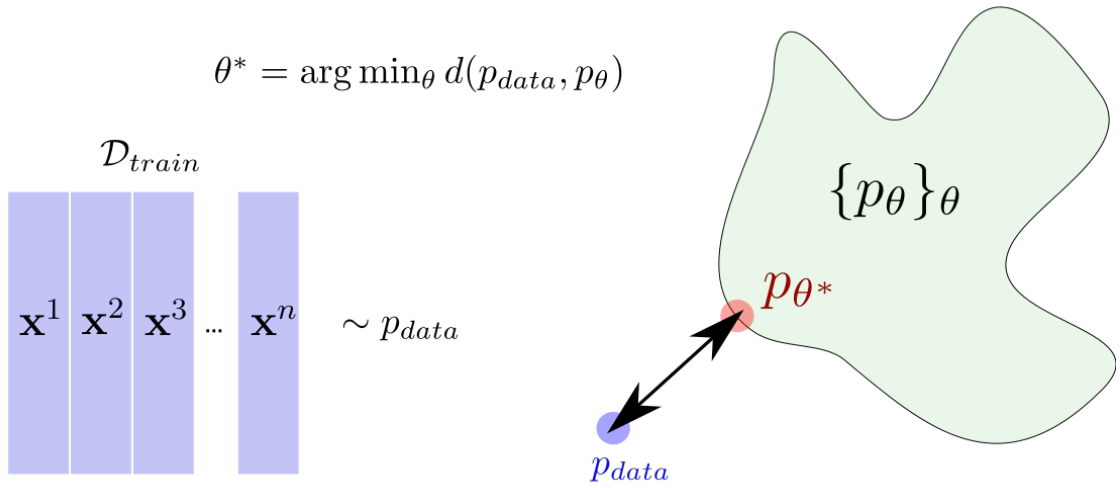
Auto-regressive models

October 19, 2022



What we want: given n training examples $\mathbf{x}^i$, we would like to have some representation for $p(\mathbf{x})$. Typically the dimension m of these data is large (think in images $\mathbf{x}^i = (\mathbf{x}^i_1, ..., \mathbf{x}^i_m)^T$), where m is in the order of 10^3 to 10^6.

Sometimes, we just need $p(\mathbf{y}|\mathbf{x})$ since $\mathbf{x}$ is given and the required "answer" to the problem is about $\mathbf{y}$.

With a **generative model**, we need specify/learn both $p(\mathbf{y})$ and $p(\mathbf{x}|\mathbf{y})$, then we compute $p(\mathbf{y}|\mathbf{x})$ with the Bayes rule.

With a **discriminative model**, we simply evaluate $p(\mathbf{y}|\mathbf{x})$, with no need to have the distribution of $\mathbf{x}$.

This representation should allow us:

- to **sample new data** distributed according to $p(\mathbf{x})$,

$$\mathbf{x}' \sim p(\mathbf{x}),$$

- to **evaluate** data $\mathbf{x}'$ on this distribution (e.g. to detect out-of-distribution data or to define some regularization prior):

$$p(\mathbf{x}'),$$

- to learn something about the **struture of these data**.

Modeling of $p(\mathbf{x})$: think in binary random values, first:

- 2 possible values per coordinate,
- how many parameters do you need to define the distribution?

$$2^m - 1$$

Modeling of $p(\mathbf{x})$ for $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_m)^T$ through the **chain rule**:

$$p(\mathbf{x}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)...p(\mathbf{x}_m|\mathbf{x}_{m-1}, ..., \mathbf{x}_1).$$

What about using **neural models** for the different conditional probabilities:

$$p(\mathbf{x}) \approx p(\mathbf{x}_1)f_{\theta_2}(\mathbf{x}_2|\mathbf{x}_1)f_{\theta_3}(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2)...f_{\theta_m}(\mathbf{x}_m|\mathbf{x}_{m-1}, ..., \mathbf{x}_1).$$

Consider the case of binary r.v. i.e. the images of handwritten digits from MNIST.

In that case $m = 2828 = 784$ pixels.

Given a dataset $\mathcal{D}$ of these images, we would like to train a neural network to represent

$$p(\mathbf{x}) = p(\mathbf{x}_1, ..., \mathbf{x}_{784}).$$

Suppose we consider an **ordering** of the pixels within the images: for example, row-wise, from the top-left to the bottom-right.

We will use this ordering through the chain rule:

$$p(\mathbf{x}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)...p(\mathbf{x}_{784}|\mathbf{x}_{783}, ..., \mathbf{x}_1),$$

with $p(\mathbf{x}_1)$ given by a table (**one probability** α^1) and will use **simple networks** (parameterized functions) for:

$$p(\mathbf{x}_i|\mathbf{x}_{i-1}, ..., \mathbf{x}_1).$$

On of the most simple solutions: use a **logistic regression** for each of these terms:

$$p(\mathbf{x}_i|\mathbf{x}_{i-1}, ..., \mathbf{x}_1) \triangleq \sigma(\alpha_0^i + \alpha_1^i\mathbf{x}_1 + ... + \alpha_{i-1}\mathbf{x}_{i-1}).$$

We are using **parameterized functions** (e.g., logistic regression above) to predict the next pixel value given all the previous ones.

Called **autoregressive model**.

Note that the joint distribution is given by the following **product of Bernoulli distributions**

$$p(\mathbf{x}) = \mathcal{B}(\mathbf{x}_1; \alpha^1)\times\mathcal{B}(\mathbf{x}_2; \sigma(\alpha_0^2+\alpha_1^2\mathbf{x}_1))\times...\times\mathcal{B}(\mathbf{x}_i; \sigma(\alpha_0^i+\alpha_1^i\mathbf{x}_1+...+\alpha_{i-1}\mathbf{x}_{i-1}))...\times\mathcal{B}(\mathbf{x}_{784}; \sigma(\alpha_0^{784}+\alpha_1^{784}\mathbf{x}_1+...+\alpha_{783}^{784}\mathbf{x}$$

Once the model is trained, this gives us a direct way of **evaluating** the likelihood of an image under this distribution: given an image with values $\mathbf{x}_1, ..., \mathbf{x}_{784}$, as follows:

