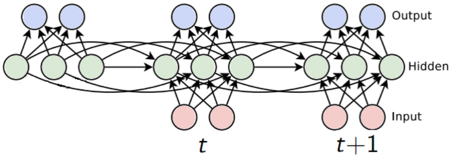


$p(x_1|x_2,x_3)$
$p(x_2)$
$p(x_3|x_2)$
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$
$\mathbf{V}$
$\mathbf{W}^2$
$\mathbf{W}^1$
x_1 x_2 x_3
$= \mathbf{M}^{\mathbf{V}}$
$= \mathbf{M}^{\mathbf{W}^2}$
$= \mathbf{M}^{\mathbf{W}^1}$
Autoencoder x Masks → MADE

1. Parameter sharing: use a single multi-layer neural network
2. **Challenge**: need to make sure it's autoregressive (DAG structure)
3. **Solution**: use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1.
    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. For each unit, pick a number i in $[1, n-1]$. That unit is only allowed to depend only on the first i inputs (according to the chosen ordering).
    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: Recursive 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

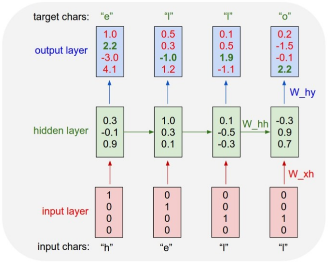


o_{t+1}
h_{t+1}
x_{t+1}

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

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!

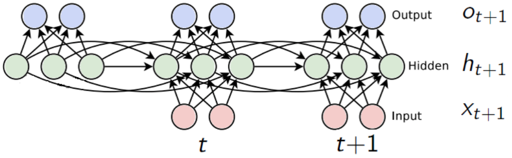
Example: Character RNN (from Andrej Karpathy)



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**: $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$
3. For example,

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

RNN: Recursive 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)
4. Can be difficult to train (vanishing/exploding gradients)

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:

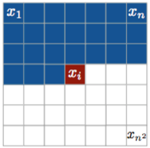
```
{ { cite journal — id=Cerling Nonforest Department—format=Newlymeslated—none } }
"www.e-complete".
"'See also"': [[List of ethical consent processing]]

== 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)

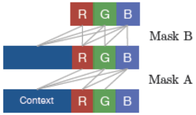
Train on data set of baby names. Then sample from the model:

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

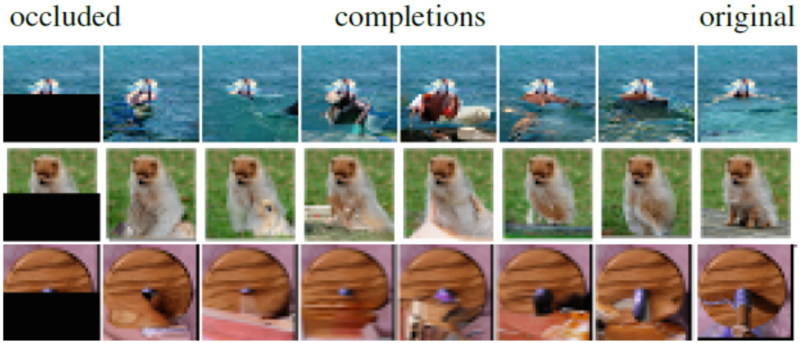
Pixel RNN (van den Oord, 2016)



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 RNN variants. LSTMs + masking (like MADE)

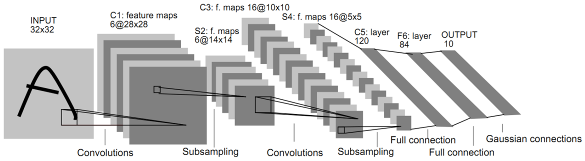


Pixel RNN results



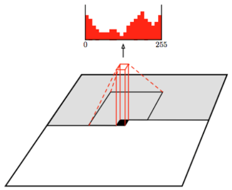
Results on downsampled ImageNet. Very slow: sequential likelihood evaluation.

Convolutional Architectures

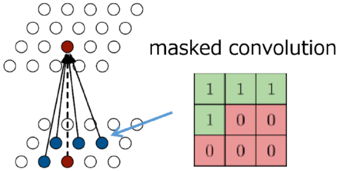


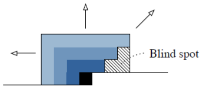
Convolutions are natural for image data, and easy to parallelize on modern hardware

PixelCNN

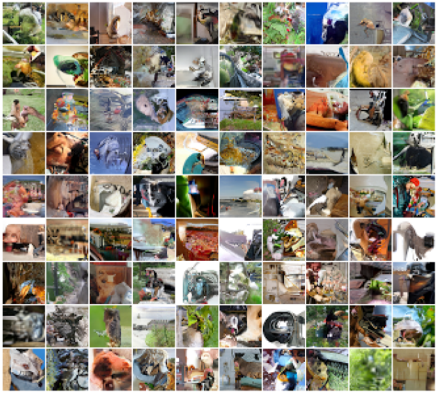


Idea: Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).
Challenge: Has to be autoregressive. Masked convolutions preserve raster scan order. Additional masking for colors order.





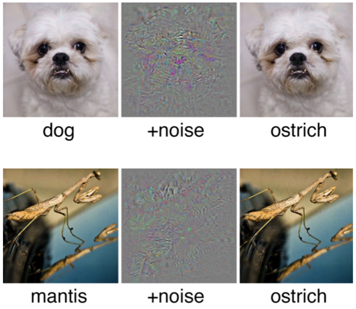
PixelCNN



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

Adversarial Attacks and Anomaly detection

Machine learning methods are vulnerable to adversarial examples



Can we detect them?

Pixel Defend

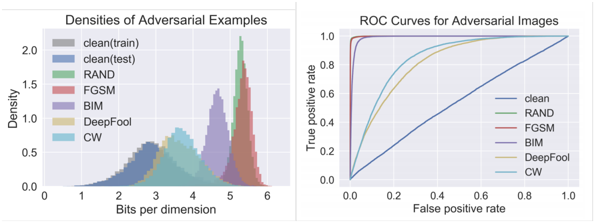
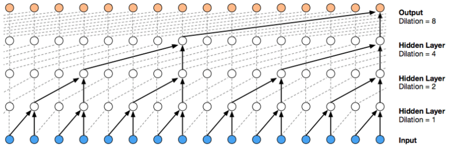


Figure: (Left) *Likelihoods of different adversarial examples.* (Right) *ROC curves for detecting various attacks.*

- 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

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