DUL: Autoregressive Models

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

May 29, 2025

Table of Contents

Motivation: The Challenge of High-Dimensional Data

- **The Core Problem:** We aim to estimate distributions of complex, high-dimensional data.
  - *Example:* A 128x128x3 RGB image has $128 \times 128 \times 3 = 49,152$ dimensions. Directly modeling such a high-dimensional space is practically impossible due to the "curse of dimensionality."
- **Desired Properties for our Models:**
  - **Computational Efficiency:**
    - Efficient training (fast convergence, reasonable resource usage).
    - Efficient model representation (compact size).
  - **Statistical Efficiency:**
    - Expressiveness (ability to capture complex data relationships).
    - Generalization (performing well on unseen data from the same distribution).
  - **Performance Metrics:**
    - High sampling quality and speed (for generating new data).
    - Good compression rate and speed (for data storage and transmission).

- **What are Autoregressive Models?**
  - **Goal:** To model and generate complex, high-dimensional data distributions.
  - **Core Idea:** Factorise the joint distribution of data using the **chain rule**, expressing it as a product of conditional probabilities:

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

This allows us to model complex data one element (or "token") at a time, conditioned on previously generated elements.

Fundamentals and Motivation (cont.)

- **Why do we need them?**
  - **Curse of Dimensionality:** Directly modeling high-dimensional data distributions is computationally infeasible. Autoregressive models overcome this by breaking down the problem.
  - **Solving Key Problems:**
    - **Data Generation:** Synthesize realistic images, audio, video, and text.
    - **Data Compression:** Create efficient representations for storage and transmission.
    - **Anomaly Detection:** Identify data points that deviate from the learned distribution.
- **Applications**
  - **Text Generation:** Large Language Models (e.g., GPT), RNNs.
  - **Image Generation:** PixelCNN, PixelRNN.
  - **Audio Synthesis:** WaveNet.

Practical Implementations and Considerations

- **Key Architectures and Models**
  - **Classic Approaches:**
    - **Bayesian Networks:** Graphical models that encode conditional dependencies.
    - **Recurrent Neural Networks (RNNs):** Naturally suited for sequential data modeling.
  - **Modern Neural Approaches:**
    - **Masked Autoencoder for Distribution Estimation (MADE):** A feed-forward neural network with masks to enforce the autoregressive property.
    - **Causal Masked Neural Models:** Neural networks designed for autoregressive generation by preventing information flow from future elements.
    - **Convolutional:** PixelCNN, WaveNet (use masked convolutions).
    - **Attention-based: Transformers** with causal masks (enable flexible context modeling).

- **Design Considerations**
  - **Tokenization:** Discretizing continuous data (e.g., images into patches, audio into samples) or using natural discrete units (words, bytes) to form sequences for modeling.
  - **Caching:** Essential for efficient inference and generation in autoregressive models, by reusing previously computed states.
  - **Architecture Choices:**
    - **Decoder-only vs. Encoder-Decoder:** Trade-offs exist for generation and conditioning capabilities.
    - **Newer Recurrent Models:** Evolution of RNNs and hybrid architectures continue to emerge.

Key Components and Architectures

- Autoregressive models are a class of statistical models where observations from previous time steps are used to predict the value at the current time step.
- They are commonly used for analyzing and forecasting time series data.
- The core principle is that the current value of a time series can be explained by its past values.

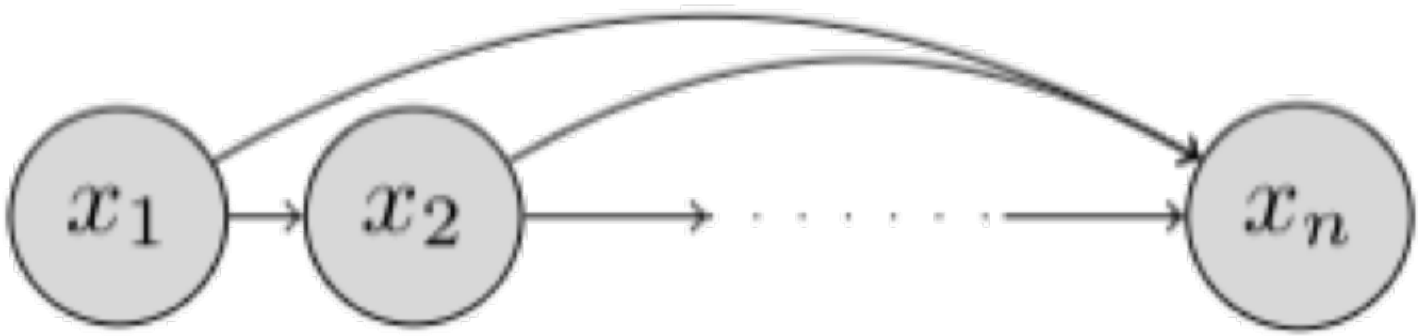


Figure 2: Graphical model for an autoregressive network

Key Components and Architectures (cont.)

- In the MNIST dataset example, we can set an ordering for all the random variables from top-left X_1 to bottom-right X_{784}
- Without loss of generality, we can use chain rule for factorization

$$\mathcal{P}(x_1, \cdots, x_{784}) = \mathcal{P}(x_1)\mathcal{P}(x_2|x_1)\mathcal{P}(x_3|x_1, x_2)\cdots\mathcal{P}(x_{784}|x_1, x_2, \cdots, x_{784})$$

- Parameterizing the above and using the sigmoid function for binarization,
  - $\mathcal{P}_{CPT}(X_1 = 1; \alpha^1) = \alpha^1, \mathcal{P}(X_1 = 0) = 1 - \alpha^1$
  - $\mathcal{P}_{logit}(X_2 = 1|x_1; \alpha^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1)$
  - $\mathcal{P}_{logit}(X_3 = 1|x_1, x_2; \alpha^3) = \sigma(\alpha_0^3 + \alpha_1^3 x_1 + \alpha_2^3 x_2)$

Autoregressive Models: Historical Models

Before the advent of modern architectures such as Transformers and PixelCNN, several foundational neural autoregressive models established the basis for tractable density estimation using neural networks. Notable among these are:

- **FVSBN (Fully Visible Sigmoid Belief Networks):** Introduced a factorized approach for modeling complex distributions, enabling efficient computation of likelihoods.
- **NADE (Neural Autoregressive Distribution Estimator):** Extended the autoregressive framework by leveraging neural networks to model high-dimensional binary data, allowing for scalable and tractable density estimation.
- **RNADE (Real-valued NADE):** Adapted the NADE model to handle real-valued data, broadening the applicability of autoregressive models to a wider range of tasks.

These early models not only demonstrated the feasibility of neural autoregressive density estimation but also inspired the development of more sophisticated architectures that followed.

Fully Visible Sigmoid Belief Network (FVSBN)

Introduced by: Bengio and Bengio (2000)

Key Idea:

- A Bayesian network defined over observed variables, where each conditional distribution is modeled using a sigmoid (logistic) unit.
- All variables are visible; there are no latent or hidden variables in the model.

Key Features:

- Each variable is conditionally independent of all others given its predecessors.
- The model captures complex dependencies between variables through a factorized representation.
- It allows for efficient computation of the joint probability distribution over the observed variables.

Fully Visible Sigmoid Belief Network (FVSBN) (cont.)

Formulation:

- The joint probability of the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is factorized as:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

where $x_{<i} = (x_1, x_2, \ldots, x_{i-1})$.

- Each conditional is parameterized as:

$$p(x_i = 1 \mid x_{<i}) = \sigma(W_i x_{<i} + b_i)$$

where W_i is a row vector of weights for the i-th variable, b_i is a bias term, and $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

Fully Visible Sigmoid Belief Network (FVSBN) (cont.)

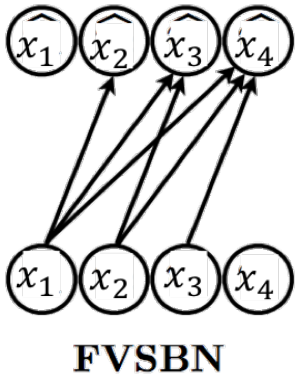


Figure 3: A fully visible sigmoid belief network over 4 variables

Fully Visible Sigmoid Belief Network (FVSBN) (cont.)