- $\pi_1 = \left[\alpha^1\right]^{\mathbf{x}_1} \left[1 - \alpha^1\right]^{1-\mathbf{x}_1}$,
- $\pi_2 = \left[\sigma(\alpha_0^2 + \alpha_1^2 \mathbf{x}_1)\right]^{\mathbf{x}_2} \left[1 - \sigma(\alpha_0^2 + \alpha_1^2 \mathbf{x}_1)\right]^{1-\mathbf{x}_2}$,
- $\pi_3 = \left[\sigma(\alpha_0^3 + \alpha_1^3 \mathbf{x}_1 + \alpha_2^3 \mathbf{x}_2)\right]^{\mathbf{x}_3} \left[1 - \sigma(\alpha_0^3 + \alpha_1^3 \mathbf{x}_1 + \alpha_2^3 \mathbf{x}_2)\right]^{1-\mathbf{x}_3}$,
- ...
- $\pi_{784} = \left[\sigma(\alpha_0^{784} + \alpha_1^{784} \mathbf{x}_1 + \alpha_2^{784} \mathbf{x}_2 + ... + \alpha_{783}^{784} \mathbf{x}_{783})\right]^{\mathbf{x}_{784}} \left[1 - \sigma(\alpha_0^{784} + \alpha_1^{784} \mathbf{x}_1 + \alpha_2^{784} \mathbf{x}_2 + ... + \alpha_{783}^{784} \mathbf{x}_{783})\right]^{1-\mathbf{x}_{78}}$

and the total likelihood would be:

$$\pi = \prod_{i=1}^{784} \pi_i.$$

Sampling from $p(\mathbf{x}_1, ..., \mathbf{x}_{784})$ is also very simple, with a strong **sequential nature**:

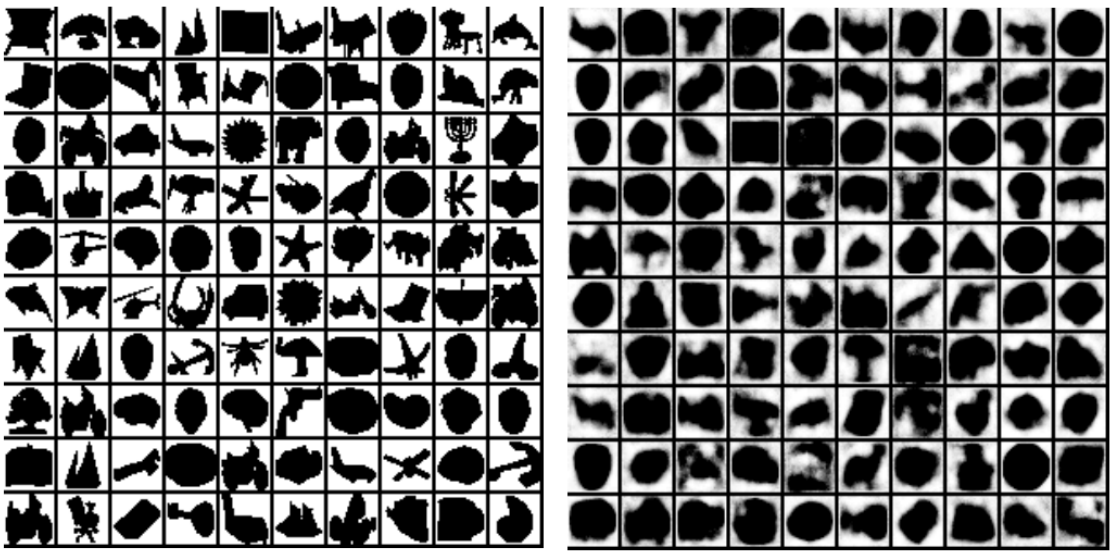
- Sample $\mathbf{x}_1 \sim \mathcal{B}(\mathbf{x}_1; \alpha^1)$.
- Sample $\mathbf{x}_2 \sim \mathcal{B}(\mathbf{x}_2; \sigma(\alpha_0^2 + \alpha_1^2 \mathbf{x}_1))$.
- Sample $\mathbf{x}_3 \sim \mathcal{B}(\mathbf{x}_3; \sigma(\alpha_0^3 + \alpha_1^3 \mathbf{x}_1 + \alpha_2^3 \mathbf{x}_2))$.
- ...
- Sample $\mathbf{x}_{784} \sim \mathcal{B}(\mathbf{x}_{784}; \sigma(\alpha_0^{784} + \alpha_1^{784} \mathbf{x}_1 + \alpha_2^{784} \mathbf{x}_2 + ... + \alpha_{783}^{784} \mathbf{x}_{783}))$.

[1]:
```
import numpy as np
np.random.choice(2, 1, p=[0.35,0.65])
```

[1]:
```
array([1])
```

The **number of parameters** is:

$$1 + 2 + ... + m = \frac{1}{2} m(m+1).$$



Learning Deep Sigmoid Belief Networks with Data Augmentation. Zhe Gan, Ricardo Henao, David Carlson, Lawrence Carin Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, PMLR 38:268-276, 2015.

0.1 Deep auto-regressive models

One simple idea to have more complex functions to represent the conditional probabilities: use **additional neural network layers** (from Multiple Layer Perception, MLP) instead of just logistic regression.

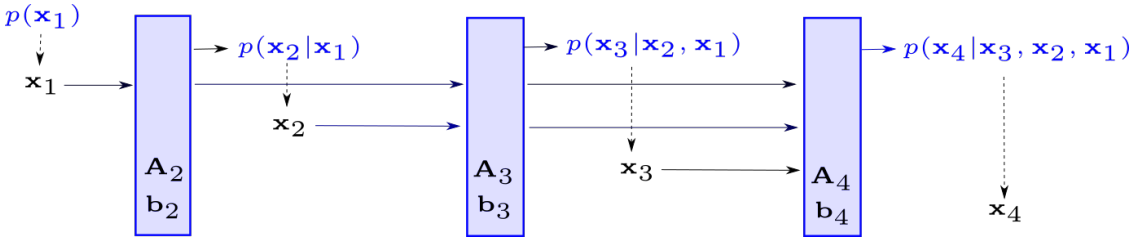
For $i > 1$

$$\mathbf{h}_i = h(\mathbf{A}_i \mathbf{x}_{<i} + \mathbf{b}_i) \quad \text{(hidden layer, dimension } d) \tag{1}$$

$$\mathbf{x}_i \sim p(\mathbf{x}_i|\mathbf{x}_1, ..., \mathbf{x}_{i-1}; \mathbf{A}_i, \mathbf{b}_i, \alpha^i) = \mathcal{B}(\mathbf{x}_i; \sigma(\alpha^i [\mathbf{h}_i^T, 1]^T) \tag{2}$$

where:

- h is an activation fuction,
- $\mathbf{h}_i \in \mathbb{R}^d$,
- $\mathbf{A}_i \in \mathbb{R}^{d \times (i-1)}$, $\mathbf{b}_i \in \mathbb{R}^d$,
- $\mathbf{x}_{<i} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ ... \\ \mathbf{x}_{i-1} \end{pmatrix}$,
- $\alpha^i \in \mathbb{R}^{d+1}$.

