On generative models learning complex distributions

1st Bahr Lukas
RWTH Aachen University

2nd Kai Lagemann
RWTH Aachen University

***Abstract*—One disadvantage of neural networks is the missing capability to estimate uncertainties. As highly expressive general function estimators neural networks can approximate arbitrarily complex functions but come at the drawback that epistemic uncertainty is not reflected. Usually, neural networks are trained by a maximum likelihood approach which only permits aleatoric uncertainty estimates inherent in the training data, simply because neural networks are not designed to account for epistemic uncertainty. Bayesian neural networks address this problem by using a prior probability distribution, e.g. Gaussian distribution, for the model parameters that is updated during training. Some priors converge to posterior distributions that can be approximated well, e.g., Gaussian processes, Brownian, frationally Brownian or non-Gaussian stable processes. However, in general, there is no guarantee that a complex posterior distribution can be covered precisely by a simple prior during training. Generative models are one of the most promising approaches to analyze and understand the true posterior distribution of a data set. Normalizing flows transform this simple Gaussian distribution into a more complex one [11]. We show that a new approach with continuous-depth hidden states called Free-form Continuous Dynamics for Scalable Reversible Generative Models (FFJORD) extends this approach by making discrete flows continuous [7]. We show that modern generative networks are capable of learning the observation and corresponding labels. This is shown by evaluating different generative models on a data set with complex distribution.**

I. Introduction

The research field of Machine Learning (ML) focuses on learning statistical models by representing the underlying distribution incorporated in the data set. In general, the parameters of a model are adapted by maximizing the probability of the output given the input. A loss function, computing the error between the predicted outputs and the ground truth labels, is used to update the parameters incrementally. We can refer to this kind of neural networks as variational inference networks, for which every data point is associated to a local latent variable. Furthermore, we can optimize the variational inference network. Instead of learning the set of free parameters, an amortized inference network learns the parameters of a parameterized function. For simplicity a Gaussian function is often used, for which the decoder network outputs the parameters mean and variance.

Kingma and Welling show that using variational inference lead to a proper representation of the true posterior distribution incorporated in the data set and also introduced a method to trace the parameterized function called reparameterization trick [11]. Even with increasing data size the number of parameters are constant and new observations can be easily fitted in the latent space without needing to train the network completely again.

But amortized inference networks come with a trade-off in approximating the true posterior distribution. A Gaussian function might be a simple solution because only two parameters are needed and to its closed form expressions offer a good traceability, but only normally distributed data can be approximated properly.

Rezende et al. addressed this problem by transforming the Gaussian distribution through invertible mapping functions also referred to Normalizing Flows (NF) [15]. As a result, the posterior distribution can be arbitrarily complex and scalable. The transformation of the distribution is referred to flow and follows the theorem for change of variables [15]. Rezende et al. introduce two different forms of flows, planar and radial flows. A sequence of flows outputs the final density function and its distribution. A simple initial distribution is transformed in discrete steps by invertible functions until the desired complexity of the distribution is reached. To this end, a nonlinear deterministic function is learned which maps the data into the latent space from which new samples can be generated.

Although the distribution of NF is closer to the true distribution, the method used by NF introduces a single-unit bottleneck in which we perceive an information loss during the training process. Van den Berg et al. remove this bottleneck by using Sylvester flows [1]. Sylvester Normalizing Flows (SNF) achieve more flexible posterior distribution and can be seen as generalization of planar flows.

Neural Ordinary Differential Equations (NODE) is a new type of deep neural network models. NODEs show promising results 1) for supervised learning 2) continuous normalizing flows and 3) time series models [2]. In NODEs there is no need to set a discrete sequence of hidden layers. Instead, the layers are parameterized by the derivative of the hidden states using a neural network. For solving the output of the network a black-box differential equation solver is used.

When using normalizing flows a Jacobian determinant needs to be calculated which can be computational demanding. Free-form Continuous Dynamics for Scalable Reversible Generative Models (FFJORD) make use of continuous depth models. As a result, only the trace of the Jacobian needs to be calculated which is done by Hutchinson's trace estimator leading to a scalable unbiased estimation of the log-density and more accurate posterior distribution.

The contribution of this paper is the evaluation of FFJORD based on the architecture of NODE [2] achieving similar

results in comparison to traditional generative networks based on VAE like SNF [1].

