Autoregressive Models

Volodymyr Kuleshov

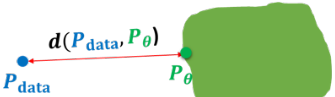
Cornell Tech

CS 5788 Guest Lecture

The Task of Generative Modeling

Suppose we are given a training set of examples, e.g., images of dogs

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



$\theta \in M$

Model family

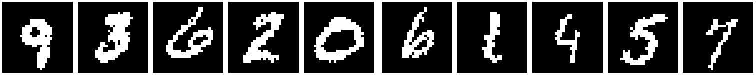
Our Goal: define a probability distribution $p(x)$ over images x.

1. **Modeling**: Define a family of distributions p_θ parameterized by θ.
2. **Learning**: Search for model parameters θ based on training data $\mathcal{D}$.
3. **Inference**: Generate data x by sampling from $p_\theta(x)$.

Previous lectures: VAEs, EBMs, diffusion. This lecture: Autoregressive models.

Running Example: A Generative Model for MNIST

Suppose we are given a dataset $\mathcal{D}$ of handwritten digits (binarized MNIST)



- Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1).
- **Our Goal:** Learn a probability distribution $p(x) = p(x_1, \cdots, x_{784})$ over $x \in \{0, 1\}^{784}$ such that when $x \sim p(x)$, x looks like a digit.

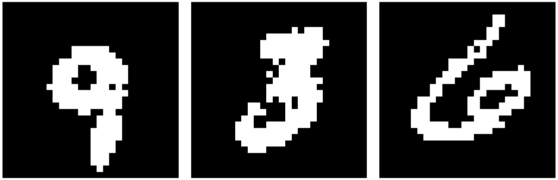
Probabilistic Models: Basic Discrete Distributions

How do we model a pixel in image? We can use a discrete distribution. For example:

- Bernoulli distribution:
  - Domain: $\{White, Black\}$
  - Specify $P(X = White) = p$. Then $P(X = Black) = 1 - p$.
  - Write: $X \sim Ber(p)$
  - Sampling: flip a (biased) coin
- Categorical distribution: a (biased) m-sided dice
  - Domain: $\{1, \cdots, m\}$
  - Specify $P(Y = i) = p_i$, such that $\sum p_i = 1$
  - Representation: A big table with $m - 1$ rows. Row i contains p_i.

The Curse of Dimensionality in Probabilistic Models

Suppose we want to model a BW image of digit with $n = 28 \cdot 28$ pixels.



- Pixels $X_1, \ldots, X_n$ are modeled as binary (Bernoulli) random variables, i.e., $\text{Val}(X_i) = \{0, 1\} = \{\text{Black}, \text{White}\}$.
- How many possible images exist?

$$\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ times}} = 2^n$$

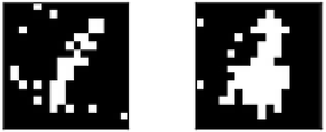
- **Modeling Attempt 1**: Define $p(x_1, \ldots, x_n)$ as a *categorical* distribution: a big table with one row per image $(x_1, \ldots, x_n)$. How long is the table? $2^n - 1$. This is a big problem!

Parameter-Efficient Probabilistic Models

- **Modeling Attempt 2**: Model pixels as independent Bernoullis, and define the probability of an image as their product:

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2)\cdots p(x_n)$$

- How many parameters to specify the joint distribution $p(x_1, \ldots, x_n)$?
- 2^n **probabilities can be described by** n **numbers** (if $|\text{Val}(X_i)| = 2$)!
- Independence assumption is too strong. Model not likely to be useful
  - For example, each pixel chosen independently when we sample from it.



Structure through conditional independence

- **Modeling Attempt 3**: By the Chain Rule, any distribution is a product of conditionals:

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

  This still needs $2^n - 1$ entries, chain rule does not buy us anything.

- Now make modeling choice that each x_i depends only on x_{i-1}, then

$$\begin{aligned} p(x_1, \ldots, x_n) &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid \cancel{x_1}, x_2) \cdots p(x_n \mid \cancel{x_1, \cdots,} x_{n-1}) \\ &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2) \cdots p(x_n \mid x_{n-1}) \end{aligned}$$

- How many parameters? $2n - 1$. Exponential reduction, as long as context length is bounded.

