Generative Autoregressive Models

Danilo Comminiello

Generative Deep Learning 2021/2022

June 7, 2022

Sapienza
Università di Roma

Lecture content highlights

- We will focus on autoregressive models to estimate the distribution of sequence data $p(\mathbf{x})$.
- We show how to parameterize autoregressive models by neural networks in order to build generative autoregressive models.
- We introduce the PixelCNN as an generative autoregressive image model.

Table of contents

1. Autoregressive Models for Sequence Data
2. Parameterization of AR Models Through Neural Networks
3. PixelCNN: An Autoregressive Generative Model for Images

1 Autoregressive Models for Sequence Data

Statistical Tools for Sequence Data
Autoregressive Models with Finite Memory

Autoregressive models for sequence data

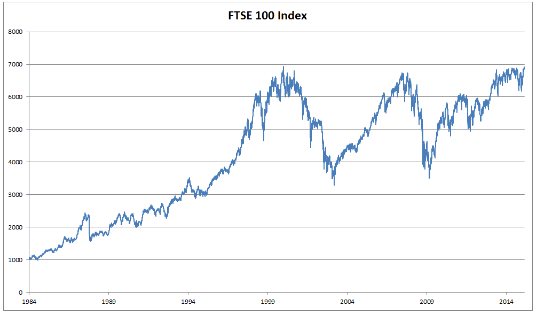

Figure 1: Example of sequential signal: stock prices index over 30 years [1].

Let's denote the prices by $x_t \geq 0$, i.e., at time $t \in \mathbb{N}$ we observe some price x_t. To do well in the stock market on day t, a trader should want to predict x_t via:

$$x_t \sim p(x_t \mid x_{t-1}, \ldots, x_1).$$

Statistical tools for sequence data: autoregressive models I

The trader could use a *regressor*, but there is just a major problem: the number of inputs, $x_{t-1}, \ldots, x_1$ varies, depending on t.

The number of inputs increases with the amount of data, thus potentially being computationally intractable.

The deep learning methods that we are going to introduce now will mainly revolve around how to estimate $p(x_t \mid x_{t-1}, \ldots, x_1)$ efficiently.

We can consider two main strategies requiring specialized statistical tools for estimation:

- Assume that a potentially rather long sequence is not really necessary and adopt an autoregressive (AR) model.
- Keep some summary h_t of the past observations, at the same time update h_t in addition to the prediction $\hat{x}_t$ and adopt a latent autoregressive models.

Statistical tools for sequence data: autoregressive models II

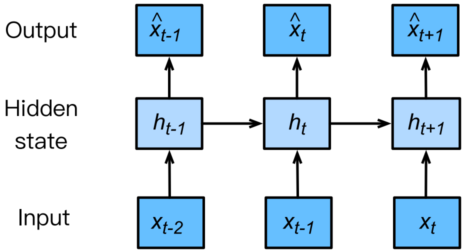

Figure 2: A latent autoregressive model [1].

Autoregressive models estimate the training time series as:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \ldots, x_1).$$

Finite memory: Markov models

The first attempt to limiting the complexity of a conditional model is to assume a finite memory.

Whenever the AR approximation of using only $x_{t-1}, \ldots, x_{t-\tau}$ instead of $x_{t-1}, \ldots, x_1$ holds to estimate x_t, we say that the sequence satisfies a **Markov condition**.

If $\tau = 1$, we have a first-order Markov model and $p(x)$ is given by

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}) \text{ where } p(x_1 \mid x_0) = p(x_1).$$

Such models are particularly nice whenever x_t assumes only a discrete value, since in this case dynamic programming can be used to compute values along the chain exactly.

Casuality and bidirectionality

In principle, there is nothing wrong with unfolding $p(x_1, \dots, x_T)$ in reverse order:

$$p(x_1, \dots, x_T) = \prod_{t=T}^{1} p(x_t \mid x_{t+1}, \dots, x_T).$$

In fact, if we have a Markov model, we can obtain a reverse conditional probability distribution, too.

In many cases, however, there exists a natural direction for the signals, namely *going forward in time*.

It is clear that future events cannot influence the past. Hence, if we change x_t, we may be able to influence what happens for x_{t+1} going forward but not the converse.

