Deep Generative Models

Autoregressive Model

Hamid Beigy

Sharif University of Technology

April 8, 2024

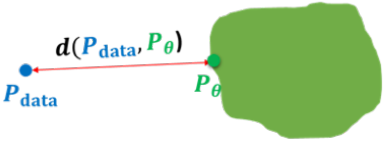
Table of contents

Introduction

Introduction (Statistical / Deep Generative Models)

1. **A Generative model** (GM) is a probability distribution $p(\mathbf{x})$.
   - A statistical GM is a trainable probabilistic model, $p_\theta(\mathbf{x})$.
   - A deep GM is a statistical generative model parametrized by a neural network.
2. A generative model needs
   - Data ($\mathbf{x}$): Complex, unstructured samples such as images, speech, molecules, text, etc.
   - Prior knowledge: parametric form (e.g., Gaussian, mixture, softmax), loss function (e.g., maximum likelihood, divergence), optimization algorithm, etc.

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

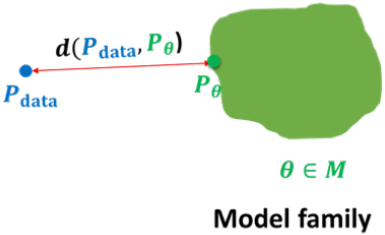


Model family

Credit: Aditya Grover

Introduction (Key Questions)

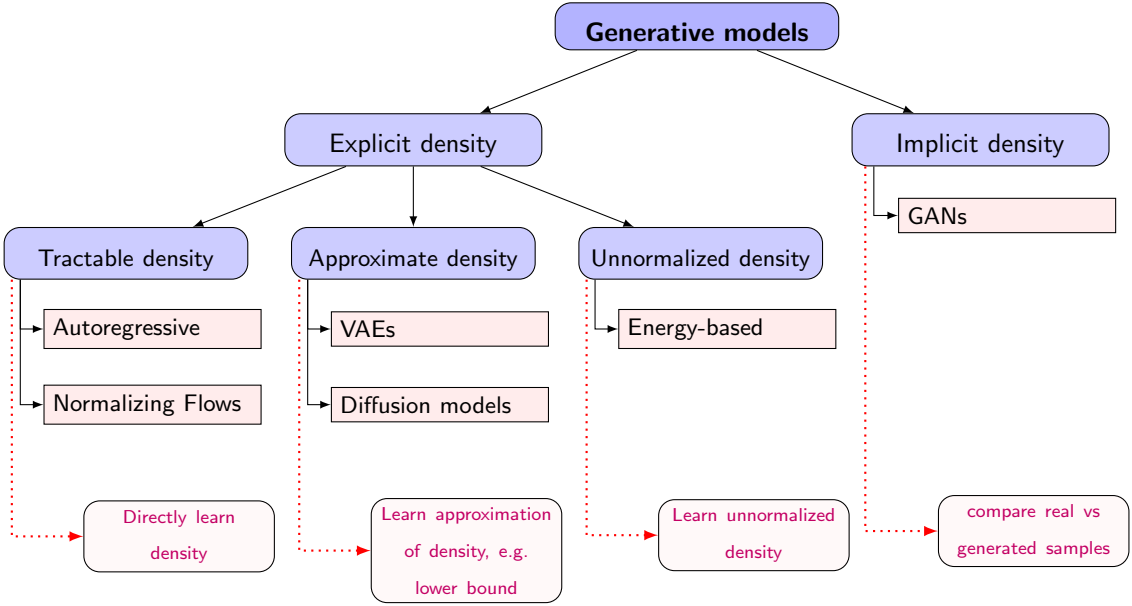
$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



1. **A Representation:** how do we parameterize the joint distribution of many random variables?
2. **A Learning:** what is the right way to compare probability distributions?
3. **A Inference:** how do we invert (or encode) the generation process?

Credit: Aditya Grover

Generative models categories



Autoregressive models

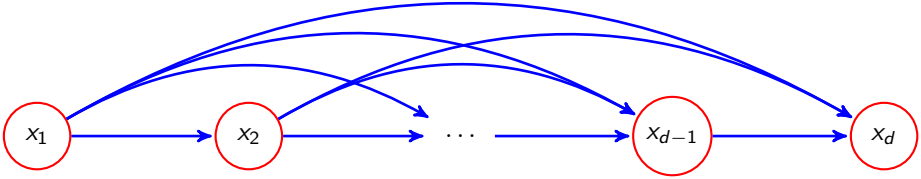
Autoregressive models

1. Suppose we have a dataset $S = \{x_1, x_2, \ldots, x_m\}$ of n-dimensional points x.
2. For simplicity, we assume points are binary, i.e., $x \in \{0,1\}^n$.
3. Using chain rule, we can factorize the joint distribution as

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

   where $\mathbf{x}_{<i} = [x_1, x_2, \ldots, x_{i-1}]$ denotes vector of random variables with index less than i.

4. The chain rule factorization can be expressed graphically as a Bayesian network.



Autoregressive models

1. The autoregressive constraint is a way to model sequential data.
2. The factorization contains n factors and some of these factors contain many parameters ($O(2^n)$ in total).
3. It is infeasible to learn such an exponential number of parameters.
4. AR models use (deep) neural network to parameterize these factors $p(x_i|x_{<i})$.
5. We assume the conditional distributions $p(x_i|x_{<i})$ correspond to Bernoulli random variables and learn a function that maps the proceeding random variables $x_1, x_2, \ldots, x_{i-1}$ to the mean of this distribution as

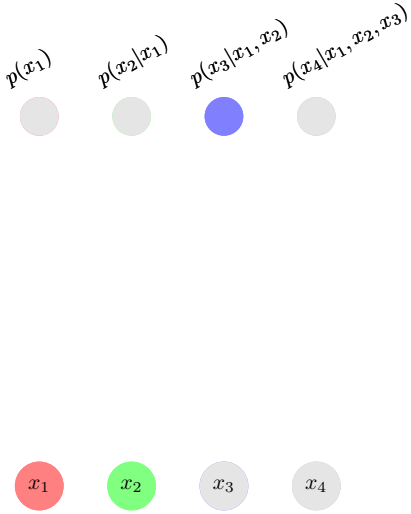
$$p_{\theta_i}(x_i|x_{<i}) = Bern(f_i(x_1, x_2, \ldots, x_{i-1}))$$

   where θ_i denotes the set of parameters used to specify the mean function $f_i : \{0,1\}^{i-1} \mapsto [0,1]$.
6. The number of parameters of an autoregressive generative model equals to $\sum_{i=1}^{n} |\theta_i|$.
7. Tractable exact likelihood computations.
8. No complex integral over latent variables in likelihood
9. Slow sequential sampling process.