Two Ways of Modeling Conditionals

Autoregressive models specify p by via its conditionals $p(x_i \mid x_{i-1}, \cdots, x_1)$

- **Bayesian Nets** (classical): Conditionals are categoricals (tables)

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

  How many parameters to specify a categorical distribution $p(x_i \mid x_{i-1}, \cdots, x_{i-k})$ conditioned on k previous binary pixels? 2^k

- **Deep AR Models** (this lecture!): Conditionals could be too complex to be stored in tabular form; we parameterize them using **neural nets**:

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

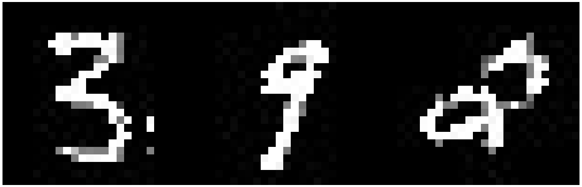
  A sufficiently deep neural net can approximate any function.

Deep Autoregressive Models: Overview

How do we model one term $p_{\mathrm{Neural}}(x_i \mid x_{i-1}, \cdots, x_1)$ using a neural net?

1. Choose distribution family over pixel x_i with parameters $\hat{x}_i$
   - If x_i is binary, use a Bernoulli distribution: $x_i \sim \mathrm{Ber}(\hat{x}_i)$, $\hat{x}_i \in [0, 1]$.
2. Define a neural network with parameters θ that predicts $\hat{x}_i$ from the previous pixels $x_{i-1}, \cdots, x_1$.
   - For example, $\hat{x}_i = \hat{x}_\theta(x_{i-1}, \cdots, x_1)$, where $\hat{x}_\theta$ is a multi-layer perceptron (MLP) with parameters θ.
   - Two different types of parameters: θ (parameters of the neural net) and $\hat{x}_i$ (parameters of the distribution, output of neural net)

Deep Autoregressive Models: Overview



Autoregressive models support standard generative modeling tasks:

- Easy to sample from
  1. Sample $\overline{x}_0 \sim \text{Ber}(\hat{x}_0)$. Compute $\hat{x}_1 = \hat{x}_\theta(\overline{x}_0)$
  2. Sample $\overline{x}_1 \sim \text{Ber}(\hat{x}_1)$. Compute $\hat{x}_2 = \hat{x}_\theta(\overline{x}_0, \overline{x}_1)$
  3. $\cdots$
- Easy to compute probability $p(x = \overline{x})$
  1. Compute $p(x_0 = \overline{x}_0)$ as $\hat{x}_0$ if $\bar{x}_0 = 1$ else $1 - \hat{x}_0$; let $\hat{x}_1 = \hat{x}_\theta(\overline{x}_0)$
  2. Compute $p(x_1 = \overline{x}_1 \mid x_0 = \overline{x}_0)$ as $\hat{x}_1$ if $\bar{x}_1 = 1$ else $1 - \hat{x}_1$
  3. $\cdots$
  4. Multiply together (sum their logarithms)

Deep Autoregressive Models: Looking At One Pixel

How do we define a Bernoulli probability $\hat{x}_i$ over binary pixel X_i given the previous pixel values $\mathbf{x}_{<i} = (x_1, \ldots, x_{i-1})$?

$$\hat{x}_i = p(X_i = 1 \mid \mathbf{x}_{<i}; \boldsymbol{\theta}) = \hat{x}_{\boldsymbol{\theta}}(\mathbf{x}_{<i})$$

- **Logistic Regression**: The simplest binary classification model (an artificial "neuron"):
  - Let $z(\boldsymbol{\theta}, \mathbf{x}) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$.
  - Let $\hat{x}_i = \hat{x}_{\boldsymbol{\theta}}(\mathbf{x}_{<i}) = \sigma(z(\boldsymbol{\theta}, \mathbf{x}))$, where $\sigma(z) = 1/(1 + e^{-z})$
- How do we interpret $\hat{x}_i$?
  - The probability $p(X_i = 1 \mid \mathbf{x}_{<i})$ that pixel i is on.
  - A “soft” prediction of the next pixel x_i given the previous pixels $\mathbf{x}_{<i}$.
- Can also use stacked layers of neurons, transformers. etc.

An Autoregressive Model From Logistic Regression

Let's define our first model. First, pick an ordering of the variables, e.g., a 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

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

- We parameterize the conditionals using **logistic regression**:

$$p(x_1, \cdots, x_{784}) = p(x_1; \theta) p_{\text{logit}}(x_2 \mid x_1; \boldsymbol{\theta}) p_{\text{logit}}(x_3 \mid x_1, x_2; \boldsymbol{\theta}) \cdots \\ p_{\text{logit}}(x_n \mid x_1, \cdots, x_{n-1}; \boldsymbol{\theta})$$

- More explicitly:
  - $p(X_1 = 1; \theta) = \theta$, $p(X_1 = 0) = 1 - \theta$
  - $p_{\text{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 x_1)$
  - $p_{\text{logit}}(X_3 = 1 \mid x_1, x_2; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 x_1 + \boldsymbol{\theta}_2 x_2)$
- We are using parameterized functions (e.g., logistic regression above) to predict next pixel given all the previous ones.