II. Related Work

Using generative networks to output both observation and labels has not been proposed to this date. However, many generative models have preceded outputting images.

Variational Autoencoder Kingma and Welling introduced a parameterization function in order to learn an inference network for which standard stochastic gradient methods can be used [11]. Even with increasing data size the number of parameters are constant and new observations can be easily fitted into the latent space without needing to train the network completely again.

Normalizing Flows Instead of approximating the posterior distribution with a simple parameterization function, Rezende et al. demonstrated that with a sequence of invertible mappings the initial density function is turned into a more complex one [15]. Van den Berg et al. removed the bottleneck from planar flows using Sylvester normalizing flows [1].

Continuous-Depth Models Chen et al. proposed a new architecture of neural networks with continuous-dept hidden states [2]. Also, a generative model with continouous normalizing flows is demonstrated which can be trained by maximum likelihood. In order to solve the ordinary differential equations the adjoint sensitivity method is used [13].

Scalable Reversible Generative Models In order to reduce the computational cost of normalizing flows, Gratwohl et al. showed that by modelling the transformation by an ordinary differential equation the Jacobian can be calculated using the trace. The result is an unbiased density estimation.

III. Background Theory

Generative models perform maximum likelihood in order to learn the parameters θ of the model $p_\theta(x, \mathbf{z})$, where x denotes the observation and $\mathbf{z}$ the latent variable. Amortized inference networks reduce the amount of parameters needed to learn by the network. Kingma and Welling proposed using a Gaussian function for which the variational autoencoder learns the parameters mean and variance to represent the observations in the latent space [11]. New data is acquired by sampling from this distribution. Even with increasing data size the number of parameters are constant and new observations can be easily fitted in the latent space without needing to train the network again.

Denote $x \in \chi$ as the observation and $\mathbf{z} \in \mathbb{R}^D$ the latent variables. We want to construct a distribution so that we can sample $\mathbf{z}$ that are likely to produce x aiming to maximize the marginal likelihood of the model $p_\theta(x, \mathbf{z})$. Formally the probability of x being the true observation is given by

$$p(x) = \int p(x|\mathbf{z};\theta)p(\mathbf{z})d\mathbf{z}. \tag{1}$$

In order to get highly accurate estimations of $p(x)$ we need to sample $\mathbf{z}$ from the latent space. If we would sample all possible $\mathbf{z}$ from the latent space, e.g. Gaussian distribution, it would take a long run to find $\mathbf{z}$ which are likely to have produced x. Instead, Kingma and Welling propose constructing a distribution $q_\phi(\mathbf{z}|x)$ which only outputs $\mathbf{z}$ that are likely to have reconstructed x [11]. We can construct $q_\phi(\mathbf{z}|x)$ as $q_\phi(\mathbf{z}|x) = \mathbb{N}(\mathbf{z}|\mu_\phi(x), \sigma_\phi(x))$ and train it using an inference network. Mean $\mu_\phi(x)$ and the diagonal variance $\sigma_\phi(x)$ are deterministic functions trained with the parameters ϕ by the inference network.

The parameters θ and ϕ are trained by minimizing the negative evidence of lower bound (ELBO). Jordan et al. propose that we can approximate $q_\phi(\mathbf{z}|x)$ by constructing lower bounds on the log marginal likelihood [8]. The ELBO gives us the desired measurement of the distribution of the respective observation and the distribution of a latent vector. Hence, an approximate posterior can be calculated by transforming the originally KL divergence $D[q(\mathbf{z})||p(\mathbf{z}|x)] = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log \frac{q(\mathbf{z})}{p(\mathbf{z}|x)}] = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log q(\mathbf{z}) - \log p(\mathbf{z}|x)]$ so that the lower bound follows as

$$\begin{aligned} \log p_\theta(x) &\geq \log p_\theta(x) - D[q_\phi(\mathbf{z}|x)||p_\theta(\mathbf{z}|x)] \\ &= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log p_\theta(x|\mathbf{z})] - D[q_\phi(\mathbf{z}|x)||p(\mathbf{z})] \\ &=: -\mathcal{F}(\theta, \phi). \end{aligned} \tag{2}$$