- The conditional variables $X_i|X_1, X_2, \cdots X_{i-1}$ are Bernoulli with parameters

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

- To evaluate $\mathcal{P}(x)$, we just multiply all the conditionals.
- To sample new images, we sample each X_i chronologically.
  - Sample $\overline{x}_1 \sim \mathcal{P}(x_1)$ np.random.choice([1,0], p=$[\hat{x}, 1-\hat{x}]$)
  - Sample $\overline{x}_2 \sim \mathcal{P}(x_2|X_1 = \overline{x}_1)$
  - Sample $\overline{x}_3 \sim \mathcal{P}(x_3|X_1 = \overline{x}_1, X_2 = \overline{x}_2)$
- Total number of parameters $= 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$
- **Pros:** The likelihood is tractable and can be computed exactly.
- **Cons:** Poor scalability: each variable x_i requires its own set of weights and bias, leading to a quadratic number of parameters as the number of variables increases.

FVSBN: Results

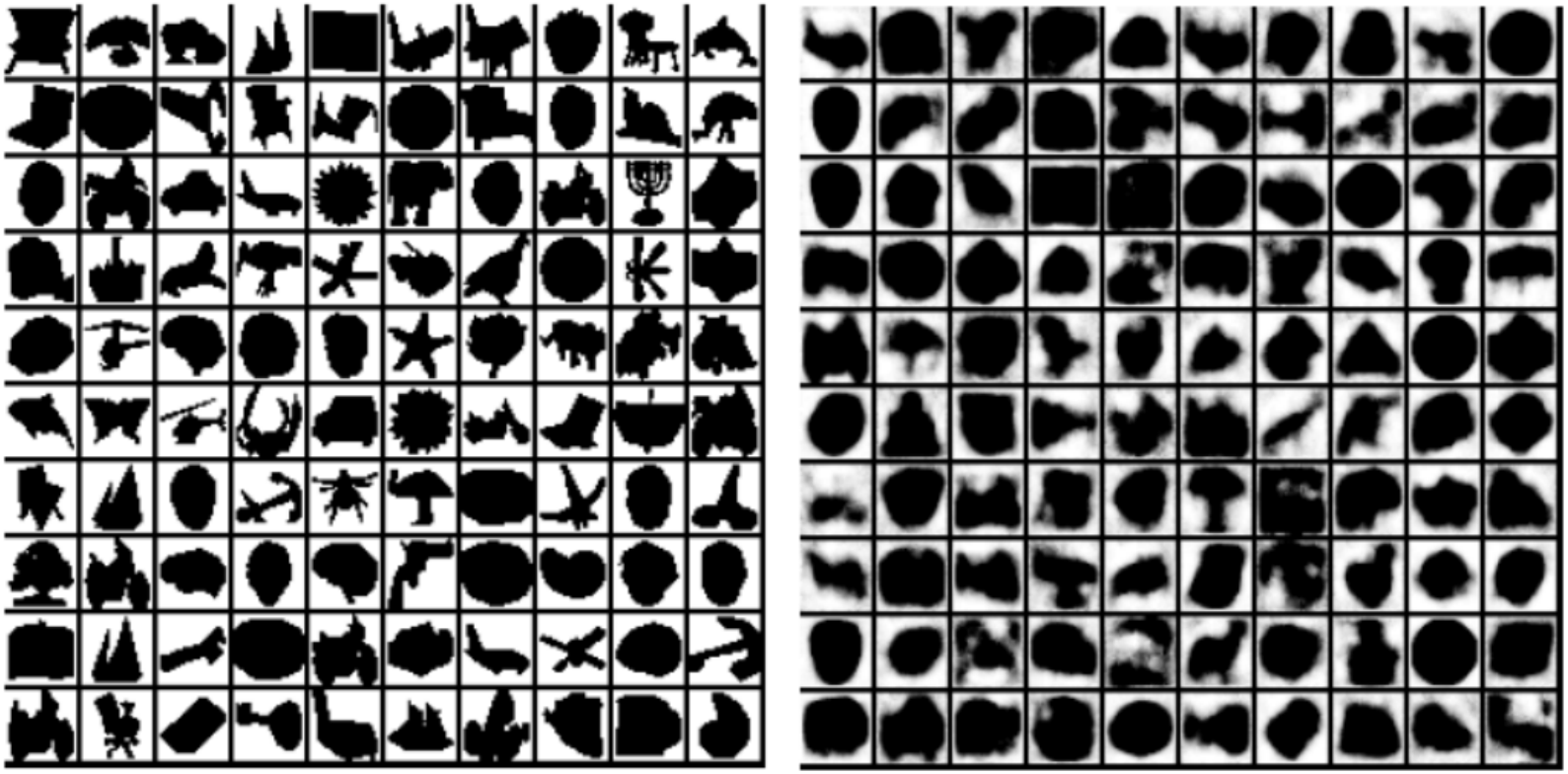


Figure 4: FVSBN results. Left: Training data. Right: Samples generated by model (Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015)

Neural Autoregressive Density Estimation (NADE)

Introduced by: Larochelle and Murray (2011)

Goal: Provide an efficient and scalable version of Fully Visible Sigmoid Belief Networks (FVSBN) by using shared weights and a feedforward neural network.

Main Idea:

- Uses a masked neural network to estimate each conditional probability:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

- Efficiently computes gradients via weight sharing and dynamic masking.

Advantages:

- Tractable and scalable.
- Inspired later models such as MADE and normalizing flows.

Neural Autoregressive Density Estimation (NADE) (cont.)

Model Structure:

- Each variable x_i is modeled conditionally on all previous variables $x_{<i}$.
- Uses a neural network to compute the conditional probabilities.
- The weights are shared across different variables, reducing the number of parameters significantly.

Formulation:

- The conditional probability is modeled as:

$$p(x_i \mid x_{<i}) = \sigma(\alpha^{(i)} h_i + b_i)$$

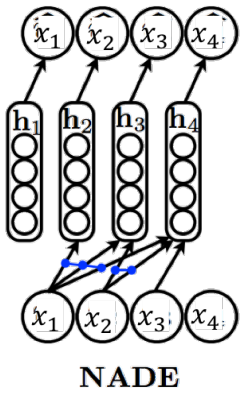
where h_i is a hidden representation computed from the previous variables, $\alpha^{(i)}$ are the weights, and b_i is a bias term.

Neural Network Layer:

NADE uses a neural network layer instead of just logistic regression to improve model performance.

$$h_i = \sigma(\mathbf{W}_{.,<i} x_{<i} + c)$$

$$f_i(x_1, x_2, \cdots, x_{i-1}) = \sigma(\alpha^{(i)} h_i + b_i)$$



NADE

Figure 5: A neural autoregressive density estimator over four variables. Blue connections donate shared weights.

NADE Results

Figure 6: NADE results. Left: Samples generated by model. Right: Conditional Probabilities $\hat{x}_i$ (The Neural Autoregressive Distribution Estimator, 2011)

Non-Binary Outputs

- What if the output variable is not binary? For example, we have to predict pixel value between 0 and 255.
- One solution: Let $\hat{\mathbf{x}}_i$ parameterize a categorical distribution

$$h_i = \sigma(\mathbf{W}_{.,<i} x_{<i} + c) \tag{1}$$

$$\mathcal{P}(x_i | x_1, \cdots, x_{i-1}) = \text{Cat}(p_i^1, \cdots, p_i^K) \tag{2}$$

$$\hat{\mathbf{x}}_i = (p_i^1, \cdots, p_i^K) = \text{softmax}(A_i h_i + b_i) \tag{3}$$

- Softmax generalizes the sigmoid function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K probabilities (non-negative, sum to 1).

RNADE – Real-valued NADE

- How to model continuous random variables $X_i \in \mathbb{R}$?
- Solution: Let $\hat{\mathbf{x}}_i$ parameterize a continuous distribution.
- This was introduced in RNADE.
- $\hat{\mathbf{x}}_i$ defines the mean and stddev of Gaussian Random Variable (μ_i^j, σ_i^j)

RNADE – Real-valued NADE (cont.)

Introduced by: Uria et al. (2013)

RNADE (Real-valued Neural Autoregressive Density Estimator) extends NADE to handle continuous data.

- **Extension:** Uses a mixture of Gaussians for each conditional distribution.
- **Conditional Density:**

$$p(x_i \mid x_{<i}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i \mid \mu_k, \sigma_k^2)$$

- The parameters of each mixture component (weights π_k, means μ_k, variances σ_k^2) are produced by a neural network conditioned on $x_{<i}$.

