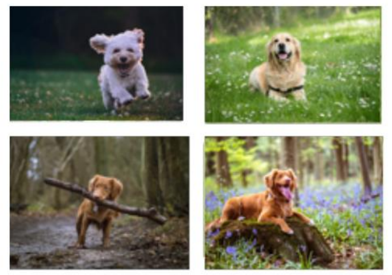
Autoregressive Models

Dr. Zulqarnain Khan

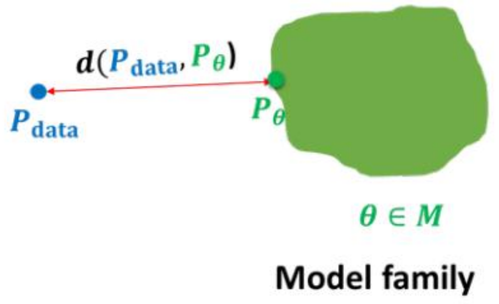
(Slides from: Stefano Ermon)

Learning a Generative Model

- We are given a training set of examples, e.g., images of dogs
- We want to learn a probability distribution $p(x)$ over images 'x' such that we can do:
  - **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like a dog (sampling)
  - **Density Estimation:** $p(x)$ should be high if 'x' looks like a dog, and low otherwise (anomaly detection)
  - **Unsupervised representation learning:** We should be able to learn what these images have in common, e.g., ears, tail, etc. (features) – possible with at least some of the generative models.
- First question: how to represent $p(x)$ – today we'll talk about AR models
- Second question: how to learn it!



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



Recap: Bayes Nets vs Neural Models

- Bayes Net (assumes conditional independencies)

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

- Neural Models (assume specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.)

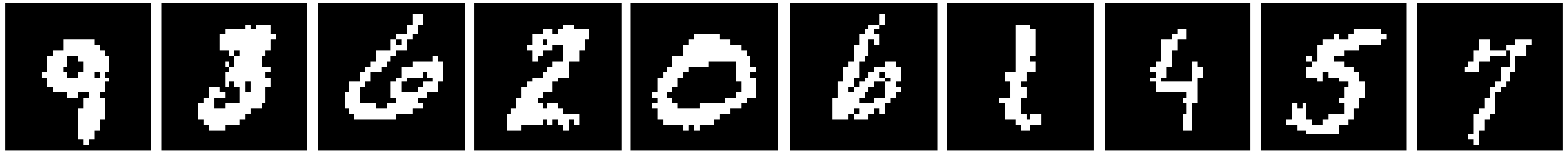
$$p(x_1, x_2, x_3, x_4) \approx p(x_1)p(x_2 \mid x_1)p_{\text{Neural}}(x_3 \mid x_1, x_2)p_{\text{Neural}}(x_4 \mid x_1, x_2, x_3)$$

Recap: Neural Models for Classification

- In discriminative models, for binary classification we assume that
$$p(Y = 1 \mid \mathbf{x}; \alpha) = f(\mathbf{x}, \alpha)$$
- Linear dependence: Let $z(\alpha, x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i$. Then, $p(Y = 1 \,|\boldsymbol{x}; \boldsymbol{\alpha}) = \sigma\big(z(\alpha, x)\big), where\ \sigma(z) = 1\backslash 1 + e^{-z}\ is\ the\ logistic\ function$. This dependence might be too simple.
- Non-Linear dependence: Let $h(A, b, x) = f(Ax + b)$ be a non-linear transformation of the inputs. Which makes $p(Y = 1 \,|\boldsymbol{x}; \boldsymbol{\alpha}, \boldsymbol{A}, \boldsymbol{b}) = f(\alpha_0 + \sum_{i=1}^{h} \alpha_i h_i)$
  - More flexible
  - More parameters: A, b, α
  - Can repeat multiple times to get a neural network

Running Example: MNIST

- **Given:** a dataset D of handwritten digits (binarized MNIST)



- Each image has n = 28 × 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. (Remember, 2^(784) possible images!)
- **Two Steps:**
  - Parameterize a model family $\{p_\theta(x), \theta \in \Theta\}$
  - Search for model parameters θ based on training data D

Autoregressive Models

- We can pick an ordering of all the random variables, e.g., 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:

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

- Avoid complex conditionals, assume a functional form:

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

- Specifically (if we use logistic regression):

$$p_{\mathrm{CPT}}(X_1 = 1; \alpha^1) = \alpha^1, \; p(X_1 = 0) = 1 - \alpha^1$$
$$p_{\mathrm{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\alpha}^2) = \sigma(\boldsymbol{\alpha}_0^2 + \boldsymbol{\alpha}_1^2 x_1)$$
$$p_{\mathrm{logit}}(X_3 = 1 \mid x_1, x_2; \boldsymbol{\alpha}^3) = \sigma(\boldsymbol{\alpha}_0^3 + \boldsymbol{\alpha}_1^3 x_1 + \boldsymbol{\alpha}_2^3 x_2)$$

Autoregressive Models

- Avoid complex conditionals, assume a functional form:

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

- Specifically (remember logistic regression):

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

$$\begin{tikzcd}
\text{Self (as opposed to an external label)} & \text{Autoregressive} \arrow[l] \arrow[r] & \text{Logistic Regression}
\end{tikzcd}$$

Autoregressive Models

- Fully Visible Sigmoid Belief Network (FVSBN)
- Neural Autoregressive Density Estimation (NADE)
- Real-valued NADE (RNADE)
- Masked Autoencoder for Distribution Estimation (MADE)
- Recurrent Neural Nets (RNNs)
- Convolutional Autoregressive models

Fully Visible Sigmoid Belief Network (FVSBN)

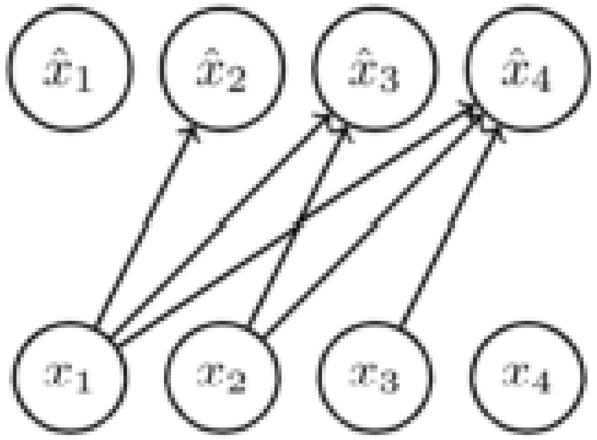
- The conditional variables $X_i | X_1, \dots, X_{i-1}$ are Bernoulli with parameters

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

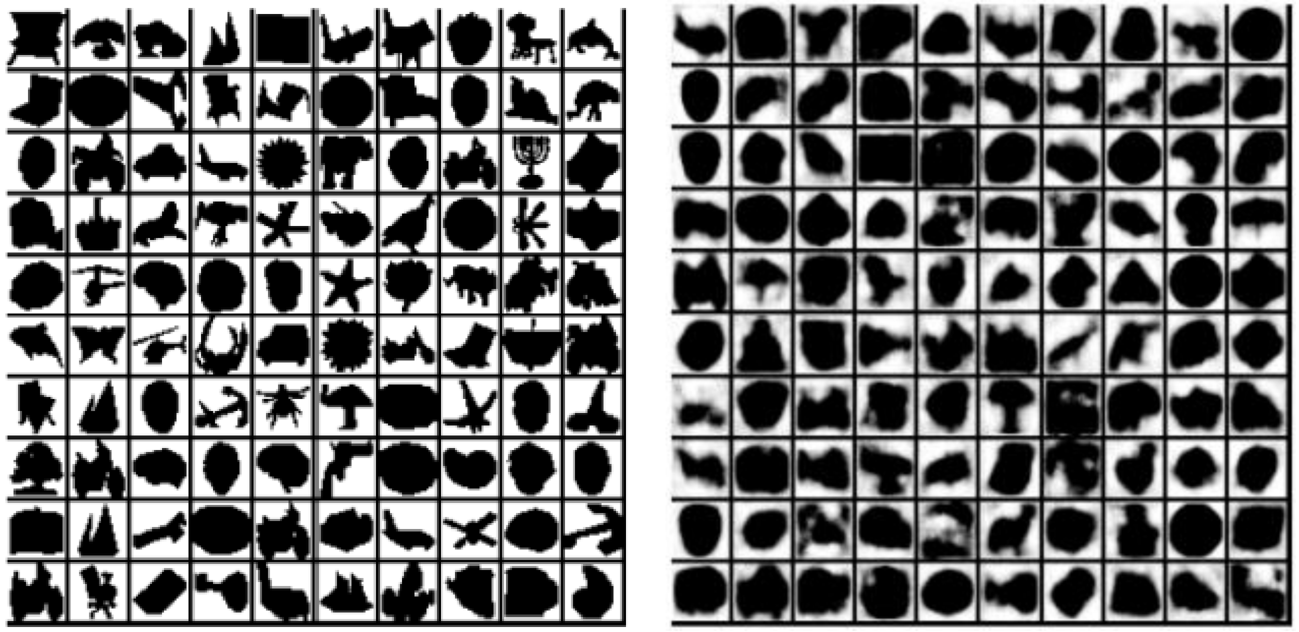
- How to evaluate $p(x_1, \dots, x_4)$? Multiply all the conditionals, e.g.

