Probabilistic Generative Models

Jonas Hübotter

May 7th, 2022

This is intended as an introduction to the use of uncertainty in generative models. We will start without assuming any prior knowledge of generative models and progress quickly to discuss the most fundamental ideas behind some of the most powerful generative models to date.

1 *Generative Models*

A generative model is a model that learns to imitate some pattern-generating process. That is, we have (high-dimensional) patterns $\boldsymbol{x} \in \mathcal{X}$ that are generated according to a probability distribution p on $\mathcal{X}$. We want to learn a model $p_{\boldsymbol{\theta}}$ that is indistinguishable from p. In particular, our model should

(1) avoid missing patterns that are generated by p (this is called *mode collapse*) and
(2) avoid generating patterns that are not supported by p.

A famous example is the example of generating human faces, where we want to generate facial features proportional to how frequently they occur in real human faces.

1.1 *Density Estimation*

The problem as stated is above is also called a *density estimation* problem, that is, we want to estimate a density p using a family of densities $p_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$. The canonical method for parameter estimation (also called inference) is *maximum likelihood estimation*, for short. In MLE, we pick $\boldsymbol{\theta}$ such that it explains the patterns generated by p best,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim p}[p_{\boldsymbol{\theta}}(\boldsymbol{x})] \tag{1}$$

$$= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim p}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]. \tag{2}$$

using monotonicity of the logarithm

$p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is also called the *evidence* of $\boldsymbol{x}$. So maximum likelihood estimation corresponds to maximizing the evidence of $\boldsymbol{x}$ under the model $p_{\boldsymbol{\theta}}$ when $\boldsymbol{x}$ is sampled according to the true distribution p.

Given that we can differentiate $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ with respect to $\boldsymbol{\theta}$, we can easily obtain unbiased gradient estimates, which we can use to find a sample-based approximation of $\boldsymbol{\theta}^*$.[1]

[1] Because we can only use finitely many samples $\boldsymbol{x}$, picking a single point estimate of $\hat{\boldsymbol{\theta}}$ might not be the best choice. Often, it is helpful to encode the uncertainty in our choice of $\boldsymbol{\theta}^*$ using a distribution.

Learning a density p directly — for example using standard techniques from deep learning — is hard. This is because our model has to fulfill structural constraints to be a valid density. Moreover, learning the densities directly does not allow us to efficiently sample from the resulting distribution.

Simple parametric models such as *autoregressive models*, which break down the problem of generating $\boldsymbol{x}$ into the problem of sequentially generating components $\boldsymbol{x}(i)$ given previously generated components $\boldsymbol{x}(1), \ldots, \boldsymbol{x}(i-1)$, turn out not to be expressive enough for challenging, high-dimensional generative tasks.[2] We have seen that throwing deep neural networks (DNNs) at the task directly is not very useful, but we do need their expressiveness for modeling p accurately.

[2] Autoregressive models are very useful for time-series prediction.

1.2 Implicit Models

A useful approach to make DNNs work in a constrained setting, is to encode the constraints implicitly in our model. This idea leads to the family of *implicit models*, where we generate $\boldsymbol{x}$ by first generating a random vector $\boldsymbol{z}$ from a known distribution $\mathcal{D}$ and then use a deterministic DNN $f_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ to "lift" $\boldsymbol{z}$ into the set of patterns $\mathcal{X}$. We can think of this as a transformation of random vectors,

$$\mathbf{X} \doteq f_{\boldsymbol{\theta}}(\mathbf{Z}), \quad \mathbf{Z} \sim \mathcal{D}. \tag{3}$$

Often, we sample the *encoding* $\boldsymbol{z}$ according to the standard normal distribution, $\mathcal{D} \doteq \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

For general DNNs f, parameter estimation becomes intractable. To perform parameter estimation, we need to recover the density $p_{\boldsymbol{\theta}}$ of our model. This can be done using so-called *pushforward measures* when f is invertible and it is easy to compute $\det \boldsymbol{J}_f$, where $\boldsymbol{J}_f$ is the Jacobian of f. Such f are also called *normalizing flows*.[3]

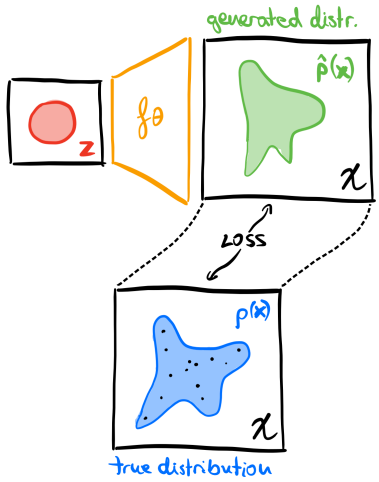


Figure 1: An implicit model "lifts" $\boldsymbol{z}$ from the *code space* (also called *latent space*) to the *pattern space* using a DNN f parametrized by $\boldsymbol{\theta}$.

