Review of Deep Generative Models

Mohammed El mendili *November 2, 2020*

Deep Learning has managed to solve many problems in the last decade [1]. However, the field stills suffer from two major problems [2]: Data efficiency (i.e. ability to learn from small samples) and Generalization (most AI models perform very poorly on out-of-sample data). Solving these problems would naturally lead to breakthroughs in many areas such as probabilistic time series forecasting, multimedia generation, image completion, etc [1]. In this review, we will discuss Deep Generative Models (DGMs), a combination of generative models and Deep Neural Networks (DNNs), that aims to efficiently compress the data into a low-dimensional representation in order to generate it. More specifically, we will discuss three types of likelihood-based methods: An **Auto-regressive generative model**: *Locally Masked Convolutions for Autoregressive Models [1]*, a **Flow-based generative model**: *Glow: Generative Flow with Invertible 1X1 Convolutions [2]*, and a **Variational Encoder model**: *NVAE: A Deep Hierarchichal Variational Autoencoder [3].*

1 Locally Masked Convolutions for Auto-regressive Models [1]

Autoregressive models aims at decomposing the joint distribution of input data (e.g. images) into a product of **conditional parameterized distributions** modelled by a DNN. More formally, the decomposition takes the following form (θ is a parameter of the NN, D denotes the dimension of input data):

$$p_\theta(x) = p_\theta(x_1, ..., x_D) = p_\theta(x_{\sigma(1)}) \prod_{i=2}^{D} p_\theta(x_{\sigma(i)} | x_{\sigma(1)}, x_{\sigma(2)}, ..., x_{\sigma(i-1)})$$

where $\sigma : [D] \to [D]$ is a permutation over $[D]$. This enables to divide the initial (high-dimensional) space into simple parts (here single dimensions) and then capture the inter-dependencies between these low-dimensions (auto-regressive logic) using a graphical model (encoded in the permutation σ). The crucial point here lies in the definition of σ. In fact, one can define $D!$ different permutations over a space of dimension D, and most autoregressive models actually differ in the way they define the factorization in the previous equation [1]. Moreover, given a fixed permutation σ, the sampling process is forward and follows a single generation order (sample $x_{\sigma(1)}$, then $x_{\sigma(2)}$, and so on). This enables to have **exact** density and likelihood $L(\theta)$ estimation [1]:

$$L(\theta) = \mathbf{E}_{x \sim p_x} \sum_{i=1}^{D} log\Big(p_\theta(x_{\sigma(i)} | x_{\sigma(1)}, x_{\sigma(2)}, ..., x_{\sigma(i-1)})\Big)$$

While this sampling order is natural for temporal and sequential data, this is not always the case especially for spatial data such as images. For example, the order is crucial for the success of tasks such as image completion, as it is conceptually better to begin by known areas and then move to sample the unknown areas. More precisely, the so-called **raster scan** (i.e. the top left pixel in an image is modelled unconditionally and then the order is defined accross each rows (left to right) from top to down) is unable to complete the top parts of an image when the bottom part is given [1]. The idea of [1] is to estimate the model θ by maximizing a likelihood L^* over **several orderings** decoded using a uniform distribution p_σ over these orderings:

$$L^*(\theta) = \mathbf{E}_{x \sim p_x} \mathbf{E}_{\sigma \sim p_\sigma} log\Big(p_\theta(x_{\sigma(1)}, x_{\sigma(2)}, ..., x_{\sigma(D)}, \sigma)\Big)$$

Optimization is handled using stochastic gradient descent with order-agnostic training [1]. Note also that a single-order is used per batch, allowing to extract relevant models in the test-time (e.g. for image completion, if we have m observed dimensions $k_1, k_2, .., k_m$, then we use the model that was trained with σ verifying $\sigma(1) = k_1, ..., \sigma(m) = k_m$ and sample the next dimensions according to that order).
Another important implementation point is the use of NN to model the conditional probabilities. Given an $H \times W$ multi channel image, convolutional autoregressive models transform this image into a tensor of log-probabilities such as the coordinate i, j in the output defines the conditional (respective to the considered ordering) distribution of $x_{i,j}$. One potential problem of this approach is that these distributions shouldn't depend on forward looking (in the sense of the considered permutation) observations in the Bayesian network [1]. The solution considered in [1] is to mask *patches of the input (image) to each layer* by defining, **for each permutation** σ, a binary mask matrix (same dimensions as input) and then applying it to the input X using $X = M \otimes X$. This techniques is the core idea of *Locally masked convolutions* (**LMCONV**) presented in [1] and it enables to test (using the same NN) with different image generation

orders (i.e. permutations).
The authors in [1] showed, in their experiments, that replacing convolution layers in the DNN by LMCONV layers as described above and optimizing the corresponding likelihood (different orders p_σ where used in training) leads to better likelihood results than the same NN architecture with convolutional layers. Also, they argued that this approach could generalize well to unseen permutations.