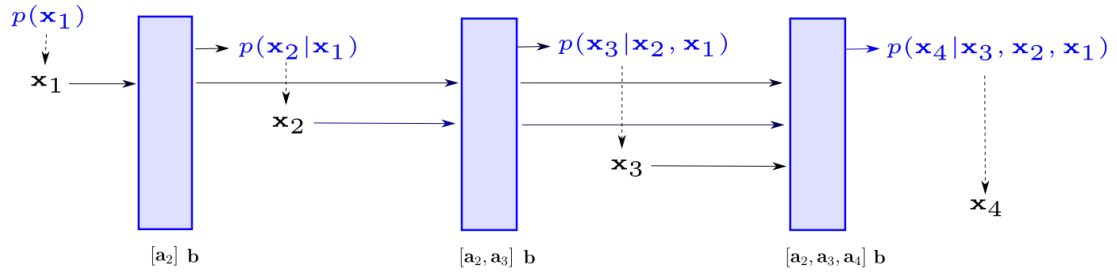


Called **Neural Autoregressive Density Estimation** (NADE).

An important improvement: consider a **single** weight matrix $\mathbf{A} \in \mathbb{R}^{d \times m-1}$ and

$$\mathbf{A}_i \triangleq \mathbf{A}_{*,<i}$$

i.e., we consider for i the sub-matrix of **A with its first** $i-1$ **columns**.

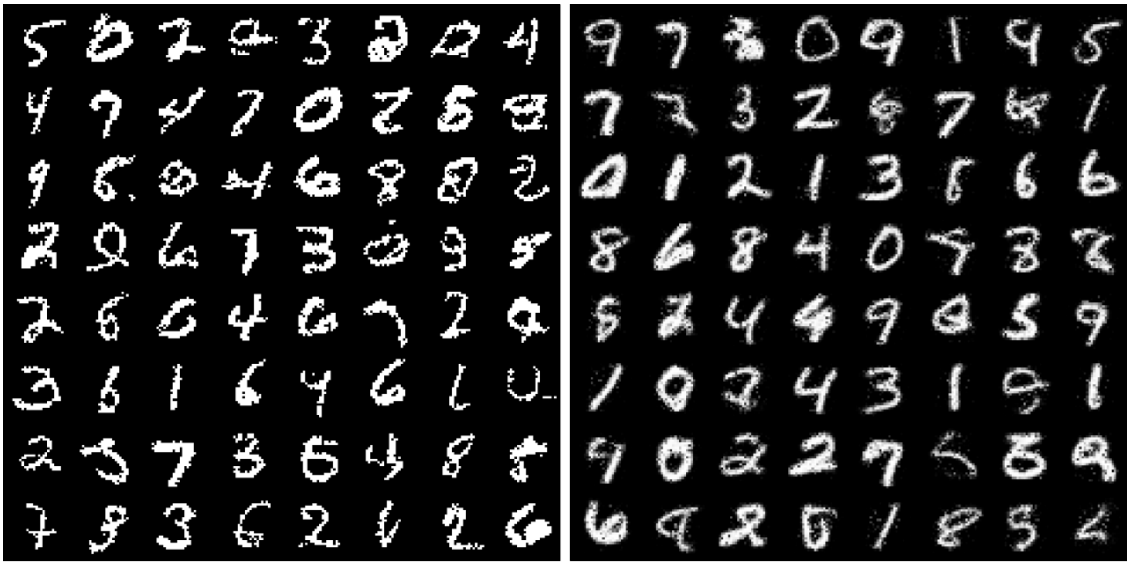


How many parameters?

- A single weights matrix $\mathbf{A} \in \mathbb{R}^{d \times (m-1)}$,
- A single vector $\mathbf{b} \in \mathbb{R}^d$,

- m logistic regression coefficient vectors $\alpha_i \in \mathbb{R}^{d+1}$.

$O(dm)$.



Samples and conditional distributions.

The Neural Autoregressive Distribution Estimation, 2011.

Generalization to **discrete random variables**?

$$\mathbf{x}_i \in \{1, ..., K\}.$$

For example, **pixel intensities varying from** 0 **to** 255.

Nearly the same, changing the sigmoid

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

$$p(\mathbf{x}_i | \mathbf{x}_{i-1}, ..., \mathbf{x}_1) = (p_{i,1}, ..., p_{i,K}) = softmax(\mathbf{W}_i [\mathbf{h}_i, 1]^T).$$

with

$$softmax(\mathbf{v}) = (\frac{\exp(\mathbf{v}_1)}{\sum_{k=1}^K \exp(\mathbf{v}_k)}, ..., \frac{\exp(\mathbf{v}_K)}{\sum_{k=1}^K \exp(\mathbf{v}_k)}).$$

Generalization to **continuous variables**?

One possibility: adapt the above architecture to produce each conditional probability as a uniform mixture of K Gaussians.

RNADE: The real-valued neural autoregressive density-estimator. B. Uria, I. Murray, H. Larochelle *https://arxiv.org/pdf/1306.0186.pdf*

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

$$p(\mathbf{x}_i | \mathbf{x}_{i-1}, ..., \mathbf{x}_1) = \sum_{k=1}^{K} \frac{1}{K} \mathcal{N}(\mathbf{x}_i | \mu_{i,k}, \sigma_{i,k}).$$

The parameters to output at coordinate i are the means and variances:

$$(\mu_{i,1}..\mu_{i,K}, \sigma_{i,1}, ..., \sigma_{i,K})$$

and they may be estimated through another neural network (for example) n_i:

$$(\mu_{i,1}..\mu_{i,K}, \sigma_{i,1}, ..., \sigma_{i,K})^T = n_i(\mathbf{h}_i).$$

A detail: the output of n_i should be designed such that $\sigma_{i,k}$ is **strictly positive**.

