# Autoregressive Models

MUSTAFA HAJIJ

# Introduction

Autoregressive model over $x$ factor the joint distribution as the following product of conditionals:

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

The problem becomes modeling the conditional probability distribution : $P(x_i | x_1, x_2, \ldots, x_{i-1})$

# Introduction

Autoregressive model over $x$ factor the joint distribution as the following product of conditionals:

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

The problem becomes modeling the conditional probability distribution : $P(x_i|x_1, x_2, \ldots, x_{i-1})$

Interpretation : Given previous values $x_1, x_2, \ldots, x_{i-1}$ , these models do not output a value for $x_i$, they output the predictive probability distribution $P(x_i|x_1, x_2, \ldots, x_{i-1})$ for $x_i$.

# Introduction

Any autoregressive model can be run sequentially to generate a new sequence : start with your seed $x_1, x_2, \ldots, x_k$ and predict $x_{k+1}$. Then use $x_2, x_3, \ldots, x_{k+1}$ to predict $x_{k+2}$ , and so on.

# PixelCNN

- **PixelCNN** is a generative model that predicts the next pixel in an image given the previous pixels.
- It leverages the idea of autoregressive modeling, where the image is generated pixel by pixel in a specific order.

# PixelCNN

- **Architecture Overview:**

- Consists of multiple convolutional layers.
- Each layer is designed to ensure that the generation of a pixel depends only on previously generated pixels.

# PixelCNN

•**Masking Concept:**Masks are used in the convolutional layers to prevent information from flowing from future pixels to the current pixel.

•**Types of Masks:**

- **A-type Mask:** Used in the first layer to ensure that a pixel does not depend on itself.
- **B-type Mask:** Used in subsequent layers, allowing the pixel to depend on itself and previous pixels.

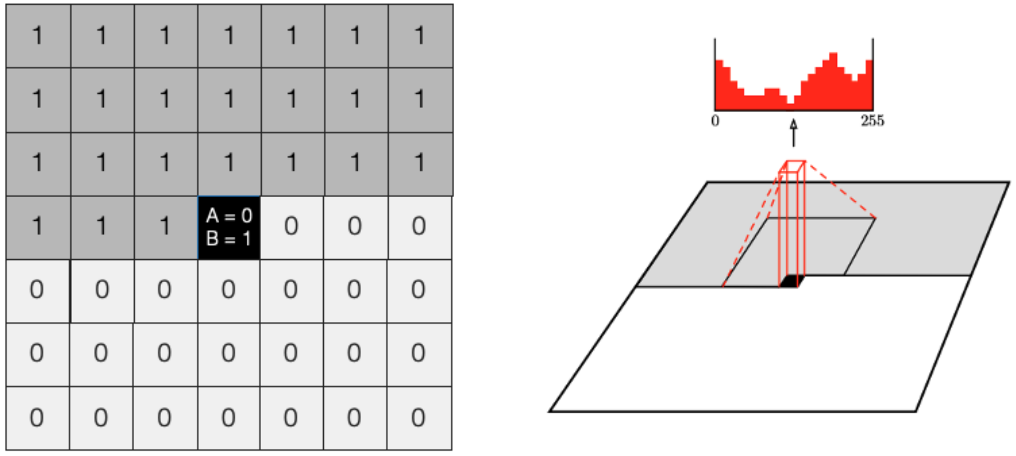
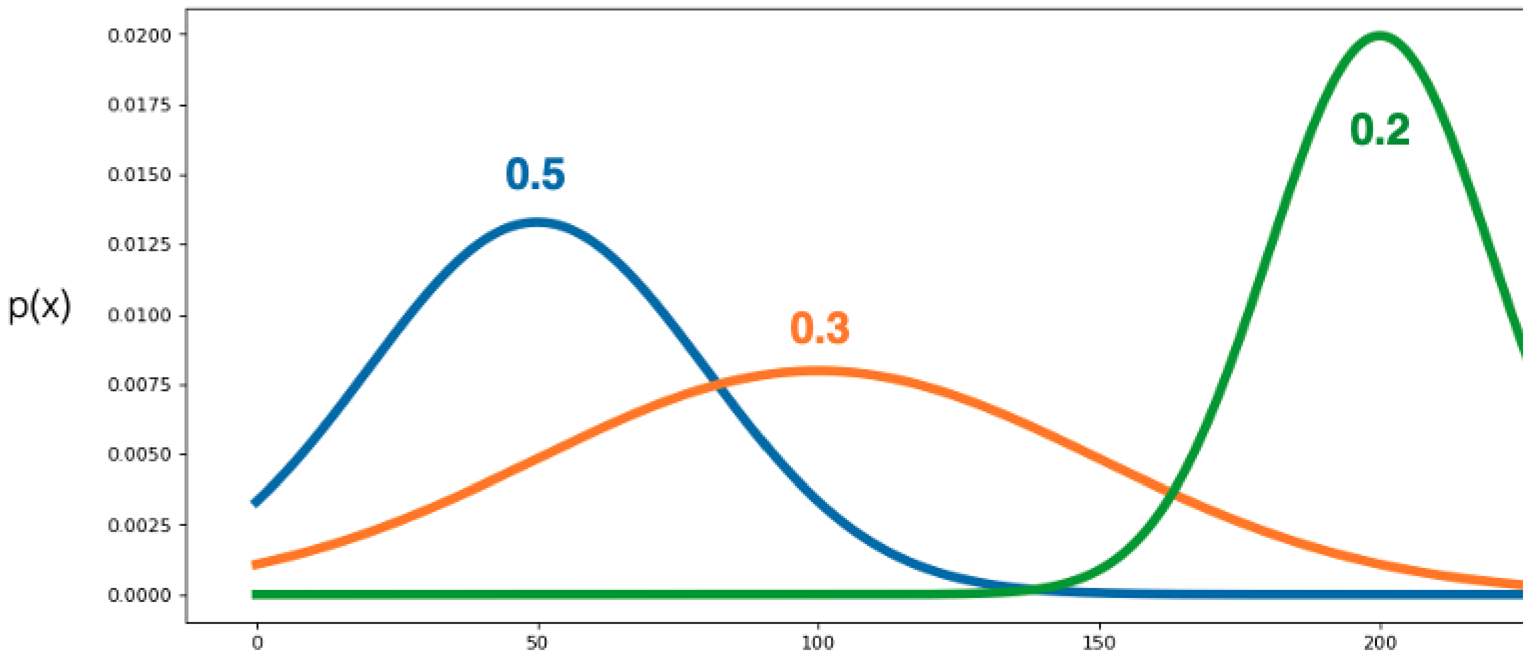