Autoregressive models

1. The nth output should only be connected to the previous $n-1$ inputs.

2. For example, when computing $p(x_4|x_3, x_2, x_1)$ the only inputs that we should consider are x_1, x_2, x_3 because these are the only variables given to us while computing the conditional probability.



Autoregressive models

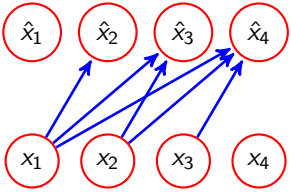
1. In the simplest case, we can specify the function as a linear combination of the input elements followed by a sigmoid non-linearity (to restrict the output to lie between 0 and 1).

2. This gives us the formulation of a fully-visible sigmoid belief network (FVSBN).

$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma\left(a_0^i + \sum_{j=1}^{i-1} a_j^i x_j\right)$$

   where σ is sigmoid function and $\theta_i = \{a_0^i, \ldots, a_{i-1}^i\}$.

3. At the output layer we want to predict n conditional probability distributions while at the input layer we are given the n input variables.

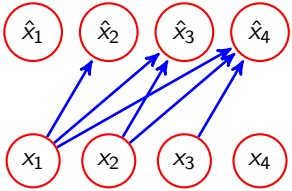


4. The conditional variables $x_i | x_1, \ldots, x_{i-1}$ are Bernoulli with parameters

$$\hat{x}_i = p(x_i = 1 | x_1, \ldots, x_{i-1}; \theta_i) = \sigma\left(a_0^i + \sum_{j=1}^{i-1} a_j^i x_j\right)$$

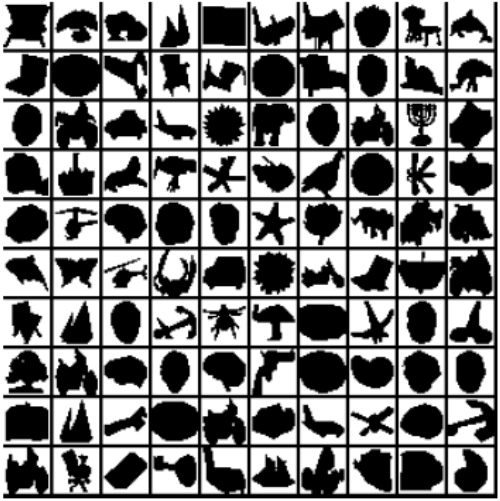
Autoregressive models

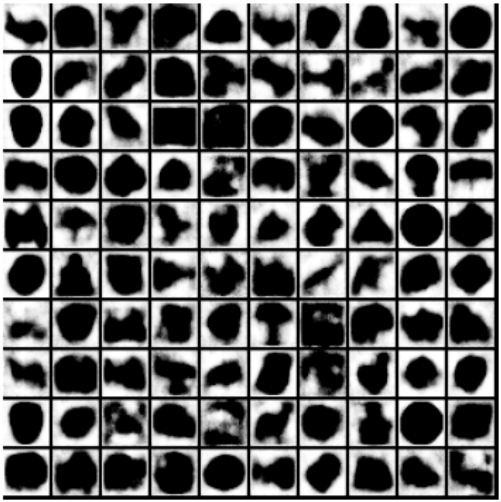
1. How to evaluate $p(x_1, \ldots, x_{900})$?
2. Multiply all the conditionals factors.
3. How to sample from $p(x_1, \ldots, x_{900})$?
   - Sample $\bar{x}_1 \sim p(x_1)$.
   - Sample $\bar{x}_2 \sim p(x_2|x_1 = \bar{x}_1)$.
   - Sample $\bar{x}_3 \sim p(x_3|x_1 = \bar{x}_1, x_2 = \bar{x}_2)$.
4. How many parameters? $1 + 2 + 3 + \ldots + n \approx \frac{n^2}{2}$



FVSBN results (Gan et al. 2015)

1. Left: Training (Caltech 101 Silhouettes) Right: Samples from the model





Neural Autoregressive Density Estimator

1. To increase the expressiveness of an autoregressive generative model, we can use more flexible parameterizations for the mean function such as MLP instead of logistic regression.

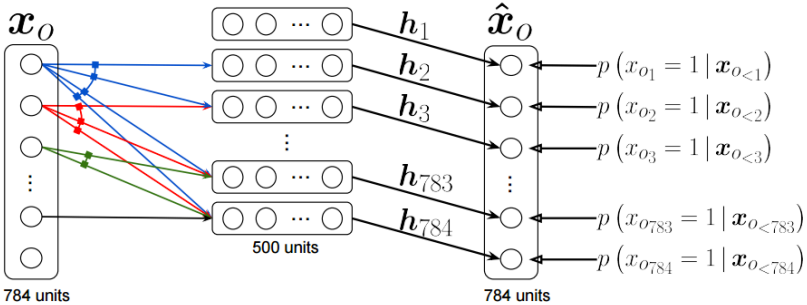
2. For example, consider the case of a neural network with one hidden layer.

3. The mean function for variable i can be expressed as

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$
$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma\left(\mathbf{a}^i \mathbf{h}_i + b_i\right)$$

   where $\mathbf{h}_i \in \mathbb{R}^d$ is hidden layer activations of MLP.