Interpret the output as the **logarithm of the variance**; then, take the **exponential of it**.

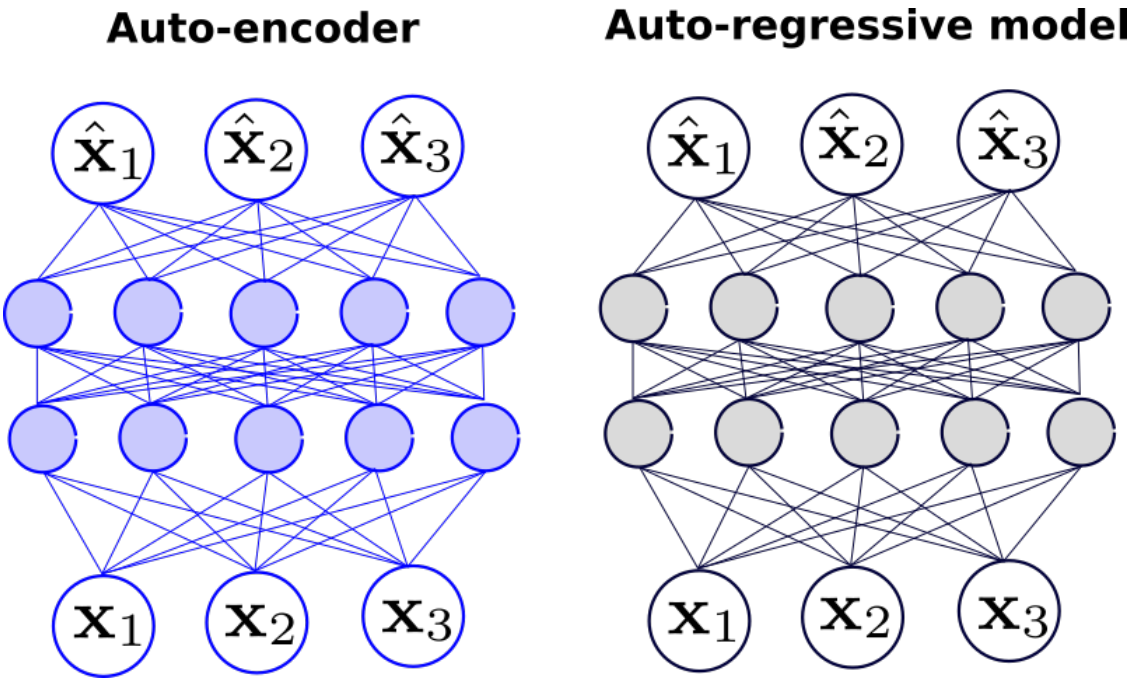
0.2 Auto-regressive models and auto-encoders

Review: **auto-encoder** is a family of networks that, given an input $\mathbf{x} \in \mathbb{R}^m$, follows an **encoder-decoder** architecture "coming back" to $\mathbb{R}^m$:

$$\mathbf{v} = e_\phi(\mathbf{x}), \tag{3}$$

$$\hat{\mathbf{x}} = d_\theta(\mathbf{v}). \tag{4}$$

The intermediate representation $\mathbf{v}$ may be used as a **compressed representation** for $\mathbf{x}$.



- In one case (AE), $\hat{\mathbf{x}}_i$ are **deterministic**.
- In the other case (AR), $\hat{\mathbf{x}}_i$ are **stochastic**: $\hat{\mathbf{x}}_i \triangleq p(\mathbf{x}_i|\mathbf{x}_{i-1}...\mathbf{x}_1)$.

The auto-encoder generates its output (the whole sequence) in one step only.

Can we do the same when **training an autoregressive model**?

What is the main difference between both structures?

- **Not any graph** is possible in the case of the auto-regressive model!
- For example, with:

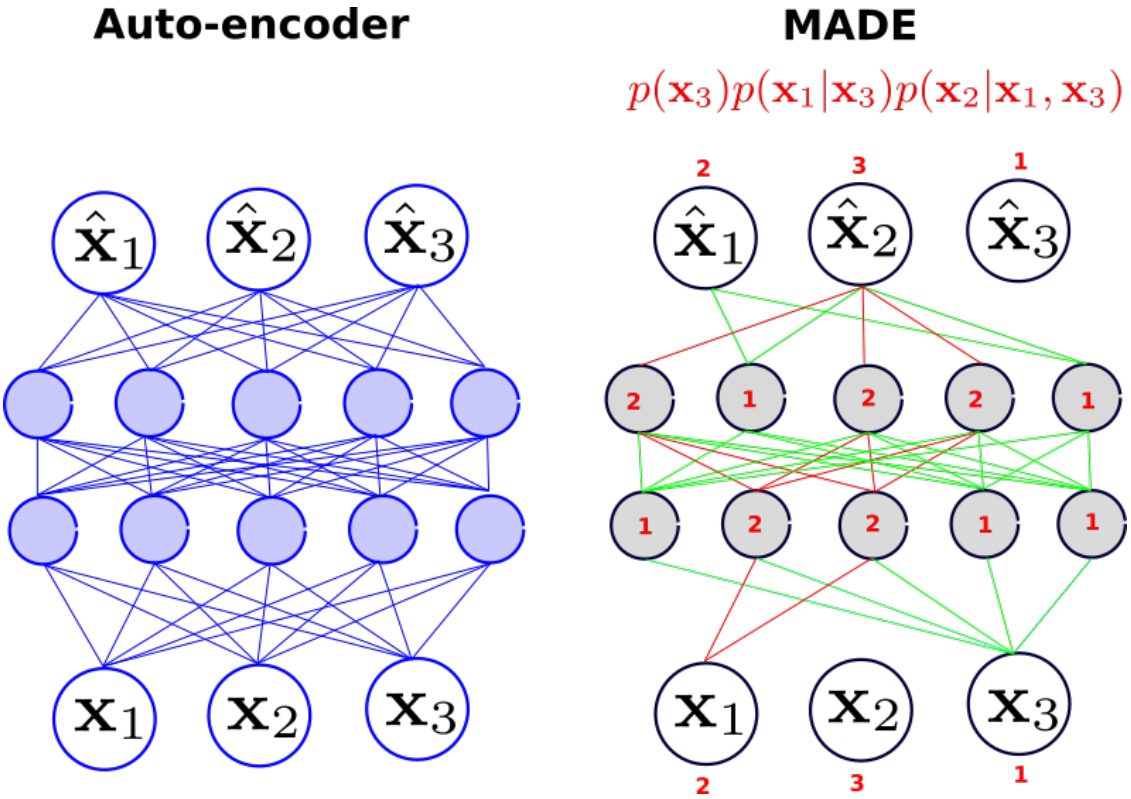
$$p(\mathbf{x}_3)p(\mathbf{x}_1|\mathbf{x}_3)p(\mathbf{x}_2|\mathbf{x}_1,\mathbf{x}_3),$$

