A review of invertible neural networks: theory and applications

Bhavya Mehta
Department of Computer Science Engineering
Veermata Jijabai Technological Institute, Mumbai.
bdmehta_b19@ce.vjti.ac.in

Adithya Dev K P
Department of Mathematics
Birla Institute of Technology and Science, Pilani
f20190867@hyderabad.bits-pilani.ac.in

Purav Singla
Department of Mathematics
Birla Institute of Technology and Science, Pilani
puravsingla011@gmail.com

Rohit Singh Rathaur
Department of Mathematics
Birla Institute of Technology, Mesra
rohitrathore.imh55@gmail.com

Rahul Sarkar
Institute for Computational and Mathematical Engineering
Stanford University
rsarkar@stanford.edu

Contents

1 Introduction

2 Earliest Invertible Architectures

2.1 Iterative Inversion

Iterative inversion is a powerful technique used in the field of neural networks to approximate the input that generates a given output. It is a process that iteratively adjusts the input to a neural network based on the difference between the network's output and the desired output.
Developed by A. Linden and J. Kindermann, Inversion of Multilayer Networks gave the base to all further developments in the field of Invertible Neural Networks. A standard Least Mean Square Error along with gradient search in the input space is used to compute the input that gives the desired output.

Computing the Inverse
Let O denotes the set of indices in the output space, I the set of indices in the input space and T the desired output. The loss fucntion is given by:

$$L = \sum_{i \in O} (t_i - a_i)^2$$

We train the network in the forward direction as usual. We choose a starting point in the input space say I^0. We compute the output:

$$O^0 = f_L(f_{L-1}).....f_2(f_1(I^0, W_1), W_2).....W_{L-1}), W_L)$$

where f_i is the activation function for the i'th layer and W_i is the weight matrix for the i'th layer. We compute the Error and error signals are backpropagated to the previous layers without making changes to the weights. The input vector is updated as follows

$$I^k = I^{k-1} - \eta \nabla_{I^{k-1}} L(f_L(f_{L-1}).....f_2(f_1(I^0, W_1), W_2).....W_{L-1}), W_L)$$

where η is the step size and ∇ is the gradient with respect to I^{k-1}. The update happens until a local minima is attained and we get the input that gives an output with error minimized locally.
The approach, even though novel in it's time is computationally expensive compared to the architectures we'll be reviewing later on in this paper. It is also not necessary that this method will give the global minimum and often will not give the most optimal input for the desired output.

2.2 Inversion of Feed Forward Neural Networks Jensen et al. [1999]

This method explains that inverting a neural network implies finding the input pattern that corresponds to a desired output pattern while maintaining the original weights and activation functions.

Inverting Feedforward Neural Networks: Addressing Non-Uniqueness and Non-Existence of Solutions
In the problem of inverting a feedforward neural network (FNN), the goal is to find an input pattern x that produces a desired output pattern y^* when passed through the network. Formally, the problem can be defined as:

Find x^* such that $y^* = f(x^*)$, where $f(x)$ represents the FNN. However, several challenges arise when attempting to invert an FNN:

1. **Non-uniqueness of solutions**: There may be multiple input patterns x that produce the same output pattern y^*. This is because FNNs can have complex and non-linear activation functions, and their internal representations may compress information, leading to a many-to-one mapping from input patterns to output patterns. Mathematically this is represented as:

$$f(x_1) = f(x_2) = ... = f(x_n) = y^* \tag{1}$$

   where $x_1, x_2, ..., x_n$ are different input patterns that produce the same output pattern y^*.

2. **Non-existence of solutions**: In some cases, there might not be any input pattern x that produces the desired output pattern y^*. This situation can occur if the desired output lies outside the range of the FNN's output function or if the FNN is not capable of generating the desired output due to its architecture or learned weights.

According to Jensen et al. neural network inversion algorithms can be placed into three broad classes:

1) exhaustive search.

2) single-element search methods.

3) multicomponent (population-based) evolutionary methods that operate on a plurality of potential solutions.

2.2.1 Single-Element Search

Given some neural network function $f : X \to Y$, for some $y \in Y$, the appropriate $x \in X$ needs to be found, such that $f(x) = y$. Or more generally, if $L : Y \to R$ is the loss function defined over the network output, an input x have to be found that minimizes $L(f(x))$.

Williams-Linder-Kindermann Inversion

The method of WLK inversion involves two main steps:

1. computing the deltas for every value

2. updating the weights manually

For the initial input vector i_0. The recursive equation of the training phase is the following:

$i_{t+1}^k = i_t^k - \eta \frac{\partial E}{\partial i_t^k}$

where t is the index of the iteration, i_t^k is the kth component of the i_t vector, and η is the learning rate. The derivatives of the neurons need to be solved by backward order from the output to the input.

2.2.2 Genetic Methods

The basic operations of the genetic algorithm on the search points are:

1) Selection based on fitness.

2) Recombination of genetic material based on crossover.

3) Mutation. When applied to neural network inversion, each network input search point was encoded as a single bit string.

When applied to neural network inversion, each network input search point can be encoded as a single bit string. A fitness function returns a fitness score based on the quality of each search element

Boundary Marking Algorithm

The process starts with N points randomly distributed over the space of interest. In each generation, M points with the worst functional error ($If(x) - cl$) are deleted and replaced by perturbations of the least crowded remaining points. Details are outlined below.

Generate N points randomly distributed (e.g., uniformly) over the space of interest. For each generation:

1. Sort the points by their $I_f(x_i) - c_I$ errors

2. Delete the M points with the worst $I_f(x_i) - c_I$ errors

3. Generate a replacement for each deleted point:

(a) Sort the remaining points in order of their average distance to their nearest m neighbors,

(b) Select a parent k from the least crowded points. Random selection from the first $\frac{N}{5}$ points in the d_i sort order is typical.

(c) Generate the new point $x_{i+1} = x_k - \kappa_n$ where $n \sim N(0, V_I)$.

3 Flow Based Invertible Networks

Flow-based models are a type of generative model that use Invertible transformations to map a simple distribution (such as a Gaussian distribution) to a target distribution. The idea behind flow-based models is to build a sequence of invertible transformations that can "flow" the simple distribution to a distribution that closely matches the target distribution. By using the inverse of these transformations, it is possible to sample from the target distribution and generate new samples. Flow-based models play a very important role in the majority of advanced neural network architectures.

Flow networks use the two key concepts of Jacobian Determinant and change of variable theorem. The change of variable theorem states that, Given a random variable z whose probability density function is known (let's say $\pi(z)$) and it has a 1-1 and invertible mapping f such that $x = f(z)$ and $z = f^{-1}(x)$ then $p(x)$ is given as follows:

$$p(x) = \pi(f^{-1}(x))|\frac{df^{-1}}{dx}| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

The multivariate version of the change of variable theorem is as follows:

$$p(x) = \pi(z)|\det \frac{dz}{dx}| = \pi(f^{-1}(x))|\det \frac{df^{-1}}{dx}|$$

where $|\det \frac{dz}{dx}|$ is the Jacobian determinant. Also note that if the Jacobian Determinant is 0, then the propability density of the target variable cannot be found and invertibility cannot be achieved.

3.1 Normalizing Flows

Danilo Jimenez Rezende, Shakir Mohamed developed the concept of Normalizing Flows to approximate a distribution much more accurately than the existing methods such as backpropagation. Normalizing flows is built on the change of variable theorem, ie. The normalizing flow model applies a series of invertible transformation functions with the help of the change of variable theorem to accurately approximate the probability density function of the final target variable. This model provided the base for majority of the recent Invertible Neural Networks

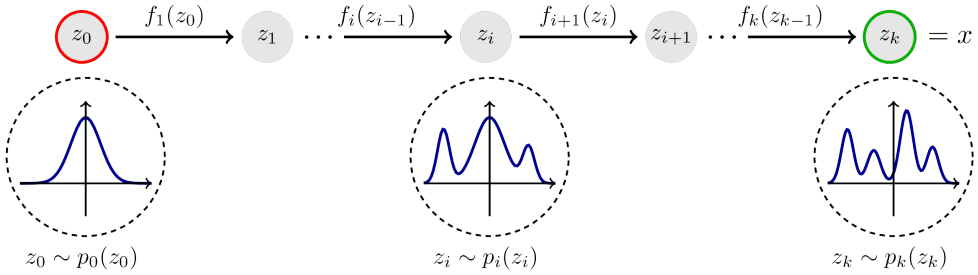


Figure 1: Illustration of a Normalizing Flow model

As illustrated,

$$\begin{aligned} p_i(\mathbf{z}_i) &= p_{i-1}\left(f_i^{-1}(\mathbf{z}_i)\right)\left|\det \frac{df_i^{-1}}{d\mathbf{z}_i}\right| \\ &= p_{i-1}(\mathbf{z}_{i-1})\left|\det\left(\frac{df_i}{dz_{i-1}}\right)^{-1}\right| \\ &= p_{i-1}(\mathbf{z}_{i-1})\left|\det \frac{df_i}{dz_{i-1}}\right|^{-1} \\ \log p_i(\mathbf{z}_i) &= \log p_{i-1}(\mathbf{z}_{i-1}) - \log\left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right| \end{aligned}$$

$$
\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1 \left(\mathbf{z}_0\right) \\
\log p(\mathbf{x}) = \log \pi_K \left(\mathbf{z}_K\right) &= \log \pi_{K-1} \left(\mathbf{z}_{K-1}\right) - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \log \pi_{K-2} \left(\mathbf{z}_{K-2}\right) - \log \left| \det \frac{df_{K-1}}{d\mathbf{z}_{K-2}} \right| - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \ldots \\
&= \log \pi_0 \left(\mathbf{z}_0\right) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|
\end{aligned}
$$

Note, the transformation function should be easily invertible and the Jacobian determinant should be easy to compute as well.

The normalizing flow models make the exact log-likelihood of the input data $\log p(x)$ tractable. Hence for training a flow-based model, the negative log-likelihood is minimized over the training dataset.

$$
\sum_{x \in X} - \log p_X(X) \tag{1}
$$

3.1.1 Normalizing Flows Main Ideas

Inverse Transform Sampling:

Inverse transformation sampling takes uniform samples of a number u between 0 and 1, interpreted as a probability, and then returns the smallest number $x \in \mathbb{R}$ such that $F(x) \geq u$ for the cumulative distribution function F of a random variable.

Probability Integral Transform:

It relates to the result that data values that are modeled as being random variables from any given continuous distribution can be converted to random variables having a standard uniform distribution.

Using both these results it can be shown that :

There always exists an invertible transform $f : X \to Z$ for any two probability densities.Lets call our neural network to be this invertible function f.