Note, that the first part of the inference network (encoder) outputs the parameter $\mu_\phi(x)$ and $\sigma_\phi(x)$ for the distribution $q_\phi(\mathbf{z}|x)$. If $\mathbf{z}$ is sampled from $q_\phi(\mathbf{z}|x)$ it will not be deterministic. In order to backpropagate through the network to update the network parameter, $\mathbf{z}$ needs to be traceable. The solution proposed by [11] suggests a reparameterization trick. In order to make $\mathbf{z}$ deterministic, a random variable ϵ is introduced. So, instead of sampling $\mathbf{z}$ directly from $q_\phi(\mathbf{z}|x)$, we multiply the standard deviation $\sqrt{\sigma^2}$ by $\epsilon \sim \mathcal{N}(0, I)$ and add it to μ. ϵ denotes the random node, so $\mathbf{z}$ becomes deterministic and we can backpropagate through the inference network.

The posterior distribution $q_\phi(\mathbf{z}|x)$ is restricted in approximating the true posterior distribution because it is modelled using a simple diagonal Gaussian function. Hence, the gap between $q_\phi(\mathbf{z}|x)$ and $p_\theta(\mathbf{z})$ is typically large. As a consequence, Rezende et al. introduced an algorithm for learning a more flexible posterior distribution closer to the true posterior distribution [15].

Applying the change of variable theorem to a random variable z with distribution $q(z)$ and an inverse function $f^{-1} = g$, the resulting of the random variable $z' = f(z)$ distribution is defined as $q(z') = q(z) \cdot |\det \frac{\partial f}{\partial z}|^{-1}$. The $|\det \frac{\partial f}{\partial z}|$ specifies about how much the function is locally shrinking or expanding. Thus, the initial density ends in a valid probability function. We refer to the sequence of invertible mapping as normalizing flows.

If we consider a chain of transformations z_k, a chain can be represented as $z_k = f_K \circ ... \circ f_2 \circ f_1(z_0)$, where k is the number of flows.

Rezende et al. proposed, that we can decrease the cost of computing the Jacobian determinant to $\mathcal{O}(D)$ by using planar flows [15]. A planar flow is given by

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b). \tag{3}$$

The vectors $\mathbf{w} \in \mathbb{R}^D, \mathbf{u} \in \mathbb{R}^D, b \in \mathbb{R}$ are learned through the inference network and $h(\cdot)$ with derivative $h'(\cdot)$ is referred as $h = \tanh$. In case of $h = \tanh$, [15] shows that a transformation is invertible if $\mathbf{u}^T\mathbf{w} \geq -1$.

The density is than given by

$$\ln q_k(z) = \ln q_o(z) - \sum_{k=1}^{K} \ln|1 + \mathbf{u}_k^T \psi_k(\mathbf{z}_{k-1})|. \tag{4}$$

A flow given by eq. 4 modifies its initial density q_0 through a series of contractions and expansions in the direction perpendicular to the hyperplane $w^{\mathbf{T}}\mathbf{z} + b = 0$.

Van den Berg. et al. propose a generalization of planar flow and removes the single-unit bottleneck [10] resulting in more flexible transformations [1]. A more general transformation follows as $\mathbf{z'} = \mathbf{z} + \mathbf{A}h(\mathbf{B}\mathbf{z} + \mathbf{b})$, where $\mathbf{z'} = \mathbf{z} \in \mathbb{R}^D, \mathbf{A} \in \mathbb{R}^{D\times M}, \mathbf{B} \in \mathbb{R}^{M\times D}, \mathbf{b} \in \mathbb{R}^M$ and $M \leq D$ with M being the number of hidden units. Using Sylvester's determinant identity theorem, we can define the determinant to

$$\det\left(\frac{\partial \mathbf{z}'}{\partial \mathbf{z}}\right) = \det\left(\mathbf{I}_M + dia(h'(\mathbf{B}\mathbf{z} + \mathbf{b}))\mathbf{B}\mathbf{A}\right). \tag{5}$$

$\mathbf{A}$ and $\mathbf{B}$ can be parameterized by $\mathbf{z}' = \mathbf{z} + \mathbf{Q}\mathbf{R}h(\tilde{\mathbf{R}}\mathbf{Q}^{\mathbf{T}}\mathbf{z} + \mathbf{b}) = \phi(z)$, where $\mathbf{R}$ and $\tilde{\mathbf{R}}$ are upper triangular $M \times M$ matrices and $\mathbf{Q} = (q_1 ... q_m)$ with the columns $\mathbf{q}_m \in \mathbb{R}^D$ forming an orthonormal set of vectors.