4. Hence, we have the following architecture



Neural Autoregressive Density Estimator

1. To improve model, use a neural network with one hidden layer instead of logistic regression.

$$\begin{aligned} \mathbf{h}_i &= \sigma(A_i \mathbf{x}_{<\mathbf{i}} + \mathbf{c}_i) \\ \hat{x}_i = p(x_i = 1 | x_1, \ldots, x_{i-1}; \boldsymbol{\theta}^i) &= \sigma(\boldsymbol{\alpha}^{(i)} \mathbf{h}_i + b_i) \\ \boldsymbol{\theta}^i &= \{A_i, \mathbf{c}_i, \boldsymbol{\alpha}^{(i)}, b_i\} \end{aligned}$$

2. $\mathbf{h}_i \in \mathbb{R}^d$ denotes the hidden layer activations for the MLP.

3. $\theta_i = \{A_i \in \mathbb{R}^{d \times (i-1)}, \mathbf{c}_i \in \mathbb{R}^d, \boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d, b_i \in \mathbb{R}\}$ are the set of parameters.

4. Hidden layer parameters are shared and only the relevant columns of $\mathbf{A}$ are used for each i.

5. The total number of parameters in this model is dominated by the matrices A_i and given by $O(nd + n)$.

Neural Autoregressive Density Estimator

1. The Neural Autoregressive Density Estimator (NADE) provides an alternate MLP-based parameterization that is more statistically and computationally efficient than the given approach (Larochelle and Murray 2011).

2. In NADE, parameters are shared across the functions used for evaluating the conditionals.

3. The hidden layer activations are specified as

$$\mathbf{h}_i = \sigma(W_{.,<i}\mathbf{x}_{<\mathbf{i}} + \mathbf{c})$$
$$\hat{x}_i = p(x_i = 1|x_1, \ldots, x_{i-1}; \boldsymbol{\theta}^i) = \sigma(\boldsymbol{\alpha}^{(i)}\mathbf{h}_i + b_i)$$

4. $\theta = \{W \in \mathbb{R}^{d\times n}, \mathbf{c} \in \mathbb{R}^d, \{\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d\}_{i=1}^n, \{b_i \in \mathbb{R}\}_{i=1}^n\}$ is the full set of parameters.

5. The weight matrix W and the bias vector c are shared across the conditionals.

Neural Autoregressive Density Estimator

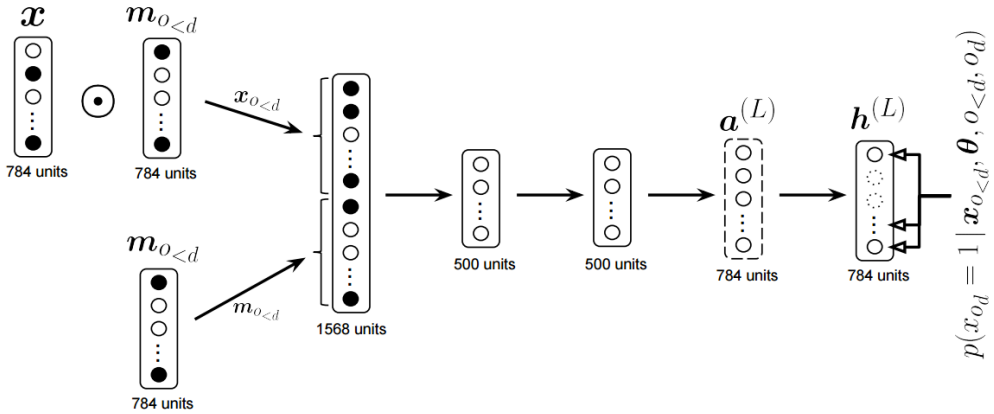
1. Sharing parameters has two benefits:
   - The total number of parameters gets reduced from $O(n^2 d)$ to $O(nd)$.
   - Hidden unit activations can be evaluated in $O(nd)$ time via

     $$\mathbf{h}_i = \sigma(\mathbf{a}_i)$$
     $$\mathbf{a}_{i+1} = \mathbf{a}_i + W[., i]x_i$$

     with the base case given by $\mathbf{a}_1 = \mathbf{c}$.
2. Training of NADE is done by minimizing $-\frac{1}{T}\sum_{i=1}^{T} \log p(x_i)$
3. Samples from NADE trained on a binary version of MNIST.

Deep NADE

1. The input to the network (DeepNADE) is the concatenation of the masked data and the mask itself (Uria, Côté, et al. 2016).
2. This allows the network to identify cases when input data is truly zero from cases when input data is zero because of the mask.
3. NADE also explored other autoencoder architectures such as convolutional neural networks
4. DeepNade with two hidden layers



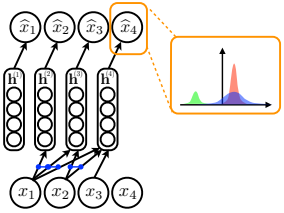
Real-Valued NADE

1. The RNADE algorithm extends NADE to learn generative models over real-valued data (Uria, Murray, and Larochelle 2013).

2. The conditionals are modeled via a continuous distribution such as mixture of K Gaussian.

$$p(x_i|x_{<i}) = \sum_{j=1}^{K} \pi_{ij}\mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$$

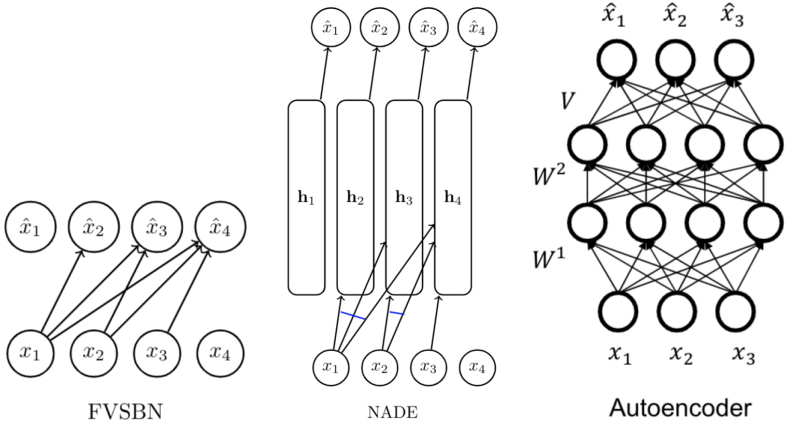
- Output of the network are parameters of a mixture model for $p(x_k|x_{<k})$
- Means are $\mu_{i,k} = b_{i,k}^{\mu_i} + \boldsymbol{\alpha}_{i,k}^{\mu_i} h_i$
- Standard deviations are $\sigma_{i,k} = \exp\left(b_{i,k}^{\sigma_i} + \boldsymbol{\alpha}_{i,k}^{\sigma_i} h_i\right)$
- Mixing weights are $\pi_{i,k} = softmax\left(b_{i,k}^{\pi_i} + \boldsymbol{\alpha}_{i,k}^{\pi_i} h_i\right)$