Change of variables in the probability density function:

Let's assume a continuous random variable Z with n dimensions having joint density p_Z and an invertible as well as differentiable function g, define $X = g(Z)$, then density of X is as follows:

$$
p_X(x) = p_Z(z) \left| \det \left(\frac{\partial z}{\partial x} \right) \right| = p_Z(g^{-1}(x)) \left| \det \left(\frac{\partial g^{-1}(x)}{\partial x} \right) \right| = p_Z(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x} \right) \right| \tag{2}
$$

where $f := g^{-1}(x)$ and $\det \left(\frac{\partial f(x)}{\partial x} \right)$ is the determinant of the Jacobian matrix.

Using the above result, we can compute the (log-)likelihood of any variable $X = f^{-1}(Z)$ with density Z and the function f.

Finally, we can train a deep latent variable model directly with simple latent variables Z and an invertible deep neural network (f) to model some unknown complex distribution X.

We look at some of the models to build a Deep Neural Network that is simultaneously invertible and can represent the values of a complex transform.

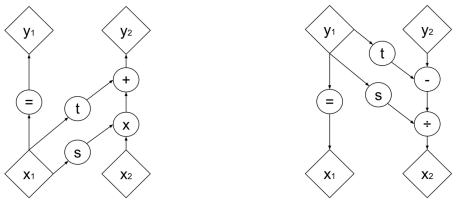
3.1.2 RealNVP

By layering numerous bijective functions on top of one another, the RealNVP (Real-valued Non-Volume Preserving; Dinh et al., 2017) model implements the normalising flow model. Each layer's input dimension is divided into two parts. If the model has a total of D dimensions, the first d dimension remain the same, while the $d + 1$ to D dimension undergoes a scale and translation operation with the first d dimensions as function parameters.

$$
f : x \to y(bijective function)
$$

$$y_{1:d} = x_{1:d}$$
$$y_{d+1:D} = x_{d+1:D} \circledcirc exp(s(x_{1:d}) + t(x_{1:d})$$

where $s()$ and $t()$ are the scale and translate function. It can be noted from the equation that computing the inverse doesn't require the inverse of s and t to be calculated. They can be arbitrarily complex and can be modelled by deep neural networks. The model also ensures that none of the component are left unchanged by reversing the order of components in some affline coupling layers.



(a) Forward propagation (b) Inverse propagation

Figure 2: Forward and reverse computations of affine coupling layer

Masking Schemes in RealNVP:

Let X be any distribution, $z = f_\theta(x)$ be a deep neural network parameterized by θ and $p_Z(z)$ be our prior on latent variables Z.We have :

$$\log p_X(x) = \log(p_Z(f_\theta(x)) \mid\mid \det(\frac{\partial f_\theta(x)}{\partial x}) \mid\mid) = \log p_Z(f_\theta(x)) + \log(\mid\mid \det(\frac{\partial f_\theta(x)}{\partial x}) \mid\mid) \text{ by Eq. 3} \tag{3}$$

From our coupling layer, our jacobian is as follows:

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & diag(exp[s(x_{1:D})]) \end{bmatrix}$$

Upon Simplification:

$$\log \left| \det \left(\frac{\partial y}{\partial x^T} \right) \right| = \sum_j s_j(x_1 : d) \tag{4}$$

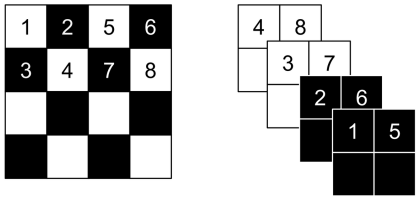


Figure 3: Masking schemes for coupling layers indicated by black and white: spatial checkerboard (left) and channel wise (right). Squeeze operation (right) indicated by numbers.

3.1.3 NICE

The NICE (Non-linear Independent Component Estimation; Dinh, et al. 2015) is a model developed before RealNVP. The difference is that the affline coupling layer in NICE does not have the scale term.

3.1.4 Glow

The Glow (Kingma and Dhariwal, 2018) model extends the previous reversible generative models, NICE and RealNVP, and simplifies the architecture by replacing the reverse permutation operation on the channel ordering with invertible 1x1 convolutions. Each step of flow of a GLOW model is divided into three substeps as illustrated in Figure 2 and Figure 3. The affline coupling layers remains the same as the RealNVP model.

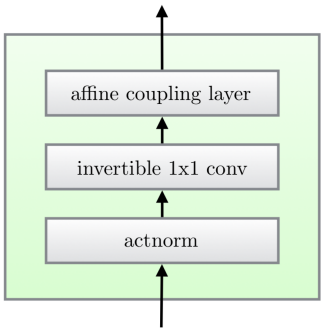


Figure 4: Each step of flow of the GLOW model

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

Figure 5: Table illustrating the substeps in the GLOW model

3.2 Models with Autoregressive Flows

The autoregressive content is a way to model sequential data, $x = [x_1, x_2, ..., x_D]$: each output only depends on the data observed in the past, but not on the future ones. In other words, the probability of observing x_i is the conditioned on x_1, x_2, x_{i-1} and the product of these conditional probabilities gives us the probability of observing the full sequence:

$$p(x) = \Pi_{i=1}^{D} p(x_i | x_1, ..., x_{i-1}) = \Pi_{i=1}^{D} p(x_i | x_{1:i-1}) \tag{5}$$

how to model the conditional density is of our choice. It can be a univariate Gaussian with mean and standard deviation computed as a function of $x_{1:i-1}$, or a multilayer neural network with $x_{1:i-1}$ as the input.

If a flow transformation in a normalizing flow is framed as an autoregressive model - each dimension is a vector variable is conditioned on the previous dimensions - this is an autoregressive flow.

This section starts with several classic autoregressive models (MADE, PixelRNN, WaveNet) and then we dive into autoregressive flow models (AMF and IAF).

3.2.1 MADE

MADE (Masked Autoencoder for Distribution Estimation Germain et al. [2015]) is a unique architecture designed to enforce the autoregressive property in an autoencoder efficiently. In an autoregressive model, the goal is to predict the conditional probabilities of each variable in a given sequence based on the values of its preceding variables. Traditional autoencoders would require feeding the model with different observation windows at multiple time steps to

capture these dependencies. However, MADE improves upon this process by applying binary mask matrices to the autoencoder's hidden units. This masking technique ensures that each input dimension is reconstructed only from previous dimensions in a given ordering, effectively capturing the autoregressive property in a single pass. As a result, MADE is more computationally efficient and better suited for modeling distributions with strong autoregressive properties, such as time series data or image generation tasks.

In a multilayer fully-connected neural network with L hidden layers and weight matrices $W^1, W^2, ..., W^L$, as well as an output layer with weight matrix V, the computation through the layers can be described as follows:

1. First, the input x is passed through the hidden layers. At each layer l, a linear transformation using the weight matrix W^l is applied, followed by an activation function (e.g., $ReLU$, $sigmoid$, or $tanh$). The output of the final hidden layer is represented as h^L.

$$h^l = f(W^l hl - 1), \qquad l = 1, 2, ..., L$$

   where $h^0 = x$ and f is the activation function.

2. The output of the final hidden layer, h^L, is then passed through the output layer with the weight matrix V. In the case of an autoregressive model, the output $\hat{x}$ has each dimension $\hat{x_i} = p(x_i|x_{1:i-1})$. A softmax function is typically applied to produce the probability distribution over the output dimensions.

$$\hat{x} = \sigma(Vh^l + c)$$

Without any mask applied to the weight matrices, the network would not be able to enforce the autoregressive property, and each output dimension would depend on all input dimensions rather than just the preceding ones. To capture the autoregressive property, one can use the masking technique introduced in MADE, which modifies the weight matrices by element-wise multiplication with binary mask matrices.

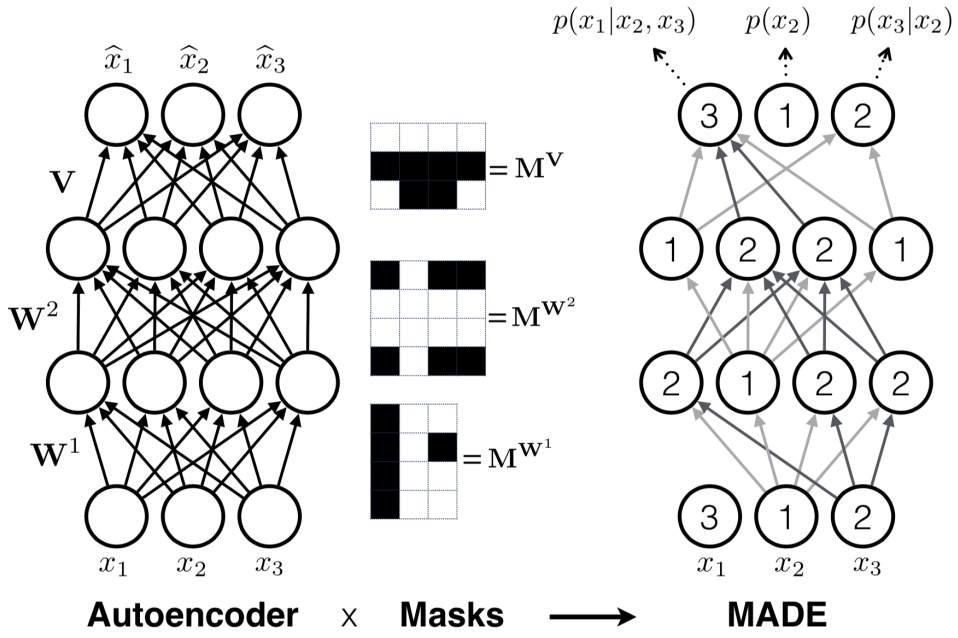


Figure 6: Demonstration of how MADE works in a three-layer feed-forward neural network. (Image Source: Germain et al. [2015])

To zero out some connections between layers, we can simply element-wise multiply every weight matrix by a binary mask matrix. Each hidden node is assigned with a random "connectivity integer" between $1 and (D - 1)$; the assigned value for the k^th unit in the l^th layer is denoted by m^l_k. The binary mask matrix is determined by element-wise comparing values of two nodes in two layers.

$$
\begin{aligned}
\mathbf{h}^l &= \text{activation}^l\left(\left(\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l}\right)\mathbf{h}^{l-1} + \mathbf{b}^l\right) \\
\hat{\mathbf{x}} &= \sigma\left(\left(\mathbf{V} \odot \mathbf{M}^{\mathrm{V}}\right)\mathbf{h}^L + \mathbf{c}\right) \\
M^{\mathbf{W}^l}_{k',k} &= \mathbf{1}_{m^l_{k'} \geq m^{l-1}_k} = \begin{cases} 1, & \text{if } m^l_{k'} \geq m^{l-1}_k \\ 0, & \text{otherwise} \end{cases} \\
M^{\mathbf{V}}_{d,k} &= \mathbf{1}_{d \geq m^L_k} = \begin{cases} 1, & \text{if } d > m^L_k \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{6}
$$

A unit in the current layer can only be connected to other units with equal or smaller numbers in the previous layer and this type of dependency easily propagates through the network up to the output layer. Once the numbers are assigned to all the units and layers, the ordering of input dimensions is fixed and the conditional probability is produced with respect to it. See a great illustration in Fig. 2. To make sure all the hidden units are connected to the input and output layers through some paths, the ${m^l}_k$ is sampled to be equal or greater than the minimal connectivity integer in the previous layer, $min_{k'} {m_k}^{(l-1)}$.