[3] We will not go into more depth on normalizing flows here.

1.3 Outline

Normalizing flows restrict the architecture of the DNN f to ensure that recovering the density, and thus inference, remains tractable. An alternative approach is to perform inference approximately. This approach, we will discuss in the next section on *variational autoencoders*.

Finally, we will look at a different and very powerful class of models known as *generative adversarial models*.

2 Variational Autoencoders

One common approach to perform approximate inference is to maximize a lower bound on the evidence instead of maximizing

this evidence directly. To obtain a lower bound on the log-evidence $\log p_\theta(x)$, we can relate this quantity back to the space of encodings:

$$\log p_\theta(x) = \log \int p_\theta(x \mid z) p(z)\, dz.$$

Intuitively, the probability of x is the probability of an encoding z being "lifted" to x times the probability of the encoding z to be sampled in the first place.[4] We can rewrite this to,

$$= \log \int p_\theta(x \mid z) \frac{p(z)}{q(z)} q(z)\, dz$$
$$= \log \mathbb{E}_{z\sim q}\left[p_\theta(x \mid z) \frac{p(z)}{q(z)} \right],$$

where q is any density on the code space. Using Jensen's inequality and that log is concave, we obtain,

$$\geq \mathbb{E}_{z\sim q}[\log p_\theta(x \mid z)] - \mathbb{E}_{z\sim q}\left[\log \frac{q(z)}{p(z)} \right]. \tag{4}$$

This lower bound on the log-evidence is also known as the *evidence lower bound* (or ELBO for short).

The regularization term $\mathbb{E}_{z\sim q}\left[\log \frac{q(z)}{p(z)}\right]$ is also called *(reverse) Kullback-Leibler divergence*, $\mathrm{KL}(q\|p)$, and measures how well q approximates the true distribution of encodings p. Maximizing ELBO therefore corresponds to maximizing the likelihood of x given samples of encodings z, while z are sampled according to a distribution q, which is close to p. When q is from a class of distributions parametrized by ψ, then q_ψ is also called a *variational distribution*.

[4] The true distribution over the latent space $p(z)$ is not to be confused with the true distribution over the pattern space $p(x)$.

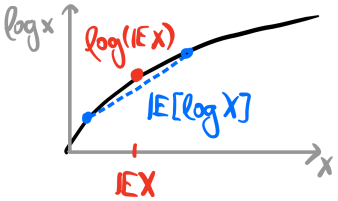


Figure 2: An illustration of Jensen's inequality.

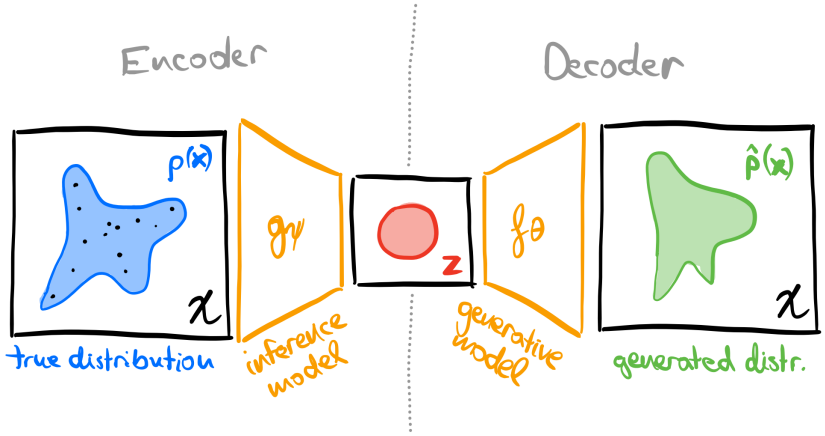


Figure 3: A variational autoencoder learns an inference model and generative model simultaneously. The learned inference model is used to produce encodings from which the generative model should produce patterns.