RNADE – Real-valued NADE (cont.)

Formulation:

- The joint probability is factorized as:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

- Each conditional is modeled as:

$$p(x_i \mid x_{<i}) = \sum_{k=1}^{K} \pi_k(x_{<i}) \mathcal{N}(x_i \mid \mu_k(x_{<i}), \sigma_k^2(x_{<i}))$$

  where π_k, μ_k, and σ_k are outputs of a neural network conditioned on $x_{<i}$.

Applications:

- Density estimation for real-valued data (e.g., audio, speech).

RNADE – Real-valued NADE (cont.)

Pros:

- Powerful for modeling complex continuous distributions.
- Influential precursor to more advanced autoregressive flows.

Cons:

- Still suffers from scalability issues with high-dimensional data.
- Requires careful tuning of the number of mixture components.

Learning Autoregressive Models

أكاديمية كاوست
KAUST ACADEMY

- The goal of learning is to return a model $\hat{\mathcal{M}}$ that precisely captures the distribution $\mathcal{P}_{data}$ from which our data was sampled
- This is in general not achievable because of
  - limited data only provides a rough approximation of the true underlying distribution
  - computational reasons
- Example. Suppose we represent each MNIST digit with a vector $\mathbf{X}$ of 784 binary variables (black vs. white pixel). How many possible states (= possible images) in the model? $2^{784} \approx 10^{236}$. Even 10^7 training examples provide extremely sparse coverage!
- We want to select $\hat{\mathcal{M}}$ to construct the "best" approximation to the underlying distribution $\mathcal{P}_{data}$

Learning Autoregressive Models (cont.)

- We want to learn the full distribution so that later we can answer any probabilistic inference query
- We want to construct $\mathcal{P}_\theta$ as "close" as possible to $\mathcal{P}_{data}$ (recall we assume we are given a dataset $\mathcal{D}$ of samples from $\mathcal{P}_{data}$)

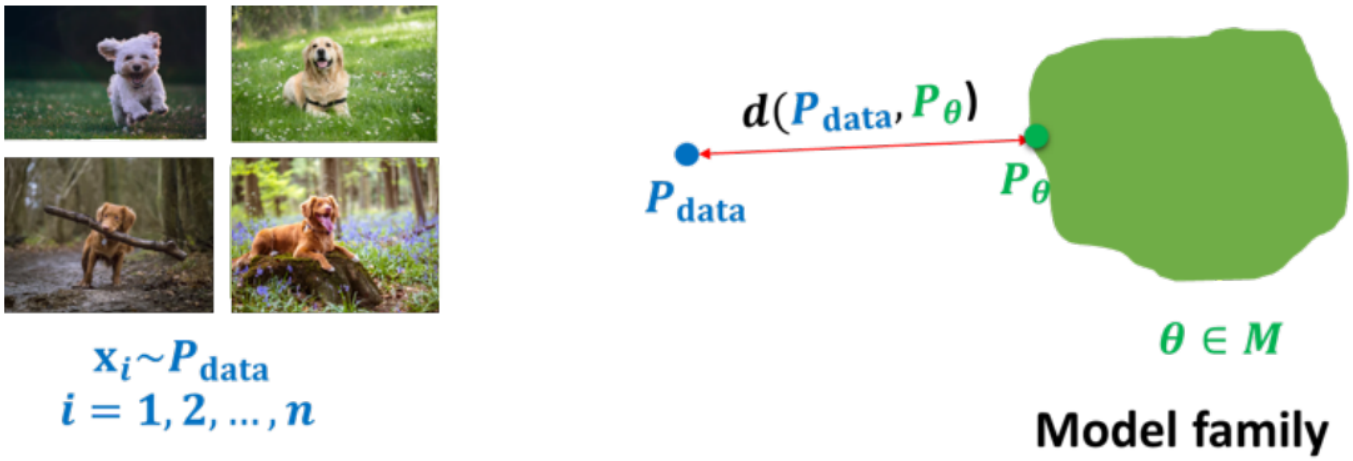


Figure 7: Learning a generative model. $d(\cdot)$ is a distance measure.

KL Divergence

- How do we evaluate "closeness" between model and data distribution?
- **Kullback-Leibler divergence** (KL-divergence) is one possibility:

$$D(\mathcal{P}_{data}||\mathcal{P}_\theta) = E_{x\sim\mathcal{P}_{data}}\left[log\left(\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}\right)\right] = \sum_x \mathcal{P}_{data}(x) log \frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}$$

- $D(\mathcal{P}_{data}||\mathcal{P}_\theta)$ iff the two distributions are the same.
- It measures the "compression loss" (in bits) of using $\mathcal{P}_\theta$ instead of $\mathcal{P}_{data}$.

Expected Log-Likelihood

- We can simplify this somewhat:

$$\begin{aligned} D(\mathcal{P}_{data}||\mathcal{P}_\theta) &= E_{x\sim\mathcal{P}_{data}}\left[log\left(\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}\right)\right] \\ &= E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_{data}(x)] - E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \end{aligned} \tag{4}$$

- The first term does not depend on $\mathcal{P}_\theta$. Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**

$$\begin{aligned} \arg\min_{\mathcal{P}_\theta} D(\mathcal{P}_{data}||\mathcal{P}_\theta) &= \arg\min_{\mathcal{P}_\theta} -E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \\ &= \arg\max_{\mathcal{P}_\theta} E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \end{aligned} \tag{5}$$

  - Asks that $\mathcal{P}_\theta$ assign high probability to instances sampled from $\mathcal{P}_{data}$, so as to reflect the true distribution
  - Because of log, samples x where $\mathcal{P}_\theta(x) \approx 0$ weigh heavily in objective

- Although we can now compare models, since we are ignoring $H(\mathcal{P}_{data})$, we don't know how close we are to the optimum.

- Problem: In general we do not know $\mathcal{P}_{data}$

Maximum Likelihood

- Approximate the expected log-likelihood

$$E_{x \sim \mathcal{P}_{data}}[log \mathcal{P}_\theta(x)]$$

with the empirical log-likelihood:

$$E_{\mathcal{D}} = \frac{1}{\mathcal{D}} \sum_{x \in \mathcal{D}} log \mathcal{P}_\theta(x)$$

- **Maximum likelihood learning** is then:

$$\arg\max_{\mathcal{P}_\theta} \frac{1}{\mathcal{D}} \sum_{x \in \mathcal{D}} log \mathcal{P}_\theta(x)$$

Recurrent Neural Networks (RNN)

أكاديمية كاوست
KAUST ACADEMY

- The next form of autoregressive models
- **Challenge**: model $\mathcal{P}(x_t|x_{1:t-1};\alpha^t)$. "History" $x_{1:t-1}$ keeps getting longer.
- **Idea**: keep a summary and recursively update it

Recurrent Neural Networks (RNN) (cont.)