2 Parameterization of AR Models Through Neural Networks

Parameterization of Finite Memory AR Models
Parameterization of Long-Range Memory AR Models

Parameterization of finite memory AR models

We can use a small neural network, e.g., an MLP, to predict the distribution of x_t.

Such a model is not only nonlinear but also its parameterization is convenient due to a relatively small number of weights to be trained.

However, the obvious drawback is a limited memory (i.e., only T last variables in our example).

Moreover, it is unclear *a priori* how many variables we should use in conditioning.

In many problems, e.g., image processing, learning long-range statistics is crucial to understand complex patterns in data; therefore, having long-range memory is essential.

Long-range memory through RNNs

A possible solution to the problem of a short-range memory modeled by an MLP relies on applying a recurrent neural network (RNN), thus the conditional distribution can be expressed as:

$$p\left(x_1, \ldots, x_T\right) = \prod_{t=1}^{T} p\left(x_t \mid \mathsf{RNN}\left(x_{t-1}, h_t\right)\right).$$

where h_t acts as a memory that allows learning long-range dependencies.

This approach gives a single parameterization, thus, it is efficient and also solves the problem of a finite memory.

However, it is slow and may suffer from exploding or vanishing gradients.

Long-range memory through CNNs

Sometimes, instead of using RNNs, it could be convenient to stack 1D convolutional layers (Conv1D) to process sequential data, as:

- Kernels are shared (i.e., an efficient parameterization).
- The processing is done in parallel that greatly speeds up computations.
- By stacking more layers, the effective kernel size grows with the network depth.

These three traits seem to place Conv1D-based neural networks as a perfect solution to our problem. However, can we indeed use them straight away?

Causal convolutions

A Conv1D can be applied to calculate embeddings, but it cannot be used for autoregressive models.

Why? Because we need convolutions to be *causal*.

Causal here means that a Conv1D layer is dependent on the last k inputs *but* the current one, or *with* the current one.

So we must “cut” the kernel in half and forbid it to look into the next variables.

If we are concerned about the effective kernel size, we can use *dilation* larger than 1.

Stacking CausalConv1D layers with the dilation larger than 1 allows us to learn long-range dependencies.

3 PixelCNN: An Autoregressive Generative Model for Images

Autoregressive Image Modeling
Autoregressive Conditional Image Modeling

Autoregressive image modeling: PixelCNN I

Pixel-by-pixel is a simple generative method wherein given an image of dimension x_{n^2}, we iterate, employ feedback and capture pixel densities from every pixel to predict our “unknown” pixel density x_i.

Autoregressive models can be employed to compute the joint distribution by a simple chain rule:

$$p(x) = \prod_{t=1}^{n^2} p(x_t \mid (x_{t-1}, \ldots, x_1))$$

where $p(x)$ is the *generated image*, n^2 is the *number of pixels*, and $p(x_t \mid (x_{t-1}, \ldots, x_1))$ is the *probability* of the tth pixel which depends on the values of all previous pixels.

Basically, an image can be modeled as a sequence of points where each pixel depends linearly on previous ones.

Autoregressive image modeling: PixelCNN II

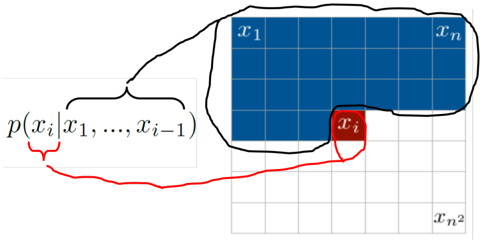

Figure 3: Representation of the joint distribution which displays that the pixels are computed pixel-by-pixel for every row, and the forthcoming pixel depends on the pixels values above and to the left of the pixel in concern. Image source: StatWiki.

The PixelCNN aims to map a neighborhood of pixels to the prediction for the next pixel.

That is, to generate pixel x_t the model can only *condition* on the previously generated pixels $x_1, \ldots, x_{t-1}$; so every conditional distribution is modeled by a CNN.

Autoregressive image modeling: PixelCNN III

To make sure the CNN can only use information about pixels above and to the left of the current pixel, the filters of the convolution are masked.