then you should ensure that the parameters of $p(\mathbf{x}_1|\mathbf{x}_3)$ do **not depend** on $\mathbf{x}_1$ (this will be the **ground truth**!) nor on $\mathbf{x}_2$ (because of the chosen **factorization**).

MADE: Masked Autoencoder for Distribution Estimation

- Idea is to use **at training** a **single neural network** to process the data $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_m)^T$: instead of m forward calls to produce the conditional probabilities, just one forward call!

- The network has a structure **similar to the one of Auto-Encoders**.
- Some connections are masked (zeroed) to **ensure that, given an order of the variables, the data processing respects that order**.



In the example above:

- An **order** is chosen, corresponding to

$$p(\mathbf{x}_3)p(\mathbf{x}_1|\mathbf{x}_3)p(\mathbf{x}_2|\mathbf{x}_1,\mathbf{x}_3).$$

  We annotate the input data (bottom) and output (top).

- All the hidden layers are associated (randomly) to an **index** $1, ..., m-1$.
- All connections that connect an index i (bottom) to $j < i$ (top) are removed. This ensures that $p(\mathbf{x}_1|\mathbf{x}_3)$ (index 2) depends only on processing of $\mathbf{x}_3$, and that $p(\mathbf{x}_3)$ (index 1) depends on nothing else.

Hence, in **training** (when you have the complete samples), you can re-write the production of the parameters for the distribution of the consecutive variables **in just one pass**, through a rather classic MLP.

0.3 Auto-regressive models with Recurrent Neural Networks

Fundamentally, with auto-regressive models we need to form a compact representation of the "history" $\mathbf{x}_{1:i-1}$

$$p(\mathbf{x}_i|\mathbf{x}_{1:i-1};\theta_i).$$

Another idea: use a representation for this summary $\mathbf{h}_{i-1}$ and recursively update it with the new coordinate that we will to represent:

$$\mathbf{h}_i = f_\theta(\mathbf{h}_{i-1}, \mathbf{x}_i). \tag{5}$$

The parameters θ are **shared along the sequence**.

This is the idea of a **recurrent neural network** (RNN):

- the "summary" of the sequence seen so far, $\mathbf{x}_{1:i}$, is called the **hidden state** $\mathbf{h}_i$.
- The output of the network at step i is identified to the parameters of the conditional distribution

$$p(\mathbf{x}_{i+1}|\mathbf{x}_{1:i}).$$

Because all the network weights are **shared** along i, the number of parameters does not depend on m. It can be applied to data $\mathbf{x}$ of arbitrary dimension.

On way to implement it is with a simple MLP. At coordinate i

$$\mathbf{h}_0 \tag{6}$$
$$\mathbf{h}_i = \tanh(\mathbf{W}_{hh}\mathbf{h}_{i-1} + \mathbf{W}_{hx}\mathbf{x}_i), \tag{7}$$
$$\mathbf{o}_i = \mathbf{W}_{ho}\mathbf{h}_i, \tag{8}$$

and $p(\mathbf{x}_{i+1}|\mathbf{x}_{1:i})$ will have parameters given by $\mathbf{o}_i$.

Note that you have a **sequence here**, and you can even **split the vector x into small vectors** (instead of processing coordinate after coordinate).

Some limitations:

- We need to define an **ordering** among the coordinates.
- Training may face some difficulties because of the length of the sequence: vanishing/exploding gradients. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Evaluating the likelihood $p_\theta(\mathbf{x})$ of one sample is slow.

0.4 PixelRNN

Pixel recurrent neural networks, A. Van Den Oord, N. Kalchbrenner, K. Kavukcuoglu. Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML'16), 2016.

Idea:

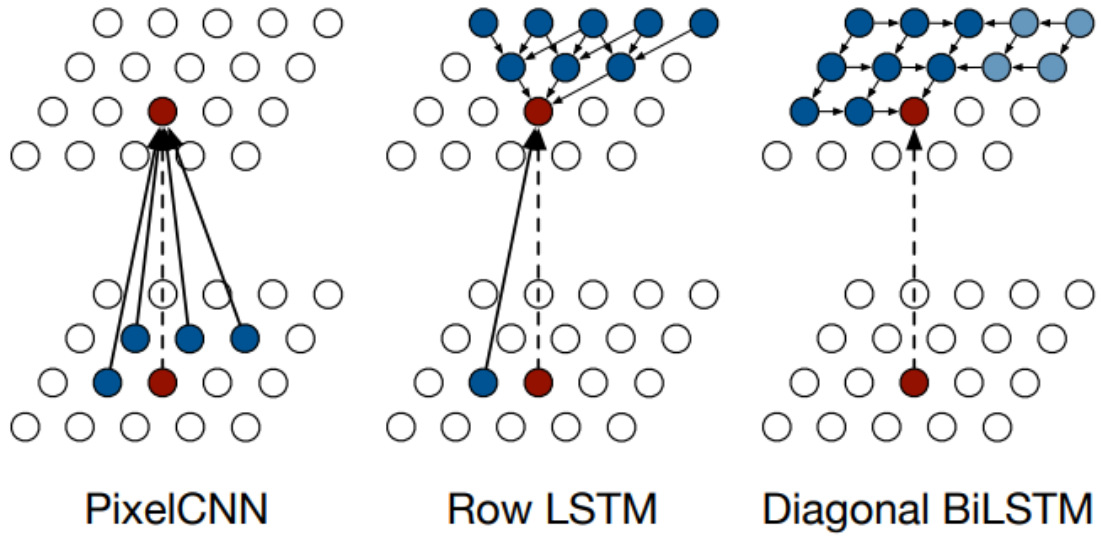
- $\mathbf{x}$ are $n \times n$ **images**; the components of the vector are the $\mathbf{x}_i$.
- Use the **raster scan order** for the pixels.
- Add each color component too:

$$p(\mathbf{x}_i|\mathbf{x}_{1:i-1}) = p(\mathbf{x}_{i,r}|\mathbf{x}_{1:i-1})p(\mathbf{x}_{i,g}|\mathbf{x}_{1:i-1}, \mathbf{x}_{i,r})p(\mathbf{x}_{i,b}|\mathbf{x}_{1:i-1}, \mathbf{x}_{i,r}, \mathbf{x}_{i,g}).$$