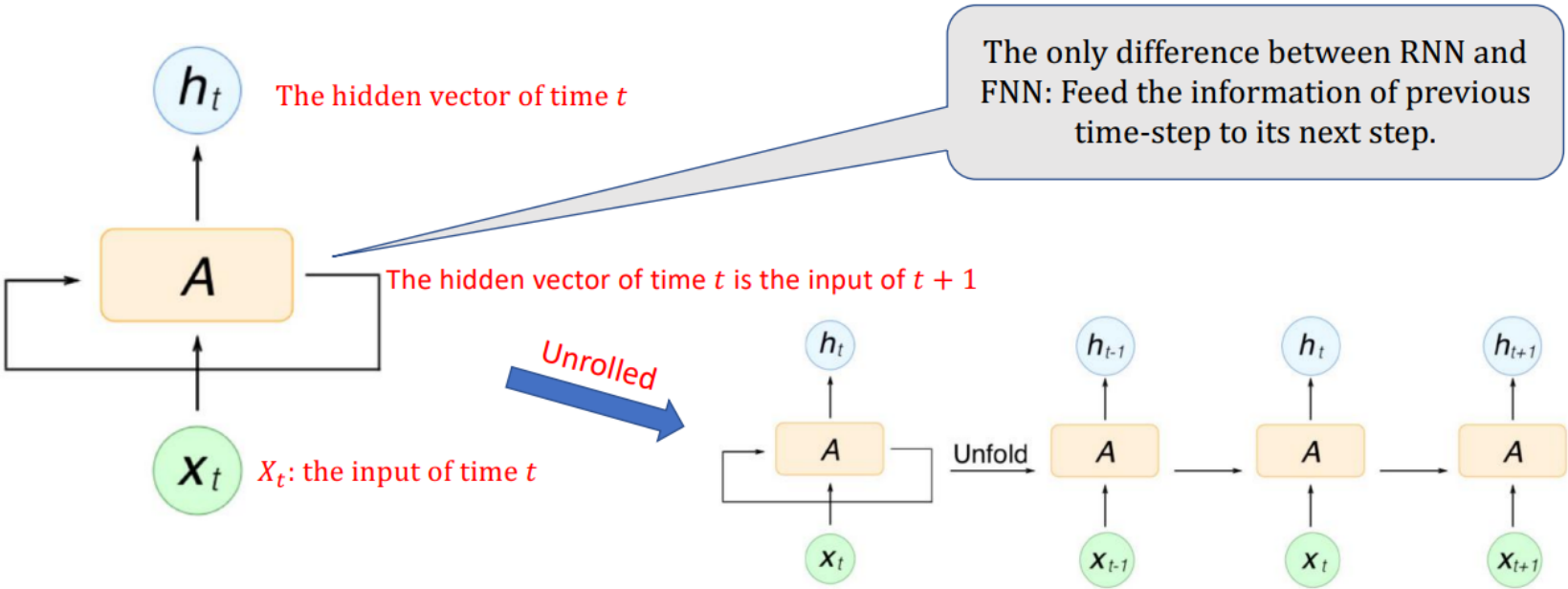


Figure 8: Recurrent Neural Network (RNN) sample architecture.

Recurrent Neural Networks (RNN) (cont.)

Pros:

- Can be applied to sequences of arbitrary length.
- Very general: For every computable function, there exists a finite RNN that can compute it

Cons:

- Still requires an ordering
- Sequential likelihood evaluation (very slow for training)
- Sequential generation (unavoidable in an autoregressive model)
- Can be difficult to train (vanishing/exploding gradients)

