COMP760:

Geometry and Generative Models

Week 3: Generative Models Primer I

Notes by: Joey Bose and Prakash Panangaden

1

GENERATIVE MODELING PRIMER I

Destroying structure by adding noise to data is straightforward; transforming noise to structured data is *generative modeling*. By learning to transmute noise, generative models seek to uncover the underlying generative factors that give rise to observed data. These factors can often be cast as inherently geometric quantities such as natural symmetries that manifest themselves as equivariances or invariances to certain transformation laws. For example, a convolutional neural network is translation invariant while a graph neural network is permutation invariant with respect to node labels. Other generative factors may instead be unobserved and understanding the geometric properties of these *latent* variables can shed light on the natural space the true data resides in—e.g. hierarchical data can be embedded with fewer dimensions in hyperbolic spaces. One may also wish to interrogate the nature of the relationships between certain observed variables (e.g. causal relationships) but within the latent space of a pre-trained generative model. In these cases, geometric manipulations such as learning attribute hyperplanes and subspace projections may help expose, under mild assumptions, the causal relationships between variables of interest, giving key insights into building better and more interpretable generative models.

This chapter includes key works concerned with deep generative modelling. We first characterize different families of generative models and outline their respective modelling strengths and limitations. At the core of this chapter are recent advancements in generative models, which include variational approaches, implicit density models, and invertible models. For modern iterations, where data is often extremely high dimensional and the volume of data necessitates the use of larger computational resources, we turn to deep learning methods that are known to succeed in these settings. Specifically, modern generative models are often parametric approaches where the parameters are deep neural networks that are optimized through stochastic gradient-based techniques enabling scalable modeling of high-dimensional data.

1.1 Latent Variable Models

Probablistic reasoning represents a unified framework for model building, inference, prediction and decision making. Contrary to conventional deep learning methods that make little assumptions about the data, probabilistic methods such as graphical models impose explicit structure on the data generation process that can be exploited to avoid common deep learning pitfalls such as accounting for uncertainty in prediction. Given the complexity of high dimensional data, one reasonable assumption that can be imposed is that the data is generated by some underlying factors of variation. More specifically, the data that can be observed can be thought to be the result of some hidden random variables that seek to explain underlying causes that could potentially explain a specific data sample. Building these so called latent variable models can either be done via prescribed models which use observed likelihoods with some noise or implicit models that are entirely likelihood free as further expanded on in section 1.2. Latent variable models offer many appealing properties such as a simple formalism to encode specific structure known to the practitioner about the data, making it a natural choice for generative modelling of structured data.

1.1.1 Variational Auto-Encoder

Variational Inference (VI) can be thought of as a general purpose learning principle for probabilistic models that require estimating intractable probability density often due to the partition function. Typically, VI requires variational parameters for each data point which can quickly become expensive in large data regimes that are common for modern deep learning. The Variational AutoEncoder (VAE) [Kingma and Welling, 2014] accounts for this by introducing a separate recognition model in addition to the generative model. VAE's are perhaps one of the biggest success stories in modern deep learning as it marries probabilitic graphical models and deep neural networks in a principled manner. The key idea is that we *amortize* the inference process, that is the same parameters of the recognition network are shared for all data points. The VAE has many close parallels to the original Helmholtz machine [Dayan et al., 1995] in that they both use a recognition model along with a separate generative model. However, a Helmholtz machine can have multiple stochastic hidden layers while the recognition model in a VAE has a determinitic encoding to a stochastic latent code. Thus, the VAE is a special case of a Helmholtz machine. Another important difference is that to train the Helmholtz machine we use the wake-sleep algorithm that optimizes two different objectives, while as we will see the VAE makes use of the reparametrization trick to optimize both the recognition and generative models under one unified objective.