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

- How to sample $p(x_1, \dots, x_{784})$?
  - Sample $x_1' \sim p(x_1)$
  - Sample $x_2' \sim p(x_2 | x_1 = x_1')$
  - Sample $x_3' \sim p(x_3 | x_1 = x_1', x_2 = x_2')$ …
- How many parameters? $1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$



FVSBN Results



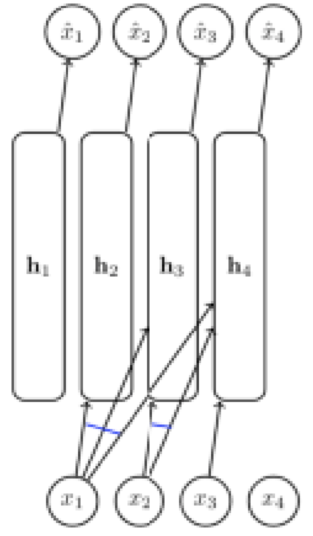
- Training data on the left (Caltech 101 Silhouettes). Samples from the model on the right. Figure from Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015.

NADE: Neural Autoregressive Density Estimation

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

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

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



NADE

- For example,

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

NADE

- Tie weights to reduce the number of parameters and speed up computation:

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

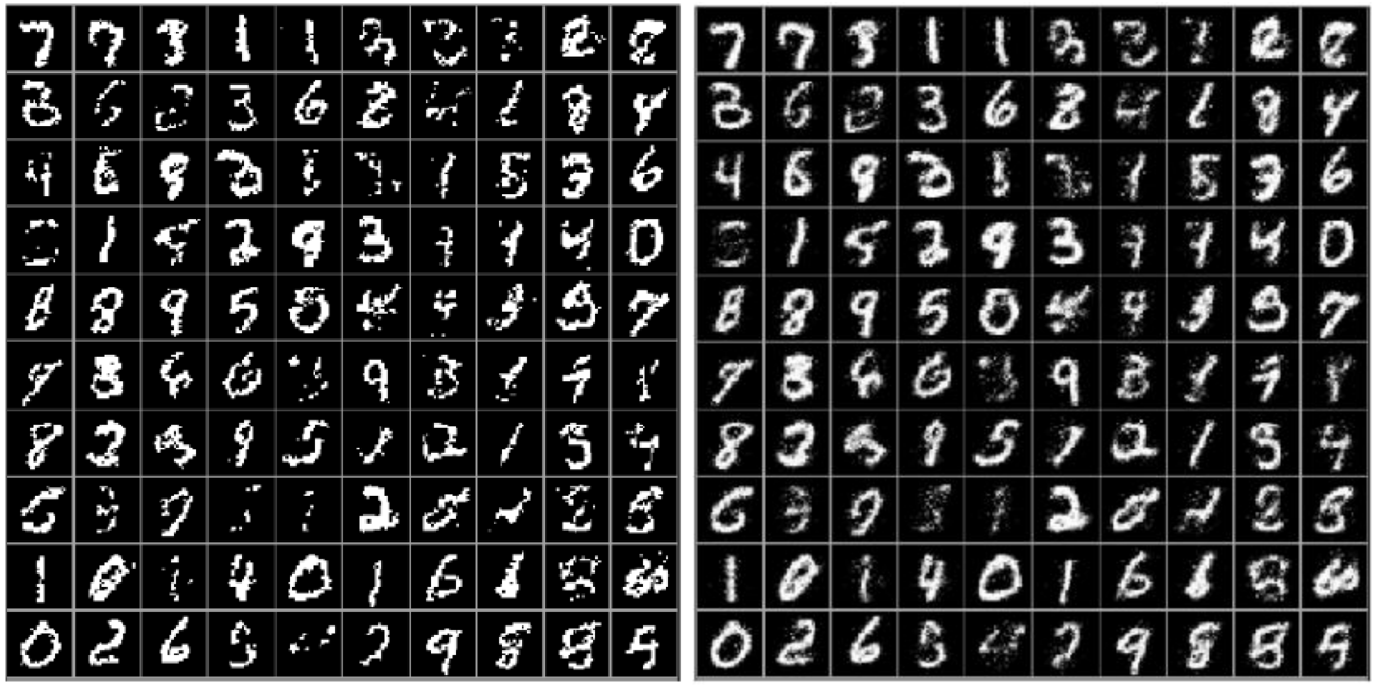
$$\hat{x}_i = p(x_i | x_1, \cdots, x_{i-1}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

- For example,

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

- If $h_i \in R^d$, how many total parameters? Linear in n: weights $W \in R^{d\times n}$, biases $c \in R^d$, and n logistic regression coefficient vectors $\alpha_i , b_i \in R^{d+1}$. And probability is evaluated in O(nd)

NADE Results



- Samples from a model trained on MNIST on the left. Conditional probabilities $\widehat{x_i}$ on the right. Figure from The Neural Autoregressive Distribution Estimator, 2011.

NADE for 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

Solution: Let $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ 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 σ(·) and transforms a vector of K numbers into a vector of K probabilities (non-negative, sum to 1).

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

Real-valued NADE (RNADE)

- How to model continuous random variables Xi ∈ R? E.g., speech signals

Solution: let $\hat{x}_i$ parameterize a continuous distribution E.g., In a mixture of K Gaussians,

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

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

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

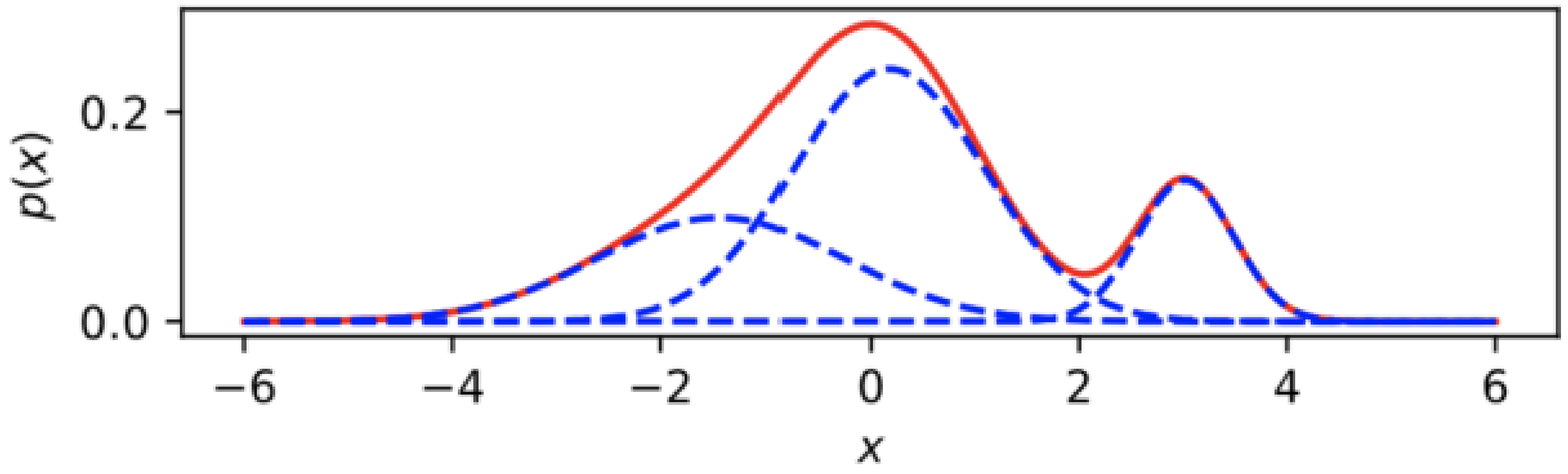
$\hat{x}_i$ defines the mean and standard deviation of each of the K Gaussians (μ_i^j , σ_i^j). Can use exponential exp(·) to ensure non-negativity