3. Please study DocNADE.

Autoregressive models vs. Autoencoders

1. Considering the following models.



2. FVSBN and NADE look similar to an autoencoder.

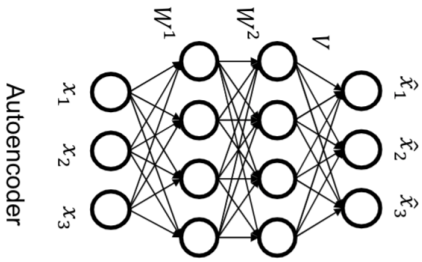
3. An encoder computing hidden.

4. A decoder computing densities.

5. A loss function, which is likelihood.

Autoregressive Autoencoders

1. An autoencoder is not a generative model: it does not define a distribution over x for sampling new data points.



2. Can we get a generative model from an autoencoder?

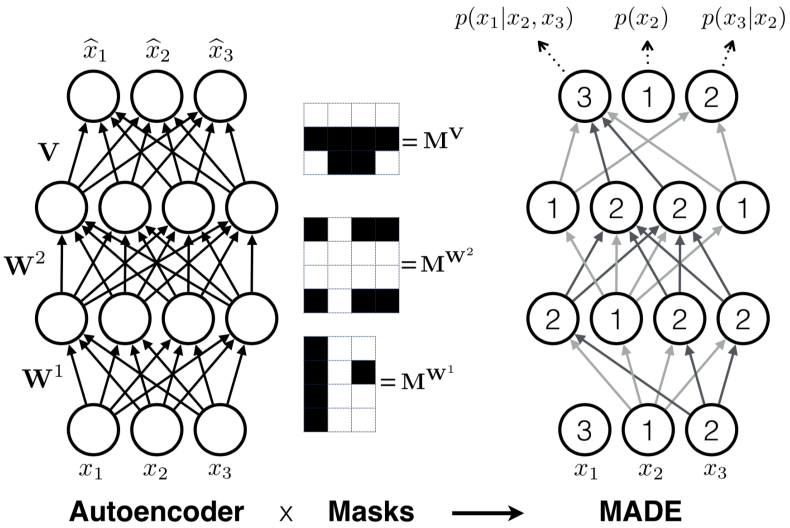
3. We need to make sure it corresponds to a valid Bayesian Network, i.e., we need an ordering. If the ordering is 1, 2, 3, then

   - $\hat{x}_1$ cannot depend on any input x.
   - $\hat{x}_2$ can only depend on x_1.

4. We can use a single neural network to produce all the parameters.

Masked Autoencoder for Distribution Estimation (MADE)

1. MADE is an autoencoder that preserves autoregressive property (Germain et al. 2015).



Masked Autoencoder for Distribution Estimation (MADE)

1. MADE is a specially designed architecture to enforce the autoregressive property in the autoencoder efficiently.

2. MADE removes the contribution of certain hidden units by using mask matrices so that each input dimension is reconstructed only from previous dimensions in a given ordering in a single pass.

3. In a multilayer fully-connected neural network, say, we have L hidden layers with weight matrices $\mathbf{W}^1, \ldots, \mathbf{W}^L$ and an output layer with weight matrix V. The output $\hat{x}$ has dimensions $\hat{x}_i = p(x_i | x_{1:i-1})$

4. Without any mask, we have

$$
\begin{aligned}
\mathbf{h}^0 &= \mathbf{x} \\
\mathbf{h}^l &= \text{activation}^l(\mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}^l) \\
\hat{\mathbf{x}} &= \sigma(\mathbf{V}\mathbf{h}^L + \mathbf{c})
\end{aligned}
$$

Masked Autoencoder for Distribution Estimation (MADE)

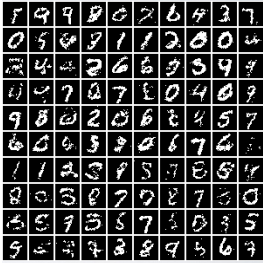
1. Without any mask, we have

$$\mathbf{h}^0 = \mathbf{x}$$
$$\mathbf{h}^l = \text{activation}^l(\mathbf{W}^l\mathbf{h}^{l-1} + \mathbf{b}^l)$$
$$\hat{\mathbf{x}} = \sigma(\mathbf{V}\mathbf{h}^L + \mathbf{c})$$

2. To zero out some connections between layers, we can simply element-wise multiply every weight matrix by a binary mask matrix.

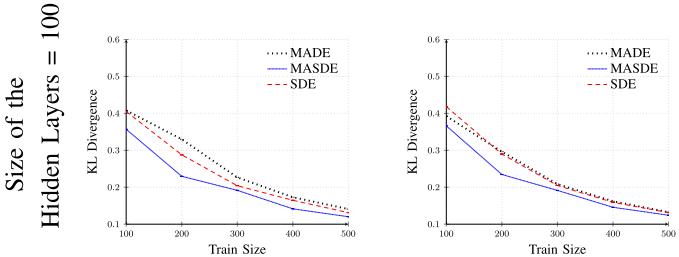
$$\mathbf{h}^l = \text{activation}^l((\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{h}^{l-1} + \mathbf{b}^l)$$
$$\hat{\mathbf{x}} = \sigma((\mathbf{V} \odot \mathbf{M}^{\mathbf{V}})\mathbf{h}^L + \mathbf{c})$$