Training an Autoregressive Model

Given training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$, **maximum likelihood learning** solves:

$$\max_{P_\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log P_\theta(\mathbf{x})$$

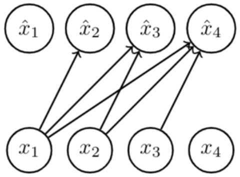
We want the model to assign high probability to the data $\mathcal{D}$.

In an autoregressive model, this problem becomes

$$\max_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i=1}^{n} \log p_{\mathrm{neural}}(x_i | \mathbf{x}_{<i}; \theta)$$

To optimize (and differentiate the objective), we need to compute (in parallel) the log-likelihood of each pixel x_i given the previous pixels $\mathbf{x}_{<i}$.

Likelihood Evaluation in an Autoregressive Model



FVSBN

- How to evaluate $p(x_1, \cdots, x_{784})$?
  - First, we compute the parameters $\hat{x}_i$ of each Bernoulli distribution
  - Then we evaluate $p(x_1, \cdots, x_{784})$. In the above example:

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

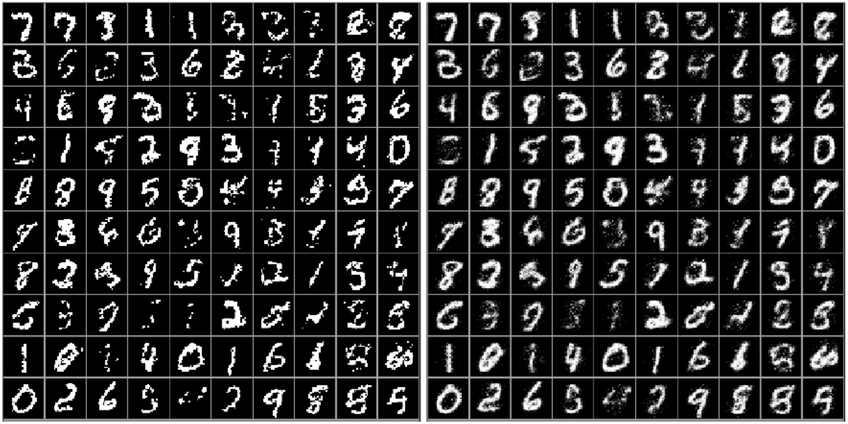
- Likelihood evaluation is **parallelizable**, hence is fast.
- Once we compute the likelihood, we can optimize the parameters θ using gradient descent.

Sampling in an Autoregressive Model

How do we use this model for the generative modeling tasks we defined earlier?

- How to sample from $p(x_1, \cdots, x_{784})$?
  1. Sample $\overline{x}_1 \sim p(x_1)$ (`np.random.choice([1,0],p=[`$\hat{x}_1, 1 - \hat{x}_1$`]))`)
  2. Sample $\overline{x}_2 \sim p(x_2 \mid x_1 = \overline{x}_1) = \text{Ber}(\hat{x}_2)$.
  3. Sample $\overline{x}_3 \sim p(x_3 \mid x_1 = \overline{x}_1, x_2 = \overline{x}_2) = \text{Ber}(\hat{x}_3) \cdots$
- Recall that each $\hat{x}_i$ is the output of a neural network $x_\theta(\mathbf{x}_{<i})$. Here, $\mathbf{x}_{<i}$ are the previously sampled pixels. We must sample all previous pixels before x_i.
  - For example, to sample x_3, we must compute $\hat{x}_3 = \hat{x}_\theta(x_1, x_2)$, and so we must have previously sampled x_1, x_2.
  - This means autoregressive models are **sequential** generative models. Sampling is not parallelizable, hence is typically slow.

Samples From an Autoregressive Model on MNIST



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

Continuous Variables

Autogressive models are not limited to discrete variables.