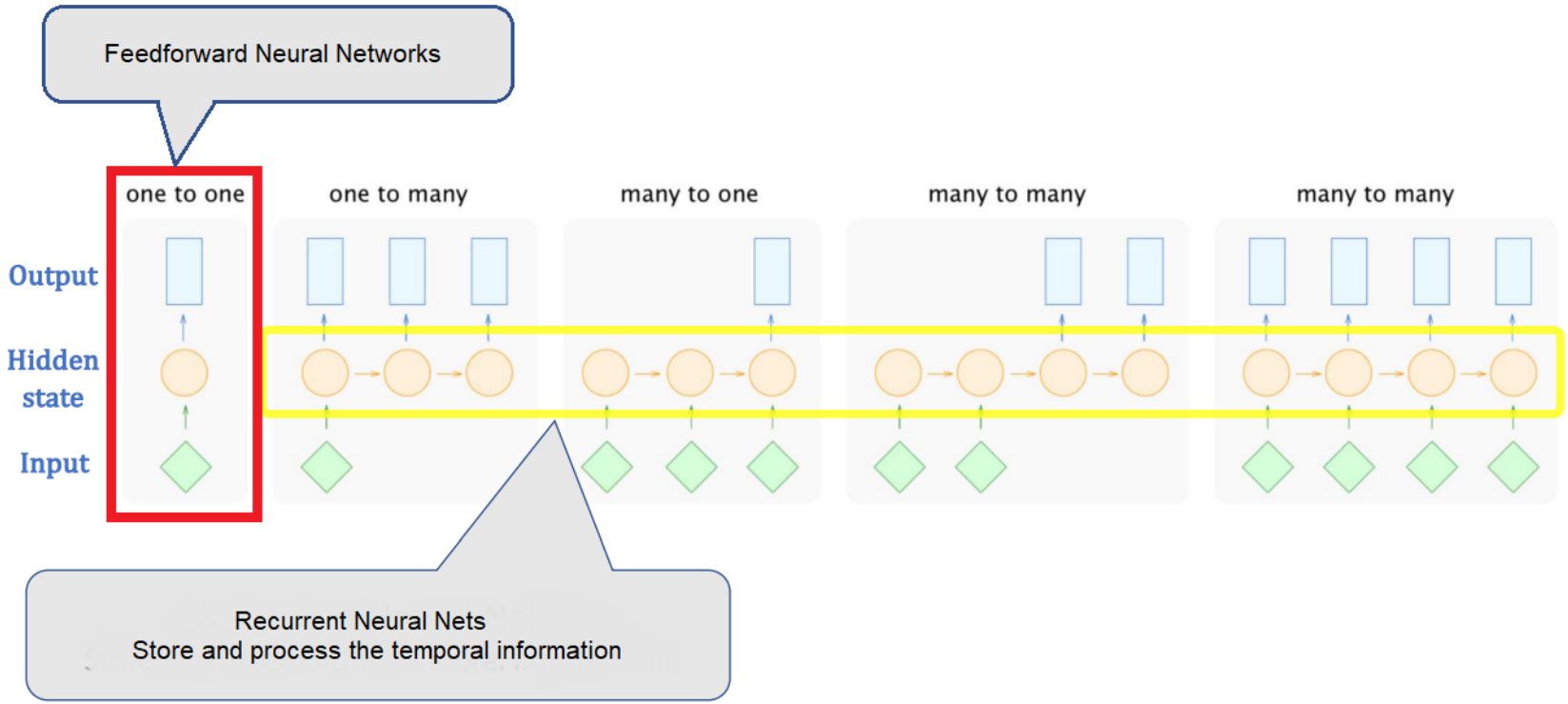


Figure 9: Different sequential modeling types

Image Captioning

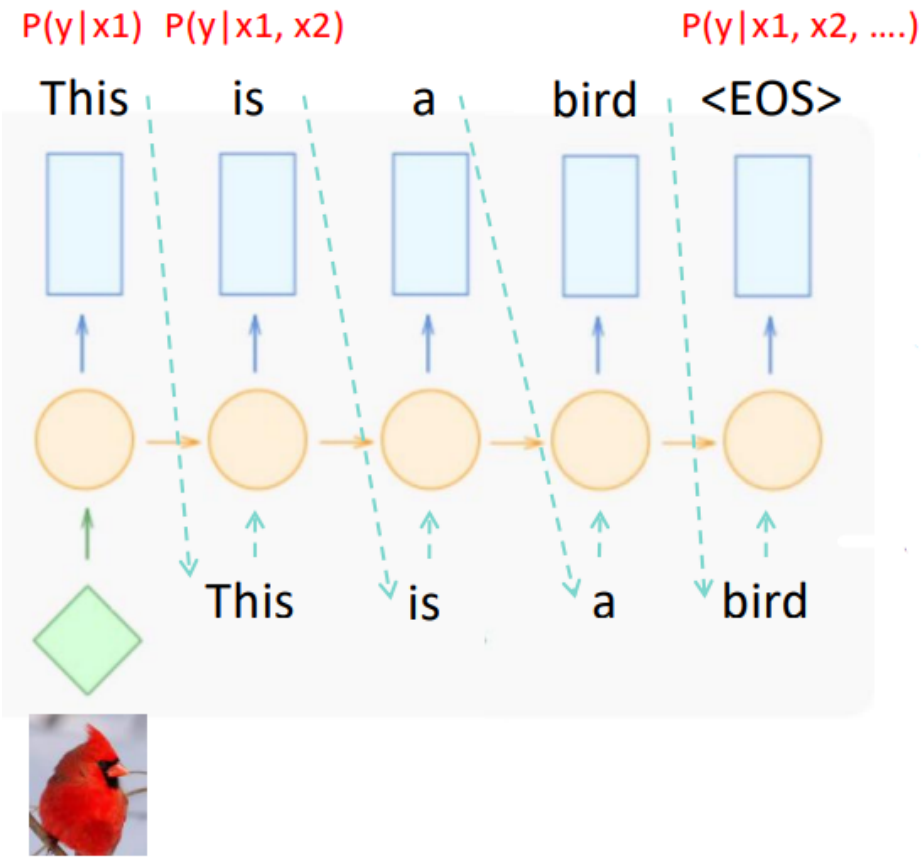


Figure 10: Image captioning with RNNs

Text Generation

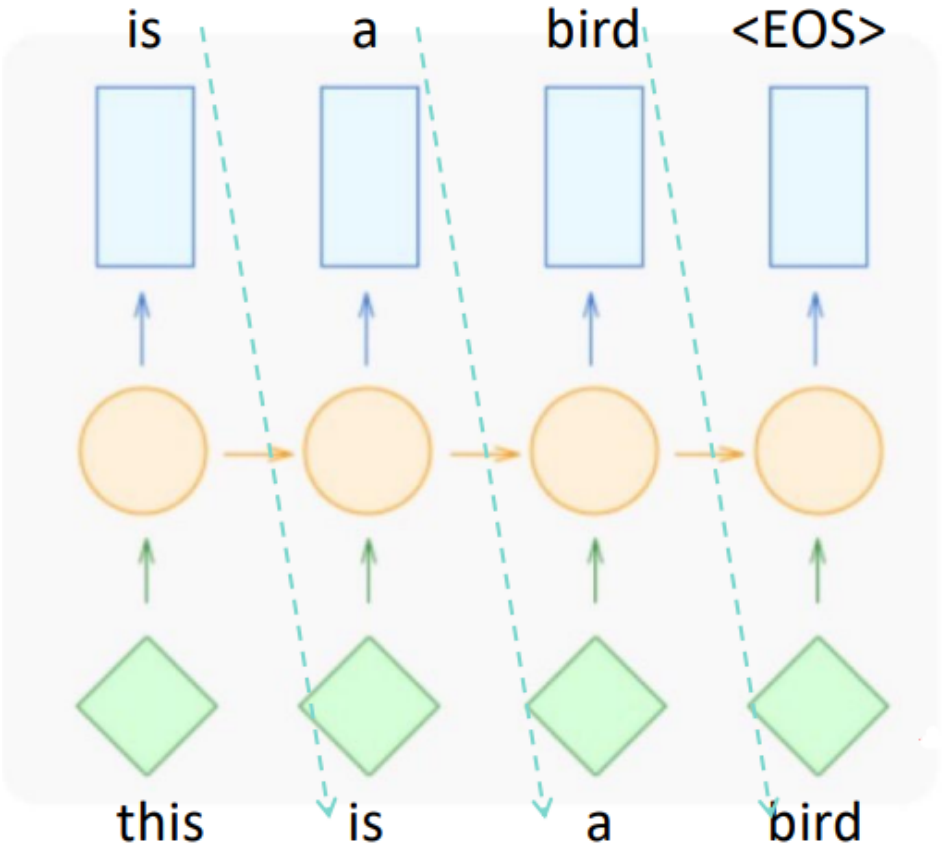


Figure 11: Text generation with RNNs

Chatbot

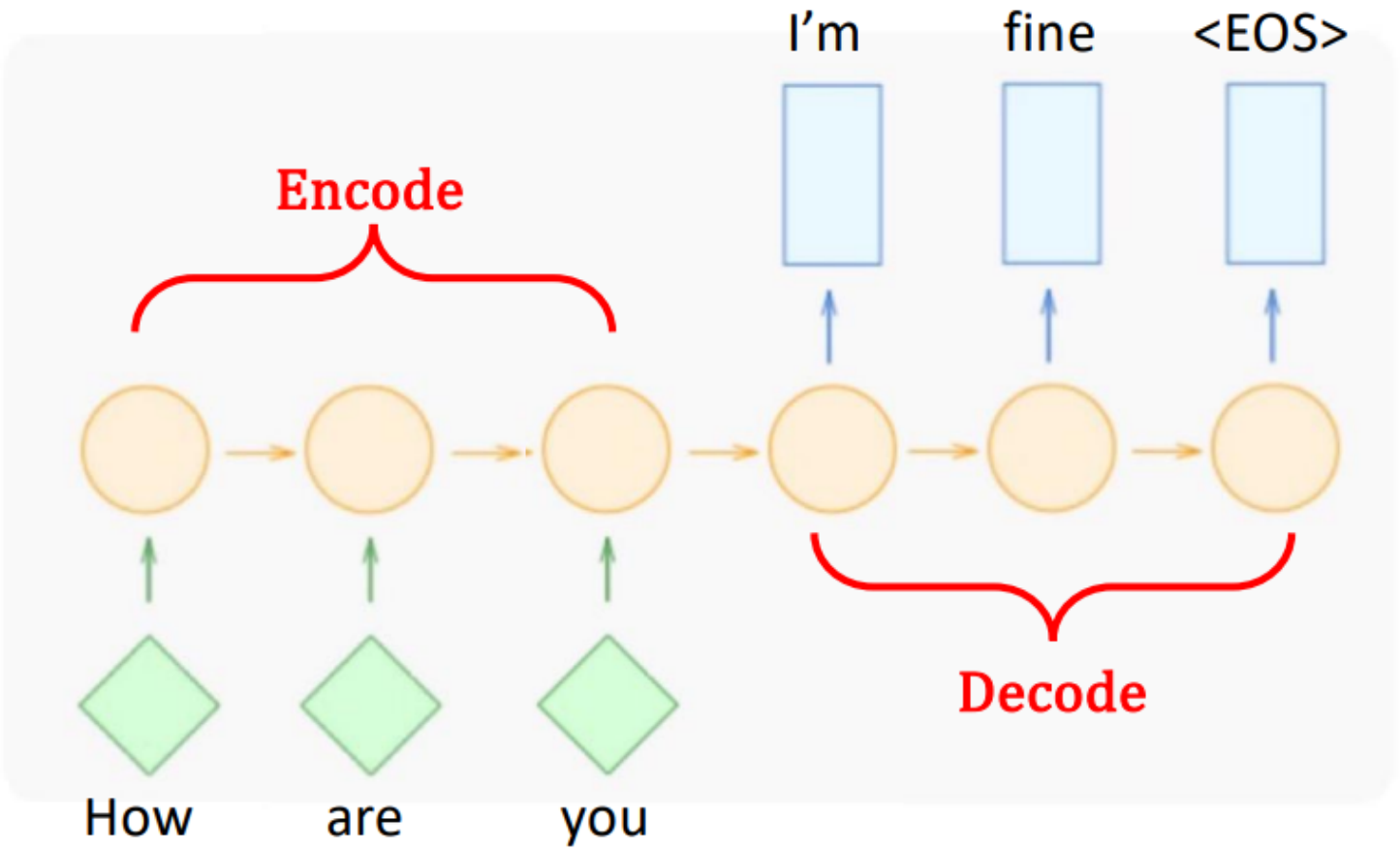


Figure 12: Chatbot with RNNs

PixelRNN – Pixel Recurrent Neural Network

Introduced by: van den Oord et al. (2016)

What is PixelRNN?

- PixelRNN is a generative model specifically designed for image generation.
- It models the joint distribution of all pixels in an image by predicting one pixel at a time.
- Pixels are generated in a raster-scan order (left to right, top to bottom).
- Uses an autoregressive approach, where each pixel is generated conditioned on all previously generated pixels.

PixelRNN – Pixel Recurrent Neural Network (cont.)

Core Idea: The model factorizes the image distribution as:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

where each pixel x_i is conditioned on all previous pixels $(x_1, x_2, \ldots, x_{i-1})$.

Each pixel's value (such as its RGB channels) is predicted based on all preceding pixels using a recurrent neural network, allowing the model to capture complex dependencies and structures within the image.

PixelRNN – Pixel Recurrent Neural Network (cont.)

Architecture of PixelRNN

1. **Input Representation:**
   - Each image is a 2D grid of pixels.
   - Each pixel can have multiple channels (e.g., RGB).
2. **Autoregressive Modeling:**
   - For each pixel, the model learns $p(x_{i,j} \mid x_{<i,j})$, where $x_{<i,j}$ are all pixels above and to the left of the current pixel (i, j).
3. **RNN Layers:**
   - Uses RNNs along the rows of the image:
     - **Row LSTM:** Processes one row at a time from left to right.
     - **Diagonal BiLSTM:** Processes diagonals in the image to improve context.
4. **Masked Convolutions:**
   - Prevent the model from "seeing the future" pixels.
   - Each convolution is masked to preserve the autoregressive property.

PixelRNN – Pixel Recurrent Neural Network (cont.)