3. Mask matrix is constructed by a labeling process.



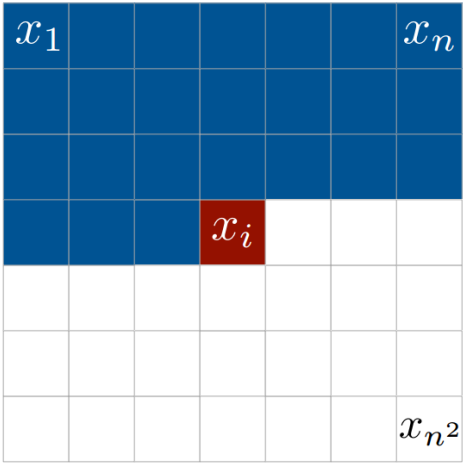
Masked Autoencoder for Structured Distribution Estimation (MASDE)

1. This method is used when the structure (Markov random field) of the data is known (Khajenezhad, Madani, and Beigy 2021).
2. In structured distributions, the graph structure of the variables declares their conditional dependencies.
3. Therefore, having a graph structure, each of the chain rule conditional terms might be presentable by a conditional probability on a smaller set of variables.
4. For each i, we assume there is a subset $B_i \subseteq \{1, \ldots, i-1\}$ such that $p(x_i|x_{<i}) = p(x_i|x_{B_i})$.
5. Use an auoencoder that has the above autoregressive property and mask matrix is constructed by a labeling process.
6. MASDE needs a smaller training set in comparison with its counterparts.



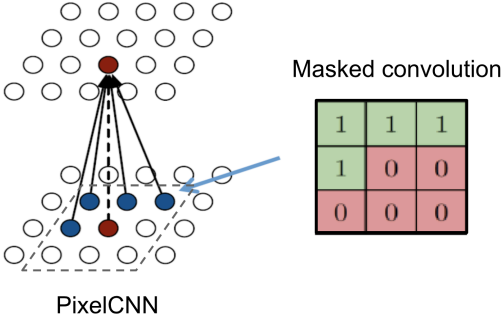
PixelRNN

1. PixelRNN is a deep generative model for images (Oord, Kalchbrenner, and Kavukcuoglu 2016).
2. Dependency on previous pixels modeled using an RNN (LSTM).



PixelCNN

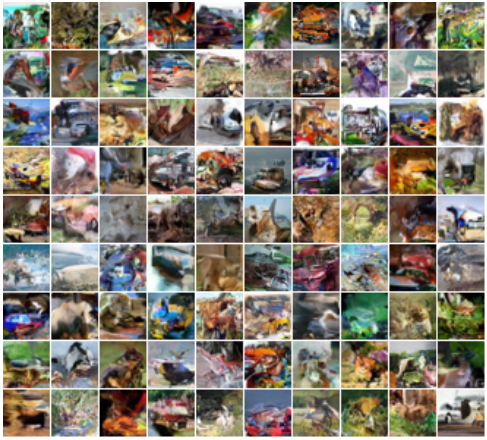
1. The main drawback of PixelRNN is that training is very slow.
2. PixelCNN uses standard convolutional layers to capture a bounded receptive field and compute features for all pixel positions at once (Oord, Kalchbrenner, Espeholt, et al. 2016).
3. In PixelCNN, pooling layers are not used.
4. Masks are adopted in the convolutions to restrict the model from violating the conditional dependence.

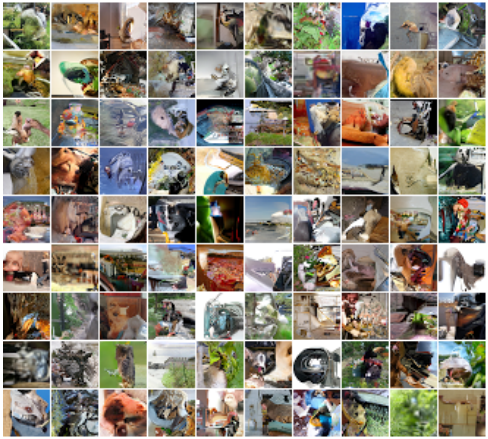


5. Please also PixelCNN++ (Salimans et al. 2017).

PixelCNN

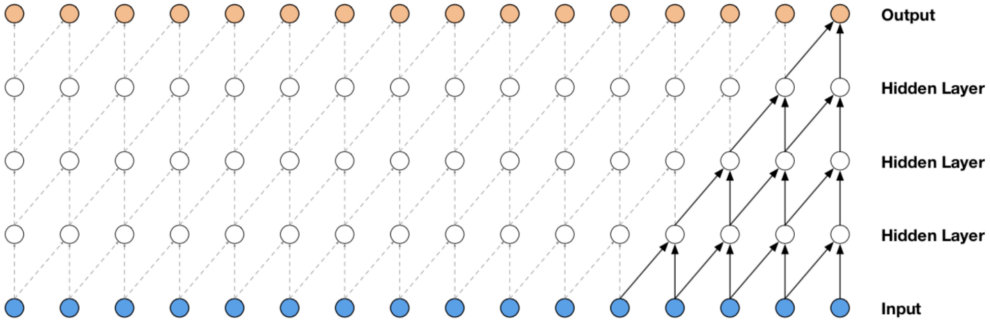
1. The training set (CIFAR-10 (left)) and the samples generated by the PixelCNN (right).





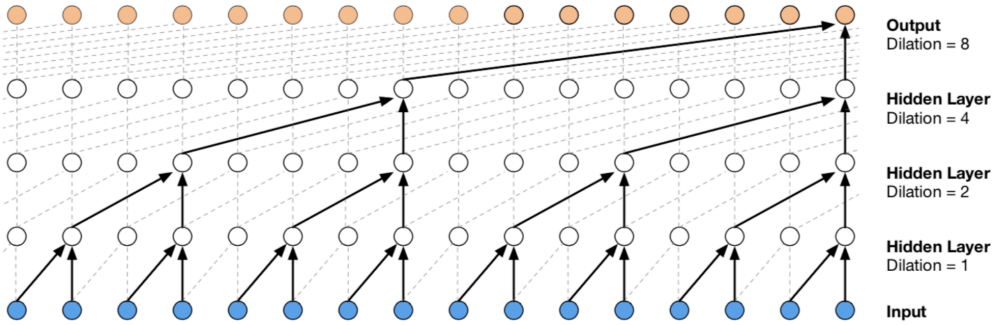
WaveNet

1. WaveNet is very similar to PixelCNN but applied to 1-D audio signals (Oord, Dieleman, et al. 2016).
2. WaveNet consists of a stack of causal convolution which is a convolution operation designed to respect the ordering.
3. Causal convolutions used for temporal data which ensures the model cannot violate the ordering in which we model the data: the prediction $p(x_{t+1}|x_1, \ldots, x_t)$.
4. The causal convolution in WaveNet is simply to shift the output by a number of timestamps to the future so that the output is aligned with the last input element.