The key idea of variational autoencoders is to simultaneously learn a generative model p_θ and inference model q_ψ. So given a pattern x

sampled according to the true distribution of patterns p, the objective to be maximized is,

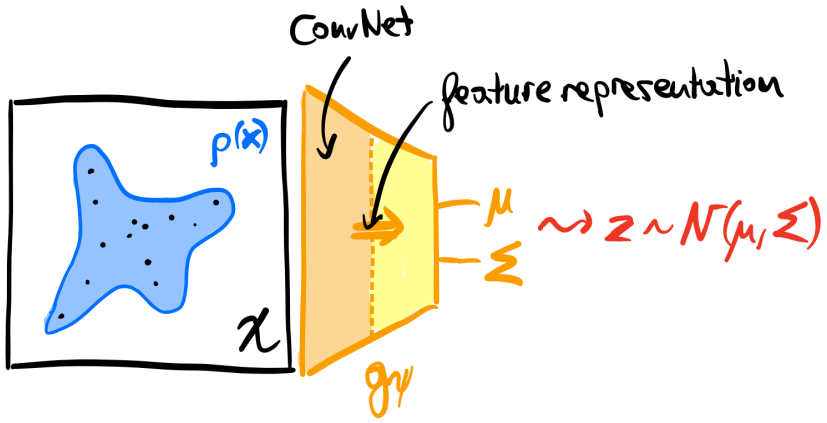
$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \gtrapprox \mathbb{E}_{z\sim q_{\boldsymbol{\psi}}(\cdot|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})] - \mathrm{KL}\big(q_{\boldsymbol{\psi}}(\cdot \mid \boldsymbol{x}) \| p\big) \doteq \ell(\boldsymbol{\theta}, \boldsymbol{\psi}; \boldsymbol{x}). \tag{5}$$

The KL-divergence can often be expressed analytically, and we can use a sample-based approximation for the expectation.

2.1 *The Inference Model*

You may (rightfully) object that we have simply extended our task of finding a generative model $p_{\boldsymbol{\theta}}$ by finding yet another generative model $g_{\boldsymbol{\psi}}$. However, the latter generative model is from a high- to a low-dimensional space, namely from the pattern space to the code space. It turns out that this task is much simpler than the task that we started with.

Typically, the inference model is chosen to be a non-isotropic statistical transformation of the outputs of a DNN. More concretely, we use a DNN $g_{\boldsymbol{\psi}}$ to first transform the high-dimensional pattern $\boldsymbol{x}$ to a low-dimensional feature representation and then to outputs $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\psi}}$.[5] We then choose $q_{\boldsymbol{\psi}}$ to be a simple distribution with mean $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and covariance $\boldsymbol{\Sigma}_{\boldsymbol{\psi}}$ (i.e., $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\psi}}, \boldsymbol{\Sigma}_{\boldsymbol{\psi}})$) that is easy to sample from.



[5] For example, by using a convolutional neural network to obtain the feature representation and a fully connected neural network to learn mappings to $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\psi}}$.

Figure 4: Illustration of a typical inference model. First, a pattern is transformed into a feature representation, which is then used to obtain statistics $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ that parametrize a distribution over encodings.

We call such a model *non-isotropic* because the statistics $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\psi}}$ depend on the pattern $\boldsymbol{x}$.

2.2 *Inference*

Now, to perform inference, we need to be able to obtain gradients with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ of $\ell(\boldsymbol{\theta}, \boldsymbol{\psi}; \boldsymbol{x})$. As we mentioned, the KL-divergence can usually be expressed analytically. This is also true

for its gradient. The gradient of $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ can be computed using backpropagation. The difficulty lies in finding the gradient of the expectation $\mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\psi}}(\cdot \mid \boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]$ with respect to $\boldsymbol{\psi}$. Because the expectation integrates over a measure that depends on the parameters $\boldsymbol{\psi}$, we cannot simply draw the gradient into the expectation using linearity.

Here, we can use the so-called *reparameterization trick* by writing $\boldsymbol{z} \sim q_{\boldsymbol{\psi}}(\cdot, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\psi}}, \boldsymbol{\Sigma}_{\boldsymbol{\psi}})$ as,

$$\boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\psi}} + \boldsymbol{\Sigma}_{\boldsymbol{\psi}}^{1/2} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{6}$$

where $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\psi}}$ are the outputs of the DNN $g_{\boldsymbol{\psi}}$. We obtain,

$$\mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\psi}}(\cdot \mid \boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})] = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{\mu}_{\boldsymbol{\psi}} + \boldsymbol{\Sigma}_{\boldsymbol{\psi}}^{1/2} \boldsymbol{\epsilon})\right]. \tag{7}$$

Of this expectation, we can now easily obtain unbiased gradient estimates, which we can use to maximize the ELBO using stochastic gradient descent. For more details on the reparameterization trick for Gaussians or other distributions and the computation of gradients of the ELBO, refer to section 5.2.7 of my notes on probabilistic artificial intelligence.