To preserve the orthogonality of Q, van den Berg et al. introduced three different flows 1) Orthogonal Sylvester flows, 2) Householder Sylvester flows and 3) Triangular Sylvester flows [1].

The orthogonality of $\mathbf{Q}$ can be preserved with an iterative procedure

$$\mathbf{Q}^{k+1} = \mathbf{Q}^k\left(\mathbf{I} + \frac{1}{2}(\mathbf{I} - \mathbf{Q}^{(k)T}\mathbf{Q}^{(k)})\right) \tag{6}$$

and a sufficient condition convergence given by $||\mathbf{Q}^{(0)T}\mathbf{Q}^{(0)} - \mathbf{I}||_2 < 1$. On the basis that eq. 6 is differentiable, any standard solver can be used for optimization. The free energy bound for Orthogonal Sylvester Normalizing flows follows in analogy to eq. 2.

$$\begin{aligned}\mathcal{F}(x) = \mathbb{E}_{q_0(z_0)}[\ln q_0(\mathbf{z}_0) - \log p(\mathbf{z}_K)] - \mathbb{E}_{q_0(z_0)}[\log p(X|\mathbf{z}_K)] \\ - \mathbb{E}_{q_0(z_0)}[\ln|1 + diag(h'(\tilde{\mathbf{R}}\mathbf{Q}^{\mathbf{T}}\mathbf{z}_{\mathbf{K-1}} + \mathbf{b}))\tilde{\mathbf{R}}\mathbf{R}|].\end{aligned} \tag{7}$$

IV. Continuous Depth Models

A. Neural Ordinary Differential Equations

Neural Ordinary Differential Equations (NODE) are a new family of deep neural network architectures [2]. With NODEs there is no need to set a discrete sequence of hidden layers. Instead, the layers are parameterized by the derivative of the hidden states using a neural network. For solving the output of the network a black-box differential equation solver is used. Chen et al. used NODE to replace residual network in supervised learning, proposed a way to generate time-series models, and improved normalizing flows to continuous normalizing flows [2].

Chen et al. propose that a transformation of a residual $\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \theta)$ can be performed multiple times and the interval of time steps can be reduced [2]. So, we end up parameterizing the continuous dynamics using an ordinary differential equation (ODE) modeled by a neural network

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta). \tag{8}$$

The initial value can be derived from the solution of the output layer $\mathbf{z}(T)$ at time T by using a differential equation solver.

1) The Adjoint Sensitivity Method: Calculating a state $\mathbf{z}(t_1)$ can be computed in analogy to other network architectures using a forward pass. The forward pass follows as

$$\begin{aligned}\mathbf{z}(t_1) &= \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt \\ &= ODESolve(\mathbf{z}(t_0), f, t_0, t_1, \theta).\end{aligned} \tag{9}$$

The next step is to evaluate the error introduced by the model. This is done by calculating the loss of the ODE. Given a differentiable loss function and a ODE solver we could backpropagate through the solver. The loss for $\mathbf{z}(t_1)$, can be computed by

$$\begin{aligned}L(\mathbf{z}(t_1)) &= L(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt) \\ &= L(ODESolve(\mathbf{z}(t_0), f, t_0, t_1, \theta)).\end{aligned} \tag{10}$$

However, this operation is time consuming and scales poorly with more complex ODE functions. Instead of backpropagating through the ODE, solver Chen et al. make use of a computationally and memory efficient method called adjoint sensitivity method [13]. The adjoint sensitivity method is a numerical method and gives information about how sensitive a change to a function's input effects the output by computing the gradient of a function. Keeping track of the time dynamics is achieved with an adjoint state, which is defined by

$$a(t) = \frac{dL}{d\mathbf{z}(t)}. \tag{11}$$

The adjoint state represents how the loss depends on the hidden state, which is continuous in time t. We can take the derivative w.r.t. time and calculate the dynamics

$$\frac{da(t)}{dt} = -a(t)\frac{\partial f(\mathbf{z}(t), t; \theta)}{\partial \mathbf{z}(t)}. \tag{12}$$

This can now be solved by an ODE solver calculating backwards in time starting from the derivative of the loss at $\mathbf{z}_{t+1}$, $\frac{dL}{d\mathbf{z}(t_1)}$ to the desired state $\mathbf{z}(t)$.