WaveNet

1. One big drawback of convolution layer is a very limited size of receptive field.
2. WaveNet therefore adopts dilated convolution, where the kernel is applied to an evenly-distributed subset of samples in a much larger receptive field of the input.

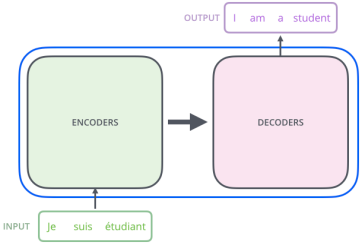


Autoregressive Transformers

Transformers model

1. The attention make it possible to do sequence to sequence modeling without recurrent network units (Vaswani et al. 2017).

2. The transformer model is entirely built on the self-attention mechanisms without using sequence-aligned recurrent architecture.



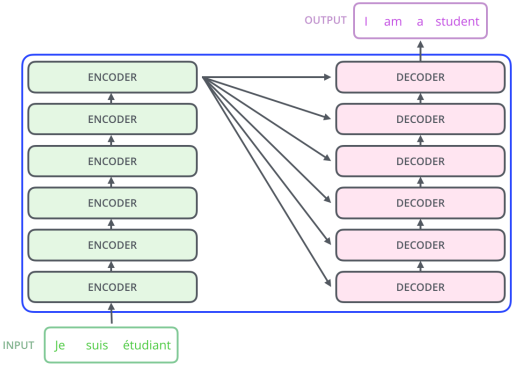


Figure: Jay Alammar

3. The encoding component is a stack of six encoders.

4. The decoding component is a stack of decoders of the same number.

Transformers training

1. The Transformers works slightly differently during training and inference.
2. Input sequence: You are welcome in English.
3. Target sequence: De nada in Spanish

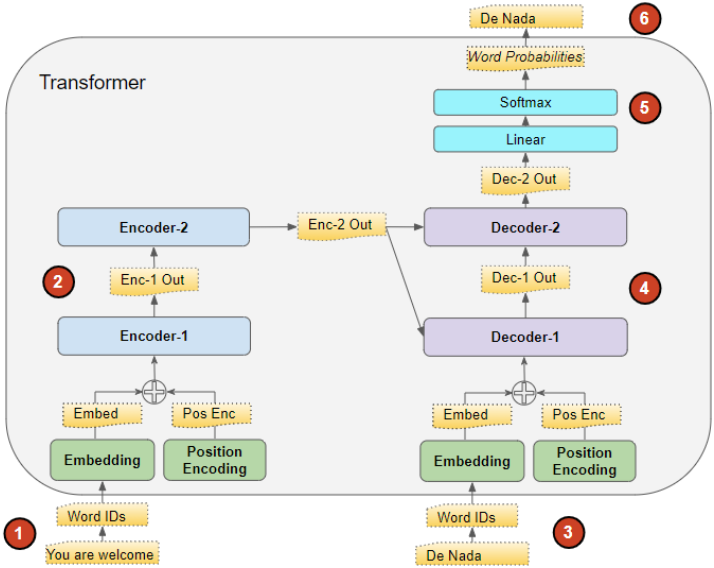


Figure:Ketan Doshi

Transformers inference

1. During Inference, we have only the input sequence and don't have the target sequence to pass as input to the Decoder.
2. The goal is to produce the target sequence from the input sequence alone.

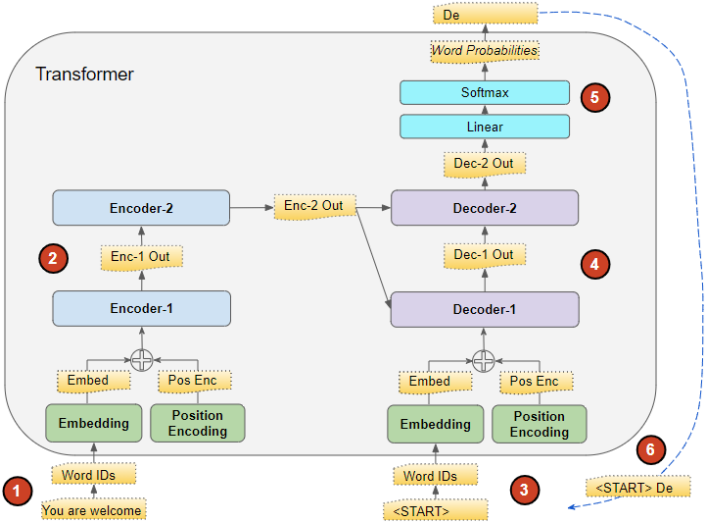
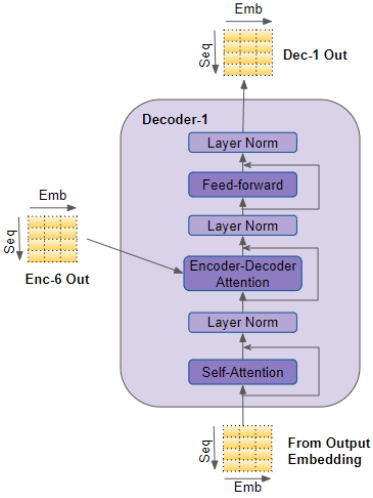


Figure:Ketan Doshi

Transformers decoder

1. The Decoder passes its input into a Multi-head Self-attention layer.
2. This operates in a slightly different way than the one in the Encoder.
3. It is only allowed to attend to earlier positions in the sequence. This is done by masking future positions.