MADE training can be further facilitated by:

1. Order-agnostic training: shuffle the input dimensions, so that MADE is able to model any arbitrary ordering; can create an ensemble of autoregressive models at the runtime.
2. Connectivity-agnostic training: to avoid a model being tied up to specific connectivity pattern constraints, resample ${m_k}^l$ for each training minibatch.

3.2.2 PixelRNN

PixelRNN, as described in the paper by van den Oord et al. [2016b], is a deep generative model for images. In this model, the image is generated pixel by pixel, with each new pixel being sampled conditionally based on the previously seen pixels. Considering an image of size nxn, with pixels denoted as $x = x_1, x_2, ..., x_{(}n^2)$, the model initiates pixel generation at the top-left corner, then proceeds from left to right and from top to bottom (as illustrated in Fig. 3). Each pixel is generated sequentially, taking into account the past context of previously generated pixels. This allows the model to capture and learn the dependencies between the pixels in the image.

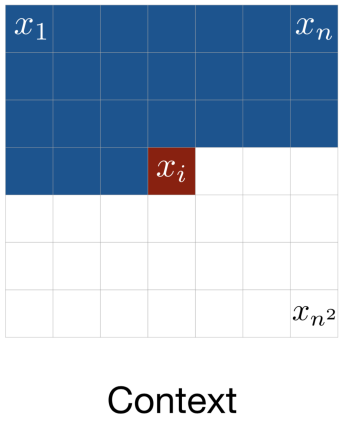


Figure 7: The context for generating for one pixel in PixelRNN. (Image Source: van den Oord et al. [2016b])

Every pixel, denoted as x_i, is sampled from a probability distribution that is conditional on the previous context, which includes pixels above it or to its left in the same row. Although the definition of this context seems arbitrary, as attention to an image can be more flexible, a generative model with such strong assumptions can still perform well.

An implementation that could encompass the entire context is the Diagonal BiLSTM. Initially, a skewing operation is applied to the input feature map, offsetting each row by one position relative to the row above it. This allows for parallelized computation across each row. Subsequently, the LSTM states are calculated with respect to the current

pixel and the pixels to its left.

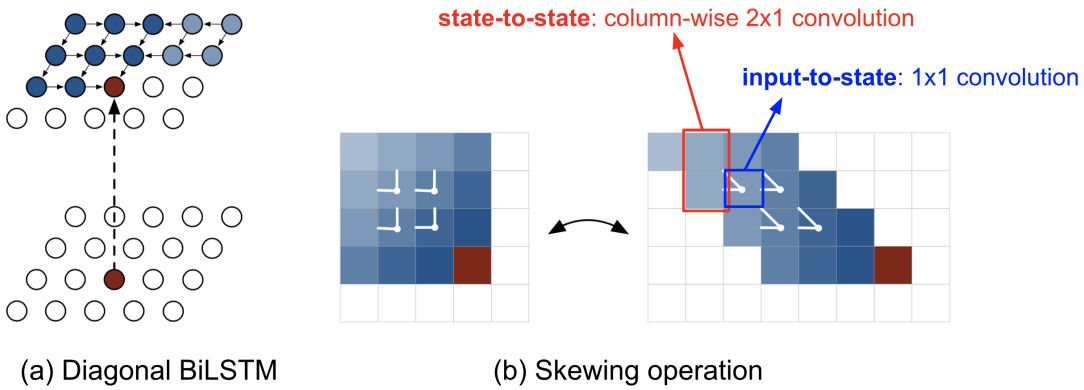


Figure 8: (a) PixelRNN with diagonal BiLSTM. (b) Skewing operation that offsets each row in the feature map by one with regards to the row above. (Image Source: van den Oord et al. [2016b])

$$
\begin{aligned}
[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] &= \sigma\left(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i\right) \\
\mathbf{c}_i &= \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i \\
\mathbf{h}_i &= \mathbf{o}_i \odot \tanh\left(\mathbf{c}_i\right)
\end{aligned}
\tag{7}
$$

where, x_i represents the i^{th} row of the input map with dimensions $h \times n \times 1$, where h is the height of the input, and n is the number of channels. The $\circledast$ symbol denotes the convolution operation, and the $\odot$ symbol represents the element-wise multiplication. The kernel weights K^{ss} and K^{is} are used for the state-to-state and input-to-state components, respectively. For the input-to-state component, the weights are precomputed. The gates o_i, f_i, and i_i refer to the output, forget, and input gates in the LSTM cell, respectively. For these gates, the logistic sigmoid function (σ) is used as the activation function. On the other hand, the content gate (g_i) uses the hyperbolic tangent ($tanh$) function as its activation. Each step in the process calculates the new state for an entire row of the input map simultaneously. This allows the model to efficiently update the state information while preserving the spatial dependencies across different rows of the input.

Diagonal BiLSTM layers offer the advantage of processing an unbounded context field; however, they can be computationally expensive due to the sequential dependency between states. A more efficient implementation employs multiple convolutional layers without pooling to define a bounded context box. This approach is known as PixelCNN. In PixelCNN, a convolution kernel is masked to prevent visibility of the future context, a technique similar to what is used in MADE. By using convolutional layers, PixelCNN can model the dependencies between pixels in a more computationally efficient manner, making it a faster alternative to Diagonal BiLSTM for capturing the context in an image.

3.2.3 WaveNet

WaveNet, developed by van den Oord et al. [2016a], is a model similar to PixelCNN but designed for $1-D$ audio signals instead of images. It comprises a stack of causal convolutions, which are convolution operations that maintain the temporal ordering of data. This ensures that predictions at a specific timestamp rely solely on past data, without any dependencies on future inputs. In PixelCNN, causal convolutions are implemented using masked convolution kernels. In contrast, WaveNet achieves causal convolutions by shifting the output a certain number of timestamps into the future, aligning the output with the last input element. This approach allows WaveNet to generate high-quality audio signals while adhering to the autoregressive property, making it a powerful model for various audio generation tasks.

A significant limitation of standard convolution layers is their relatively small receptive field. This means that the output depends on a limited range of input timesteps, which can be problematic for modeling long sequences where dependencies span hundreds or thousands of timesteps. To overcome this issue, WaveNet employs dilated convolutions. In dilated convolutions, the kernel is applied to an evenly-distributed subset of samples within a much larger receptive

field of the input. This approach allows the model to capture long-range dependencies effectively without increasing the number of parameters or computational complexity significantly. Dilated convolutions expand the receptive field exponentially with each layer, enabling the model to learn and represent dependencies over much longer sequences. As a result, WaveNet can effectively generate high-quality audio signals with complex long-term structures and dependencies.

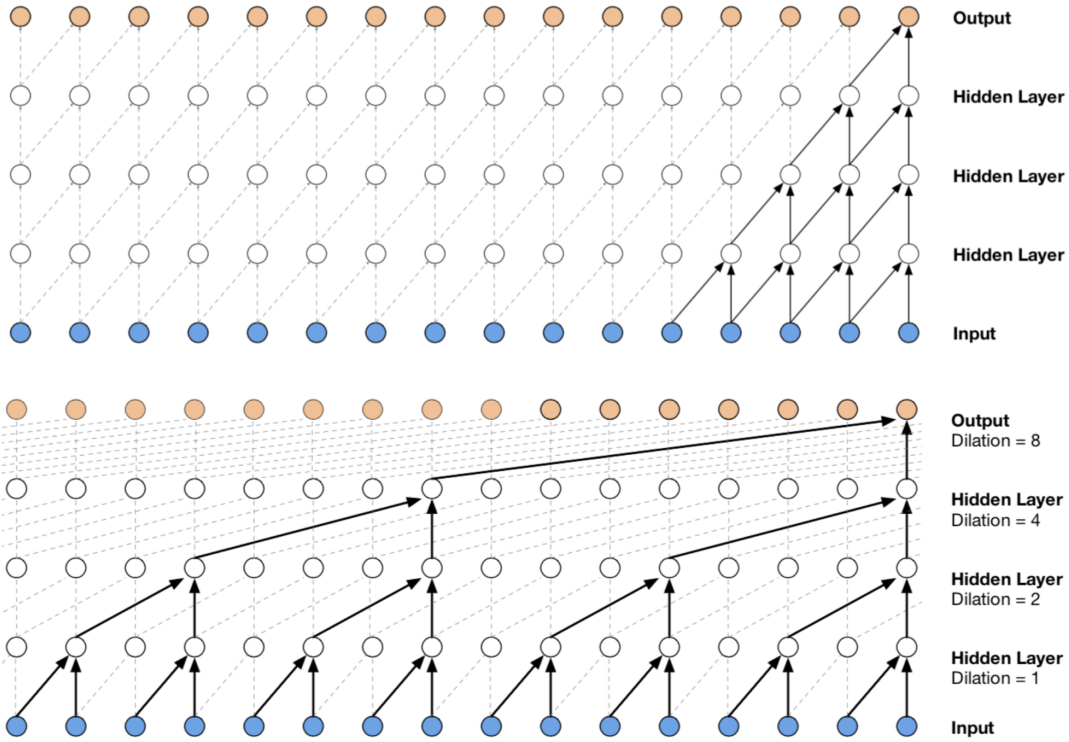


Figure 9: Visualization of WaveNet models with a stack of (top) causal convolution layers and (bottom) dilated convolution layers van den Oord et al. [2016a])

WaveNet employs gated activation units as the non-linear layer, which has been found to perform significantly better than ReLU for modeling $1-D$ audio data. The residual connection is applied after the gated activation. The gated activation unit formula for the l^th layer can be written as follows:

$$z_t{}^l = \tanh(W_f{}^l \circledast x_t{}^l) \odot \sigma(W_g{}^l \circledast x_t{}^l) \tag{8}$$

where W_f^l and W_g^l are the learnable convolution filter and gate weight matrices of the l^th layer, respectively; x_t^l represents the input at time step t for the l^th layer; $\tanh$ and σ are the hyperbolic tangents and sigmoid activation functions, respectively; and $\odot$ denotes element-wise multiplication.

The gated activation unit combines the output of two convolutions with different activation functions, allowing the model to capture complex patterns in the input data while maintaining a smooth output signal. This characteristic makes the gated activation unit particularly well-suited for modeling $1-D$ audio signals in WaveNet.