Real-valued NADE (RNADE)

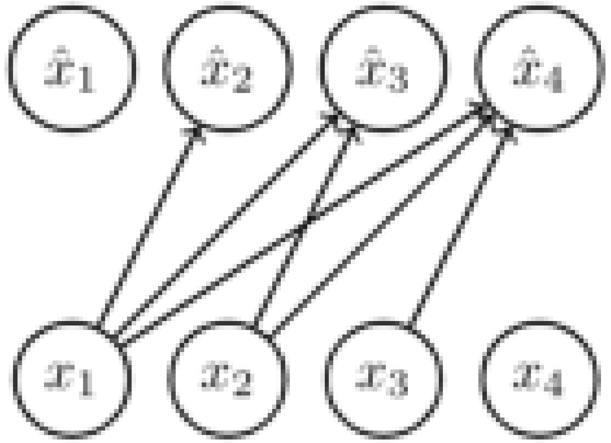
$$p(x_i|x_1,\cdots,x_{i-1}) = \sum_{j=1}^{K} \frac{1}{K}\mathcal{N}(x_i;\mu_i^j,\sigma_i^j)$$

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

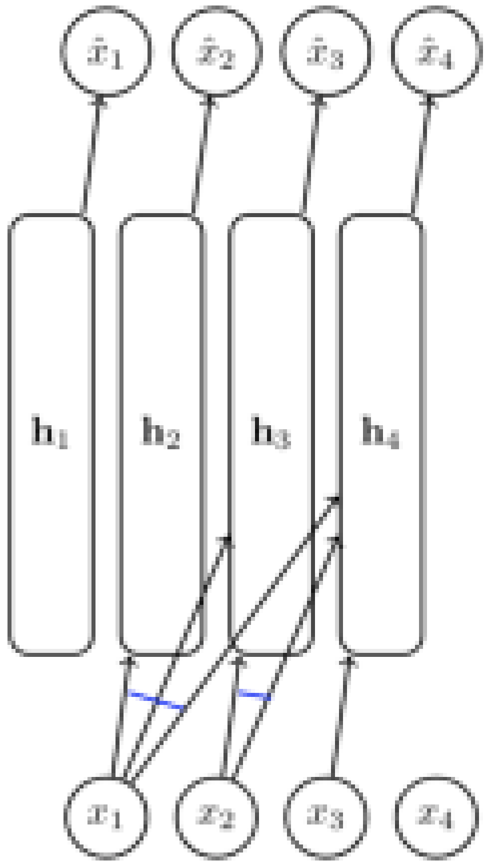
$$\hat{\mathbf{x}}_i = (\mu_i^1,\cdots,\mu_i^K,\sigma_i^1,\cdots,\sigma_i^K) = f(\mathbf{h}_i)$$



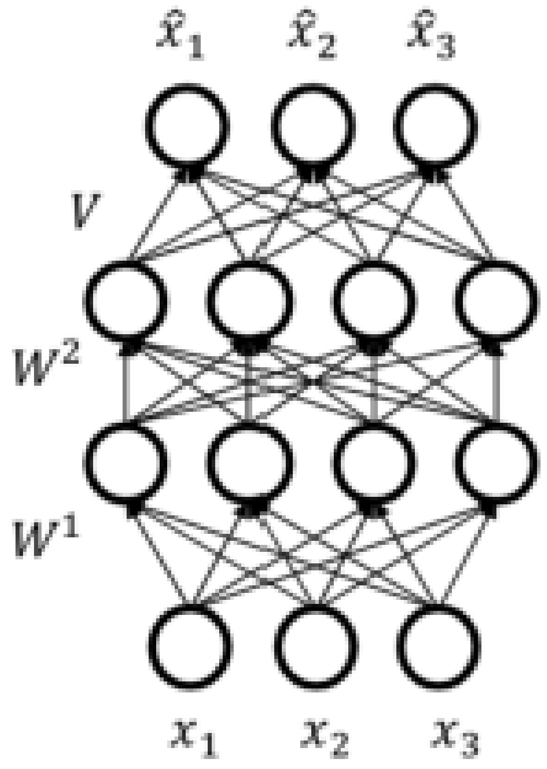
Autoregressive models vs. autoencoders



FVSBN



NADE



Autoencoder

Autoregressive models vs. autoencoders

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

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

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

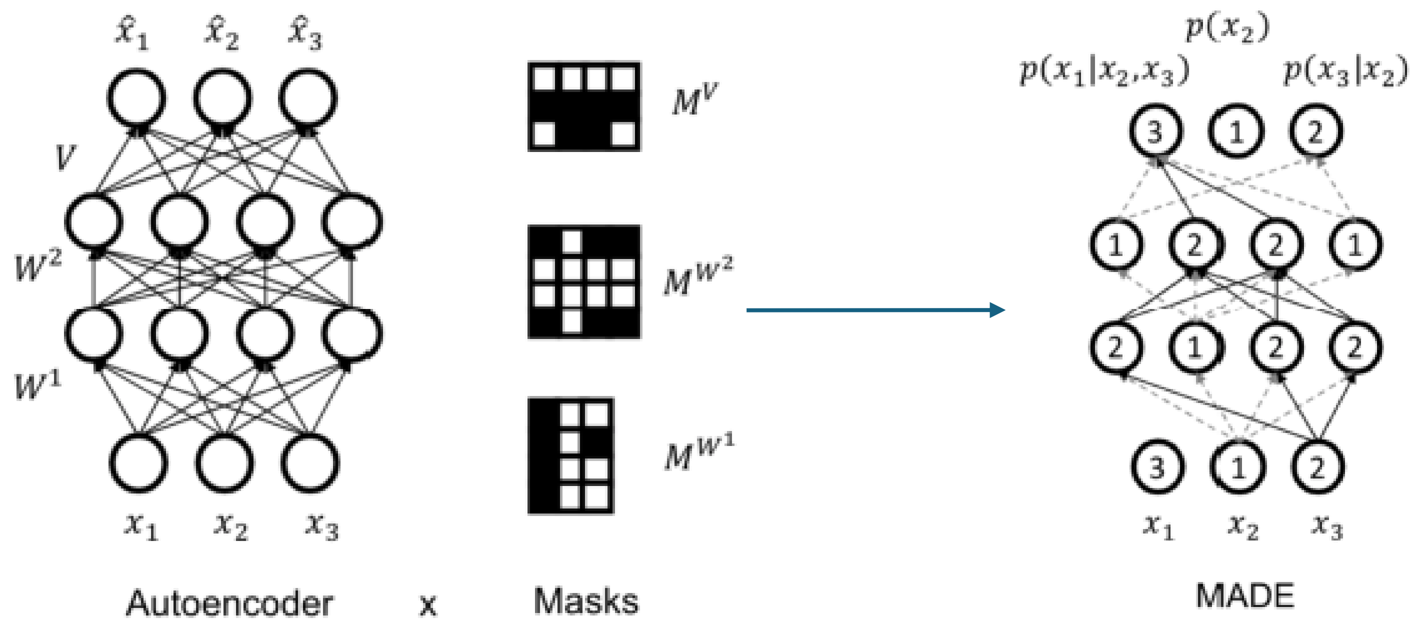
- 'e' and 'd' are constrained so that we don't learn identity mappings. Hope that e(x) is a meaningful, compressed representation of x (feature learning)
- A vanilla autoencoder is not a generative model: it does not define a distribution over x we can sample from to generate new data points.

