

Deep Generative Models (Fall 2024)

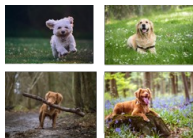
CS HUFS

Autoregressive Models

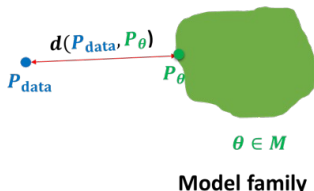
- Autoregressive Models
- Fully Visible Sigmoid Belief Network (FVSBN)
- Neural Autoregressive Density Estimation (NADE)
- Autoregressive Autoencoders
- Masked Autoencoder for Distribution Estimation (MADE)
- Recurrent Neural Networks (RNN)
- Generative Transformers
- PixelRNN, PixelCNN, PixelDefend, WaveNet

Learning a generative model

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



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



- We want to learn a probability distribution $p(x)$ over images x such that
 - 1 **Generation:** If we sample $x_{\text{new}} \sim p(x)$, 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

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

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

- Neural Models

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

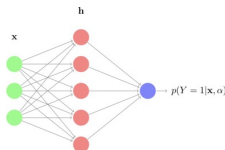
Assumes specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.

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 $p(Y | \mathbf{x})$, and assume that

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

- Logistic regression:** let $z(\alpha, \mathbf{x}) = \alpha_0 + \sum_{i=1}^n \alpha_i x_i$
 $p_{\text{logit}}(Y = 1 | \mathbf{x}; \alpha) = \sigma(z(\alpha, \mathbf{x}))$, 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. $p_{\text{Neural}}(Y = 1 | \mathbf{x}; \alpha, A, \mathbf{b}) = \sigma(\alpha_0 + \sum_{i=1}^h \alpha_i h_i)$
 - More flexible
 - More parameters: A, \mathbf{b}, α
 - Repeat multiple times to get a multilayer perceptron (neural network)



Motivating Example: MNIST

- **Given:** a dataset 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 $p(x) = p(x_1, \dots, x_{784})$ 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 D [Next lecture]

Autoregressive Models

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

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

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

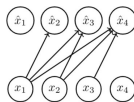
$$p(x_1, \dots, x_{784}) = p_{\text{CPT}}(x_1; \alpha^1)p_{\text{logit}}(x_2 | x_1; \alpha^2)p_{\text{logit}}(x_3 | x_1, x_2; \alpha^3) \dots p_{\text{logit}}(x_n | x_1, \dots, x_{n-1}; \alpha^n)$$

- More explicitly

- $p_{\text{CPT}}(X_1 = 1; \alpha^1) = \alpha^1, p(X_1 = 0) = 1 - \alpha^1$
- $p_{\text{logit}}(X_2 = 1 | x_1; \alpha^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1)$
- $p_{\text{logit}}(X_3 = 1 | x_1, x_2; \alpha^3) = \sigma(\alpha_0^3 + \alpha_1^3 x_1 + \alpha_2^3 x_2)$

- 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 $X_i \mid X_1, \dots, X_{i-1}$ are Bernoulli with parameters

$$\hat{x}_i = p(X_i = 1 \mid x_1, \dots, x_{i-1}; \alpha^i) = p(X_i = 1 \mid x_{<i}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

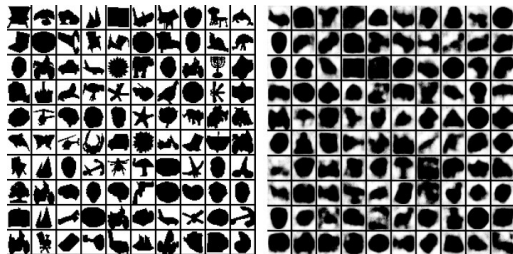
- How to evaluate $p(x_1, \dots, x_{784})$? Multiply all the conditionals (factors)
 - In the above example:

$$\begin{aligned} p(X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0) &= (1 - \hat{x}_1) \times \hat{x}_2 \times \hat{x}_3 \times (1 - \hat{x}_4) \\ &= (1 - \hat{x}_1) \times \hat{x}_2(X_1 = 0) \times \hat{x}_3(X_1 = 0, X_2 = 1) \times (1 - \hat{x}_4(X_1 = 0, X_2 = 1, X_3 = 1)) \end{aligned}$$

- How to sample from $p(x_1, \dots, x_{784})$?