3.2.4 Masked Autoregressive Flow

Masked Autoregressive Flow Papamakarios et al. [2018] is a kind of normalizing flow in which the transformation layer is designed as an autoregressive neural network. MAF shares many similarities with Inverse Autoregressive Flow (IAF), which was introduced later. Both MAF and IAF focus on transforming a base distribution (usually a simple distribution like a Gaussian) into a more complex distribution by applying a series of invertible and differentiable transformations. The main difference between MAF and IAF lies in the direction of their autoregressive models.

MAF learns the forward autoregressive model, which means it models the transformation from the base distribution to the target distribution. In contrast, IAF learns the inverse autoregressive model, which models the transformation from the target distribution back to the base distribution. This difference in modeling direction leads to a trade-off between MAF and IAF. MAF has a fast density estimation but a slow sampling, as it requires a sequential process for generating samples. On the other hand, IAF has fast sampling but slow density estimation because it needs to sequentially compute the probability density of a given sample. Despite their differences, both MAF and IAF can be used for a variety of applications, such as density estimation, generative modeling, and variational inference. The choice between MAF and IAF depends on the specific problem requirements and the trade-off between sampling and density estimation.

Given two random variables, x and y, and a known probability density function $p_x(x)$, MAF aims to learn $p_y(y)$. MAF generates each dimension of y_i conditioned on the past dimensions $y_{1:i-1}$. More precisely, the conditional probability is an affine transformation of x_i, where the scale and shift terms are functions of the observed part of $y_{1:i-1}$.

- Data generation, producing a new sample y:

$$y_i = s_i(x_{1:i-1})x_i + t_i(x_{1:i-1}) \tag{9}$$

- Density estimation, given a known sample y:

$$p_y(y) = p_x(x)\Pi_{i=1}^{D} \tag{10}$$

4 Architectures

4.1 Standard Invertible Neural Networks

We start with the model from Ardizzone et al. In the analysis of complex physical systems, it is often the case that the parameters of interest, which we'll refer to as hidden parameters,denoted as x, cannot be directly measured. Instead, scientists rely on measurable quantities which can be generated from the study of x, denoted as y, to gain insights into these hidden parameters. The relationship between the hidden parameters x and the measurable quantities y is described by a mapping called the forward process,denoted as y = s(x). The model from Ardizzone title as Invertible Neural Networks (INNs) tries to estimate the complete posterior ($p(x|y)$) of the hidden parameters. Motivated by flow based networks INNs are also characterzied by three properties:

i The mapping from x to y is bijective

ii Forward and Inverse mapping are efficiently computable

iii The Jacobian of the mappings are tractable.

The forward process $s(x)$ leads to an inherent loss of information, to counteract this the model introduces additional latent variable at the output z. z stores the additional information about x that y was not able to store. The model also ensures that $p(z)$ is gaussian. Therefore we have a forward mapping $f(x) = [y, z]$ and an inverse mapping $x = f^{-1}(y, z) = g(y, z)$. The forward training inherently optimized the mapping $f(x)$ and implicitly determines it's inverse $g(y, z)$.
In other words, the INN takes information from y and uses it along with the distribution of z to generate the desired output x. It does this by applying a deterministic function that pushes or transforms the distribution of z into the space of x, while taking into account the information in y. The result is a mapping that connects the input y, the distribution of z, and the output x in a way that satisfies the desired posterior distribution $p(x|y)$. The model uses Unsupervised

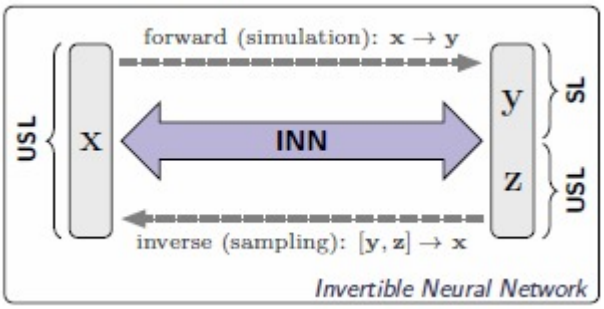


Figure 10: Invertible Neural Network

Loss for z and x, specifically MMD (Maximum Mean Discrepancy) and supervised loss for y such as square loss for regression and cross-entropy for categorical values.

The Problem

The model considers the problem with the given specification $x \in R^D$ (the phenomenon of interest), $y \in R^M$ (observed variable). The function $y = s(x)$ is already derived from the research work conducted and is used to generate the training samples. Note: The number of dimensions of the input and the output has to be the same for the INN architecture. Since there is always an information loss, $D < M$ and we introduce the variable z with $D - M$ dimension. The model aims at approximating $p(x|y)$ by a tractable model $q(x|y)$ and approximating the forward process $s(x)$ with $f(x)$. The functions are re-parametrized with a neural network with parameters θ:

$$x = g(y, z; \theta)$$

$$p(z) = N(z; 0; I_K)$$

$$f(x; \theta) \approx s(x)$$

$$[y, z] = f(x; \theta) = [f_y(x; \theta), f_z(x; \theta)] = g^{-1}(x; \theta)$$

A single Invertible Neural Network is used to implement both f and g and hence they share the same parameter θ. The dimensions are made to match on either side. In the case where M>D, x is padded with zeros to offset the difference. From these definitions, the model expresses the approximated complete posterior as:

$$q(\mathbf{x} = g(\mathbf{y}, \mathbf{z}; \theta) \mid \mathbf{y}) = p(\mathbf{z}) \left|J_{\mathbf{x}}\right|^{-1}, \quad J_{\mathbf{x}} = \det \left(\frac{\partial g(\mathbf{y}, \mathbf{z}; \theta)}{\partial [\mathbf{y}, \mathbf{z}]} \bigg|_{\mathbf{y}, f_{\mathbf{z}}(\mathbf{x})} \right)$$

Architecture

The model follows the architecture proposed in RealNVP. The scale and shit transformations are modelled using simple fully connected layers with a leaky ReLU activation. In the case, where the input dimension are very small, both the output and the input is padded with respective number of zeros. Permutation layers between the coupling layers are also added just like in RealNVP architecture.

Training

Training is done bidirectionally and it is made possible because of the Invertible Architecture. The backward and forward iterations are done in an alternating fashion. The model is trained using two different losses:

- $L_y(y_i, f_y(x_i))$: Penalizes the deviation between the simulated outcome $y_i = s(x)$ and the outcome predicted by the neural network $f_y(x_i)$. A squared loss is used for regression and a cross entropy loss is used for categorical variables.
- $L_z(q(y, z), p(y)p(z)$: The loss for latent variables penalizes the mismatch between the joint distribution between of network outputs $q(y = f_y(x), z = f_z(x)) = p(x)/|J_y z|$ and the product of marginal distributions of the simulation outcomes $p(y = s(x)) = p(x)/|J_s|$. During training, The gradients of L_z with respect to y are blocked to ensure the resulting updates only affect the prediction of z. Hence L_z ensures that the generated z follows the normal distribution and also ensure that y and z are independent. The loss is implemented by Maximum Mean Discrepancy (MMD) which gives us the oppurtunity to quantify the difference between two probability distributions using samples from the distribution and without any further information about the distributions itself. MMD assigns a point in a high-dimensional space to each distribution, capturing its properties. This is known as mean embedding. Once the mean embeddings of distributions P and Q are obtained, the MMD is calculated as the distance between these embeddings. The distance is defined using a specific kernel function. The kernel used for this specific architecture is given below:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = 1 / \left(1 + \left\|\left(\mathbf{x} - \mathbf{x}'\right) / h\right\|_2^2\right)$$

4.2 Conditional Invertibl Neural Network

A conditional invertible neural network is an extension of the INN concept that incorporates additional conditioning variables into the model. In a cINN, the transformation from the input to the output is conditioned on some additional information. This conditioning information can be any auxiliary data, such as class labels, control variables, or context information.

The inclusion of conditioning variables allows a cINN to model conditional distributions. It means that given the input and the conditioning information, a cINN can generate an output that depends on both of them. The conditioning variables provide additional context and guidance to the model, enabling it to learn complex dependencies and capture conditional variations in the data.

cINNs are particularly useful in tasks where the output is dependent on specific conditions or attributes. For example, image-to-image translation tasks, where an input image needs to be transformed into an output image while preserving certain attributes, can be effectively tackled using cINNs.

The architecture consists of two different networks, a conditional neural network, and a flow-based RealNVP invertible neural network. A RealNVP flow-based network applies a scale and shift transformation to multi-dimensional data where the scale and shift operation is a function of the first d dimensions, the scale and shift transformation is then applied to the remaining $D - d$ dimensions of the input data. The scale and shift transformation is easily invertible and the Jacobian can be calculated easily, hence satifying both the conditions of a invertible neural network. The pressure and saturation observation is given as input to the conditional neural networks, the outputs from each of the layer is concatenated with the output of each block of the RealNVP neural network which transforms the variable x to a latent variable z and is used to define the scale and shift operation applied to the outputs of the block. A Visual representation

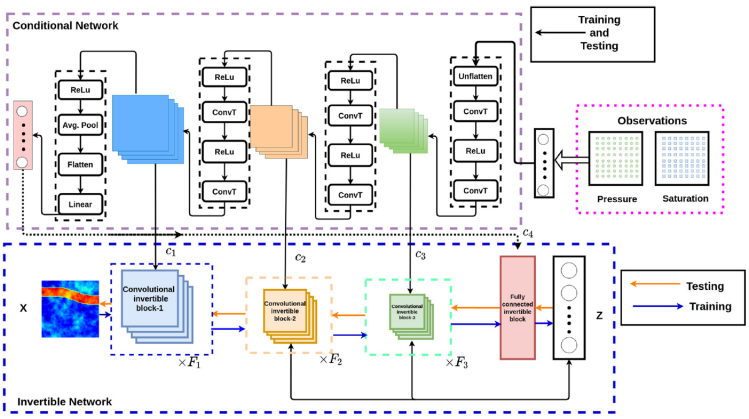


Figure 11: Architecture of a conditional invertible network

of the architecture is illustrated with an example: Comparing the example to our the standard INN, y is given by the pressure and saturation values, z is the latent variable which follows a normal distribution as discussed in the previous section, x is the hidden parameter.

4.3 Invertible Residual Networks

An invertible residual network is a modification to the traditional ResNet design that allows it to compete with both novel image classification algorithms and flow-based generative models in terms of accuracy by merely altering the normalisation technique of conventional ResNets. Behrmann et al. [2019] adopts a lenient Lipschitz constraint on the residual branches of the stringent architectural limitations that come from coupling layers and autoregressive models. This restriction infact allows for the iterative determination of the Jacobian determinant and the inverse of the model.