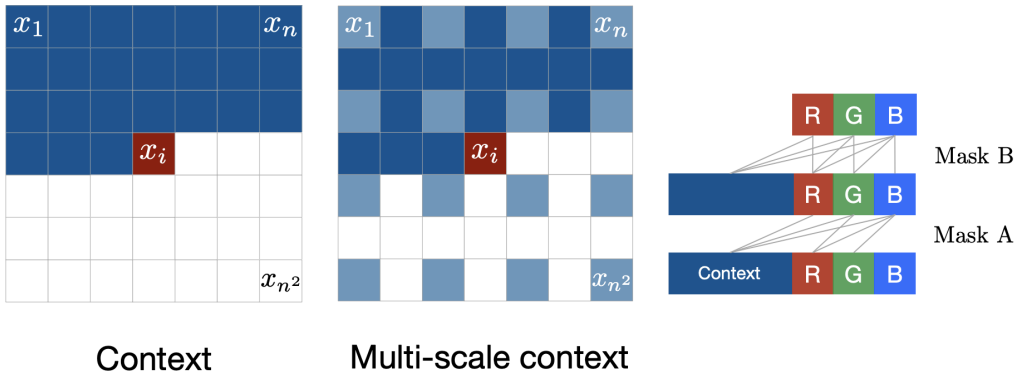


Figure 13: Left: To generate pixel xi one conditions on all the previously generated pixels left and above of xi. Center: To generate a pixel in the multi-scale case we can also condition on the subsampled image pixels (in light blue). Right: Diagram of the connectivity inside a masked convolution. In the first layer, each of the RGB channels is connected to previous channels and to the context, but is not connected to itself. In subsequent layers, the channels are also connected to themselves.

PixelRNN – Pixel Recurrent Neural Network (cont.)

Types of RNNs in PixelRNN

- **Row LSTM:**
  - Applies a 1D LSTM across each row.
  - Information flows left-to-right in each row.
  - Maintains the autoregressive structure.
- **Diagonal BiLSTM:**
  - Applies a 2D LSTM diagonally across the image.
  - Allows pixels to be conditioned on a broader context.

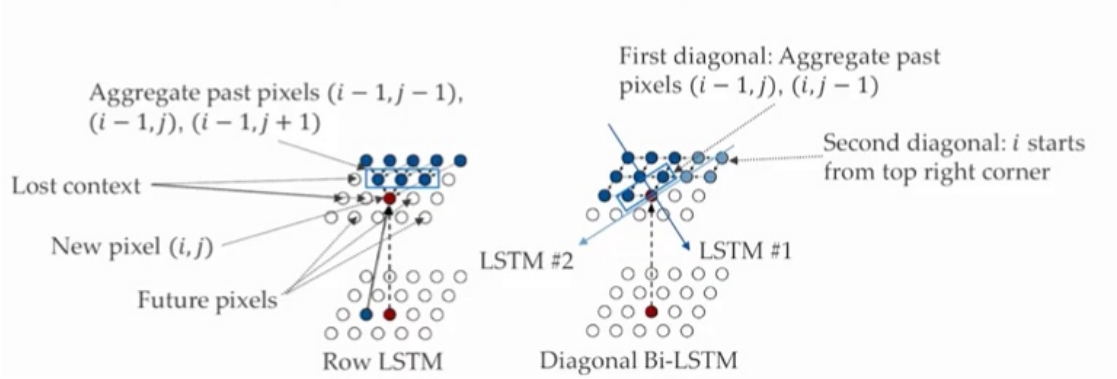


Figure 14: PixelRNN architecture with Row LSTM and Diagonal BiLSTM. The model processes pixels in a raster-scan order, ensuring that each pixel is conditioned on all previously generated pixels.

PixelRNN – Pixel Recurrent Neural Network (cont.)

Training and Inference:

- **Training:** The model is trained using maximum likelihood estimation (MLE) to maximize the joint probability of the pixels in the training images.
- **Inference:** During inference, pixels are generated sequentially, starting from a blank canvas and conditioning each new pixel on all previously generated pixels.
- **Sampling:** The model can sample new images by repeatedly predicting the next pixel based on the current image context.

PixelRNN – Pixel Recurrent Neural Network (cont.)

Figure 15: Image completions sampled from a PixelRNN.

PixelRNN – Pixel Recurrent Neural Network (cont.)

Pros:

- Capable of generating high-quality images with complex structures.
- Effective at capturing long-range dependencies in images.

Cons:

- Computationally intensive due to sequential processing of pixels.
- Slower inference compared to parallelizable models like PixelCNN.

Autoregressive Models: Modern Models

Modern autoregressive models have evolved significantly, leveraging advancements in neural network architectures and training techniques. Key developments include:

- **PixelCNN/PixelSNAIL:** Introduced convolutional layers to capture spatial dependencies in images, allowing for efficient pixel-wise generation.
- **WaveNet:** Applied dilated convolutions to model long-range dependencies in audio signals, achieving state-of-the-art results in speech synthesis.
- **Transformer-based Models:** Utilized self-attention mechanisms to capture global dependencies, enabling parallel processing and improved scalability.

These modern models have set new benchmarks in various domains, including image generation, speech synthesis, and text modeling.

Autoregressive Models: Modern Models (cont.)

PixelCNN/PixelSNAIL:

- Introduced by van den Oord et al. (2016) and later extended by Chen et al. (2017).
- Uses convolutional layers to model pixel dependencies in images.
- Each pixel is generated conditioned on previously generated pixels, allowing for efficient sampling.
- PixelCNN uses masked convolutions to ensure that each pixel is only influenced by previously generated pixels.
- PixelSNAIL extends this by incorporating self-attention mechanisms, allowing for better long-range dependencies.

Autoregressive Models: Modern Models (cont.)

WaveNet:

- Introduced by van den Oord et al. (2016) for audio generation.
- Utilizes dilated convolutions to capture long-range dependencies in audio signals.
- Each audio sample is generated conditioned on previous samples, allowing for high-quality audio synthesis.
- Achieved state-of-the-art results in speech synthesis and music generation tasks.

Autoregressive Models: Modern Models (cont.)

Transformer-based Models:

- Introduced by Vaswani et al. (2017) for natural language processing.
- Utilizes self-attention mechanisms to capture global dependencies in sequences.
- Allows for parallel processing, making it highly scalable for large datasets.
- Models like GPT-3 and BERT have set new benchmarks in various NLP tasks.
- Can be adapted for autoregressive generation tasks in images, audio, and other modalities.

Autoregressive Models: Modern Models (cont.)

Applications:

- Image generation (e.g., PixelCNN, PixelSNAIL).
- Audio synthesis (e.g., WaveNet).
- Text generation and language modeling (e.g., Transformer-based models).
- Video generation and other multimodal tasks.

Autoregressive Models: Modern Models (cont.)

Pros:

- High-quality generation across various modalities.
- Ability to capture complex dependencies in data.
- Scalability and efficiency in training and inference.

Cons:

- Computationally intensive, especially for large models.
- Requires large amounts of training data for optimal performance.
- Still faces challenges in handling very high-dimensional data efficiently.

Masked Spatial (2D) Convolution - PixelCNN

- Images can be flattened into 1D vectors, but they are fundamentally 2D.
- We can use a masked variant of ConvNet to exploit this knowledge.
- First, we impose an autoregressive ordering on 2D images:

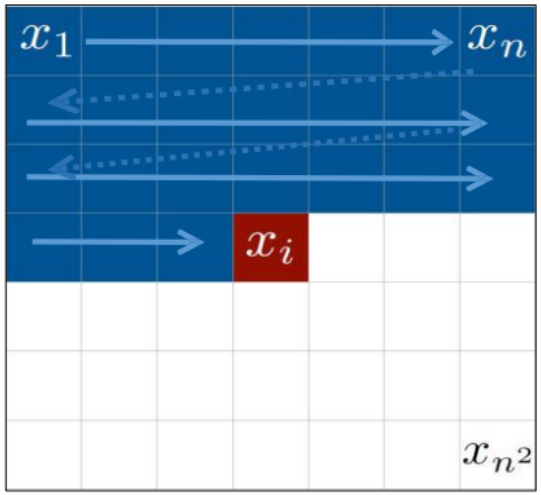


Figure 16: Raster scan ordering of a 2D image. The pixels are processed in a left-to-right, top-to-bottom manner, similar to reading text.

PixelCNN (cont.)

- The autoregressive ordering allows us to use a convolutional neural network (CNN) to predict the next pixel based on the previously generated pixels.
- We use masked convolutions to ensure that the model only has access to the pixels that have already been generated.
- The convolutional layers are designed to respect the autoregressive ordering, meaning that each pixel is predicted based on its left and top neighbours, but not the right or bottom neighbours.