That means the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions.

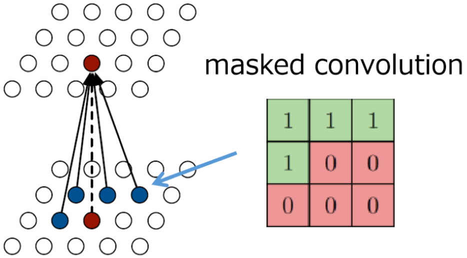

Figure 4: Masked convolution for a 3×3 filter. Image source: StatWiki.

Autoregressive image modeling: PixelCNN IV

To avoid any blind spot in the masking, two filters are considered (horizontal and vertical stacks) in conjunction to allow for capturing the whole receptive field.

By splitting the convolution into two different operations enables the model to access all pixels prior to the pixel of interest.

The Gated PixelCNN [2] uses explicit probability densities to generate new images using AR connections to model images through pixel-by-pixel computation by decomposing the joint image distribution as a product of conditionals.

Autoregressive image modeling: PixelCNN V

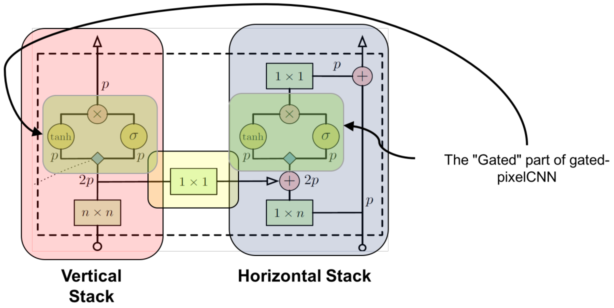

Figure 5: Illustration of a single layer in the Gated PixelCNN architecture; wherein the vertical stack contributes to the horizontal stack with the 1×1 convolution - going the other way would break the conditional distribution. The horizontal and vertical stacks are sort of independent, wherein vertical stack should not access any information horizontal stack has - otherwise it will have access to pixels it shouldn't see. Convolution operations are shown in green (which are masked), element-wise multiplications and additions are shown in red. Image source: StatWiki.

Autoregressive conditional image modeling I

Conditioning is a smart word for saying that we're feeding the network some high-level information.

For instance, providing an image to the network with the associated classes in MNIST/CIFAR datasets.

During training you feed image as well as class to your network to make sure network would learn to incorporate that information as well.

During inference you can specify what class your output image should belong to. You can pass any information you want with conditioning, we'll start with just classes.

Autoregressive conditional image modeling II

For a conditional PixelCNN, we represent a provided high-level image description as a latent vector $\mathbf{h}$, wherein the purpose of the latent vector is to model the conditional distribution $p\left(\mathbf{x}|\mathbf{h}\right)$ such that we get a probability as to if the images suites this description.

The conditional PixelCNN models based on the following distribution:

$$p\left(\mathbf{x}|\mathbf{h}\right) = \prod_{t=1}^{n^2} p\left(x_t \mid \left(x_{t-1}, \ldots, x_1, \mathbf{h}\right)\right)$$

Next lecture

- We will introduce deep generative models relying on latent variables.
  - Unsupervised learning of latent representations is a key point for a successful generation process.
- We introduce the most significant generative latent-based model, that is the variational autoencoder (VAE).
  - VAE involves an encoder to learn the deep latent variables and a decoder to generate new data.

References I

[1] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive Into Deep Learning*, 2020.

[2] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, “Conditional image generation with PixelCNN decoders,” *arXiv preprint arXiv:1606.05328*, Jun. 2016.

[3] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018.

[4] D. Foster, *Generative Deep Learning – Teaching Machines to Paint, Write, Compose and Play*. O’Reilly Media, Inc., Jun. 2019.

[5] J. M. Tomczak, *Deep Generative Modeling*. Springer Nature Switzerland AG, 2022.

Generative Autoregressive Models

Generative Deep Learning
2021/2022

Danilo Comminiello

PhD Course in Information and Communication Technology (ICT)

http://danilocomminiello.site.uniroma1.it

danilo.comminiello@uniroma1.it