According to the following theorem, a straightforward requirement is all that is required to make the dynamics solvable, which makes the ResNet invertible:

Let $x \in \mathbb{R}^d$ and $F(x)$ denote a ResNet with blocks $f(x) = x + g(x)$. Then, the ResNet $F(x)$ is invertible if

$$Lip(g(x)) < 1$$

where $Lip(g(x))$ is the Lipschitz-constant of $g(x)$. Specifically, $g(x)$ is a residual block, which is decomposed into a sequence of invertible functions:

$$g(x) = f_1(f_2(f_3(x)))$$

where each function $f_i(x)$ is an invertible function that can be easily inverted. In other words, given the output $y = g(x)$, we can compute the input $x = f_3^{-1}(f_2^{-1}(f_1^{-1}(y)))$.The full invertible residual block can then be defined as:

$$h(x) = x + f_3^{-1}(f_2^{-1}(f_1^{-1}(g(x))))$$

Here, $h(x)$ is the output of the invertible residual block with input x. If the above condition is satisfied, then it holds the following equation with $Lip(g(x)) = L$;

$$Lip(F) \leq 1 + L \text{ and } Lip(F^{-1}) \leq \frac{1}{1-L}$$

Thus, in contrast to previous approaches, the i-ResNet block has a Lipschitz bound for both forward and inverse passes, whereas other approaches do not by design. As a result, i-ResNets might be a promising direction for stability-critical applications.

Forward pass of an invertible ResNet with Lipschitz constraint and log-determinant approximation

Input: Data point x, network F, residual block g, number of power series terms n

Algorithm 1 Forward pass of an invertible ResNet

Forward pass of an invertible ResNet with Lipschitz constraint and log-determinant approximation

Input : Data point x, network F, residual block g, number of power series terms n
Output :
foreach *Residual block* **do**
  Lip constraint: $\hat{W}_j := \text{SN}(W_j, x)$ for linear Layer W_j
  Draw v from $\mathcal{N}(0, I)$
  $w^T := v^T$, $\ln \det := 0$

  for $k = 1$ *to* n **do**
    $w^T := w^T Jg$ (vector-Jacobian product)
    $\ln \det := \ln \det + (-1)^{k+1} \frac{w^T v}{k}$
  end
end

4.4 Autoencoders

Autoencoders are unsupervised artificial neural networks designed to learn encoded representations of data and generate input data as closely as possible from the learned encoded representations. The output of an autoencoder is its prediction for the input.

The architecture of a basic autoencoder is shown in Figure 12. It consists of an encoder and a decoder. The input data, denoted as x, undergoes an encoding process through an affine transformation defined by W_h followed by a squashing function, resulting in an intermediate hidden layer h. This hidden layer is then subjected to the decoder, which applies another affine transformation defined by W_x followed by another squashing function. The output $\hat{x}$ represents the reconstructed input.

Mathematically, the autoencoder can be represented as:

$$h = f(W_h x + b_h) \quad \text{(encoding)}$$

$$\hat{x} = g(W_x h + b_x) \quad \text{(decoding)}$$

where f and g represent the activation functions, and b_h and b_x are the bias terms.

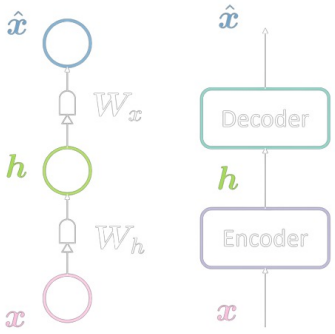


Figure 12: Autoencoder Architecture

Autoencoders find applications in anomaly detection and image denoising. They can reconstruct data that lies on a manifold and are insensitive to perturbations outside that manifold. Autoencoders can also serve as image compressors by using an intermediate dimensionality d lower than the input dimensionality n, where the encoder acts as a compressor and the hidden representations capture most of the input information while occupying less space.

For the reconstruction loss, the average per-sample loss is calculated as:

$$L = \frac{1}{m} \sum_{j=1}^{m} \ell(x^{(j)}, x^{(\hat{j})})$$

The choice of loss function depends on the nature of the input. For categorical input, the Cross-Entropy loss is used:

$$\ell(x, \hat{x}) = -\sum_{i=1}^{n} [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)]$$

For real-valued input, the Mean Squared Error (MSE) loss is employed:

$$\ell(x, \hat{x}) = \frac{1}{2} \|x - \hat{x}\|^2$$

This formulation allows the autoencoder to learn and minimize the reconstruction error during training, improving its ability to generate accurate representations of the input data.

4.5 Autoregressive flows

We will comprehend the fundamental building blocks of models based on autoregressive flow, including Masked Autoregressive Flow (MAF) and Inverse Autoregressive Flow (IAF). We first cover the change of variable rules for 1D and higher Dimensional scenarios to better understand the computational cost of Normalising Flows before talking about models such as Masked Autoregressive Flows, Inverse Autoregressive Flows, etc.

The change of variable rule can be stated by beginning with a foundational distribution, let's call it u, and employing a bijective transformation, denoted as ϕ. This transformation is used to create a new variable x, where x is obtained by applying the bijective transformation ϕ to u, expressed mathematically as $x = \phi(u)$. With these components, the change of variables rule can be succinctly written as such:

$$\log p_X(x) = \log p_U(u) - \log \left| \frac{\partial \phi(u)}{\partial u} \right| \tag{11}$$

In Normalizing Flows, we employ a sequence of bijective transformations, or "bijectors", by chaining them together to progressively morph a basic distribution into a more intricate one. To illustrate, if we have K bijective transformations, we can apply them in succession to alter our initial distribution (for instance, denoted as u_0) until it transforms into the complex distribution we desire, represented as x.

$$\phi = \phi_k \cdot \phi_{k-1} \cdot \phi_{k-2} \ldots \phi_1 \tag{12}$$

$$where u_k = \phi_k \left(u_{k-1}\right); k = 1, 2, \ldots K \text{ base distribution } u_0, \& x = u_K$$

For a sequence of K transformations, we can adapt Equation 11 accordingly as follows:

$$\log p_X(x) = \log p_U(u) - \sum_{k=1}^{K} \log \left| \frac{\partial \phi_k \left(u_k\right)}{\partial u_{k-1}} \right| \tag{13}$$

A significant challenge in implementing Normalizing Flows is the computational burden associated with calculating the log-determinant of the Jacobian matrix, often abbreviated as log-det-Jacobian Bhattacharyya [2021]. When computing the determinant of an nn Jacobian matrix through methods such as Gaussian Elimination, the runtime complexity is cubic in the order of the matrix, or $O(n^3)$, which can be computationally intensive for large matrices. The selection of how to model this conditional density is flexible, and there has been a variety of propositions ranging from basic univariate Gaussian models to more sophisticated neural networks. Let's delve into some of the prominent options!

Masked Autoregressive Flows (MAF) is an approach to modeling multivariate distributions by breaking them down into a product of one-dimensional Gaussian conditionals, leveraging the chain rule of probability as outlined by Papamakarios et al. [2018]. In a similar vein, Inverse Autoregressive Flows (IAF) deconstruct the latent distribution into simpler forms as discussed by Kingma et al. [2017]. To create models that are asymptotically invertible, MAF

integrates standard feed-forward neural networks for the reverse direction, drawing inspiration from Parallel WaveNets as developed by van den Oord et al. [2018]. This incorporation is critical as it enables both forward and backward mappings with efficiency.

$$\mathcal{L}(\mathbf{y}, \mathbf{z}) = \frac{1}{2} \cdot \left(\frac{1}{\sigma^2} \cdot (\mathbf{y} - \mathbf{y}_{\text{gt}})^2 + \mathbf{z}^2 \right) - \log \left| \det J_{\mathbf{x} \mapsto [\mathbf{y}, \mathbf{z}]} \right|, \tag{14}$$

During training, a specialized loss function, which includes a term based on Equation 14 and an additional cycle loss, is used to ensure the network learns an effective mapping. The used cycle loss is:

$$\mathcal{L}(\mathbf{y}, \mathbf{z}, \hat{\mathbf{x}}) = \frac{1}{2} \cdot \left(\frac{1}{\sigma^2} \cdot (\mathbf{y} - \mathbf{y}_{\text{gt}})^2 + \mathbf{z}^2 \right) - \log \left| \det J_{\mathbf{x} \mapsto [\mathbf{y}, \mathbf{z}]} \right| + \alpha \cdot (\mathbf{x} - \hat{\mathbf{x}})^2 \tag{15}$$

The cycle loss is particularly important as it ensures that the transformations are consistent in both directions. This means that if you transform a point from the original space to the latent space and back, it should end up near where it started, and vice versa. This consistency is crucial for ensuring that the learned transformations are meaningful representations of the data. MAF and IAF have been particularly useful in scenarios where complex, high-dimensional data distributions are involved. They have applications in generative modeling, variational inference, and other areas where efficiently computing the likelihood of data points is important.

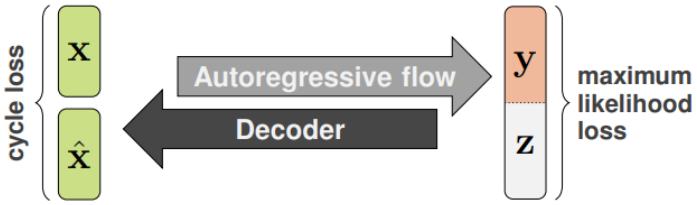


Figure 13: Autoregressive Flow

However, it's important to note that while MAF can be more efficient in density estimation due to the decomposition, it might be slower at sampling compared to other normalizing flows. This trade-off between density estimation and sampling efficiency is essential to consider depending on the application at hand. Additionally, the choice between MAF and IAF might also depend on the specific characteristics of the data and problem requirements.

4.6 Invertible Autoencoders

The unsupervised image-to-image translation is a challenging task that seeks to discover a transformation between a source image domain (A) and a target image domain (B) without the availability of aligned image pairs during training. This task is inherently ill-posed as it entails deducing the joint probability distribution using only marginal distributions. State-of-the-art techniques such as CycleGAN, as highlighted by Zhu et al. [2020], often resort to simultaneous learning of two mappings, $FAB : A \text{ß} B$ and $FBA : B \text{ß} A$, by incorporating a cycle consistency constraint. Essentially, this implies that $FAB(FBA(B))$ should approximate B and conversely, $FBA(FAB(A))$ should approximate A. The cycle consistency is vital as it ensures that mutual information between the input and the translated images is retained. However, it doesn't specifically mandate that FBA should act as the inverse of FAB.

In this regard, an innovative deep learning architecture termed the Invertible Autoencoder (InvAuto) Teng et al. [2018] is introduced to explicitly enforce this inverse relationship. This is achieved by architecturally constraining the encoder to be the reverse of the decoder, with the layers corresponding to each other executing inverse transformations, and sharing their parameters. Furthermore, the mappings are restricted to be orthonormal. Remarkably, this architecture culminates in a reduction of the number of parameters that need to be learned, in some cases by up to 50%. Empirical results on standard datasets are presented, showcasing the efficacy of InvAuto in image translation tasks, where it exhibits cutting-edge performance. Additionally, InvAuto's versatility is demonstrated in a domain adaptation scenario involving the conversion of road videos. High-quality video conversions are achieved, and the effectiveness of NVIDIA's PilotNet,

a neural network designed for autonomous driving, is demonstrated when trained on real road videos and tested on the transformed ones using InvAuto. This underscores InvAuto's potential in practical applications, especially where domain adaptation is crucial.