2 Glow: Generative Flow with Invertible 1×1 convolutions [2]

In this section, we will present another type of generative modelling: *Generative Flows*. In this case, the generative process is defined as follows:

$$z \sim p_\theta(z), x = g_\theta(z)$$

where z is the latent variable (same dimention as input variables) and $p_\theta(z)$ is a simple distribution (often set to a spherical gaussian distribution), and g_θ is an invertible function (let's denote f_θ its inverse function). In generative flows, we consider functions that could be written as the composition of a sequences of other K invertible functions (called normalizing flows):

$$z = f_\theta(x) = f_1 \circ f_2 \circ ... \circ f_K(x)$$

This also (like in previous section) lead to (in general) tractable density estimations [2]:

$$log(p_\theta(x)) = log(p_\theta(z)) + \sum_{i=1}^{K} log|det(\frac{df_i}{df_{i-1}})|$$

where $f_0 = Id_x$ (identity function on input space), $f_K = Id_z$ (identity function on latent space), and $\frac{df_i}{df_{i-1}}$ is the Jacobian matrix (the determinent of this matrix is called **log determinant** and its value is the change of density when going from step $i-1$ to step i in the composition described above). One step of the generative flow f_i suggested in [2] uses the three following layers:

- **Actnorm layer:** This is a layer to perform acitvation normalization, it consists on an affine transformation ($y_{i,j} = s \otimes x_{i,j} + b$ where i, j are spatial indexes in this case of images) using a scale (s) and bias (b) parameter per channel. The parameters of the affine transformation are set in a way that ensures zero mean and unit variance for post-actnorm activation (given an initial minibatch of data). The log-determinant has a simple expression: $h.w.sum(log(|s|))$, where h, w are respectively the height and the width of the input x.
- **Invertible 1×1 convolution:** This consists on a simple convolution $\forall i, j, y_{i,j} = W.x_{i,j}$. Again, the log-determinant has a closed form: $h.w.log(det(W))$.
- **Affine coupling layer:** This layer consists mainly on (1) splitting the input tensor into two halves along the channel dimension x_a, x_b, then (2) feeding one of the them x_b to a NN and performing an affine transformation on the other x_a and finally (3) concatenate the two resulting tensors. A closed form for the log-determinant is also given in [2]. Tricks such as Zero initialization of the last layer of the NN and permutation of input dimensions were also used in [2]

Finally, this flow is injected in a multi-scale architecture of depth K (number of steps in the flow) and L levels (for more details, please refer to [4]).
Authors in [2] showed that training this flow based generative model on plain log likelihood yields to significant improvements on standard benchmarks, and that the model is able to generate realistic images.

3 NVAE: A Deep Hierarchichal Variational Autoencoder [3]

Variational Autoencoder (VAE) is yet another framework for generative modelling. The goal of VAEs is to train a model in the form $p(x, z) = p(z)p(x|z)$ where $p(z)$ is a prior distribution over latent variables and $p(x|z)$ is the decoder that generates x from z. Generally, the exact posterior $p(z|x)$ is not tractable and hence the model is trained using an approximate posterior distribution $q(z|x)$ (or encoder). In deep hierarchichal VAEs [3], latent variables in the prior and the posterior are factorized using a partition of the latent space $z = \{z_1, .., z_L\}, p(z) = \prod_i p(z_i|z_{<i}), q(z|x) = \prod_i q(z_i|z_{<i}, x)$ where each term in this factorization is modelled by NNs. In [3], authors focus on building a suitable NN for VAEs rather than reproducing NNs that were designed for a different task. Hence, they build a multi-group hierarchichal VAE (the multi-scale approach helps capture the correlations between variables at top and bottom of the architecture). In addition, they designed the encoder and the decoder using Deep Residual Networks [3] by using depthwise

convolutions instead of regular ones and by performing different operations such as *Batch Normalization*(BN), *Swish activation*($f(u) = \frac{u}{1+e^{-u}}$), *Squeeze and Excitation (SE)* (a gating layer [3]). They also used Residual Normal Distributions to model prior and approximate posterior and stabilized the training using *Spectral Regularization (SR)*[3]. Figure 1 presents the overall architecture.