Autoregressive Autoencoders

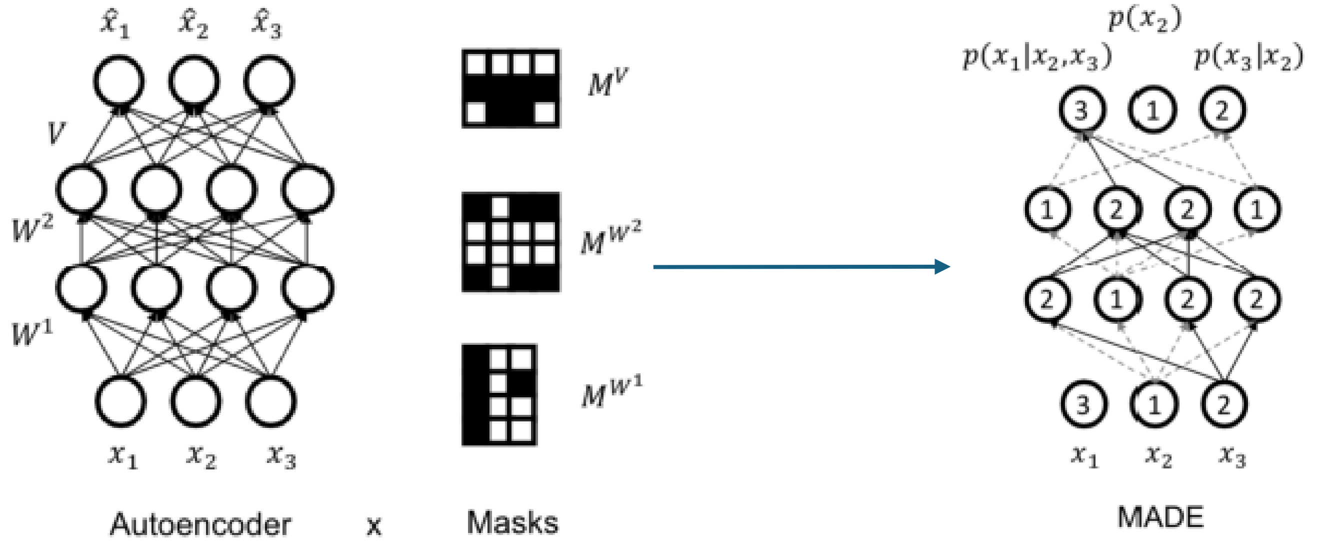
- Can we get a (autoregressive) generative model from an autoencoder?
- We need to make sure it corresponds to a valid Bayesian Network (DAG structure), i.e., we need an ordering for chain rule. If ordering is 1, 2, 3, then:
  - $\widehat{x_1}$ cannot depend on any input $x = (x_1, x_2, x_3)$. Then at generation time we don't need any input to get started
  - $\widehat{x_2}$ can only depend on x_1, …
- Benefit: 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

- Goal: An autoencoder that is autoregressive (DAG structure)



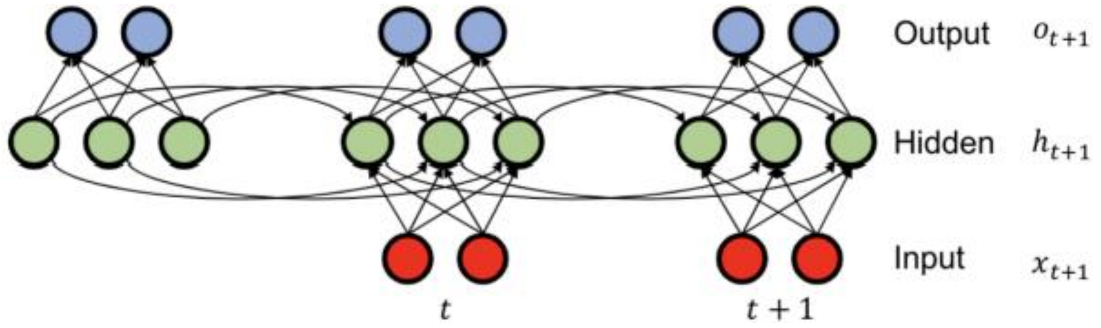
MADE: Masked Autoencoder for Distribution Estimation



- Solution: Use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1, so $p(x_1, x_2, x_3) = p(x_2)p(x_3 \mid x_2)p(x_1 \mid x_2, x_3)$.
  - The unit producing the parameters for $\widehat{x_2} = p(x_2)$ 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)

RNN: Recurrent Neural Nets

- Goal: model $p(x_t \,|x_{1:t-1};\ \boldsymbol{\alpha}^t\ )$. “History” $x_{1:t-1}$ keeps getting longer.
- Solution: keep a summary and recursively update it



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

- Hidden layer h_t is a summary of the inputs seen till time t
- Output layer o_{t-1} specifies parameters for conditional $p(x_t \,|x_{1:t-1})$
- Parameterized by b_0 (initialization), and matrices W_{hh}, W_{xh}, W_{hy} .
  Constant number of parameters (doesn’t depend on n)

Example: Character RNN (from Andrej Karpathy)

- Suppose $x_i \in$ {h, e, l, o}. Use one-hot encoding.

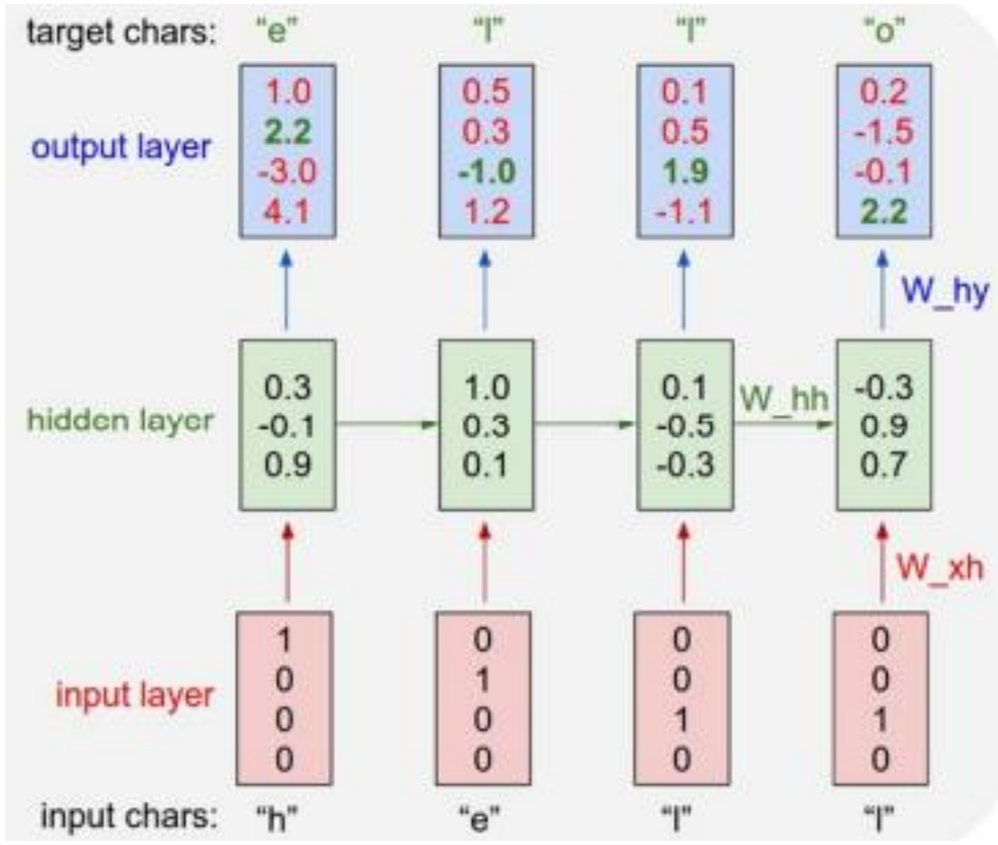
- Autoregressive:

$$p(x = hello)$$
$$= p(x_1$$
$$= h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \dots$$
$$p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$$

- For example,

$$p(x_2 = e|x_1 = h) = softmax(o_1)$$
$$= \frac{\exp(2.2)}{exp1(1.0) + \cdots + \exp(4.1)}$$

$$o_1 = W_{hy}h_1, h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1)$$



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.