PixelCNN (cont.)

PixelCNNs: Use the neighbour pixels to predict the new pixel.

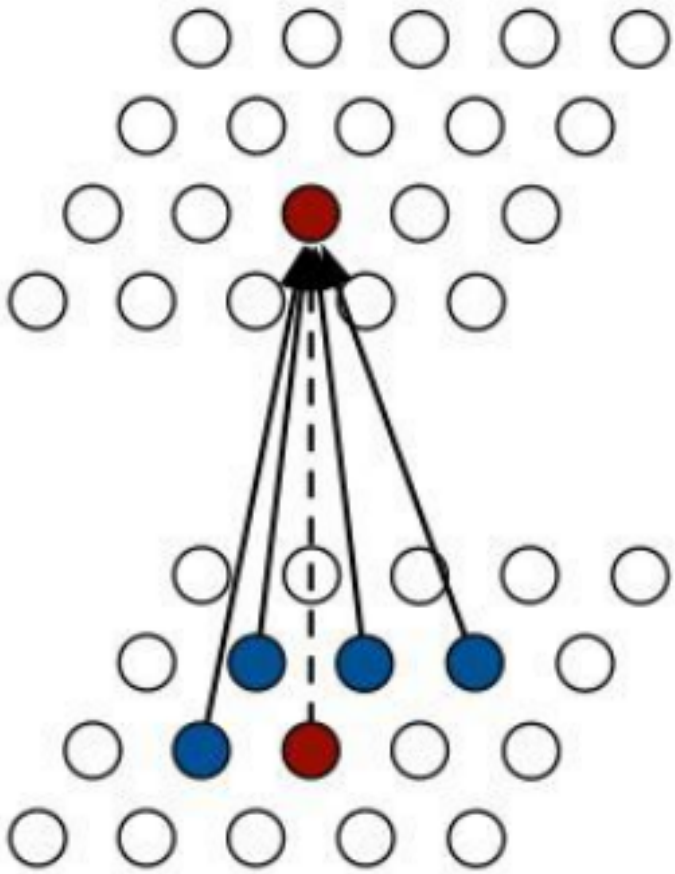


Figure 17: Image generation with pixelCNN

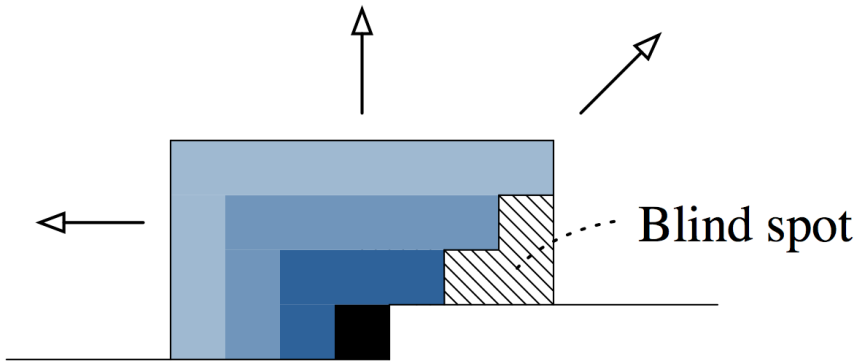


Figure 18: PixelCNN-style masking has one problem: blind spot in receptive field. The model cannot see the pixels to the right and below the current pixel, which can lead to artifacts in generated images.

PixelCNN (cont.)

- PixelCNN results

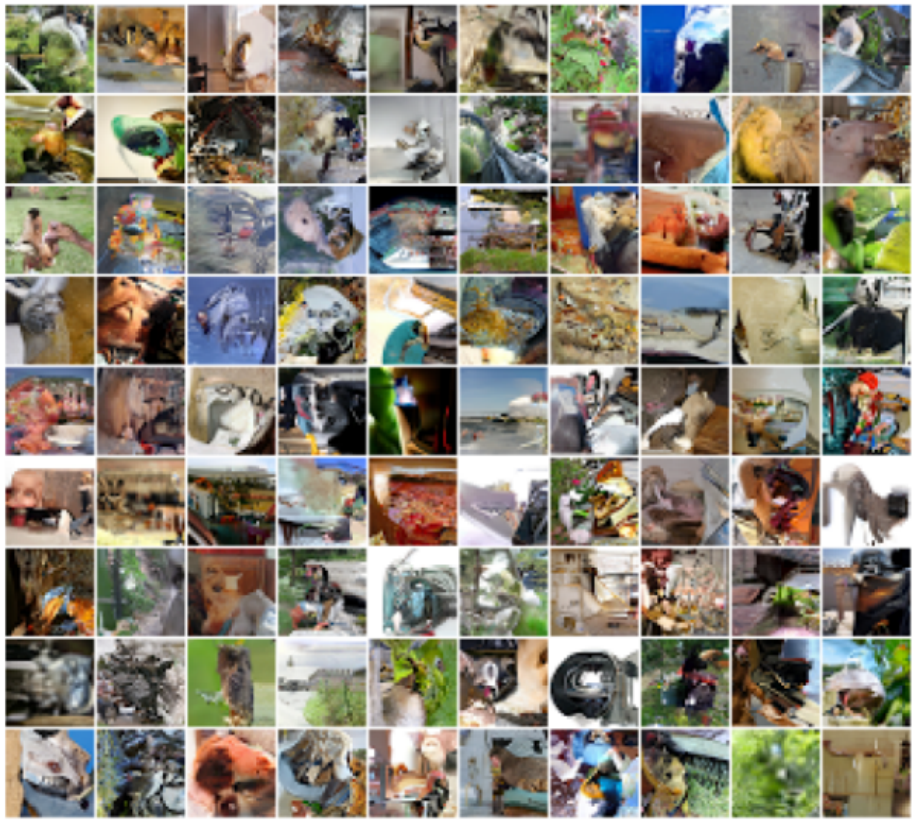


Figure 19: Image generation with pixelCNN. Model trained on Imagenet (32 x 32 pixels)

Autoregressive Models: Summary

- Define a specific order for the variables in the data (e.g., $x_1, x_2, \ldots, x_n$).
- Model the joint probability as a product of conditional probabilities:

$$\mathcal{P}(X = \bar{x}) = \prod_{i=1}^{n} \mathcal{P}(x_i \mid x_{<i})$$

- Sampling is performed recursively: generate x_1 first, then x_2 conditioned on x_1, and so on.
- Efficient to compute likelihoods and log-likelihoods for both discrete and continuous variables.
- Flexible: can be extended to multi-class, multi-dimensional, and structured data.

Autoregressive Models: Summary (cont.)

- Limitations:
  - No natural way to extract global features or representations.
  - Not inherently suited for clustering or unsupervised learning tasks.
  - The choice of variable ordering can significantly affect performance.

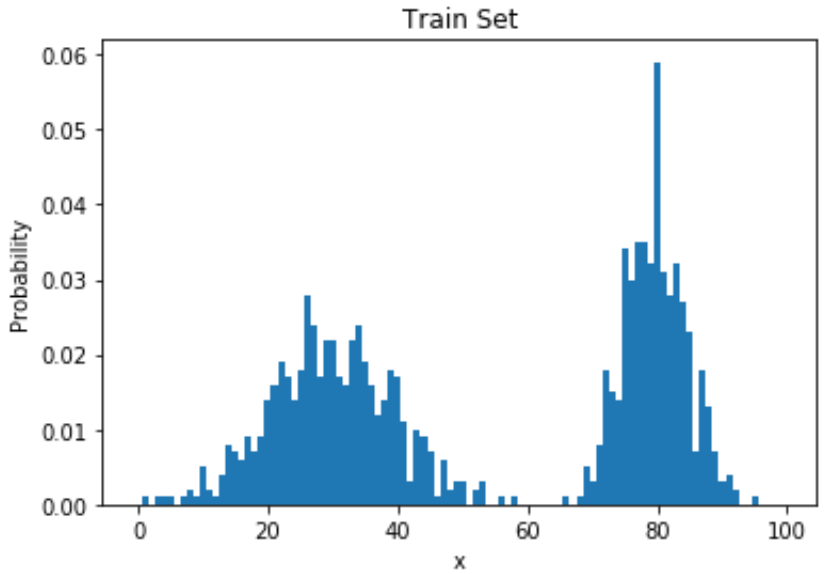
References

Reference Slides

- Fei-Fei Li "Generative Deep Learning" CS231
- Hao Dong "Deep Generative Models"