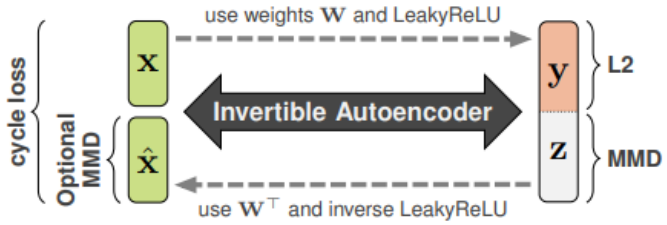


Figure 14: Invertible Autoencoders

The model introduced by Teng et al. [2018] employs invertible non-linear functions coupled with weight matrices that are orthogonal, to effectively attain invertibility. Initially, the weight matrices are populated with random values, but as the training progresses, they evolve to become orthogonal. This convergence is achieved through the incorporation of a cycle loss in the training process. The given cycle loss is:

$$\mathcal{L}(\mathbf{y}, \mathbf{z}, \hat{\mathbf{x}}) = \mathrm{L2}(\mathbf{y}) + \alpha \cdot \mathrm{MMD}(\mathbf{z}) + \beta \cdot (\mathbf{x} - \hat{\mathbf{x}})^2 \tag{16}$$

In a more detailed context, the orthogonal nature of the weight matrices ensures that the matrix inversion is computationally efficient since the inverse of an orthogonal matrix is simply its transpose. The orthogonal constraint can be enforced during training by regularizing the weights. For an orthogonal matrix W, the following property holds:

$$W^T W = I$$

where W^T is the transpose of the weight matrix W, and I is the identity matrix. During training, a regularization term can be added to the loss function to enforce this property, such as:

$$L = L_{\text{original}} + \lambda \|W^T W - I\|^2$$

where L is the total loss, L_{original} is the original loss without the regularization, λ is a hyperparameter controlling the strength of the regularization, and $\|.\|$ denotes the Frobenius norm.

This approach not only ensures the efficient invertibility of the network but also aids in preserving the complex structure and information content of the data through the invertible non-linearities. Such a design can be especially beneficial in applications like generative modeling or domain adaptation where invertibility and efficient computation are essential.

4.7 Conditional Variational Autoencoders

The Variational Autoencoder (VAE) Kingma and Welling [2022] is a type of generative model that utilizes directed graphical modeling and has achieved impressive outcomes, placing it among the top methods in generative modeling. The VAE operates under the assumption that data is produced by a certain stochastic process, which includes a continuous latent variable denoted as z. The model presumes that z originates from a specific prior distribution, denoted as $P_\theta(z)$, and the data, represented as X, is produced from a certain conditional distribution, denoted as $P_\theta(X|Z)$. In this context, z is occasionally referred to as the hidden representation of data X. The architecture of a Variational Autoencoder, like all autoencoder structures, consists of two main components: an encoder and a decoder. The encoder's function is to learn $q_\phi(z|x)$, which essentially means learning the hidden representation of data X or translating X into its hidden form. This is often referred to as a probabilistic encoder. The decoder's role, on the other hand, is to learn $P_\theta(X|z)$, which involves translating the hidden representation back into the input space. The structure of this graphical model can be visualized in the diagram below. The model is trained to minimize the objective function:

$$\underbrace{-\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})]}_{\text{reconstruction error}} + \underbrace{\mathrm{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right)}_{\text{regularization}} \tag{17}$$

The initial component of this loss function is the reconstruction loss, also known as the expected negative log-likelihood of a data point. This expectation is determined by taking a small number of samples based on the encoder's distribution over the representations.

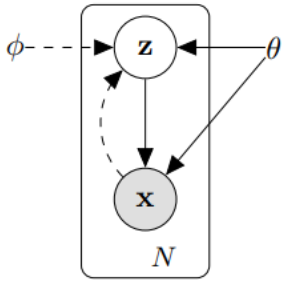


Figure 1: The type of directed graphical model under consideration. Solid lines denote the generative model $p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, dashed lines denote the variational approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ to the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The variational parameters $\boldsymbol{\phi}$ are learned jointly with the generative model parameters $\boldsymbol{\theta}$.

Figure 15: Structure of Graphical Model
Source: Autoencoding Variational Bayes

The purpose of this term is to guide the decoder to learn how to reconstitute the data using samples from the latent distribution. A high reconstruction error is an indicator that the decoder is not capable of accurately recreating the data. The second component of this loss function is the Kullback-Leibler (KL) divergence between the encoder's distribution, represented as $q_\phi(z|x)$, and the prior distribution, $p(z)$. The KL divergence quantifies the information loss when q is used to approximate the prior distribution over the latent variable z. This component encourages the values of the encoder's distribution to align with a Gaussian distribution. During generation, samples from $N(0, 1)$ is simply fed into the decoder.

While the Variational Autoencoder (VAE) has been outlined here in a concise manner due to its relevance, it isn't the primary focus. One of the challenges with using a VAE for data generation is the lack of control over the type of data it produces. For instance, if a VAE is trained on the MNIST dataset and then used to generate images by inputting $Z \sim N(0, 1)$ into the decoder, it will produce random digits. While with proper training the images will be of high quality, there's no way to specify which digit the VAE should generate. For example, we can't instruct the VAE to generate an image of the digit $'2'$.

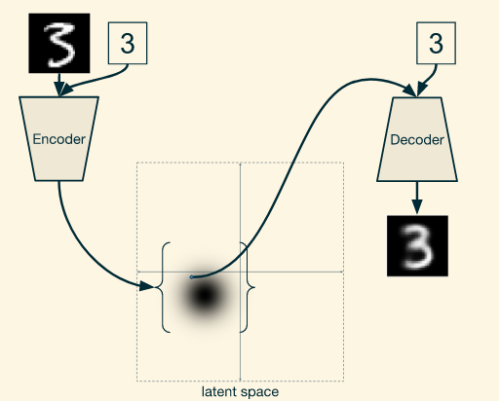


Figure 16: Visual Representation Task in Conditional VAE
Source: cvae

To achieve this level of control, a slight modification to the VAE architecture Sohn et al. [2015] is required. Suppose we want our generative model to produce output X (an image) given an input Y (the label of the image). The VAE process would then be altered as follows: given an observation y, z is drawn from the prior distribution $P_\theta(z|y)$, and the output x is generated from the distribution $P_\theta(x|y,z)$. It's worth noting that for a simple VAE, the prior is $P_\theta(z)$, and the output is generated by $P_\theta(x|z)$. In this case, the encoder's role is to learn $q_\phi(z|x,y)$, which equates to learning the hidden representation of data X, or transforming X into its hidden form, given the condition y. Correspondingly, the decoder's task is to learn $P_\theta(X|z,y)$, which involves translating the hidden representation back into the input space, given the condition 'y'.

So Variational Autoencoders (VAEs), introduced by Kingma and Welling [2022], employ a Bayesian methodology, making them particularly well-suited for tasks involving distribution predictions. Given our interest in conditional distributions and the fact that it eases the training process, our attention is primarily centered on the conditional VAE model, which was put forward by Sohn et al. [2015]. This model incorporates a loss function that significantly influences its performance and effectiveness.

$$\mathcal{L}\left(\mu_z, \sigma_z, \hat{\mathrm{x}}\right) = (\mathrm{x} - \hat{\mathrm{x}})^2 - \frac{1}{2}\alpha \cdot \left(1 + \log \sigma_z - \mu_z^2 - \sigma_z\right) \tag{18}$$

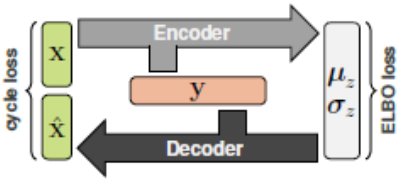


Figure 17: Conditional VAE

5 Applications

There are numerous uses for invertible neural networks in machine learning. They have been used to research deep classifier representations, comprehend the root of adversarial cases, learn transition operators for MCMC, develop generative models that are directly trainable by maximum likelihood, and carry out approximation inference. Some of the applications go as:

5.1 Classification using iNNs

Typically neural networks for classification are not invertible. For iNNs the classifier is separated into two parts $f = f2 \cdot f1$. Feature extraction vector $z = f1(x)$ and classification vector $y = f2(z)$, where $f2$ is usually the softmax function. We say the classifier is invertible when $f1$ is invertible.

It is possible to flip standard ResNet architecture, enabling the use of the same model for generation, density estimation, and classification. Behrmann et al. [2019]'s iResNets can be created by merely altering the Resnet's default normalization technique by taking advantage of the notion of ResNets as an Euler discretization of ODEs. This method only needs a Lipschitz constant smaller than one for each residual block, allowing unrestricted architectures for each block. The resultant model outperforms both state of the art image classifiers and flow-based generative models, which has never before been accomplished with a single architecture on classifying MNIST, CIFAR10 & CIFAR100 images.

5.2 Image Reconstruction

The majority of modern imaging systems are computational in nature and use a reconstruction technique to construct an image from a set of observations. To reduce representation error in generative model-constrained image reconstruction methods, various strategies have been proposed. Image reconstruction is modeled as an inverse problem:

$$f = Au + n \tag{19}$$

Where f is the set of measurements that may be undersampled, u is the objective image to be reconstructed, A is an approximation representing the imaging system, and n is the additive noise with the measurements. Kelkar et al. [2021] proposes the use of a framework for image reconstruction that is developed in the latent space of generative models based on invertible Neural networks for overcoming the reconstruction errors associated with Generative Adversarial Networks, when the solution exceeds the range of GAN. The usage of a multiscale architecture, as described by Dinh et al. [2016], produces a compressible latent space by incorporating latent space vector at various locations throughout the network. The compressibility of the latent space of Invertible Neural Networks was used to build a regularisation method for inverse applications.

5.3 Image Steganography

ISN: Invertible Steganography Networks by Lu et al. [2021] were one of the first ones to achieve considerable accuracy for image hiding. ISNs utilize the image embedding and treat steganography and recovery of hidden images as a pair of inverse problems by introducing forward and backward propagation operations of a single invertible network. As a result of shared parameters of the single ISN architecture and an increased number of channels for the hidden image branch,ISN achieves state-of-the-art in both visual and quantitative comparisons.

RIIS in Xu et al. [2022b] is one of the novel flow-based frameworks for robust invertible image steganography. RIIS prevents the container image from distortions such as Poisson noise, Gaussian noise, and JPEG compression by using the proposed CANP and CEM modules unlike earlier ISN implementations.