Having calculated the adjoint state, we can update the parameters θ. With $\mathbf{z}(t)$ and $a(t)$, we can solve the derivative $\frac{dL}{d\theta}$ by evaluating the integral

$$\frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt. \tag{13}$$

In [2] the integrals for solving $\mathbf{z}$, a and $\frac{dL}{d\theta}$ can be computed in a single call to a solver. Therefore, we need to concatenate the initial state, the partial derivatives and the adjoint into a single vector. By using automatic differentiation the Jacobian products $\mathbf{a}(t)^T\frac{\partial f}{\partial \mathbf{z}}$ and $\mathbf{a}(t)^T\frac{\partial f}{\partial \theta}$ can be efficiently evaluated.

2) Continuous Normalizing Flows: Using the change of variable introduced in [10], it is necessary to calculate the determinant of the Jacobian $\det(\frac{\partial f}{\partial \mathbf{z}})$. The computation is of cost $\mathcal{O}(D^3)$ with D being the dimensionality of the data. A continuous transformation simplifies the computation of the change to a normalizing constant.

Using the theorem of instantaneous change of variables leads to $\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -tr(\frac{\partial f}{\partial \mathbf{z}(t)})$. Computing the trace reduces the computational cost to $\mathcal{O}(D^2)$. The log density is given by

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} tr(\frac{\partial f}{\partial \mathbf{z}(t)})dt. \tag{14}$$

B. Scalable Reversible Generative Models

Free-form Continuous Dynamics for Scalable Reversible Generative Models (FFJORD) [7] reduces the computational cost of continuous normalizing flows even further. The cost of computing $\det(\frac{\partial f}{\partial \mathbf{z}})$ is $\mathcal{O}(D^2)$ for continuous normalizing flows. However, Grathwohl et al. show that using the Hutchinsons's trace estimator reduces the computational cost to $\mathcal{O}(D)$. Using the Hutchinson's trace estimator the trace can be calculated by

$$\begin{aligned} tr(\mathbf{A}) &= tr(\mathbf{A}\mathbf{I}) = tr(\mathbf{A}\mathbb{E}[\epsilon\epsilon^\top]) \\ &= \mathbb{E}[tr(\mathbf{A}\epsilon\epsilon^\top)] = \mathbb{E}[\epsilon^\top \mathbf{A}\epsilon], \end{aligned} \tag{15}$$

where ϵ is a random noise vector. Typically, ϵ is sampled from a Gaussian or Rademacher distribution [14]. Plugging the Hutchinsons's trace into the log density function leads to

$$\begin{aligned} \log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} Tr(\frac{\partial f}{\partial \mathbf{z}(t)})dt \\ &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\epsilon)}[\epsilon^T \frac{\partial f}{\partial \mathbf{z}(t)}\epsilon]dt \\ &= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\epsilon)}[\int_{t_0}^{t_1} \epsilon^T \frac{\partial f}{\partial \mathbf{z}(t)}\epsilon dt]. \end{aligned} \tag{16}$$

V. Learning heterogeneous data sets

We propose a model using generative networks to generate annotated samples. This is achieved by training the network with heterogeneous data. We refer to learning heterogeneous data, because the network is trained on both images and labels for which the underlying distribution is disconnected and to homogeneous data if the network is trained solely on images. Instead of simply sampling images without appropriate labels from the posterior distribution, the model we propose outputs sampled images with proper labels.

A. Problem statement

Generative models are important for 1) generating synthetic images [5], 2) reinforcement learning [12] and 3) image manipulation [16]. In some use cases, especially in classification problems in which the network is trained on the observation x and outputs the classification y, labels are of great importance.

Fig. 1 shows another approach to generate annotated data. We could first train a classification network with the true ground labels and second, connect the pre-trained model with the output of a generative network. This approach has some drawbacks as the classification network needs to be trained upfront.