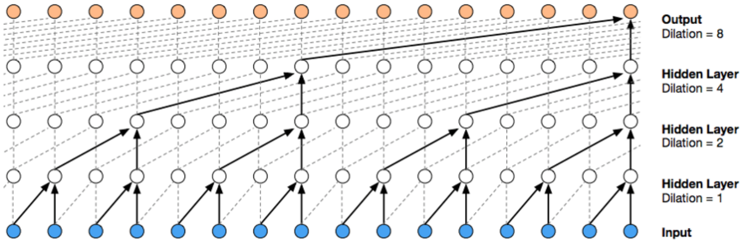
- If $\boldsymbol{X}$ is a continuous random vector, we can usually represent it using its **joint probability density function**:
  - Gaussian: if $p_X(x) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(x - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(x - \boldsymbol{\mu})\right)$
- Chain rule, Bayes rule, etc all still apply.
- Easy to extend AR to continuous variables. For example, can choose Gaussian conditionals $p(x_i \mid x_{<i}) = \mathcal{N}(\mu_\theta(x_{<i}), \Sigma_\theta(x_{<i}))$.

Pros & Cons of Autoregressive Models

1. **Pros**
   - Exact likelihood evaluation & training (unlike VAEs, EBMs, diffusion)
     - This enables efficient training that optimizes the likelihood (not an approximation)
     - Also supports applications like outlier detection, anomaly detection, etc.
   - Exact sampling (like VAEs, but unlike EBMs, diffusion models)
   - Generates sequences of arbitrary length
2. **Cons**
   - Sampling is sequential and hence is typically slow (unlike VAEs, but also like EBMs, diffusion models)
   - No good way to get latent representations (unlike VAEs, but also like EBMs, diffusion models)

Example: WaveNet (Oord et al., 2016)

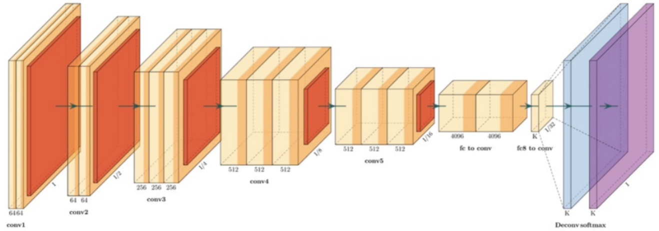
WaveNet is a sophisticated autoregressive generative model for raw audio.



- Its computational graph looks like a masked autoencoder—the output $\hat{x}$ (top) is a shifted version of the input x (bottom).
- It relies on a few technical ideas: causal convolutions and dilation.

Recall: Convolutional Neural Networks

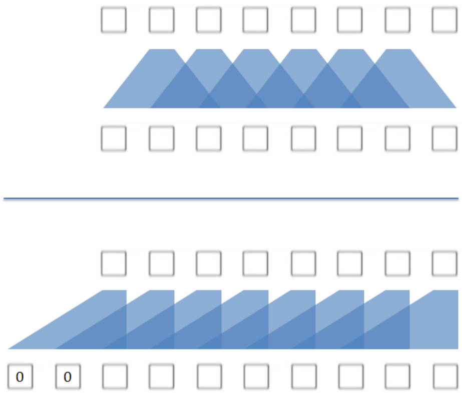
Convolutions are a translationally invariant operation commonly used in neural networks for image classification.



- WaveNet uses convolutions to parameterize a mapping of inputs x to outputs $\hat{x}_i$.
- This means that the parameters to generate each $\hat{x}_i$ from $x_{<i}$ are the same for each i (i.e., all $p(x_i|x_{<i})$ have the same parameters).

Causal Convolutions

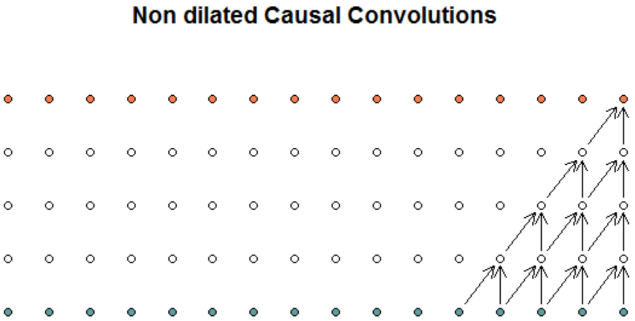
Regular 1D convolutions (top) have symmetrical receptive fields—this is a problem because x_{i+1} is part of the receptive field for predicting $\hat{x}_i$.