Ren et al. [2022]'s Steg-cINN formulates data hiding as an image colorization problem, in which the data is binarized and further mapped into a color information for a gray-scale host image. A conditional invertible neural network is used for the revealing part, which uses gray-scale image prior to guide the color generation and as a result performs data hiding in a secure way.

A more recent work by Guan et al. [2023], makes the concept of image concealing and revealing fully coupled and reversible. The invertible hiding neural network (IHNN), which is extremely adaptable and can be cascaded as many times as necessary to achieve the hiding of numerous pictures,according to the proposed DeepMIH architecture. Steganography is then guided by an importance map based on prior outcomes. To ensure that no secret information is revealed in the low-frequency sub-bands, the authors have suggested a new low-frequency wavelet loss.

5.4 Image Denoising and Deblurring

The first practical denoising investigation using invertible networks was conducted by InvDN as in Liu et al. [2021]. InvDN has the ability to create and remove noise. It converts the input into a latent representation with noise and a low-resolution clean image and swaps out the noisy latent representation for another one sampled from a prior distribution restoring the clean image. For the SIDD dataset, InvDN's denoising performance is superior to that of all other competitive methods currently in use.

Wang et al. [2021] proposes a unique codec-unified architecture with invertible network modules for picture deblurring. A wavelet invertible network is used for deblurring purposes . In order to build the multi-resolution correlations between the blur and sharp features, a U-shaped multilevel invertible network (UML-IN), inspired by U-Net is developed.

A sparsity-driven denoising network and the lifting scheme in wavelet theory served as inspiration for DnINN, which consists of an invertible neural network called LINN that is used to remove noise from the transform coefficients given provided in the Hu [2021]. While using only one-fourth of the learnable parameters, the DnINN approach produces results comparable to those of the DnCNN method.

Building upon LINNs, wavelet-inspired invertible network (WINNet) combines the merits of the wavelet-based approaches and learning-based approaches. The proposed WINNet by Huang and Dragotti [2022] consists of K -scale of lifting invertible neural networks (LINNs) and sparsity-driven denoising layers together with a noise estimation network. Simulation results show that the proposed method is highly interpretable and has strong generalization ability to unseen noise levels.

5.5 iNNs for AstroPhysics

One of the most popular types of invertible neural networks in this field is the convolutional INN. a beautiful method due to their probability-preserving bijective mapping characteristics. Current approaches in astrophysics use the time-consuming Markov Chain Monte Carlo (MCMC) sampling to infer the posterior probability. In this situation, cINNs stand up and give statisticians a much better tool that learns distributions and produces findings on par with those of current techniques in less time.

Kang et al. [2022] introduces a novel method that couples a conditional invertible neural network (cINN) with the WARPFIELD-emission predictor (WARPFIELD-EMP) to estimate the physical properties of star-forming regions from spectral observations : A cINN that predicts the posterior distribution of seven physical parameters like cloud mass, star formation efficiency, cloud density etc from the luminosity of 12 optical emission lines.

Exoplanetary science has entered the era of characterization, where new observations are used to infer physical and chemical properties of exoplanets. Haldemann et al. [2022]'s cINN following the framework for easily invertible architectures (FreIA) is trained on a database of $5.6 \cdot 106$ internal structure models of exoplanets to recover the inverse mapping between internal structure parameters and observable features like planetary mass, planetary radius etc.

cINNs have also been used for Inference of cosmic-ray source properties from cosmic-ray observations on Earth using extensive astrophysical simulations as carried out by Bister et al. [2022]. A more interesting use of INNs is done by Liang et al. [2022]in the Modelling of internal structure of differentiated asteroids where unlike previously implemented gravity based methods derivation of the heterogeneous mass distribution of an asteroid is generalized as an inverse problem.

5.6 Design of Wind Turbine Foils

Aerodynamics starts with the airfoil design challenge, in which an engineer looks for a shape with required performance characteristics. More efficient blade designs can be found by controlling the airfoil cross-sectional shapes simultaneously with the bulk blade twist and chord distributions.

INN surrogate models by Glaws et al. [2022] are capable of forward prediction of aerodynamic and structural quantities for a given airfoil shape as well as inverse recovery of airfoil shapes with specified aerodynamic and structural characteristics offering roughly a 100 times speed-up compared to iterative design procedure. Noever-Castelos et al. [2021] uses INNs to model rotor blade cross sections to match given cross-sectional parameters and a full Bayesian posterior that allows among other things the determination of confidence intervals.

Building upon similar lines, Jasa et al. [2022] trains INNs on high-fidelity airfoil aerodynamic data to a turbine design framework to enable the design of airfoil cross sections within a larger blade design problem enabling the inclusion of high-fidelity aerodynamic data earlier in the design process, reducing cycle time and increasing certainty in the performance of the optimal design.

5.7 Graph Prediction

Invertible graph neural network (iGNN) uses a deep generative model to infer the input attributes X given an outcome response Y by recasting the inverse prediction problem on graphs as a conditional generative task. Numerous real-world applications include molecular designs, power outage analysis and traffic flow analysis.

Xu et al. [2022a] proposes an iGNN architecture that takes a two-step approach. On graph data, for v in {1,2,3,4,5} as shown in Fig 18, the inverse prediction problem is to generate the conditional distribution X|Y , which is a one-to-many mapping from Y to X (indicated by dash lines). Now Y is first mapped to an intermediate feature H|Y (one-to-many) and then through an invertible neural network from X|H to X|Y (one-to-one).

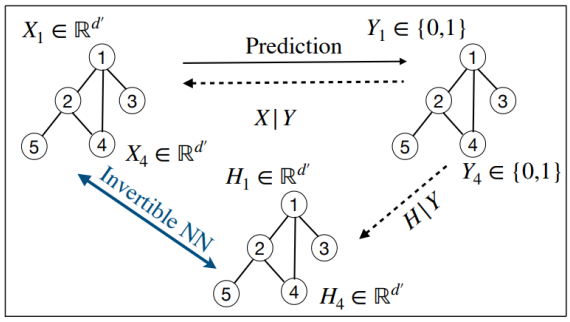


Figure 18: Illustration of iGNN model applied to graph data.

The major part of the model capacity is in the H-to-X sub-network, which is an invertible flow model based on the ResNet framework.

Experiments show that iGNNs give better results than already implemented cINN-Flow based models for the solar ramping and traffic flow anomaly detection tasks, where the generated data distribution by iGNN resembles the ground truth more accurately.

5.8 Inverse Molecule Design

Here the main topic of study is the logarithm of partition coefficient (logP) of a molecule, a measurement of its lipophilic effectiveness- one of the indications of drug-likeness. By addressing inverse molecule creation as a regression problem, the work in Hu [2021] attempts to investigate the bidirectional correlation between molecules and their chemical properties.

This model by Hu [2021] could be trained once and then sampled any number of times and for any chemical property values without the need for retraining inferring the molecules given the chemical properties, in contrast to the prior methods in the literature that could only optimize one molecule for one chemical property value at a time. With the help of two popular flow models, nonlinear independent components estimation (NICE) and real non-volume preserving (RealNVP), dimension partitioning and coupling layers transform one part of the input at a time and allow for both analytic forward and inverse mappings. As a result the model could generalize well from the training data and learn bidirectional correspondence between molecules and their chemical properties, thereby offering the ability to sample any number of molecules from any y values.

5.9 Anomaly Detection

INNs have been proposed as a method for anomaly detection because they can be used to model normal data distributions, and deviations from this distribution can be identified as anomalies. There are several different types of INNs that have been used for anomaly detection, including normalizing flow models, such as Real NVP and Glow, and autoregressive models, such as PixelCNN and WaveNet. These models have been applied to various types of data, including images, time series, and text. Several papers have proposed the use of INNs for anomaly detection.

Cho et al. [2022] proposes using Normalizing Flows (NF) to model the distribution of normal features in an unsupervised way for detecting anomalies in videos, utilizing static and dynamic features obtained from implicit two-path Autoencoder (ITAE). ITAE is used to capture representative information of normal scenes without a pre-trained network. The complex

normal distribution is handled by modeling the appearance and motion normality using the latent features of ITAE and an NF model with a manageable likelihood.

Another work, Turowski et al. [2022] suggests a new method for improving anomaly detection in energy time series by utilizing their latent space representation. This representation improves anomaly detection by clearly distinguishing between time series with and without anomalies. A representation of time series in the latent space is created, by generating a mapping from the original space to the latent space. To ensure this mapping is accurate, authors used a Variational Autoencoder or an Invertible Neural Network that takes into account typical patterns in energy time series. The difference between the two methods is essentially that Variational Autoencoders use a separately trained encoder and decoder to realize the mapping and reconstruct the original representation, while INNs only use a single bijective mapping for both encoding and decoding.

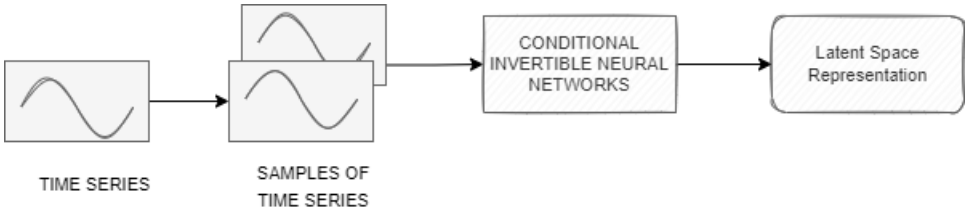


Figure 19: Illustration of cINNs for creating latent space representation of Time Series.

Gudovskiy et al. [2022] presents a model called CFLOW-AD, which utilizes a conditional normalizing flow framework for anomaly detection with the ability to locate anomalous regions. The model includes a pre-trained encoder and multiple generative decoders that estimate the likelihood of the encoded features at different scales. The Anomaly Detection model learns the distribution of normal image patches, represented by a probability density function D(z). It then trains a model that can convert this distribution to a Gaussian distribution with a different density function, D(y). Using a threshold based on the distance from the mean of this new distribution, the model is able to identify patches that are not part of the original distribution (outliers) by comparing their density to that of the normal (in-distribution) patches. The authors suggest utilizing conditional normalizing flows as decoders to improve the efficiency of CFLOW-AD for handling multi-scale feature maps in localization tasks.

In Schirrmeister et al. [2020], a method based on Generative Flow(Glow) model is suggested for detecting out-of-distribution samples. This involves using a multi-scale model like Glow and shows that low-level features are mostly captured at early scales, therefore using only the likelihood contribution of the final scale is effective for detecting high-level feature differences between in-distribution and out-of-distribution samples.