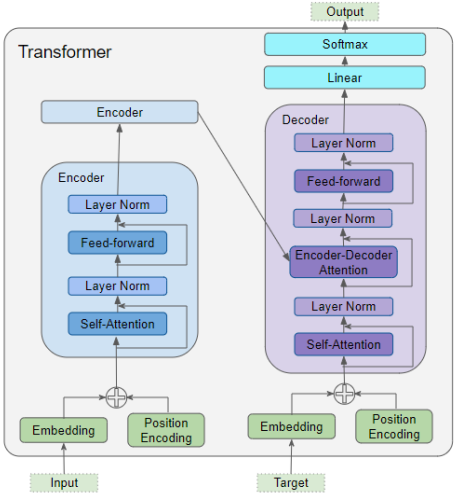


Figure: Ketan Doshi

Transformers decoder attention layers

1. The attention layers of Transformers decoder are

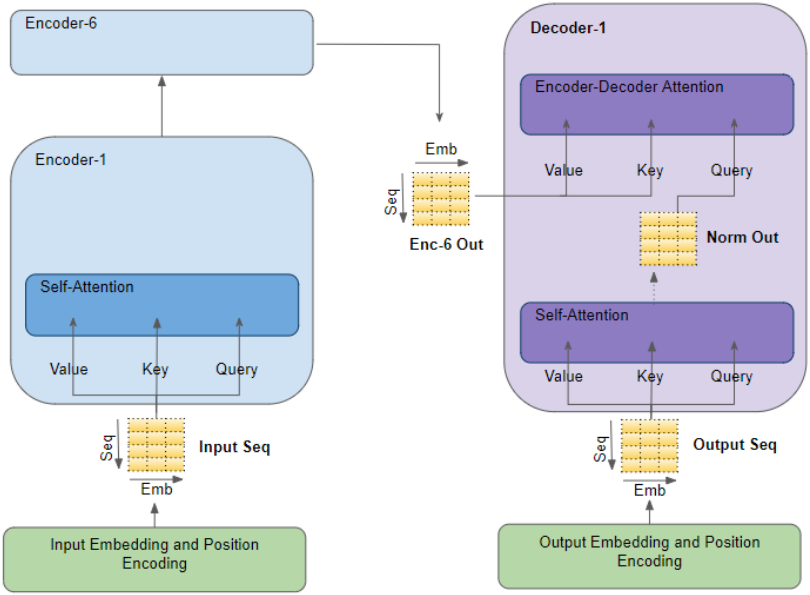
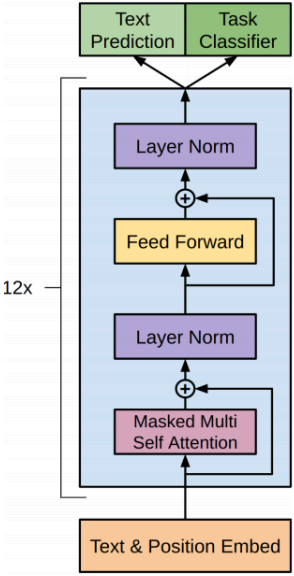


Figure: Ketan Doshi

GPT

GPT uses only the Transformers decoder blocks (Radford et al. 2018):



Transformers decoder self-attention and masking

1. The Decoder Self-Attention works just like the Encoder Self-Attention, except that it operates on each word of the target sequence.

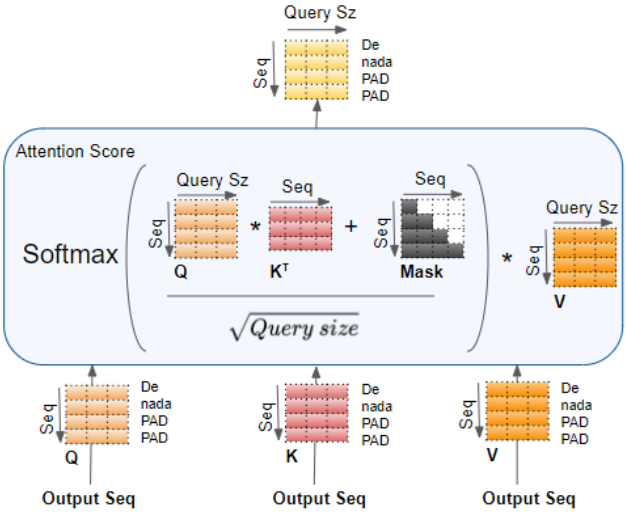


Figure: Ketan Doshi

Transformers decoder encoder-decoder attention and masking

1. The Encoder-Decoder Attention takes its input from two sources.
2. The Encoder-Decoder Attention computes the interaction between each target word with each input word.
3. The Masking masks out the Padding words in the target sequence.

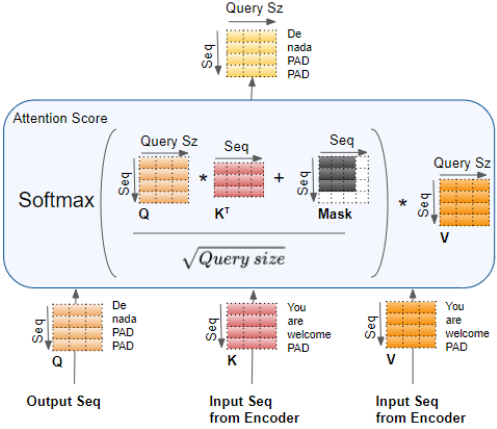


Figure: Ketan Doshi

Transformers decoder

1. At the time step n, we have input $x_1, \ldots, x_n$ tokens to the decoder.

2. The output attention tensor Y_n from the masked self-attention head is computed as follows.

$$Y_n = \text{softmax}\left(\text{Mask}\left(\frac{Q_n K_n^\top}{\sqrt{d_k}}\right)\right) V_n$$

3. In the time step $n+1$, we have next token x_{n+1} and

$$Q_{n+1} = x_{n+1}\mathbf{W}_Q = \begin{bmatrix} Q_n \\ q_{n+1} \end{bmatrix}$$

$$K_{n+1} = x_{n+1}\mathbf{W}_K = \begin{bmatrix} K_n \\ k_{n+1} \end{bmatrix}$$

$$V_{n+1} = x_{n+1}\mathbf{W}^V = \begin{bmatrix} V_n \\ v_{n+1} \end{bmatrix}$$

where $q_{n+1}, k_{n+1}, v_{n+1}$ are new attention tokens.