Note: You can think of these parameters as ones for a known distribution (e.g. Gaussian)

A VAE is a deep latent variable model where each stochastic latent code is the output of a recognition network. That is the observed data is modelled using an additional unobserved random variable z, such that $p(x) = \int p(x|z)p(z)dz$. To understand what latent codes actually lead to our observed data we need to compute the posterior distribution $p(z|x)$, which in the general case is intractable as it involves computing the likelihood of the data $p(x)$. Instead, we use VI to approximate the posterior using another neural network, q_ϕ called the recognition network or in modern deep learning parlance the *encoder*. We optimize the variational parameters ϕ such that $q(z|x) \approx p(z|x)$. As we use the same encoder network for all data points we essentially share variational parameters which can be a limiting assumption but in practice works well. The goal of optimization is to iteratively move the approximate posterior distribution closer to the real posterior. One principled measure of the difference between two distributions is the KL-divergence which is minimized when the two distributions are the same. However, to do this minimization a decoder or generative network is also needed. Essentially, the decoder maps a stochastic latent code to output space and models $p(x|z)$. A natural question then is how do we know that amortized variational inference actually increases the data likelihood? This can be easily confirmed by writing out the overall objective with both encoder and decoder networks:

$$\log p(x) = \log \int p(x|z)p(z)dz \tag{1.1}$$

$$\geq \mathbb{E}_{q(z|x)}[\log \frac{p(x,z)}{q(z|x)}] \quad \text{(Jensen's Inequality)} \tag{1.2}$$

$$= \mathbb{E}_{q(z|x)}[\log p(x|z)] + \mathbb{E}_{q(z|x)}\left[\log \frac{p(z)}{q(z|x)}\right] \tag{1.3}$$

$$= \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)||p(z)) \tag{1.4}$$

The first term in Eq. 1.4 is the reconstruction error for the decoder while the second term is the KL-divergence between the approximate posterior and a fixed prior, and acts like a regularizer to prevent the model from collapsing all of its weight on one latent code. An equivalent perspective can be arrived at by starting off with $D_{KL}(q(z|x)||p(z|x))$ which after a little bit of algebraic manipulation arrives at what is known as the variational free energy (not to be confused with RBM free energy), $\mathcal{F}$, of the system:

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)||p(z|x)) = -\mathcal{F}(x) \tag{1.5}$$

Since, the KL-divergence is a positive quantity maximizing the first quantity, called the Evidence Lower Bound (ELBO), effectively minimizes the difference between the approximate and true posteriors.

One subtle non-trivial point associated with a stochastic latent code is that there is no direct pathway for gradients needed for end to end optimization as the sampling step is non-differentiable. However a clever trick can be exploited where the non-differentiable function can be *reparameterized* through a deterministic function, $z = g(\epsilon, \theta)$, which shifts the stochasticity from the parameters to a standardized noise model —i.e. $\epsilon \sim \mathcal{N}(0, I)$. In practice, the reparametrized gradient for continuous distributions has lower variance than the REINFORCE[Williams, 1992] gradient estimator and is preferred whenever such reparametrizations exist. The VAE models the stochastic latent code as a diagonal Gaussian, which is easily reparametrizable —i.e. $z = \mu + \epsilon \odot \sigma^2$. Unfortunately, such an inflexible form for the approximate posterior can be a limiting assumption in certain applications.

$\odot$ is the Hadarmard or element-wise product

1.2 Implicit Generative Models

The goal of generative modelling is often to learn the simulator that produces the data, however in practice for some models the practitioner has access to only unnormalized probability densities, or partial knowledge of the data. How then can learning proceed in such a scenario? One approach to this problem is to learn by comparison, that is we compare the estimated distribution $q(x)$ with samples from the true distribution $p^*(x)$. At a high level the aim is to build an auxiliary model that tests how the simulated data differs from observed data under a comparison metric, after which the parameters of the model can be adjusted to move the simulated samples under q to match the real samples under the p^*. This process is known as implicit density estimation as we estimate the density of p^* via an auxiliary model q and a comparison metric.