5.10 Neural Style Transfer

Style transfer using invertible neural networks is a technique for transferring the style of one image to another image while preserving the content of the original image. An invertible neural network, such as a normalizing flow, can be trained to map the style of one image to the content of another image without losing information of hidden layers by addressing inverse problems.

The methodology presented in Lan et al. [2021] employs invertible neural networks (INNs) to accomplish unpaired stain-style transfer. The INNs are trained to learn the correspondence between the style and content of the images. The authors propose to augment the feature representation of the images by utilizing channel attention, a technique that permits the model to assign various levels of significance to different channels of the image, thus enhancing the capability to capture style information. Furthermore, the authors advocate for the use of long-range residual connections to retain a greater degree of the original information as these connections are between layers that are farther apart in the network. A loss function was also proposed to consider both pixel-level and feature-level differences while training the INN, allowing the model to strike a balance between preserving the original image and transferring the style information.

An et al. [2021] has developed a new style transfer framework called ArtFlow that addresses the issue of "Content Leak" in current style transfer techniques through a projection-transfer-reversion method. This process involves using a reversible neural flow model called the Projection Flow Network (PFN), which uses a series of invertible neural network layers called "projection layers" to transform the data and can be trained using maximum likelihood estimation for extracting deep features from the content and style images. Then, the content and style features are transferred to create a stylized image, and finally reconstructing the stylized image using the reverse propagation of PFN.

Another work, Huang and Belongie [2017] proposed a method for real-time style transfer using an invertible neural network, called Adaptive Instance Normalization (AdaIN). The network is trained to learn the underlying structure of the image data, and can then be used to generate new images that have the same content as an input image but with a different style.

5.11 Class-Incremental Learning

In recent years, Invertible neural networks have been used for class incremental learning. This approach aims to address the problem of catastrophic forgetting, which occurs when a neural network is trained on a new task and forgets how to perform the previous task. One popular approach is to use an invertible neural network, which can be used to generate new samples from the previous task, and thus prevent catastrophic forgetting.

Guillaume et al. [2019] utilized Invertible Networks for classification by training each network to output a low-magnitude vector when given data from a specific class. Once training is complete, the weights of these networks will not be changed when new classes are added. During inference, the class of a sample is determined by identifying the network that produces the smallest output.

Hocquet et al. [2021] proposes a modification to the one-versus-all invertible neural network method, a prevalent technique in class-incremental learning, aimed at decreasing memory consumption. The experimentation was carried out on the CIFAR-100 dataset, where classes were incrementally learned. Results demonstrate that the proposed approach preserves comparable accuracy while decreasing memory requirements by a maximum of five-fold relative to the traditional method.

5.12 Approximation of Manifolds and Densities

Invertible neural networks are being used for manifold and density approximation.

Manifold Approximation :

Given samples of the mapping, the problem is to construct an approximation such that

1 - The resulted approximation is continuous.

2 - The approximation error can be bounded, based on the manifold properties and the function class.

Manifolds and **Density Approximation :**

Let $p(\cdot)$ be a density on a n-dimensional Euclidean space χ. The task of density estimation is to estimate $p(\cdot)$ based on a set of independently and identically distributed data points $\{x_i\}_{i=1}^{N}$ drawn from this density.

Normalizing flows-based density estimators describe x as an invertible transformation of a latent variable z with known density, which is composed of some functions whose Jacobian is easy to calculate. Neural networks then learn the parameters of these component functions.

References

Jie An, Siyu Huang, Yibing Song, Dejing Dou, Wei Liu, and Jiebo Luo. ArtFlow: Unbiased image style transfer via reversible neural flows. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/behrmann19a.html.

Saptashwa Bhattacharyya. Understand implement masked autoregressive flow with tensorflow. 2021.

Teresa Bister, Martin Erdmann, Ullrich Köthe, and Josina Schulte. Inference of cosmic-ray source properties by conditional invertible neural networks. *The European Physical Journal C*, 82(2), February 2022. doi: 10.1140/epjc/s10052-022-10138-x. URL https://doi.org/10.1140/epjc/s10052-022-10138-x.

Myeongah Cho, Taeoh Kim, Woo Jin Kim, Suhwan Cho, and Sangyoun Lee. Unsupervised video anomaly detection via normalizing flows with implicit latent features. *Pattern Recognit.*, 129:108703, September 2022.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *ArXiv*, abs/1605.08803, 2016.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation, 2015.

Andrew Glaws, Ryan N. King, Ganesh Vijayakumar, and Shreyas Ananthan. Invertible neural networks for airfoil design. *AIAA Journal*, 60(5):3035–3047, May 2022. doi: 10.2514/1.j060866. URL https://doi.org/10.2514/1.j060866.

Zhenyu Guan, Junpeng Jing, Xin Deng, Mai Xu, Lai Jiang, Zhou Zhang, and Yipeng Li. DeepMIH: Deep invertible network for multiple image hiding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):372–390, January 2023. doi: 10.1109/tpami.2022.3141725. URL https://doi.org/10.1109/tpami.2022.3141725.

Denis Gudovskiy, Shun Ishizaka, and Kazuki Kozuka. Cflow-ad: Real-time unsupervised anomaly detection with localization via conditional normalizing flows. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 98–107, 2022.

Hocquet Guillaume, Bichler Olivier, and Querlioz Damien. OvA-INN: Continual learning with invertible neural networks. December 2019.

Jonas Haldemann, Victor Ksoll, Daniel Walter, Yann Alibert, Ralf S. Klessen, Willy Benz, Ullrich Koethe, Lynton Ardizzone, and Carsten Rother. Exoplanet characterization using conditional invertible neural networks, 2022. URL https://arxiv.org/abs/2202.00027.

Guillaume Hocquet, Olivier Bichler, and Damien Querlioz. Memory efficient invertible neural networks for Class-Incremental learning. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4, June 2021.

Wei Hu. Inverse molecule design with invertible neural networks as generative models. *Journal of Biomedical Science and Engineering*, 14(07):305–315, 2021. doi: 10.4236/jbise.2021.147026. URL https://doi.org/10.4236/jbise.2021.147026.

Jun-Jie Huang and Pier Luigi Dragotti. WINNet: Wavelet-inspired invertible network for image denoising. *IEEE Transactions on Image Processing*, 31:4377–4392, 2022. doi: 10.1109/tip.2022.3184845. URL https://doi.org/10.1109/tip.2022.3184845.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. March 2017.

John Jasa, Andrew Glaws, Pietro Bortolotti, Ganesh Vijayakumar, and Garrett Barter. Wind turbine blade design with airfoil shape control using invertible neural networks. *Journal of Physics: Conference Series*, 2265(4):042052, May 2022. doi: 10.1088/1742-6596/2265/4/042052. URL https://doi.org/10.1088/1742-6596/2265/4/042052.

C.A. Jensen, R.D. Reed, R.J. Marks, M.A. El-Sharkawi, Jae-Byung Jung, R.T. Miyamoto, G.M. Anderson, and C.J. Eggen. Inversion of feedforward neural networks: algorithms and applications. *Proceedings of the IEEE*, 87(9): 1536–1549, 1999. doi: 10.1109/5.784232.

Da Eun Kang, Eric W Pellegrini, Lynton Ardizzone, Ralf S Klessen, Ullrich Koethe, Simon C O Glover, and Victor F Ksoll. Emission-line diagnostics of H II regions using conditional invertible neural networks. *Monthly Notices of the Royal Astronomical Society*, 512(1):617–647, January 2022. doi: 10.1093/mnras/stac222. URL `https://doi.org/10.1093/mnras/stac222`.

Varun A Kelkar, Sayantan Bhadra, and Mark A Anastasio. Compressible latent-space invertible networks for generative model-constrained image reconstruction. *IEEE Transactions on Computer Imaging*, 7:209–223, 2021.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow, 2017.

Junlin Lan, Shaojin Cai, Yuyang Xue, Qinquan Gao, Min Du, Hejun Zhang, Zhida Wu, Yanglin Deng, Yuxiu Huang, Tong Tong, and Gang Chen. Unpaired stain style transfer using invertible neural networks based on channel attention and Long-Range residual. *IEEE Access*, 9:11282–11295, 2021.

Yuying Liang, Naoya Ozaki, Yasuhiro Kawakatsu, and Masaki Fujimoto. Modelling internal structure of differentiated asteroids via data-driven approach. *Monthly Notices of the Royal Astronomical Society*, November 2022. doi: 10.1093/mnras/stac3389. URL `https://doi.org/10.1093/mnras/stac3389`.

Yang Liu, Zhenyue Qin, Saeed Anwar, Pan Ji, Dongwoo Kim, Sabrina Caldwell, and Tom Gedeon. Invertible denoising network: A light solution for real noise removal. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13365–13374, June 2021.

Shao-Ping Lu, Rong Wang, Tao Zhong, and Paul L. Rosin. Large-capacity image steganography based on invertible neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10816–10825, June 2021.

Pablo Noever-Castelos, Lynton Ardizzone, and Claudio Balzani. Model updating of wind turbine blade cross sections with invertible neural networks. *Wind Energy*, 25(3):573–599, October 2021. doi: 10.1002/we.2687. URL `https://doi.org/10.1002/we.2687`.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation, 2018.

Yanzhen Ren, Ting Liu, Liming Zhai, and Lina Wang. Hiding data in colors: Secure and lossless deep image steganography via conditional invertible neural networks, 2022. URL `https://arxiv.org/abs/2201.07444`.

R Schirrmeister, Yuxuan Zhou, T Ball, and Dan Zhang. Understanding anomaly detection with deep invertible networks through hierarchies of distributions and features. *Adv. Neural Inf. Process. Syst.*, 2020.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL `https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf`.

Yunfei Teng, Anna Choromanska, and Mariusz Bojarski. Invertible autoencoder for domain adaptation, 2018.

Marian Turowski, Benedikt Heidrich, Kaleb Phipps, Kai Schmieder, Oliver Neumann, Ralf Mikut, and Veit Hagenmeyer. Enhancing anomaly detection methods for energy time series using latent space data representations. In *Proceedings of the Thirteenth ACM International Conference on Future Energy Systems*, e-Energy '22, pages 208–227, New York, NY, USA, June 2022. Association for Computing Machinery.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016a.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks, 2016b.

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast high-fidelity speech synthesis. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3918–3926. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/oord18a.html`.

Meng Wang, Tao Wen, and Haipeng Liu. A image deblurring approach using u-shaped invertible network driven by sparse latent variables. *SSRN Electronic Journal*, 2021. doi: 10.2139/ssrn.3985290. URL `https://doi.org/10.2139/ssrn.3985290`.

Chen Xu, Xiuyuan Cheng, and Yao Xie. Invertible neural networks for graph prediction. *ArXiv*, abs/2206.01163, 2022a.

Youmin Xu, Chong Mou, Yujie Hu, Jingfen Xie, and Jian Zhang. Robust invertible image steganography. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7875–7884, June 2022b.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.