Transformers decoder

$$
\begin{aligned}
Y_{n+1} &= \text{softmax}\left(\text{Mask}\left(\frac{Q_{n+1}K_{n+1}^\top}{\sqrt{d_k}}\right)\right)V_{n+1} \\
&= \text{softmax}\left(\text{Mask}\left(\frac{1}{\sqrt{d_k}}\begin{bmatrix} Q_n \\ q_{n+1}\end{bmatrix}\begin{bmatrix} K_n \\ k_{n+1}\end{bmatrix}^\top\right)\right)\begin{bmatrix} V_n \\ v_{n+1}\end{bmatrix} \\
&= \text{softmax}\left(\text{Mask}\left(\frac{1}{\sqrt{d_k}}\begin{bmatrix} Q_n \\ q_{n+1}\end{bmatrix}\left[\begin{array}{c|c} K_n^\top & k_{n+1}^\top \end{array}\right]\right)\right)\begin{bmatrix} V_n \\ v_{n+1}\end{bmatrix} \\
&= \text{softmax}\left(\text{Mask}\left(\frac{1}{\sqrt{d_k}}\left[\begin{array}{c|c} Q_nK_n^\top & Q_nk_{n+1}^\top \\ \hline q_{n+1}K_n^\top & q_{n+1}k_{n+1}^\top \end{array}\right]\right)\right)\begin{bmatrix} V_n \\ v_{n+1}\end{bmatrix} \\
&= \text{softmax}\left(\left[\begin{array}{c|c} \text{Mask}\left(\frac{1}{\sqrt{d_k}}Q_nK_n^\top\right) & -\infty \\ \hline \frac{1}{\sqrt{d_k}}q_{n+1}K_n^\top & \frac{1}{\sqrt{d_k}}q_{n+1}k_{n+1}^\top \end{array}\right]\right)\begin{bmatrix} V_n \\ v_{n+1}\end{bmatrix} \\
&= \left[\begin{array}{c} \left[\begin{array}{c|c} \text{softmax}\left(\text{Mask}\left(\frac{1}{\sqrt{d_k}}Q_nK_n^\top\right)\right) & 0 \end{array}\right] \\ \text{softmax}\left(\frac{1}{\sqrt{d_k}}q_{n+1}\left[\begin{array}{c|c} K_n^\top & k_{n+1}^\top \end{array}\right]\right) \end{array}\right]\begin{bmatrix} V_n \\ v_{n+1}\end{bmatrix} \\
&= \begin{bmatrix} \text{softmax}\left(\text{Mask}\left(\frac{1}{\sqrt{d_k}}Q_nK_n^\top\right)\right)V_n \\ \text{softmax}\left(\frac{1}{\sqrt{d_k}}q_{n+1}K_{n+1}^\top\right)V_{n+1} \end{bmatrix} = \begin{bmatrix} Y_n \\ y_{n+1}\end{bmatrix}
\end{aligned}
$$

Transformers decoder

Hence, the new attention tensor y_{n+1} can be computed using

$$
\begin{aligned}
y_{n+1} &= \text{softmax}\left(\frac{1}{\sqrt{d_k}} q_{n+1} K_{n+1}^\top\right) V_{n+1} \\
&= \text{softmax}\left(\frac{1}{\sqrt{d_k}} q_{n+1} \left[\begin{array}{c|c} K_n^\top & k_{n+1}^\top \end{array}\right]\right) \begin{bmatrix} V_n \\ v_{n+1} \end{bmatrix} \\
&= \text{softmax}\left(\frac{1}{\sqrt{d_k}} x_{n+1} W^Q \left[\begin{array}{c|c} K_n^\top & \mathbf{W}_K^\top x_{n+1}^\top \end{array}\right]\right) \begin{bmatrix} V_n \\ x_{n+1} W_V \end{bmatrix}
\end{aligned}
$$

Computing the new attention tensor y_{n+1} for the new attention token x_{n+1} is a operation $O(n)$.

References

Reading

1. Chapter 22 of Probabilistic Machine Learning: Advanced Topics (Murphy 2023).
2. Chapter 2 of Deep Generative Modeling (Tomczak 2022).

References i

- Gan, Zhe et al. (2015). "Learning Deep Sigmoid Belief Networks with Data Augmentation". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS.*
- Germain, Mathieu et al. (2015). "MADE: Masked Autoencoder for Distribution Estimation". In: *Proceedings of the 32nd International Conference on Machine Learning.*
- Khajenezhad, Ahmad, Hatef Madani, and Hamid Beigy (2021). "Masked Autoencoder for Distribution Estimation on Small Structured Data Sets". In: *IEEE Transactions on Neural Networks and Learning Systems.*
- Larochelle, Hugo and Iain Murray (2011). "The Neural Autoregressive Distribution Estimator". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS.*
- Murphy, Kevin P. (2023). *Probabilistic Machine Learning: Advanced Topics*. The MIT Press.
- Oord, Aäron van den, Sander Dieleman, et al. (2016). "WaveNet: A Generative Model for Raw Audio". In: *The 9th ISCA Speech Synthesis Workshop.*
- Oord, Aäron van den, Nal Kalchbrenner, Lasse Espeholt, et al. (2016). "Conditional Image Generation with PixelCNN Decoders". In: *Advances in Neural Information Processing Systems.*

References ii

- Oord, Aäron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (2016). "Pixel Recurrent Neural Networks". In: *Proceedings of the 33nd International Conference on Machine Learning*.
- Radford, Alec et al. (2018). "Improving Language Understanding by Generative Pre-Training". In.
- Salimans, Tim et al. (2017). "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications". In: *International Conference on Learning Representations, ICLR*.
- Tomczak, Jakub M. (2022). *Deep Generative Modeling*. Springer.
- Uria, Benigno, Marc-Alexandre Côté, et al. (2016). "Neural Autoregressive Distribution Estimation". In: *Journal of Machine Learning Research* 17.205, pp. 1–37.
- Uria, Benigno, Iain Murray, and Hugo Larochelle (2013). "RNADE: The real-valued neural autoregressive density-estimator". In: *Advances in Neural Information Processing Systems*, pp. 2175–2183.
- Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Questions?