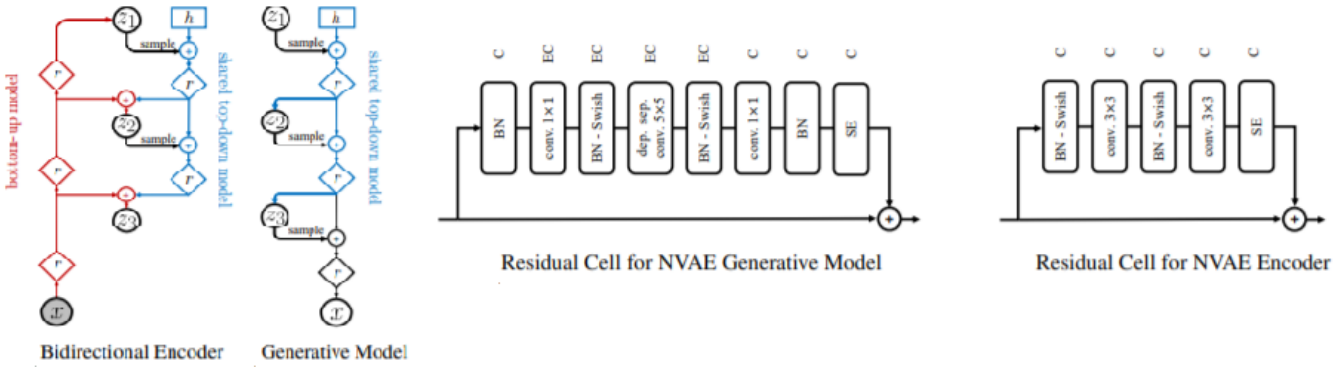

Figure 1: From Left to right: (1) Multi-scale architecture of NVAE (r stands for residual networks, + for feature combinations, and h is trainable parameters. (2) Residual Cell for NVAE generative model. (3) Residual Cell for NVAE Encoder. **Source [3]**

Regarding the results, authors in [3] showed that NVAE achieves state-of-the-art results in **non-autoregressive models**.

4 Comparison

The three approaches we presented aimed at solving the same problem, but they differ in their core conceptions. Here we draw the following comparisons:

- The auto-regressive with masked convolutions approach [1] seems to be the most powerful (state-of-the-art performance are achieved by auto-regressive mdoels [2]) and simplest one. However, it has limited *parallelizability* (due to auto-regressivity) and the complexity depends heavily on the *dimension* of the input (e.g. for videos this would be extremely very expensive).
- In the auto-regressive with masked convolutions and the Glow approaches [1,2], we showed that we have closed forms for likelihood. This could have many applications such as: Uncertainty estimation, Robustness, etc.
- Unlike [1,2], the NVAE approach [3] optimize on a lower bound of the log-likelihood. This is parallelizable [3] both for training and synthesis, but the optimization here could be extremely challenging and could suffer from serious instability issues [2].
- The auto-regressive method don't really allow access to the latent space since the marginal distributions of the hidden layers of the NN are unknown. In contrast, the Glow and NVAE approaches gives an explicit definition of the latent space. This could be particularly useful for downstream manipulations of the latent space [2].
- The three methods [1,2,3] are likelihood based. Hence, they are trained to mimic training data distribution and then they are not sensitive to biases in the dataset.

References

[1] Jain et al. , *"Locally Masked Convolution for Autoregressive Models", UAI 2020.*

[2] Kingma & Dhariwal., *"Glow: Generative Flow with Invertible 1x1 Convolutions", NeurIPS 2018*

[3] Vahdat and Kautz , *"NVAE: A Deep Hierarchical Variational Autoencoder", arXiv 2020*

[4] Dinh et al. (2016) , *Density estimation using Real NVP. arXiv preprint arXiv:1605.08803.*