Instead, we propose an approach for which the network is trained on both images and labels. The model outputs images with respect to the labels. Fig. 1 illustrates both approaches. The benefit of this method is that it is simple to train as only one generative network is involved, takes less time to train and opens a new way to manipulate images and labels.

As the network is trained on disconnected data it is essential for the network to model the underlying distribution appropriately. Modern generative models show promising results capable of learning disconnected distributions. We want to focus on two newer approaches, Sylvester Normalizing Flows (SNF) [1] and Free-form Continuous Dynamics for Scalable Reversible Generative Models (FFJORD) [7].

B. Training the networks

The networks are trained on heterogeneous data by adding a channel to the input containing the labels. For MNIST the channel is of size one, for PIV we use two channels as the label features two flow vectors. In order to make the network trainable on the heterogeneous data, we need to normalize the labels to $(0, 1)$.

VI. Experiments

To evaluate our approach, we concentrate on two datasets: MNIST and PIV (Particle Image Velocimetry). Note that we use CIFAR10 in one experiment to give a broader image with respect to the results. PIV is a private dataset which features a more complex underlying distribution in comparison to MNIST or CIFAR10.

PIV is a method to visualize flows like fluids. The PIV data set consists of a pair of flow images taken at time t and $t + 1$ and labels with directions of the flow vectors $\mathbf{u}$ and $\mathbf{v}$. In order to take pictures a PIV setup typically consists of a camera, a laser to illuminate the region of interest, and a synchronizer to shutter the camera when the laser reflects on the particles. However, we use a synthetic data set, which was generated using a fluid simulation. The train data set consists of 1.000.000 data points.

In order to generate heterogeneous data, we compare Free-form Continuous Dynamics for Scalable Reversible Generative Models (FFJORD) [7] and Householder Sylvester Normalizing Flows (H-SNF) [1]. Both networks are implemented as proposed in [7] and [1]. For all experiments Adam was used as optimizer. Both networks were trained using a learning rate scheduler on plateau starting from 0.005 and minimized to 0.00005. For H-SNF we also used a scheduler for the scalar beta which is multiplied by the KL divergence for calculating the loss. We started from beta equal to 0.01 and went to 1. The number of flows was set to 8 and the number of householder flows to 8. For FFJORD we used two stacked Continuous Normalizing Flows (CNF), a multiscaled network

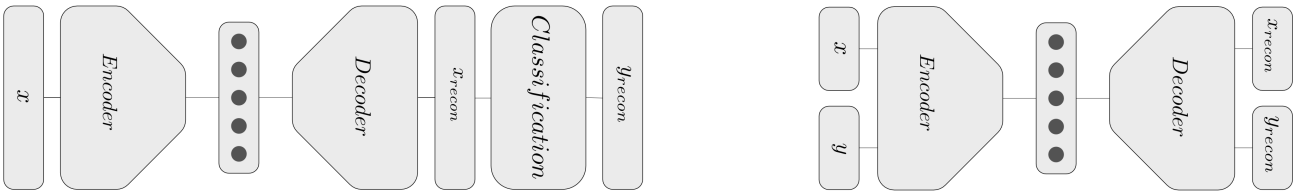

Fig. 1. In order to generate annotated labels, it is conceivable to first, train a classification network on the true images and second, to classify the sampled observations. However, our approach is to train a heterogeneous model, which is trained on both data and labels. Outputting the sampled observations with respect to the labels.

and a concatenating layer type. Both model used convolutional neurons with stride one.

To provide evidence of our approach we conducted two experiments. First, we show that the model outputs reconstructed images with labels and generates samples from the latent space of FFJORD and SNF. Second, we evaluate the goodness of the generated PIV data by sampling from the latent space with different noise vectors.