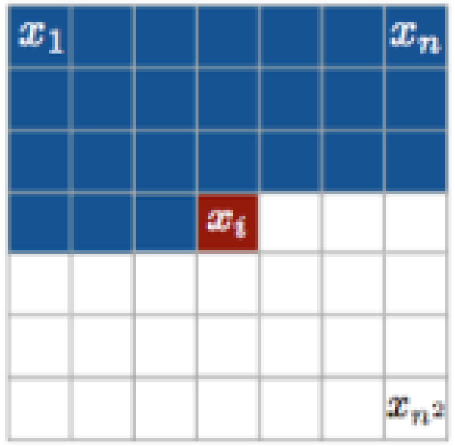
Note: generation happens character by character. Needs to learn valid words, grammar, punctuation, etc.

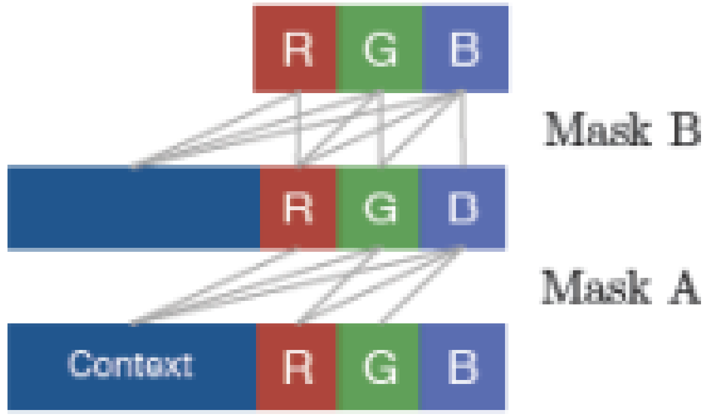
Pixel RNN (Oord et al., 2016)

- Model images pixel by pixel using raster scan order
- Each pixel conditional $p(x_t \mid x_{1:t-1})$ needs to specify 3 colors and each conditional is a categorical random variable with 256 possible values

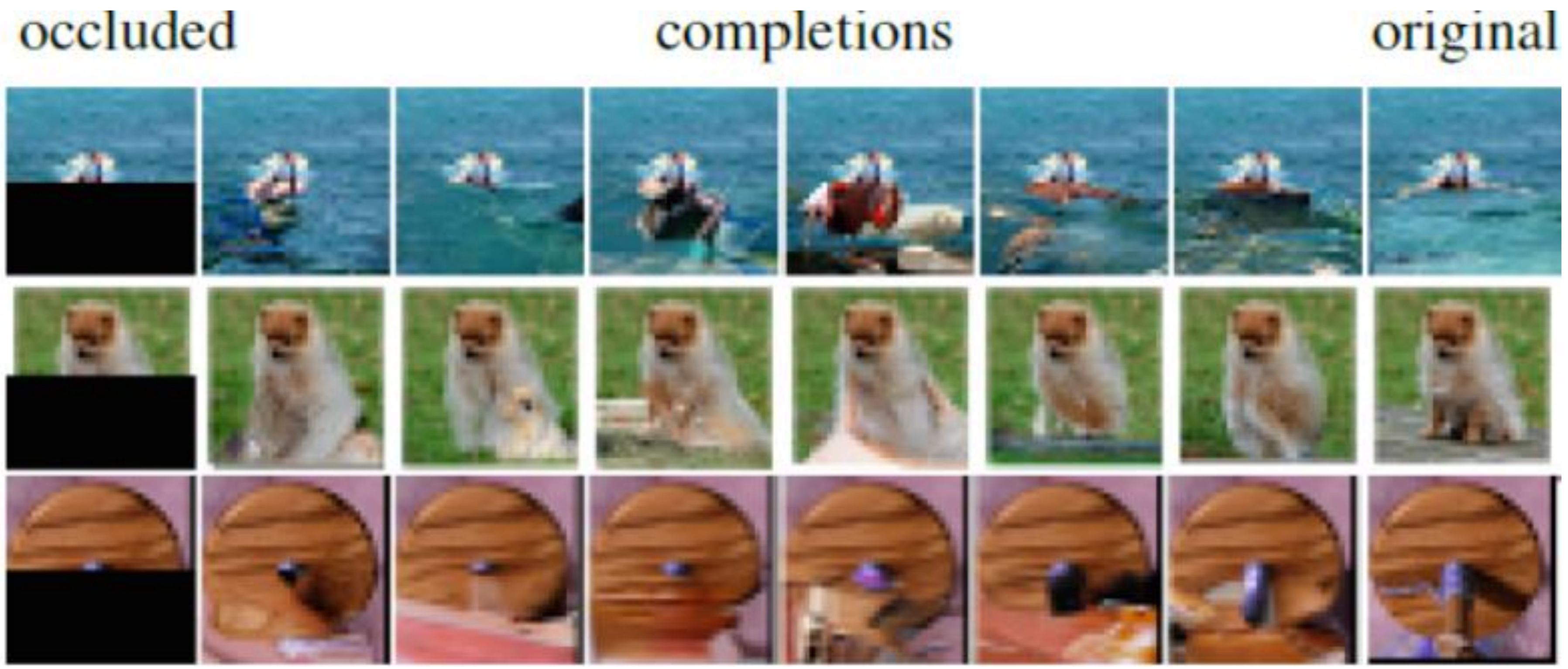
$$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})$$
$$p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$

- Conditionals modeled using RNN variants + masking (like MADE)