One simple comparison metric is the density ratio $r = p^*(x)/q(x)$, which can be used a signal to drive learning the auxiliary model parameters. To understand why this works we can introduce class labels $y \in [-1, 1]$ associated with the origin of a sample x. That is each simulated or fake sample is assigned $y = -1$ and correspondingly, a sample from the true data distribution is assigned a label $y = 1$. The ratio r can then be rewritten as follows:

$$r = \frac{p^*(x)}{q(x)} = \frac{p^*(x|y=1)}{q(x|y=-1)} \tag{1.6}$$

$$r = \frac{p^*(y=1|x)}{q(y=-1|x)}. \tag{1.7}$$

We arrive at Eqn 1.7 through a simple application of bayes rule and the cancellations of the marginal densities. Observe that Eqn 1.7 does not contain

the troublesome marginal densities $p^*(x)$ and $q(x)$ but more manageable conditional densities which can be estimated using a parametric scoring function, —i.e. $p(y = 1|x) = D_\theta(x)$ and $p(y = -1|x) = 1 - D_\theta(x)$.

1.2.1 Generative Adversarial Networks

One of the most successful applications implicit density estimation involves training two neural networks in a two-player adversarial game. Specifically, Generative Adversarial Network (GAN) training involves jointly optimizing two networks: a Generator, tasked with producing samples from some distribution that ideally mimics examples from the true data distribution, and a Discriminator, which attempts to differentiate between samples from the true data distribution and the one produced by the Generator. In its most vanilla form the GAN objective can be written as a min-max optimization problem with the following form:

$$\min_G \max_D \mathbb{E}_{p^*(x)}[\log(D_\theta(x))] + \mathbb{E}_{q_\phi(\tilde{x})}[\log(1 - D_\theta(\tilde{x}))]. \tag{1.8}$$

Here p^* denotes the true data distribution and q is the model distribution that is implicitly defined by the Generator. A generated sample $\tilde{x} = G(z), z \sim p(z)$, where z is sampled from some noise distribution such as a Gaussian. It has been shown that a Discriminator trained to optimality minimizes the Jensen Shannon Divergence (JSD). In practice it is common for the Generator to maximize $\mathbb{E}_{q(\tilde{x})}[\log(D(\tilde{x}))]$ [Goodfellow et al., 2014]. If both the Discriminator and Generator are conditioned on additional information, such as the class label, then the quality of generated samples significantly improves.

1.3 Fully Observed Models

Introducing a latent variable can help model underlying factors of variations that are difficult to account for *a priori*, but the resulting parameter estimation problem can be significantly more challenging. Instead of computing lower bounds to the log-likelihood of data like in VAE's we can infact compute the exact likelihoods using fully observed models. Fully observed models attempt to directly encode how data points are related, and more importantly the log-likelihood of the data is no longer an approximation but in fact exact. To do this, these class of models have to assume a specific ordering of variables —i.e. $p(x_0, ..., x_n) = p(x_0) \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1})$. Consequently, these models are order sensitive which makes for slow generation at inference time as to generate the current observation all the preceding observations must be known. Despite these drawbacks many state of the art generative models are in fact fully observed such as RNN's for text generation which we outline in the following example.

Example 1.3.1 ***Seq2Seq***. *Sequence to Sequence (Seq2Seq) Sutskever et al. [2014] models are a class of deep learning methods that as the name suggests consume an input sequence and generate a corresponding output sequence. Common examples of using Seq2Seq includes machine translation where the input is a sequence in a source language and the goal is to translate it to a target language. Specifically, given an input sequence* $(x_1, x_2, ..., x_n)$ *we seek to generate an output sequence* $(y_1, y_2, ..., y_n)$*. One approach is to map the input sequence to a single vector using a RNN and iteratively generate the output sequence using another RNN. However, conventional RNNs struggle to cope with long term dependencies and have common optimzization problems in exploding and vanishing gradients. Instead, Seq2Seq models use LSTM's, a variant of RNN's, that have been empirically shown to have greater capacity to model long term dependencies and alleviate some, but not all, of the optimization problems. By mapping an input sequence to an output sequence, a Seq2Seq models computes the following probability:*

$$p(y_1, ..., y_n | x_1, ...x_n) = p(y_1) \prod_{t=1}^{N} p(y_t | \nu, y_1, ..., y_{i-1}), \tag{1.9}$$

where ν *is the last hidden state of the encoding LSTM and is used to initialize the hidden state of the decoding LSTM. For modelling text, this conditional distribution is computed as a softmax over the entire vocabulary. Training consists of taking paired input-output sequences and maximizing the likelihood of the generated output as is common in supervised learning. For a new input sequence, generation simply takes the form of encoding to* ν *and iteratively decoding until a predefined stopping token is reached.*