VII. Sampling of annotated data

We show that our approach is capable to model heterogeneous data. For the experiment FFJORD and SNF were trained with PIV, MNIST and CIFAR10, each as heterogeneous and homogeneous data set. Training FFJORD and SNF on homogeneous MNIST and CIFAR10 went well as the models converged quickly and outputted reasonable reconstruction images. However, we were not able to find feasible hyperparameters for SNF trained on homogeneous PIV data. On the other hand, FFJORD showed good results on both homogeneous and heterogeneous data set. FFJORD also issued reasonable results on heteregenoeus MNIST dataset and only showed weak results on heterogeneous CIFAR10 data set. Tab. I summarizes the results. In general, we found it easier to get results from FFJORD as it was more stable to train and always converged on training. On the downside, training using FFJORD took twice the time in comparison to SNF on homogeneous data sets. Training heterogeneous data with SNF often lead to unevolved images. SNF failed to approximate the true distribution of the unconnected input. Even on the singular PIV data set SNF often did not match any distribution. In addition, we found that FFJORD 1) generates less blurry images, 2) is more stable to train and 3) converges quicker to proper results. Blurry images are a well known phenomena under VAEs. VAEs tend to distribute probability mass sparsely over the latent space, so that samples from VAEs tend to be blurry [4]. We refer to unstable training if the the loss during the training goes to infinity. That is why we concentrate on FFJORD in the following experiments.

	MNIST	CIFAR10	PIV
O-SNF	+/-	+/-	○/-
FFJORD	+/○	+/-	+/+

TABLE I
Difficulty to learn model on homogeneous and heterogeneous data sets. + simple, ○ medium, - hard to train, homogeneous/heterogeneous.

Next, we evaluate the deconstruction of the distribution by FFJORD measured in bits/dims, which evaluates how much the distribution is compressed in dimension for each epoch. In order to compare the data sets, FFJORD is trained on MNIST and PIV on both homogeneous and heterogeneous data sets with 10.000 observations. Fig. 3 demonstrates that FFJORD takes longer on greater compression rates on the homogeneous and heterogeneous PIV data sets in comparison to the MNIST data sets. Moreover, we found that the heterogeneous data sets decrease the overall compression rate on both data sets. Also we noticed that the heterogeneous data sets show fluctuations on the compression rates whereas the homogeneous data sets exhibit a stable compression rate.

With FFJORD trained on the heterogeneous PIV data set, we were able to sample pairs of flow images with respect to their flow vectors $\mathbf{u}$ and $\mathbf{v}$. Fig. 2 shows samples from the approximated posterior distribution of FFJORD with respect to the flow vectors. The model is trained on two images each having the size of 32x32 pixels and a flow vector. In order to train the model, we concatenate the data into a four dimensional tensor. The first two channels consist of the first and second image, the third and fourth channels are filled with the flow vector. The samples are sharp and show that the particles shift from first to second image. Also, we found that the flow vectors provide the correct range in comparison to the ground truth data.

We want to show how the latent space for heterogeneous PIV data set behaves. Therefore, we generated a latent vector representing the ground truth observation in the latent space. Fig. 4 shows the pair of flow images and vector with this

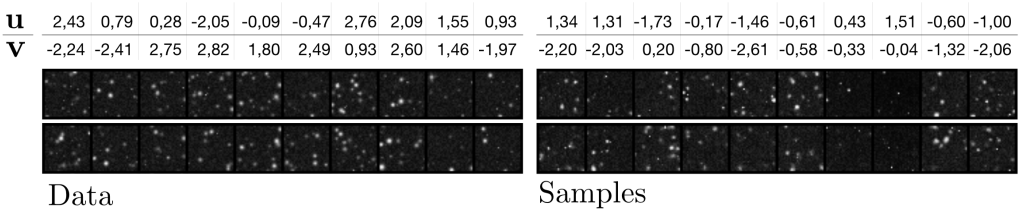

Fig. 2. PIV images with associated vectors $\mathbf{u}$ and $\mathbf{v}$ sampled from FFJORD with heterogeneous PIV data set. True ground images with flow vector on left, sampled images and vectors on right. First row shows location of the particles in t and the second row in $t+1$.

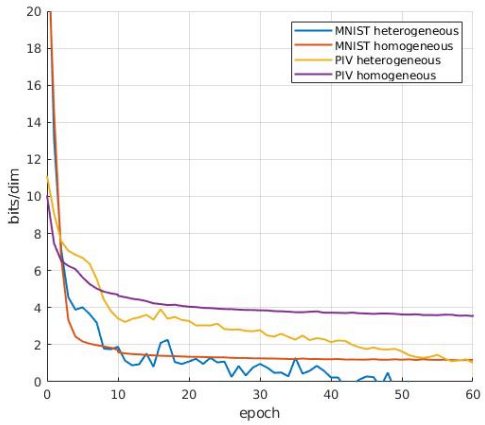

Fig. 3. Compression rate of MNIST and PIV in bits/dim per epoch. Falling monotonously for all data sets.