The papers explore several ways to represent the conditionals above:

- **LSTMs** (which is a form of RNNs), reformulated for parallelization (like MADE),
- **CNNs**,

and apply them as a **pure generative model** and as a **completion model**.



[From original paper]

- Several **consecutive layers** ("stacking") (RowLSTM, PixelCNN, Diagonal BiLSTM).
- Most of these layers have **residual** connections.
- The last layer is a **softmax layer** used to generate the color distribution.

To ensure the **auto-regressive nature of the model**, the computation of $p(\mathbf{x}_i|\mathbf{x}_{1:i-1})$ is done through **masked convolutional filters** that should depend only on $\mathbf{x}_{1:i-1}$.

0.5 RowLSTM

- Uses a **LSTM as recurrent neural network**.
- The recurrence seen through the LSTM is **along rows** (not each pixel in raster order).
- The internal operations of the LSTM are implemented as **1D convolutions** ($w \times 1$).
- **Triangular receptive field** (that ignores many previous pixels that should theoretically dependent).

Let i be the row index. LSTM implements:

$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i) \tag{9}$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i \tag{10}$$

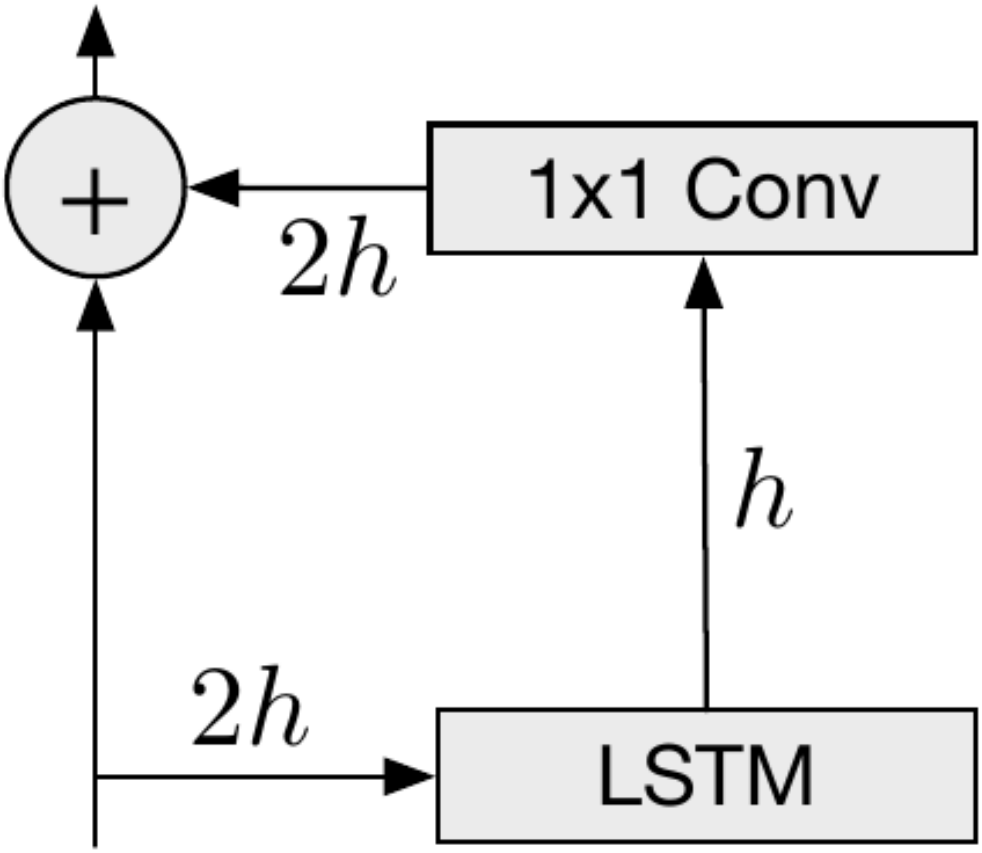
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i) \tag{11}$$

$$\tag{12}$$

$\mathbf{h}_i, \mathbf{c}_i$ are internal states $h \times n \times 1$.

- $\mathbf{K}^{ss}, \mathbf{K}^{is}$: 1D convolutions, with width w.
- Convolutions are **masked** to respect the color-wise and pixel-wise factorization.

- Produces row-wise feature maps ($h \times n \times 1$)