Learning: Estimate frequencies by counting

- Recall: the goal is to estimate p_{data} from samples $x^{(1)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$
- Suppose the samples take on values in a finite set $\{1, \ldots, k\}$
- The model: a histogram
  - (Redundantly) described by k nonnegative numbers: $p_1, \ldots, p_k$
- To train this model: count frequencies
- $p_i = \frac{\#\text{ times } i \text{ appears in the dataset}}{\#\text{ points in the dataset}}$



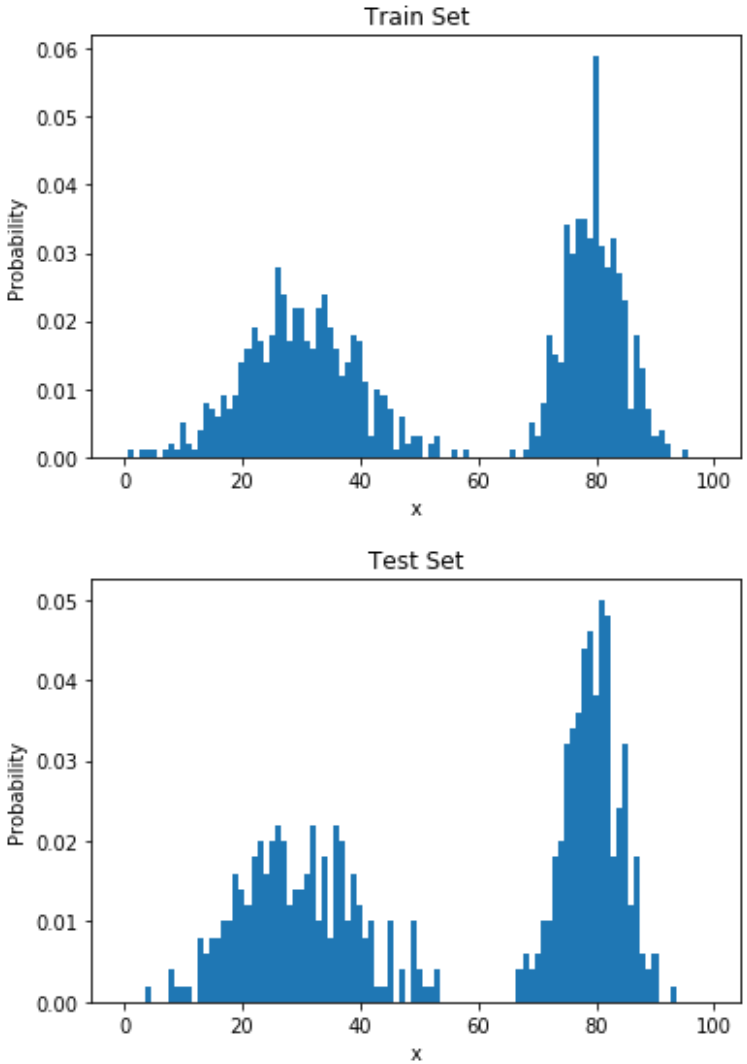
Inference and Sampling

- **Inference (querying p_i for arbitrary i):** simply a lookup into the array $p_1, \ldots, p_k$
- **Sampling (lookup into the inverse cumulative distribution function):**
  - From the model probabilities $p_1, \ldots, p_k$, compute the cumulative distribution:

$$F_i = p_1 + \cdots + p_i \quad \text{for all } i \in \{1, \ldots, k\}$$

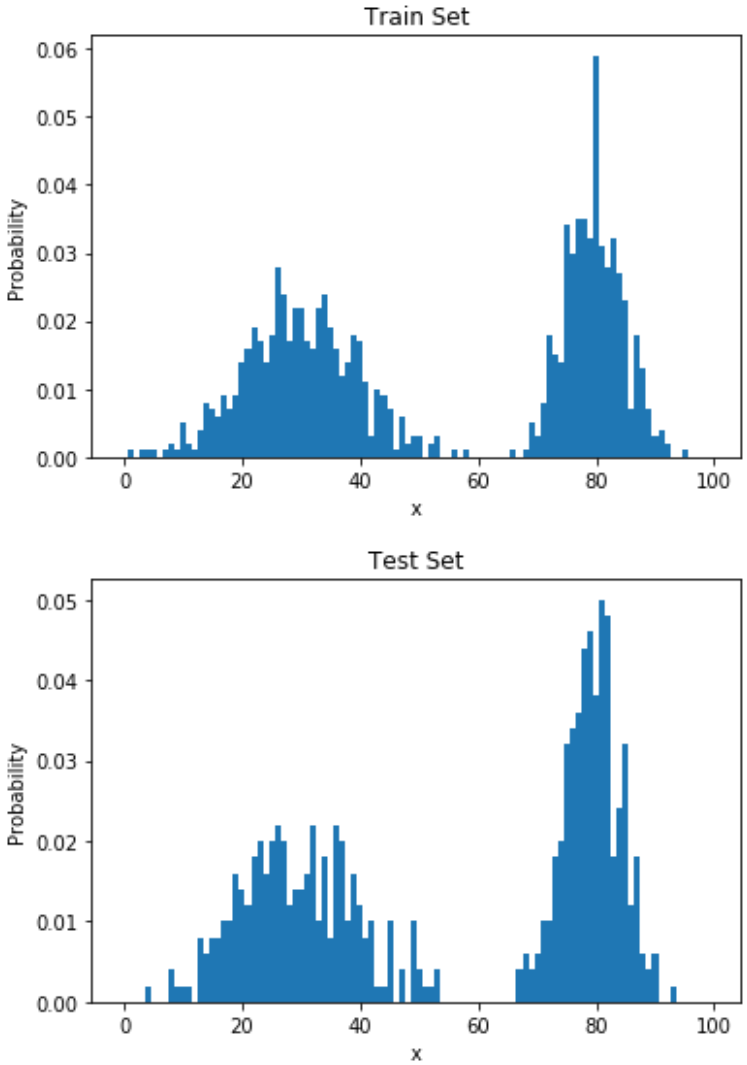
  - Draw a uniform random number $u \sim [0, 1]$
  - Return the smallest i such that $u \leq F_i$
- **Are we done?**

Problem: Lack of Generalization



learned histogram = training data distribution
$\rightarrow$ often poor generalization

Solution: Parameterized Distributions



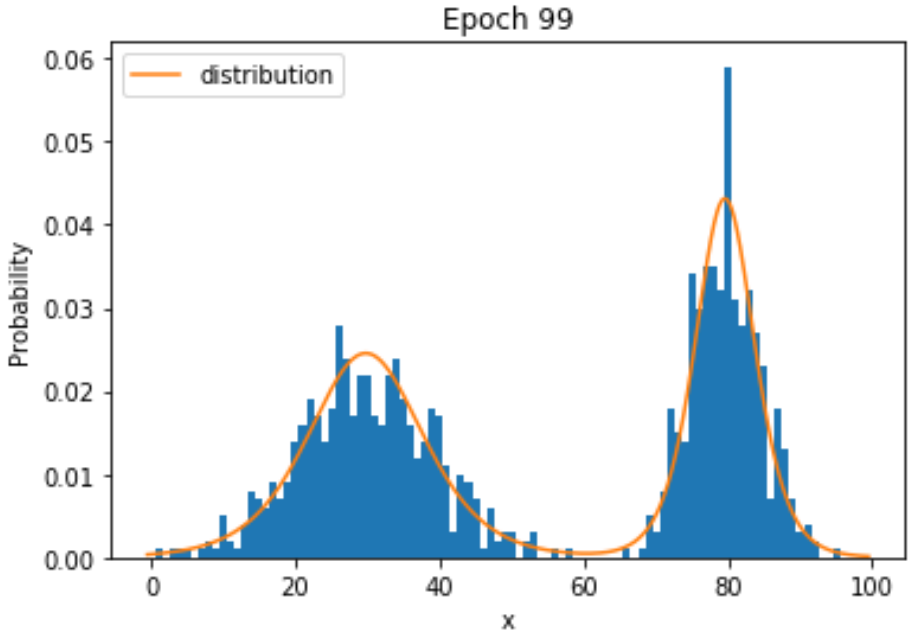


Figure 20: Fitting a parameterized distribution often generalizes better than a histogram.

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.