- 1 Sample $\bar{x}_1 \sim p(x_1)$ (`np.random.choice([1, 0], p=[\hat{x}_1 , $1 - \hat{x}_1$])`)
- 2 Sample $\bar{x}_2 \sim p(x_2 \mid x_1 = \bar{x}_1)$
- 3 Sample $\bar{x}_3 \sim p(x_3 \mid x_1 = \bar{x}_1, x_2 = \bar{x}_2) \dots$

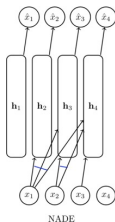
- How many parameters (in the α^i vectors)? $1 + 2 + 3 + \dots + n \approx n^2/2$



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



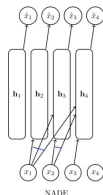
- To improve model: use one layer neural network instead of logistic regression

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

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

- For example $\mathbf{h}_2 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{\mathbf{A}_2} x_1 + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{\mathbf{c}_2} \right)$ $\mathbf{h}_3 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \vdots \end{pmatrix}}_{\mathbf{A}_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} \vdots \end{pmatrix}}_{\mathbf{c}_3} \right)$

NADE: Neural Autoregressive Density Estimation



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

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

$$x_i = p(x_i | x_1, \dots, x_{i-1}) = \sigma(\alpha_i \mathbf{h}_i + b_i)$$

- For example
$$\mathbf{h}_2 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ \mathbf{w}_1 \\ \vdots \end{pmatrix}}_{W_{\cdot, < 2}} x_1 + \mathbf{c} \right) \quad \mathbf{h}_3 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 \\ \vdots \end{pmatrix}}_{W_{\cdot, < 3}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) \quad \mathbf{h}_4 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 \\ \vdots \end{pmatrix}}_{W_{\cdot, < 4}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \right)$$

- 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 $\alpha_i, b_i \in \mathbb{R}^{d+1}$. Probability is evaluated in $O(nd)$.

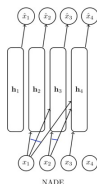
NADE results



Samples from a model trained on MNIST 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 $X_i \in \{1, \dots, K\}$? E.g., pixel intensities varying from 0 to 255

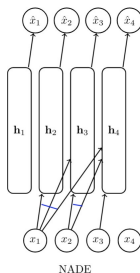
One solution: Let $\hat{\mathbf{x}}_i$ parameterize a categorical distribution

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

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).

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

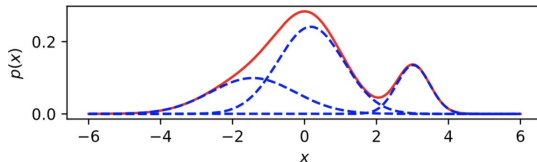
In numpy: `np.exp(a)/np.sum(np.exp(a))`

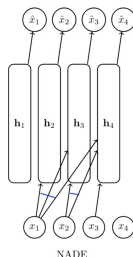


How to model continuous random variables $X_i \in \mathbb{R}$? E.g., speech signals

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

E.g., uniform mixture of K Gaussians





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,

$$p(x_i | x_1, \dots, x_{i-1}) = \sum_{j=1}^K \frac{1}{K} N(x_i; \mu_j^i, \sigma_j^i)$$

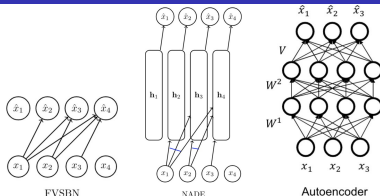
$$\mathbf{h}_i = \sigma(W_{\cdot, < i} \mathbf{x}_{< i} + \mathbf{c})$$

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

$\hat{\mathbf{x}}_i$ defines the mean and standard deviation of each of the K Gaussians (μ_j^i, σ_j^i) .