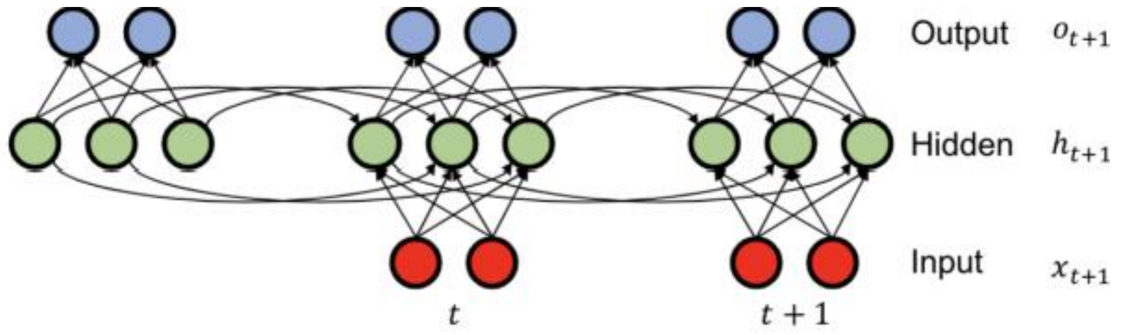


Results - Pixel RNN



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

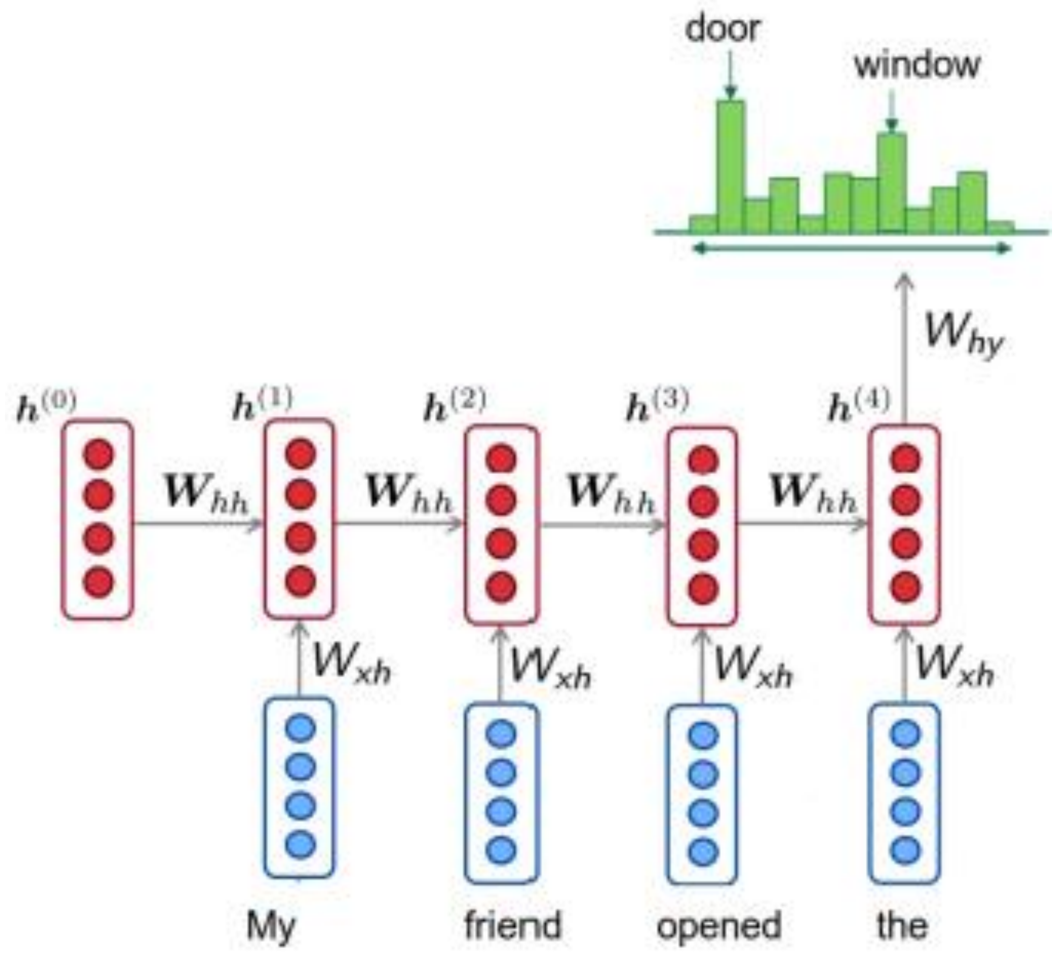
RNN: Recurrent Neural Nets



- Pros:
  - Can be applied to sequences of arbitrary length.
  - Very general: For every computable function, there exists a finite RNN that can compute it
  - Constant memory!
- Cons:
  - Still requires an ordering
  - Sequential likelihood evaluation (very slow for training)
  - Sequential generation (unavoidable in an autoregressive model)

Issues with RNNs

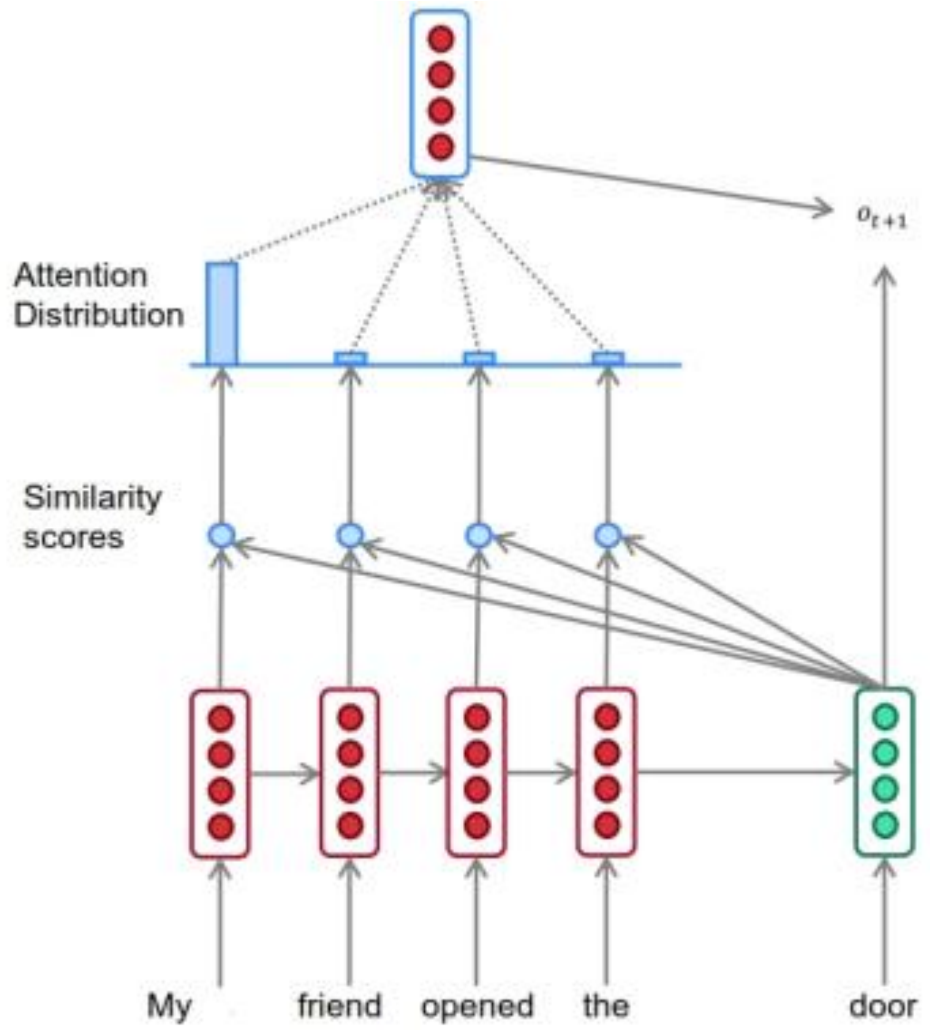
- A single hidden vector needs to summarize all the (growing) history. For example, $h^{(4)}$ needs to summarize the meaning of "My friend opened the".
- Sequential evaluation, cannot be parallelized. Slow training.
- Exploding/vanishing gradients when accessing information from many steps back