*Figure 5-13. Left - a convolutional filter mask. Type A masks the central pixel and Type B does not mask the central pixel. Right - a mask applied to a set of pixels to predict the distribution of the central pixel value (source: Conditional Image Generation with PixelCNN Decoders, van den Oord et al. https://arxiv.org/pdf/1606.05328).*

# PixelCNN

- **Problems: very slow to sample, for an image of 64 by 64, we need to sample 64 by 64!**

One solution is to replace the head of softmax over 256 by a mixture distribution.
For instance in the following example, we have a mixture with three kernals.

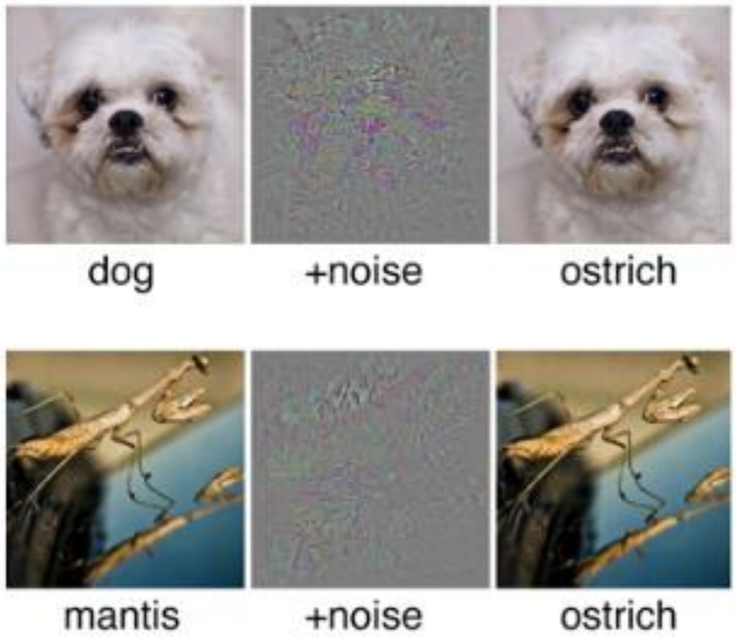



[Generative_Deep_Learning_2nd_Edition/notebooks/05_autoregressive/03_pixelcnn_md/pixelcnn_md.ipynb at main · davidADSP/Generative_Deep_Learning_2nd_Edition (github.com)]

# PixelCNN

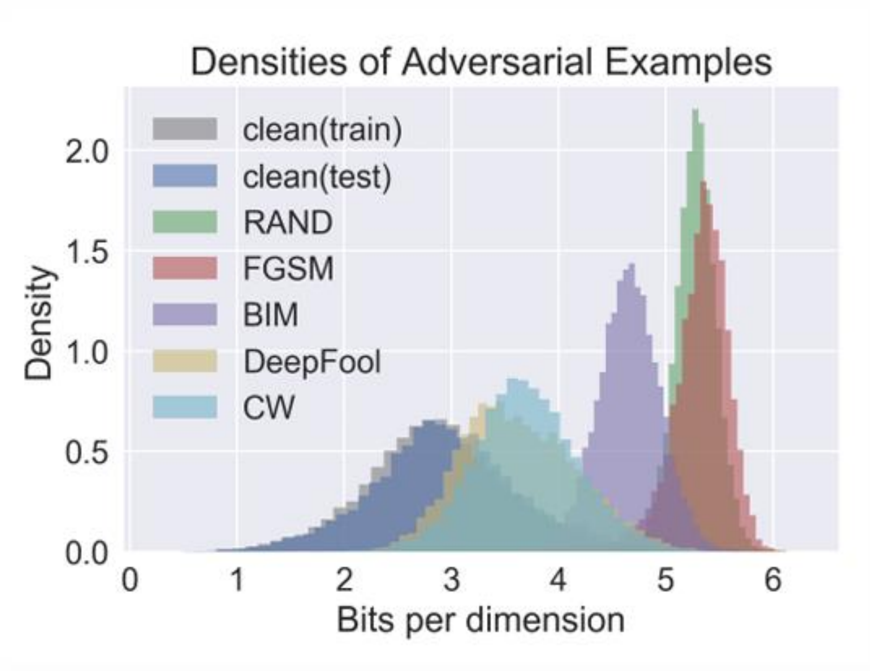
# Applications : Adversarial Attacks and Anomaly detection

Machine learning models are vulnerable to adversarial examples




Can we detect such examples?

# Applications : Adversarial Attacks and Anomaly detection




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

- 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