Figure 5: An example of faces reconstructed by a variational autoencoder. Taken from [3].

3 *Generative Adversarial Models*

We will now introduce a different class of models that are based on a very simple and beautiful idea, and which are powerful enough to yield models that produce faces that are visually indistinguishable from real faces.

Recall that our original goal was to find a distribution $p_{\boldsymbol{\theta}}$ that is indistinguishable from the distribution p. Our initial instinct was to use density estimation to find $p_{\boldsymbol{\theta}}$, but we have seen that we either have to limit the expressiveness of our model to retain tractability of the inference problem or we have to perform inference approximately. It is therefore useful to think whether we can characterize our problem in a different way.

The notion of indistinguishability leads to a very natural characterization: our goal is to make it virtually impossible for an agent to distinguish between the patterns produced by p and the patterns produced by $p_{\boldsymbol{\theta}}$. From the perspective of such an agent, this is a simple binary classification problem. Our original task can be interpreted as maximizing the loss of this agent, as a large loss corresponds to the scenario where the patterns produced by $p_{\boldsymbol{\theta}}$ cannot be distinguished from the patterns produced by p.

We write $Y \doteq \mathbb{1}\{\text{pattern is "natural"}\}$. Assuming that the classification problem is balanced,[6] it can be characterized using the joint density,

[6] That is, we present the agent with labels such that $\mathbb{P}[Y=1] = \mathbb{P}[Y=0] = 1/2$.

$$p(\boldsymbol{x}, y; \boldsymbol{\theta}) = \underbrace{\mathbb{P}[Y=y]}_{=\frac{1}{2}} \begin{cases} p(\boldsymbol{x}) & y = 1 \\ p_{\boldsymbol{\theta}}(\boldsymbol{x}) & y = 0. \end{cases} \tag{8}$$

A classifier $\pi : \mathbb{R}^n \to [0,1], \pi(\boldsymbol{x}) \approx \mathbb{P}[Y = 1 \mid \boldsymbol{x}]$ is also called a *discriminator*. Intuitively, the discriminator provides an error signal to the generative model, hinting at promising regions in the pattern space.

The canonical approach is to model the loss of a discriminator π using *binary cross-entropy* (also called a *logistic loss*),

$$\ell(\boldsymbol{\theta}, \pi) \doteq \mathbb{E}_{\boldsymbol{x}, y \sim p(\cdot;\boldsymbol{\theta})}[y \log \pi(\boldsymbol{x}) + (1-y)\log(1-\pi(\boldsymbol{x}))] \tag{9}$$

$$= \frac{1}{2}\int p(\boldsymbol{x}) \log \pi(\boldsymbol{x})\, d\boldsymbol{x} + \frac{1}{2}\int p_{\boldsymbol{\theta}}(\boldsymbol{x}) \log(1-\pi(\boldsymbol{x}))\, d\boldsymbol{x}. \tag{10}$$

Our optimization problem is then given as,

$$\min_{\boldsymbol{\theta}} \max_{\pi} \ell(\boldsymbol{\theta}, \pi). \tag{11}$$

In words, we want to make the classification task as difficult as possible for the best-possible classifier π by tuning $\boldsymbol{\theta}$.

The optimal discriminator is a Bayes-optimal classifier π^*,

$$\pi^*(\boldsymbol{x}) = \mathbb{P}[Y = 1 \mid \boldsymbol{x}] = \frac{p(\boldsymbol{x})}{p(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x})}. \tag{12}$$