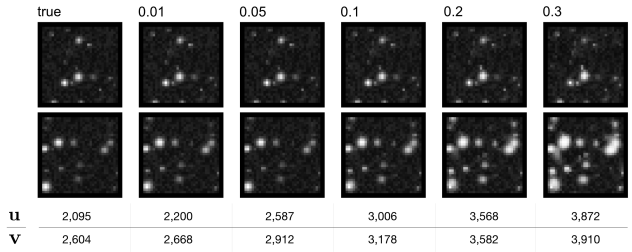

Fig. 4. PIV samples from the latent space with different noise vectors added to the ground truth latent vector of the observation on the left. The values above give the maximum value drawn from an uniform distribution for the noise vectors. First row shows location of the particles in t and the second row in $t+1$.

particular latent vector on the very left. In order to show how moving in the latent space affects the output, we added increasing noise vectors generated from a uniform distribution to the latent vector from the ground truth observation.

An increase of the noise vectors results in pairs of flow vectors for which some particles get brighter and others fade away. For a noise vector drawn from an uniform distribution with the maximum value of 0.3 the particles are the brightest. Larger noise vectors lead to grained pictures in which no particular particle was visible. Fig. 4 also illustrates the change of the vectors. Raising the value of the noise vector lead to a rise of the flow vectors. In comparison to the images for which only a slight change is noticeable, the vectors tend to change strongly relative to the true vector.

VIII. Conclusion

In this work, we have presented an approach to generate annotated samples using Free-form Continuous Dynamics for Scalable Reversible Generative Model (FFJORD) [7] and Sylvester Normalizing Flows (SNF) [1]. It is demonstrated that training modern generative models results in a valid approximation of the true posterior distribution. It is demonstrated that we can train FFJORD using heterogeneous data sets to output images with corresponding labels. This is achieved by learning a posterior distribution incorporated in the data set. Sampling from this distribution leads to annotated data.

The difficulty of learning heterogeneous data is, that we provide the network with two disconnected distributions incorporated in the data. However, we found that FFJORD performs well in approximating both distributions. We provided an evaluation of the latent space for MNIST data set using a t-SNE plot, showing proper regions for the labels. Drawing from this latent space showed good results on both images and labels. To give further evidence, we presented the results of FFJORD trained on a heterogeneous PIV data set. FFJORD managed to output sharp pairs of flow images and also depicted the shift of particles.

In future work we need to provide formal evidence why FJJORD is capable of learning heterogeneous data and SNF is not. Moreover, a deeper and more empirical evidence must be given for the results we presented on FFJORD trained on the heterogeneous MNIST and PIV data sets. Furthermore, we need to show that FFJORD also manages to learn other heterogeneous data sets. Another important point is the elaboration of heterogeneous data sets on different autoencoder architectures e.g. GAN [6], Glow [9] or RealNVP [3]. The goal must be to learn a complete continuous representation of the true posterior distribution incorporated in the data.

REFERENCES

[1] Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester Normalizing Flows for Variational Inference. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, March 2018.

[2] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *International Conference on Neural Information Processing Systems*, June 2018.

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *Computing Research Repository*, May 2016.

[4] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially Learned Inference. *International Conference on Learning Representations*, page 18, 2017.

[5] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification. *Neurocomputing*, 321:321–331, December 2018.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, June 2014.

[7] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *nternational Conference on Learning Representations*, 2019.

[8] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, 1999.

[9] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 11 Convolutions. *Neural Information Processing Systems*, page 10, 2018.

[10] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving Variational Inference with Inverse Autoregressive Flow. *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, June 2016.

[11] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *International Conference on Learning Representations*, 2013.

[12] David Pfau and Oriol Vinyals. Connecting Generative Adversarial Networks and Actor-Critic Methods. October 2016.

[13] L.S. Pontriagin. *The mathematical theory of optimal processes*. John Wiley, 1962.

[14] Hans Rademacher. *Collected papers of Hans Rademacher. 1*. MIT Press, 1974.

[15] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. *Proceedings of the 32nd International Conference on Machine Learning*, May 2015.

[16] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *IEEE International Conference on Computer Vision*, March 2017.