Can use exponential $\exp(\cdot)$ to ensure non-negativity

Autoregressive models vs. autoencoders



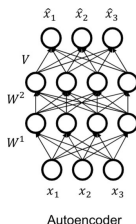
- On the surface, FVSN and NADE look similar to an **autoencoder**:
- an **encoder** $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1x + b^1) + b^2)$
- a **decoder** such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.
- Loss function for dataset D

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

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

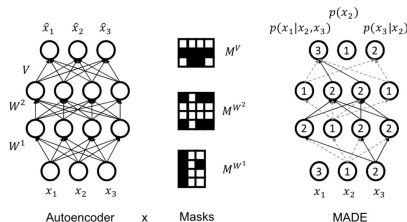
- e and d are constrained so that we don't learn identity mappings. Hope that $e(x)$ is a meaningful, compressed representation of x (feature learning)
- A vanilla autoencoder is *not* a generative model: it does not define a distribution over x we can sample from to generate new data points.

Autoregressive autoencoders



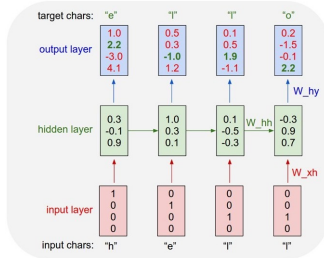
- On the surface, FVSBN and NADE look similar to an **autoencoder**. Can we get a generative model from an autoencoder?
- We need to make sure it corresponds to a valid Bayesian Network (DAG structure), i.e., we need an *ordering* for chain rule. If ordering is 1, 2, 3, then:
 - \hat{x}_1 cannot depend on any input $x = (x_1, x_2, x_3)$. Then at generation time we don't need any input to get started
 - \hat{x}_2 can only depend on x_1
 - ...
- Bonus:** we can use a single neural network (with n inputs and outputs) to produce all the parameters \hat{x} in a single pass. In contrast, NADE requires n passes. Much more efficient on modern hardware.

MADE: Masked Autoencoder for Distribution Estimation



- Challenge:** An autoencoder that is autoregressive (DAG structure)
- Solution:** use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1 , so $p(x_1, x_2, x_3) = p(x_2)p(x_3 | x_2)p(x_1 | x_2, x_3)$.
 - The unit producing the parameters for $\hat{x}_2 = p(x_2)$ is not allowed to depend on any input. Unit for $p(x_3 | x_2)$ only on x_2 . And so on...
 - 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).
 - Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly $<$ in final layer)

Example: Character RNN (from Andrej Karpathy)



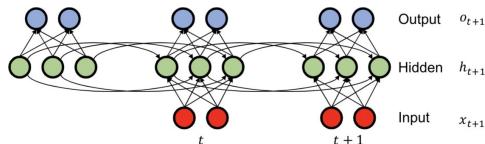
- Suppose $x_i \in \{h, e, l, o\}$. Use one-hot encoding:
 - h encoded as $[1, 0, 0, 0]$, e encoded as $[0, 1, 0, 0]$, etc.
- Autoregressive:** $p(x = \text{hello}) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$
- For example,

$$p(x_2 = e|x_1 = h) = \text{softmax}(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.1)}$$

$$o_1 = W_{hy}h_1$$

$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1)$$

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:

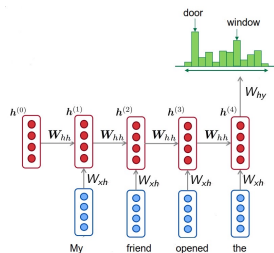
```
{ { cite journal — id=Cerling Nonforest Depart-  
ment—format=Newlymeslated—none } }  
"www.e-complete".  
""See also"": [[List of ethical consent processing]]  
  
== See also ==  
*[[Iender dome of the ED]]  
*[[Anti-autism]]  
  
== External links==  
* http://www.biblegateway.nih.gov/entrepre/ 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 El-
lia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa
Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Ve-
len 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

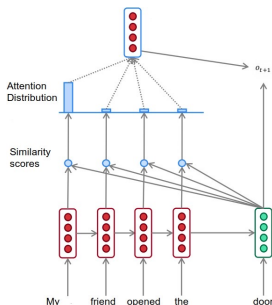
Issues with RNNs



Issues with RNN models

- A single hidden vector needs to summarize all the (growing) history. For example, $h^{(4)}$ needs to summarize the meaning of "My friend opened the".
- Sequential evaluation, cannot be parallelized
- Exploding/vanishing gradients when accessing information from many steps back

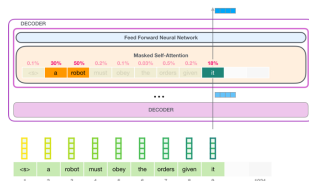
Attention based models



Attention mechanism to compare a **query** vector to a set of **key** vectors

- 1 Compare current hidden state (**query**) to all past hidden states (**keys**), e.g., by taking a dot product
- 2 Construct attention distribution to figure out what parts of the history are relevant, e.g., via a softmax
- 3 Construct a summary of the history, e.g., by weighted sum
- 4 Use summary and current hidden state to predict next token/word

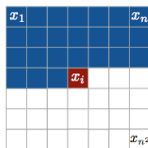
Generative Transformers



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

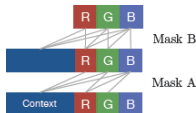
- Attention mechanisms to adaptively focus only on relevant context
- Avoid recursive computation. Use only self-attention to enable parallelization
- Needs **masked** self-attention to preserve autoregressive structure
- Demo: <https://transformer.huggingface.co/doc/gpt2-large>
- Demo: <https://huggingface.co/spaces/huggingface-projects/llama-2-13b-chat>

Pixel RNN (Oord et al., 2016)



- 1 Model images pixel by pixel using raster scan order
- 2 Each pixel conditional $p(x_t | x_{1:t-1})$ needs to specify 3 colors
$$p(x_t | x_{1:t-1}) = p(x_t^{red} | x_{1:t-1})p(x_t^{green} | x_{1:t-1}, x_t^{red})p(x_t^{blue} | x_{1:t-1}, x_t^{red}, x_t^{green})$$

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

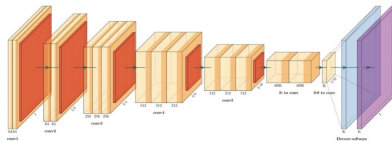


Pixel RNN



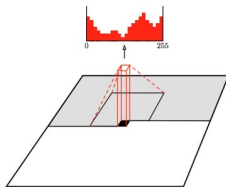
Results on downsampled ImageNet. Very slow: sequential likelihood evaluation.

Convolutional Architectures



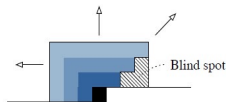
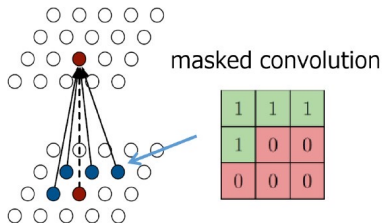
Convolutions are natural for image data and easy to parallelize on modern hardware.

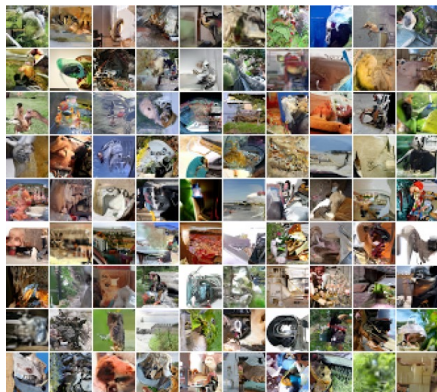
PixelCNN (Oord et al., 2016)



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.

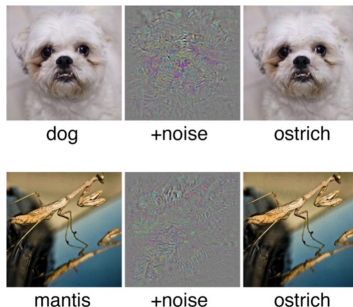




Samples from the model trained on Imagenet (32×32 pixels). Similar performance to PixelRNN, but much faster.

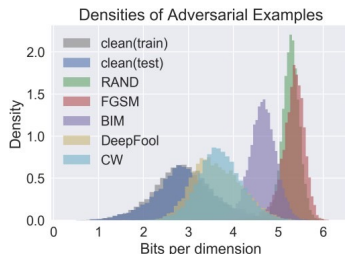
Application in Adversarial Attacks and Anomaly detection

Machine learning methods are vulnerable to adversarial examples



Can we detect them?

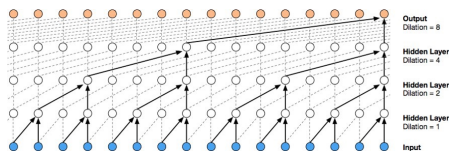
PixelDefend (Song et al., 2018)



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

WaveNet (Oord et al., 2016)

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