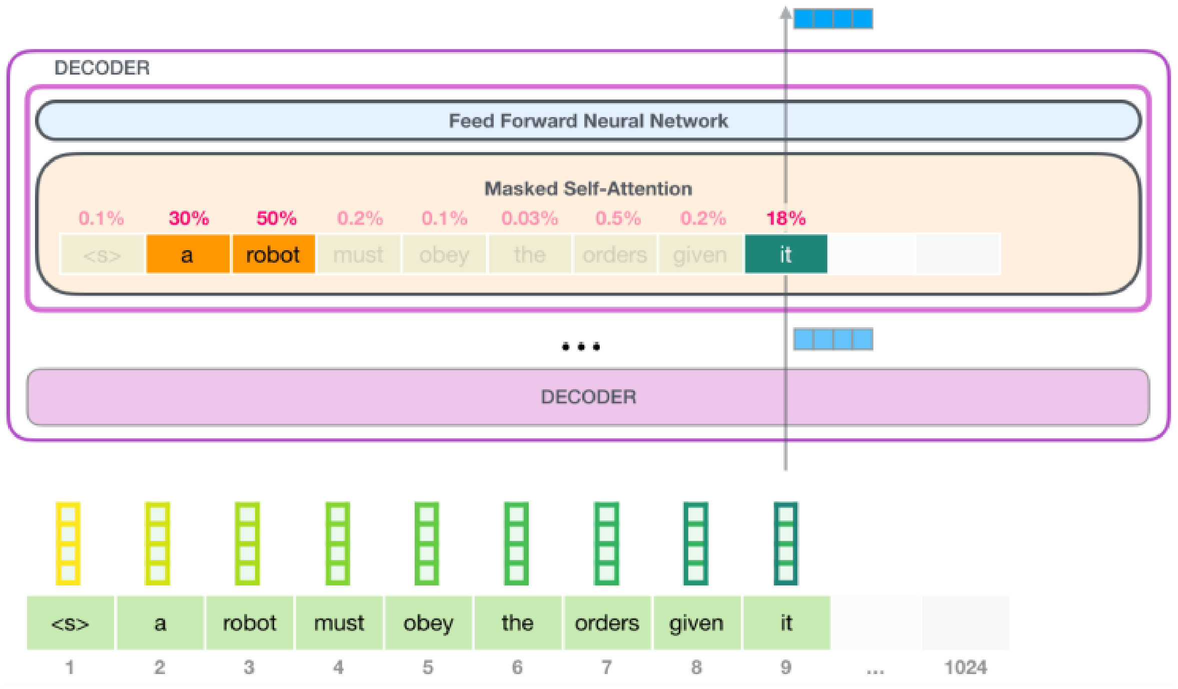
Attention based models

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

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



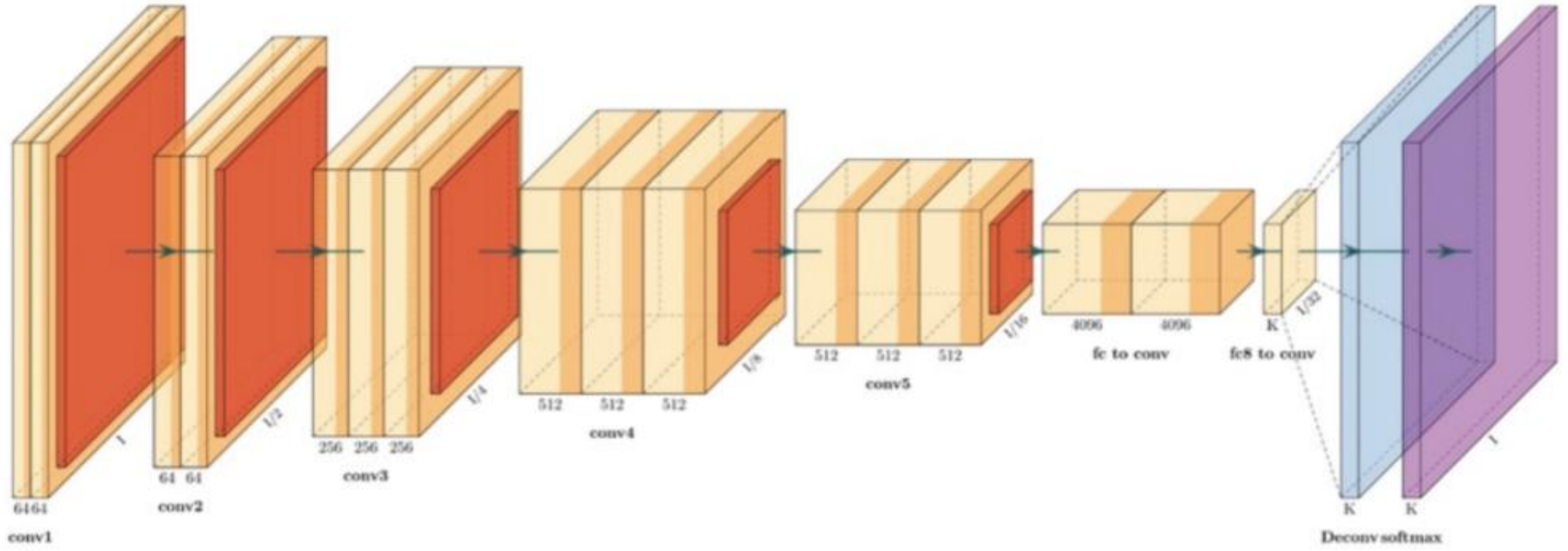
Generative Transformers



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://huggingface.co/spaces/merterbak/gpt-oss-20b-demo

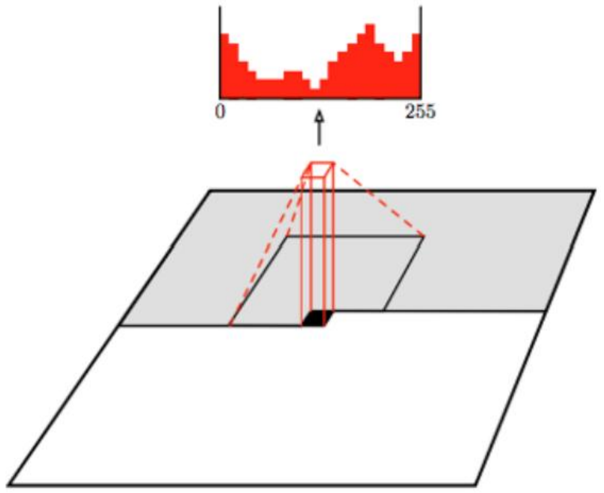
Convolutional Architectures

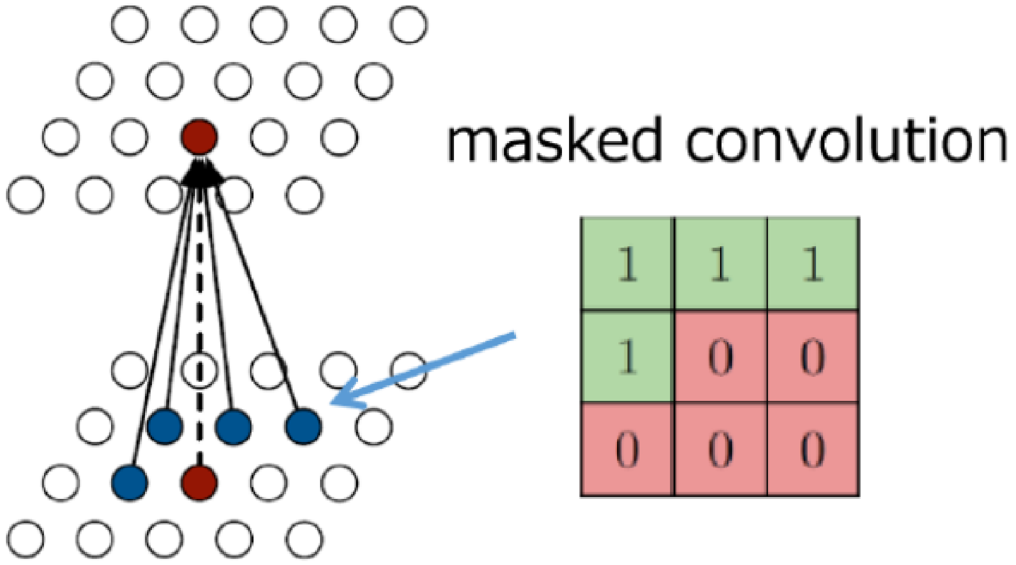


- Convolutions are natural for image data and easy to parallelize on modern hardware.

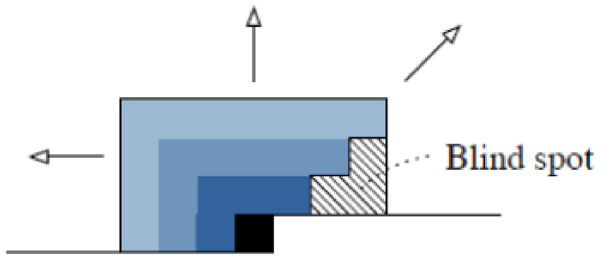
PixelCNN (Oord et al., 2016)

- Goal: 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.