2

INVERTIBLE GENERATIVE MODELS

The likelihood of a data sample, $p(x)$, often plays a key role in the design of generative models. While needed for efficient learning, the exact likelihood is not often available when parameters are given by deep neural networks. Methods such as variational inference and implicit density estimation bypass the lack of a likelihood by either computing a lower bound or by estimating the density ratio of a conditional distribution given the data. However, if such likelihoods were easily accessible then exploiting such a fact would result in a more efficient training of generative models as it is neither a lower bound nor a ratio thus providing the clearest supervisory signal when updating model parameters. An important challenge then is constructing models that do provide exact likelihoods.

One class of models that can satisfy such a condition are invertible models which map initially unstructured noise to complex data distributions and back. Indeed, if the dimensionality of the noise matches the data and a bijective transformation exists between the two distributions exact likelihoods under the data distribution can be computed via the change of variable formula for probability distributions. In contrast, GAN-based models that transform latent codes from simple probability space to data space are not bijective from their construction and thus cannot provide exact data likelihoods of generated samples. We now turn to the construction of invertible models that allow for exact likelihood estimation and fast inference in the following sections. The main focus is on providing key model instantiations that exploit clever design decisions allowing them to scale up to high dimensional structured data such as images.

2.1 Normalizing Flows

Given a parametrized density a *normalizing flow* defines a sequence of invertible transformations to a more complex density over the same space via the change of variable formula for probability distributions [Rezende and Mohamed, 2015]. Starting from a sample from a base distribution, $z_0 \sim p(z)$, a mapping $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$, with parameters θ that is both invertible and smooth, the log density of $z' = f(z_0)$ is defined as $\log p_\theta(z') = \log p(z_0) - \log \det \left| \frac{\partial f}{\partial z} \right|$.

Where, $p_\theta(z')$ is the probability of the transformed sample and $\partial f/\partial x$ is the Jacobian of f. To construct arbitrarily complex densities, a chain of functions of the same form as f can be defined through successive application of change of density for each invertible transformation in the flow. Thus the final sample from a flow is then given by $z_k = f_k \circ f_{k-1} ... \circ f_1(z_0)$ and its corresponding density can be determined by leveraging the change of variable formula for probability distributions.

In order to ensure effective and tractable learning, the class of functions f_i must satisfy three key desiderata:

1. Each function f_i must be invertible.
2. We must be able to efficiently sample from the final distribution, $z_k = f_k \circ f_{k-1} \circ \cdots \circ f_1(z_0)$.
3. We must be able to efficiently compute the associated change in volume (*i.e.*, the Jacobian determinant) of the overall transformation.

Given these requirements, the final transformed distribution is given by the change of variables formula:

$$\log p(z_k) = \log p(z_0) - \sum_{i=1}^{k} \log \det \left| \frac{\partial f_k}{\partial z_{k-1}} \right|. \tag{2.10}$$

Of practical importance when designing normalizing flows is the cost associated with computing the log determinant of the Jacobian which is computationally expensive and can range anywhere from $O(n^3) - O(n!)$ for an arbitrary matrix and a chosen algorithm. However, through an appropriate choice of f this computation cost can be brought down significantly. While there are many different choices for the transformation function, f, in this work we consider only RealNVP based flows as presented in [Dinh et al., 2017] and [Rezende and Mohamed, 2015] due to their simplicity and expressive power in capturing complex data distributions.