Causal convolutions (bottom) mask parts of the filter that touch the future

Dilated Convolutions

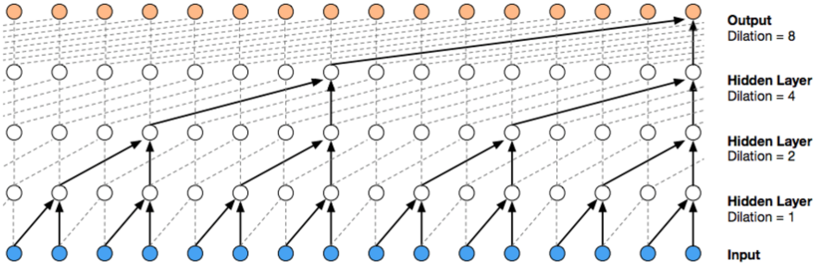
Normal convolutions in Wavenet would look like this:



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

WaveNet (Oord et al., 2016)

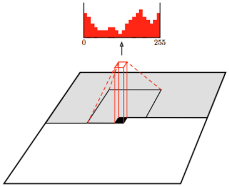
WaveNet is a powerful model for generating raw audio and speech.



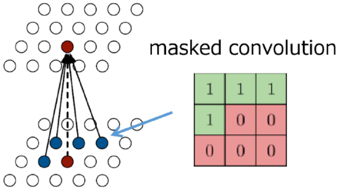
Dilated convolutions increase the receptive field: kernel only touches the signal at every 2^d entries.

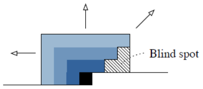
PixelCNN (Oord et al., 2016)

We can also use convolutional architecture to predict next pixel given context (a neighborhood of pixels).



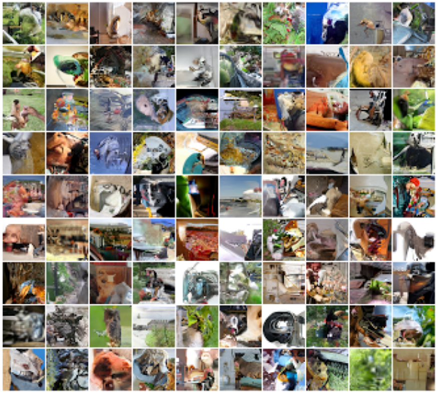
Challenge: Has to be autoregressive. Masked convolutions preserve raster scan order and ensure that $\hat{x}_i$ only depends on $x_{<i}$ (i.e., no "future" pixels).





PixelCNN

Samples from the model trained on Imagenet (32×32 pixels).



However, on image generation, AR models are not competitive with diffusion because likelihood is not a good measure of sample quality.

Example: Character-Level Autoregressive Models

Autoregressive models are especially powerful for modeling sequences of discrete tokens, such as text. Here is a toy example:

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 model:
   - $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$
3. Because x_i are discrete characters, $\mathbf{x}_\theta(\mathbf{x}_{<i})$ is a vector of probabilities for each possible character at position i.
   - For example, $\hat{x}_2 = \mathbf{x}_\theta(x_1 = h) = [0.1, 0.7, 0.1, 0.1]$ means $p(x_2 = e|x_1 = h) = 0.7$, $p(x_2 = h|x_1 = h) = 0.1$, etc.

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.

Towards Large Language Models

Autoregressive models are the basis of modern LLMs

- Their left-to-right factorization is a natural fit for modeling text, which is a sequence of discrete tokens.
- Maximizing the likelihood is an objective that works well for text.
- Variable-length generations can be sampled one token at a time.

The main differences relative to the models we have seen are:

- Transformer architecture instead of RNNs or CNNs.
- Much larger scale (billions of parameters, trained on billions of tokens).

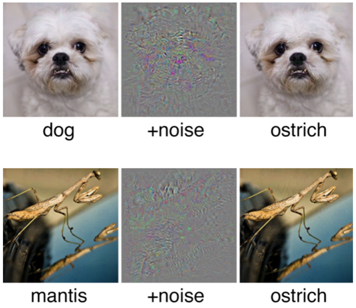
Summary of Autoregressive Models

- Easy, but slow 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
- No natural way to get latent representations, do unsupervised learning
- Autoregressive models are the basis of modern LLMs

Next lecture: parallel autoregressive models via discrete diffusion.

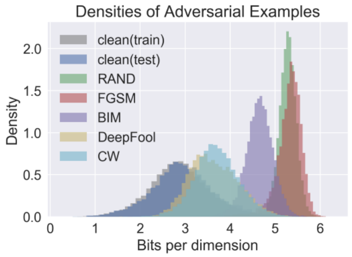
Application in Adversarial Attacks and Anomaly detection

Machine learning methods are vulnerable to adversarial examples



Can we detect them?

PixelDefend (Song et al., 2018)



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