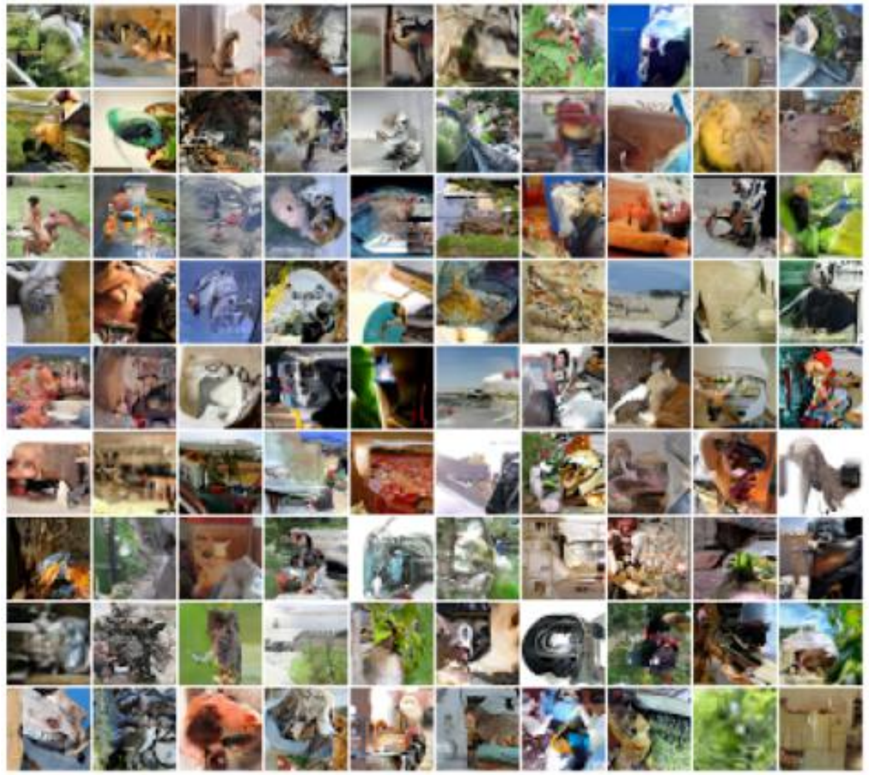




1	1	1
1	0	0
0	0	0



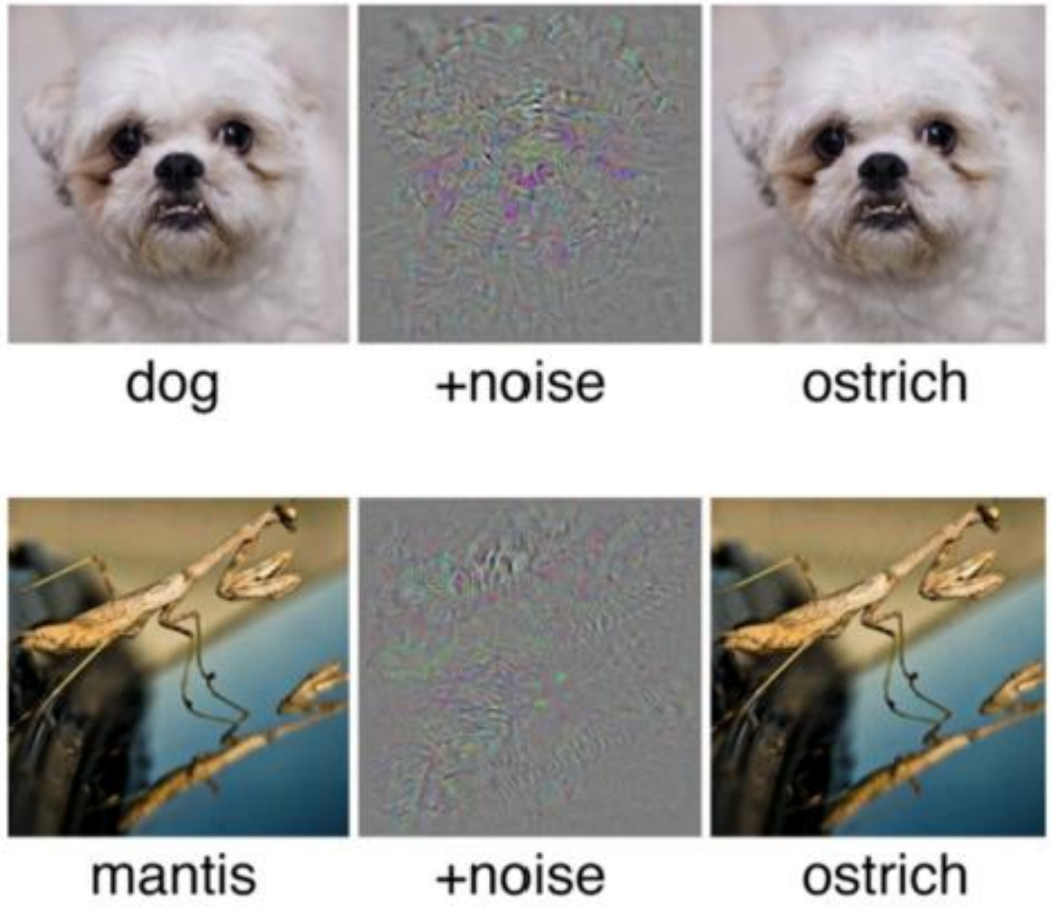
PixelCNN



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

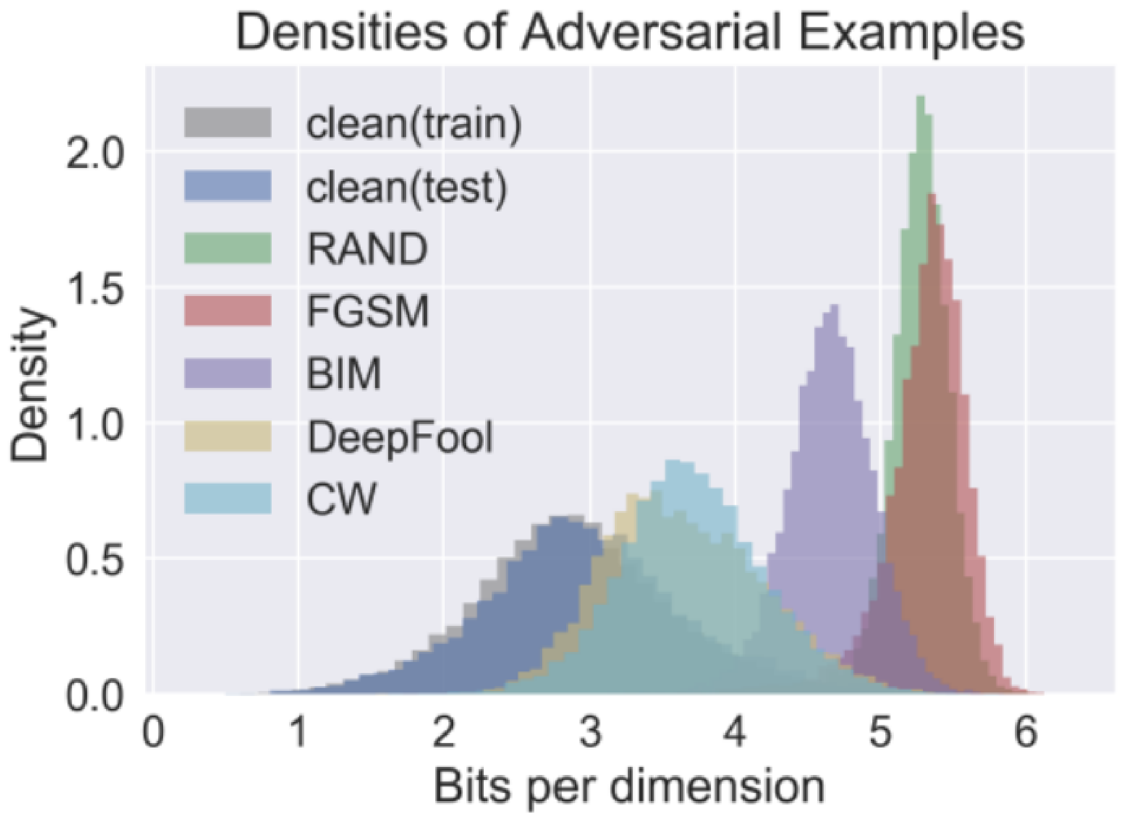
Adversarial Attacks and Anomaly detection

- Can we detect adversarial examples using these models?



PixelDefend (Song et al., 2018)

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



Summary

- Autoregressive models are easy to sample from:
  - Sample $x_0' \sim p(x_0)$
  - Sample $x_1' \sim p(x_1 | x_0 = x_0')$, …
- Autoregressive models are easy to compute probability $p(x = x')$
  - Compute $p(x_0 = x_0')$
  - Compute $p(x_1 = x_1' | x_0 = x_0')$
  - Multiply together (sum their logarithms), …
  - Ideally, can compute all these terms in parallel for fast training
- Easy to extend to continuous variables. For example, can choose Gaussian conditionals $p(x_t | x_{<t}) = N\left(\mu_\theta(x_{<t}), \Sigma_\theta(x_{<t})\right)$ or mixture of logistics
- No natural way to get features, cluster points, do unsupervised learning