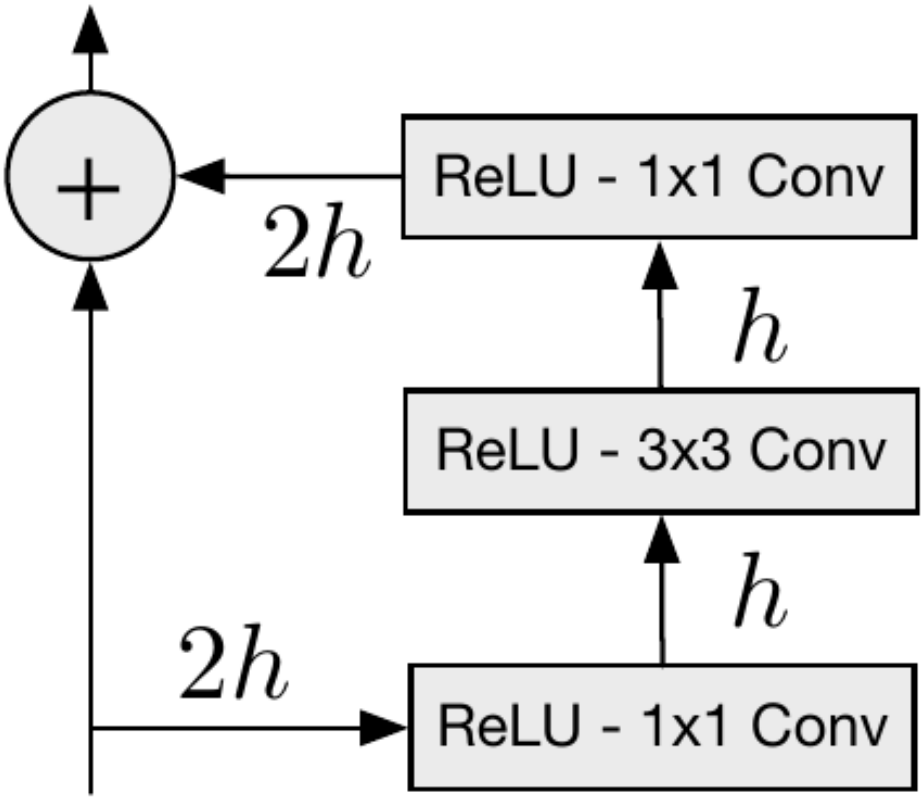
[From original paper]

0.6 Diagonal Bi-LSTM

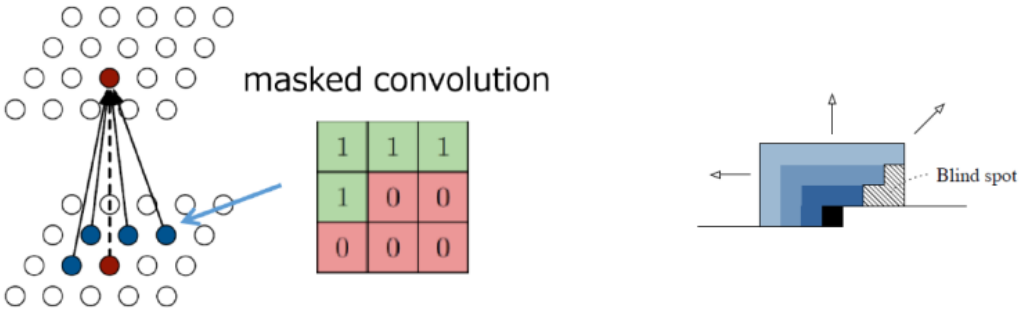
Same principle as above, but:

- instead of a row-wise LSTM, a **bidireccional LSTM with masked 1D convolutions**,
- diagonal-wise scanning of the images (along the **two** diagonals),
- allows to handle the **whole theoretical perceptual field** (as stated from the factorization).

0.7 Pixel CNN



[From original paper]

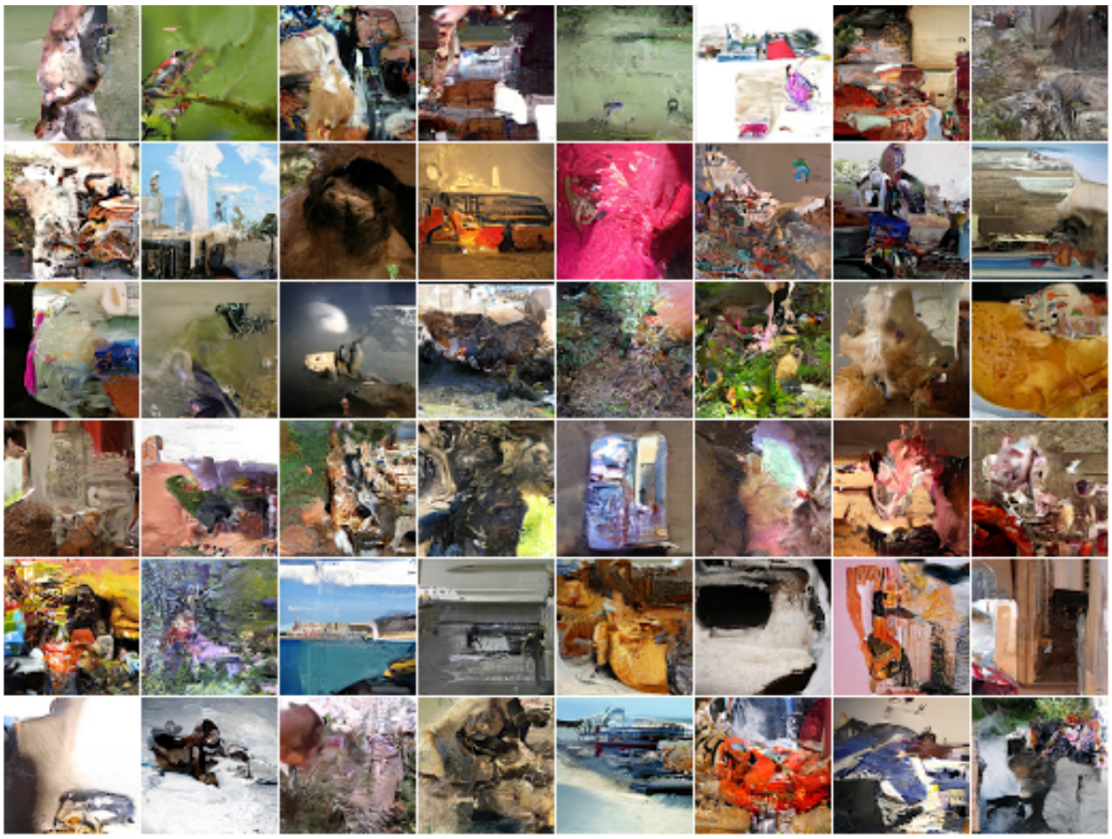


PixelCNN: the dependency is implemented indirectly:

- **square filter support** that implements the dependency on only the neighbor values within some bounded region (size of the filter);
- the square is **masked** for the dependencies to be only to the upper left data;
- 15 layers.