Example 2.1.1 ***RealNVP***. *Computing the Jacobian of functions with high-dimensional domain and codomain and computing the determinants of large matrices are in general computationally very expensive. Further complications can arise with the restriction to bijective functions making for difficult modeling of arbitrary distributions. A simple way to significantly reduce the computational burden is to design transformations such that the Jacobian matrix is triangular resulting in a determinant that is simply the product of the diagonal elements. In [Dinh et al., 2017], real-valued non-volume preserving (RealNVP) transformations are introduced as simple bijections that can be stacked but yet retain the property of having the composition of transformations having a triangular determinant. To*

achieve this each bijection updates a part of the input vector using a function that is simple to invert, but which depends on the remainder of the input vector in a complex way. Such transformations are denoted as affine coupling layers. Formally, given a D dimensional input x and $d < D$, the output y of an affine coupling layer follows the equations:

$$y_{1:d} = x_{1:d} \tag{2.11}$$

$$y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}). \tag{2.12}$$

Where, s and t are parameterized scale and translation functions. As the second part of the input depends on the first, it is easy to see that the Jacobian given by this transformation is lower triangular. Similarly, the inverse of this transformation is given by:

$$x_{1:d} = y_{1:d} \tag{2.13}$$

$$x_{d+1:D} = (y_{d+1:D} - t(y_{1:d}) \odot \exp(-s(y_{1:d})). \tag{2.14}$$

Note that the form of the inverse does not depend on calculating the inverses of either s or t allowing them to be complex functions themselves. Further note that with this simple bijection part of the input vector is never touched which can limit the expressiveness of the model. A simple remedy to this is to simply reverse the elements that undergo scale and translation transformations prior to the next coupling layer. Such an alternating pattern ensures that each dimension of the input vector depends in a complex way given a stack of couplings allowing for more expressive models.

Example 2.1.2 ***Residual Flows***. *An orthogonal approach to designing expressive normalizing flows is to instead consider residual networks as a form of Euler discretization of an ODE.*

$$x_{t+1} = x_t + h_t(x_t) \tag{2.15}$$

Here, x_t represents the activations at a given layer t (or time). A sufficient condition for invertibilty then is a constraint on the Lipschitz constant $\text{Lip}(h_t) < 1 \; \forall t = 1, \ldots, T$. A Resnet whose $h_t(x_t)$ satisfies the invertibility condition is known as an i-ResNet in the literature [Behrmann et al., 2019].

In practice, satisfying the Lipschitz constraint means constraining the spectral norm of any weight matrix —i.e. $\text{Lip}(h) < 1$ if $||W_i||_2 < 1$. While i-ResNets are guaranteed to be invertible there is no analytic form of the inverse. However, it can be obtained by a fixed-point iteration by using the output at a layer as a starting point for the iteration. Also, the computation of the Jacobian determinant can be estimated efficiently.

2.2 Variational Inference with Normalizing Flows

In a nutshell normalizing flows allow for flexible modeling of complex probability distributions while retaining the attractive properties of efficient sampling needed for fast inference. One successful application of flow-based models is in variational inference which as we discussed in previous sections builds an approximate posterior using a class of known probability distributions to the true intractable posterior. The class of approximations used is often limited, e.g., mean-field approximations, implying that no solution is ever able to resemble the true posterior distribution. We now break this bottleneck by constructing a flow in latent space which leads to significantly more complex probability distributions.

Concretely, consider amortized variational inference with a Gaussian approximate posterior that has the variational free energy $\mathcal{F}$ as defined in Eq. 1.5 and note that the bound depends on the choice of approximate posterior distribution $q(z|x)$ given by an encoder or recognition model. If the dimensionality of the latent space is $d < D$, the approximate posterior factorizes under the mean-field assumption as $q(z|x) = \prod_{i=1}^{d} q(z_i|x_i)$ where each dimension is a Gaussian with parameters given by the recognition model. Thus a sample from this posterior distribution is also a sample from a diagonal Gaussian distribution. Now defining a normalizing flow with K bijections, we can take the original posterior sample and transform it into a sample from a learned and more complex distribution. One such transformations are Planar Flows [Rezende and Mohamed, 2015] defined as $f(z) = z + uh(w^T z + b)$, where w, u, b are the learnable parameters of the model. The log determinant of the Jacobian matrix can be computed in $O(D)$ for planar flows by making using of the matrix determinant lemma and takes the following form: $|\frac{\partial f}{\partial z}| = |1 + u^T \psi(z)|$ where we have defined $\psi(z) = h'(w^T z + b)w$. Importantly, the variational free energy with this new construction is tightened as follows:

$$\mathcal{F} = \mathbb{E}_{q_\phi(z|x)}[\log q_\phi(z|x) - \log p(x, z)] \tag{2.16}$$

$$= \mathbb{E}_{q_{z_0}}[\ln q_{z_0}(z_0)] - \mathbb{E}_{q_{z_0}}[\ln p(x, z_K)] - \mathbb{E}_{q_{z_0}}[\sum_{i=1}^{K} \ln |1 + u_k^T \psi(z_{k-1})_k]. \tag{2.17}$$

A tighter bound on $\mathcal{F}$ not only leads to better posterior samples but can also result in qualitatively better reconstructions when used in a VAE.

2.3 Continuous Normalizing Flows

In the previous section, we focused on building normalizing flows as a composition of functions where each function f_i is invertible and constituted a layer in a flow network. An alternative strategy is to instead consider

the infinite limit as $i \to \infty$ of the number of invertible maps. Integrating in time now corresponds to the *infinitesimal* evolution of the probability mass from the base distribution to the target. The *continuous-time* parametrization of a normalizing flow—also termed Continuous Normalizing Flow (CNF)—can be defined in the language of ordinary differential equations which elucidate the flow's evolution in time.

Let us define z_t to be the flow's state at time t which corresponds to the index i in the finite setting. A CNF is initialized by parametrizing the time derivative of z_t with a function h_ϕ, with parameters ϕ, that defines a vector field and corresponds to the following differential equation:

$$\frac{dz_t}{dt} = h_\phi(z_t, t). \tag{2.18}$$

Unlike previous flow parametrizations the function h takes both the time t as well as the current state z_t as input. Furthermore, h is also required to be uniformly Lipschitz continuous in z_t and continuous in t. Essentially, these conditions guarantee the existence and uniqueness of the solution to the ODE defined by the CNF.

To generate a sample from a CNF we must compute the forward dynamics—i.e. integrate forward in time from a sample from a base distribution along the vector field. Specifically, a sample is found by computing:

$$x = z_{t_1} = z_0 + \int_{t=t_0}^{t_1} h_\phi(z_t, t)dt. \tag{2.19}$$

For density estimation, we need to compute the inverse transform which requires us to take an observation and map it back to a sample under the base distribution. The following equations give the inverse transform:

$$z_0 = x + \int_{t=t_1}^{t_0} h_\phi(z_t, t)dt = x - \int_{t=t_0}^{t_1} h_\phi(z_t, t)dt, \tag{2.20}$$

Equipped with this the instantaneous change in log density is then given by,

$$\frac{d \log p(z_t)}{dt} = -\mathrm{Tr}\left(J_{h_\phi(t,\cdot)}(z_t)\right), \tag{2.21}$$

where Tr is the trace operator and $J_{h_\phi(t,\cdot)}(z_t)$ is the Jacobian of h_ϕ evaluated at z_t. Plugging everything together we arrive at the following formula for the log density of a sample x:

$$\log p(x) = \log p(z_0) - \int_{t=t_0}^{t_1} \mathrm{Tr}\left(J_{h_\phi(t,\cdot)}(z_t)\right). \tag{2.22}$$

In practice, we require numerical integrators to solve the ODE which adds a significant computational burden during training as optimizing for $\log p(x)$ requires integration. However, there are more recent efforts that bypass this computational bottleneck in training [Rozen et al., 2021] by optimizing for a different objective whose solution corresponds to the desired ODE. Despite this training speedup, inference—i.e. generating samples—still requires numerical integration.

Bibliography

J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. In *The 5th International Conference on Learning Representations (ICLR), Vancouver*, 2017.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *The International Conference on Learning Representations (ICLR), Banff*, 2014.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd international conference on Machine learning*. ACM, 2015.

N. Rozen, A. Grover, M. Nickel, and Y. Lipman. Moser flow: Divergence-based generative modeling on manifolds. *Advances in Neural Information Processing Systems*, 34:17669–17680, 2021.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.