In practice, we do not have access to this discriminator, as it depends on the true distribution p. Nevertheless, we can compute the expected loss of this classifier to confirm our intuition that when this quantity is large, $p_{\boldsymbol{\theta}}$ must be "almost indistinguishable" from p. For the Bayes-optimal discriminator π^*, we obtain,

$$\begin{aligned} \ell(\boldsymbol{\theta}, \pi^*) &= \mathbb{E}_{\boldsymbol{x}, y \sim p(\cdot;\boldsymbol{\theta})}[y \log \pi^*(\boldsymbol{x}) + (1-y)\log(1-\pi^*(\boldsymbol{x})] \\ &= \frac{1}{2}\int p(\boldsymbol{x})[\log p(\boldsymbol{x}) - \log(p(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x}))]\, d\boldsymbol{x} \\ &\quad + \frac{1}{2}\int p_{\boldsymbol{\theta}}(\boldsymbol{x})[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \log(p(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x}))]\, d\boldsymbol{x} \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\int p(\boldsymbol{x}) \log p(\boldsymbol{x})\, d\boldsymbol{x} + \frac{1}{2}\int p_{\boldsymbol{\theta}}(\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x})\, d\boldsymbol{x} \\
&\quad - \int \frac{p(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x})}{2} \log\left(\frac{p(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x})}{2}\right) d\boldsymbol{x} - \log 2 \\
&= \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p}[\log p(\boldsymbol{x})] + \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})] \\
&\quad - \mathbb{E}_{\boldsymbol{x}\sim \frac{1}{2}p + \frac{1}{2}p_{\boldsymbol{\theta}}}\left[\log\left(\frac{1}{2}p(\boldsymbol{x}) + \frac{1}{2}p_{\boldsymbol{\theta}}(\boldsymbol{x})\right)\right] - \log 2 \\
&= -\frac{1}{2}H(p) - \frac{1}{2}H(p_{\boldsymbol{\theta}}) + H\left(\frac{1}{2}p + \frac{1}{2}p_{\boldsymbol{\theta}}\right) - \log 2, \qquad (13)
\end{aligned}
$$

where $H(p) \doteq \mathbb{E}_{\boldsymbol{x}\sim p}[-\log p(\boldsymbol{x})]$ is the entropy of distribution p and $\frac{1}{2}p + \frac{1}{2}p_{\boldsymbol{\theta}}$ is a convex combination of the distributions p and $p_{\boldsymbol{\theta}}$. The sum of entropies is also known as the *Jensen-Shannon divergence*,

$$
\begin{aligned}
\mathrm{JS}(p, q) &= \frac{1}{2}\mathrm{KL}\left(p \| \frac{1}{2}p + \frac{1}{2}q\right) + \frac{1}{2}\mathrm{KL}\left(q \| \frac{1}{2}p + \frac{1}{2}q\right) \qquad (14) \\
&= -\frac{1}{2}H(p) - \frac{1}{2}H(q) + H\left(\frac{1}{2}p + \frac{1}{2}q\right). \qquad (15)
\end{aligned}
$$

Consider the case where p and q are the same distribution, then the JS-divergence is zero. The described optimization problem minimizes the JS-divergence, and hence, the "distance" between p and $p_{\boldsymbol{\theta}}$.

As mentioned, we cannot compute π^* directly, nor can we compute the Jensen-Shannon divergence. Nevertheless, the above derivation motivates that replacing $\arg\max_\pi \ell(\boldsymbol{\theta}, \pi)$ by a learned discriminator that is close to the true optimum π^* yields a loss that can be used to estimate a good generator. Let us consider a class of discriminators $\pi_{\boldsymbol{\phi}}$ that is parametrized by $\boldsymbol{\phi}$. Assuming this class is expressive enough to cover π^*, we have,

$$
\ell(\boldsymbol{\theta}, \pi^*) = \max_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta}, \boldsymbol{\phi}), \qquad (16)
$$

yielding the saddle point problem,

$$
\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta}, \boldsymbol{\phi}), \quad \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta}^*, \boldsymbol{\phi}). \qquad (17)
$$

Under conditions of Nash's existence theorem for Nash equilibria, the above is equivalent to,

$$
\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \boldsymbol{\phi}^*), \quad \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \ell(\boldsymbol{\theta}^*, \boldsymbol{\phi}). \qquad (18)
$$

This formulation of generative models is known as *Generative Adversarial Networks* (GANs). We can think of this problem as a two player game with the generator trying to choose $\boldsymbol{\theta}$ so as to minimize the loss and the adversary (i.e., the discriminator) trying to choose $\boldsymbol{\phi}$ so as to maximize the loss.

Figure 6: An example of faces constructed by a generative adversarial model, StyleGAN. Taken from [2].

References

[1] Thomas Hofmann. Computational intelligence lab, 2021.

[2] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[3] Rafael S Toledo and Eric A Antonelo. Face reconstruction with variational autoencoder and face masks. *arXiv preprint arXiv:2112.02139*, 2021.