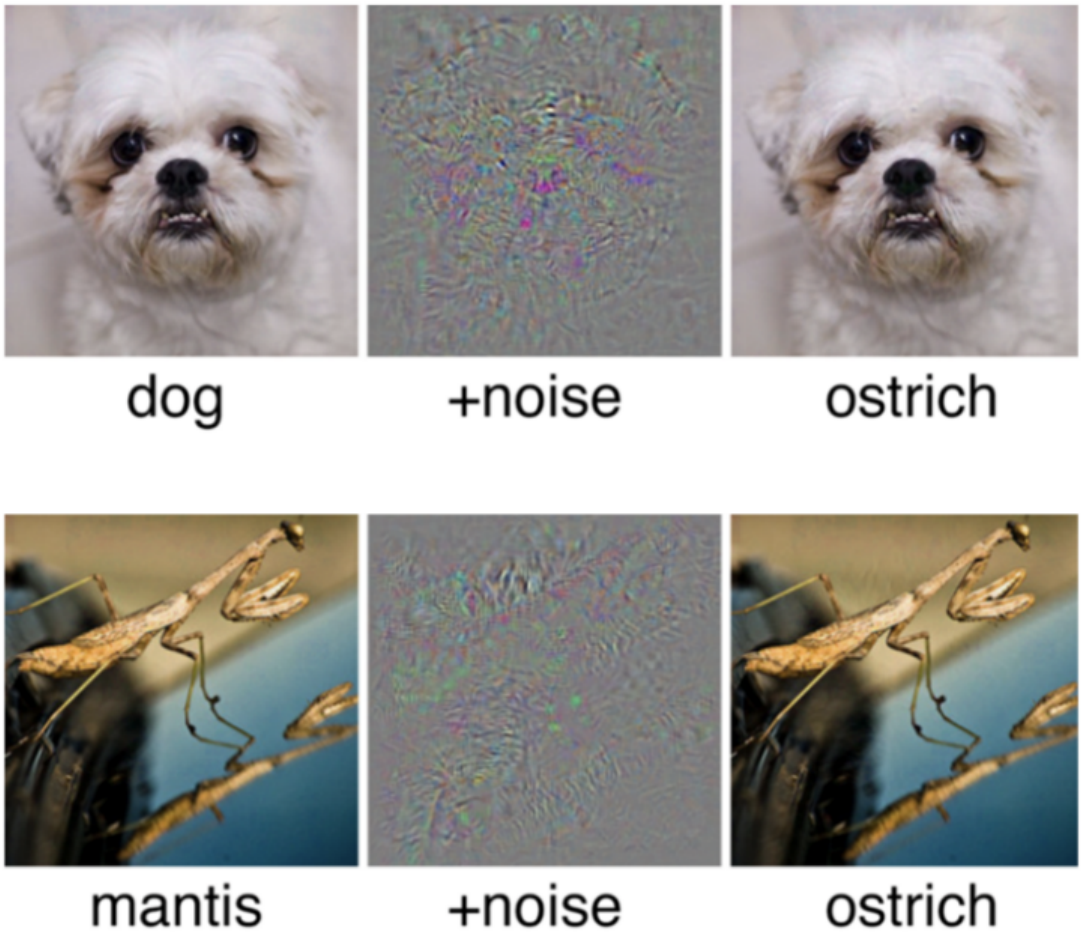
Implicitly: Markov field.

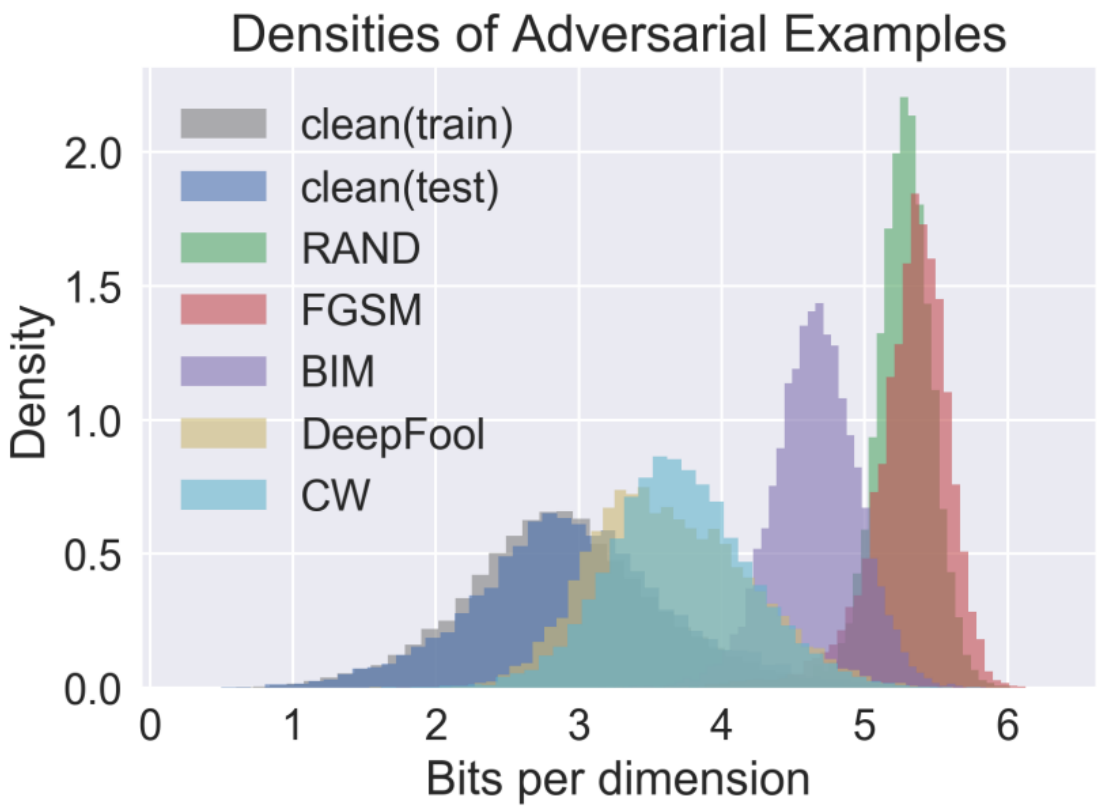
Keep in mind that the sampling is **sequential** (hence, heavy).



Application of the power of evaluation: **PixelDefend**

https://arxiv.org/pdf/1710.10766.pdf


dog
+noise
ostrich
mantis
+noise
ostrich


Densities of Adversarial Examples
clean(train)
clean(test)
RAND
FGSM
BIM
DeepFool
CW
Density
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6
Bits per dimension