



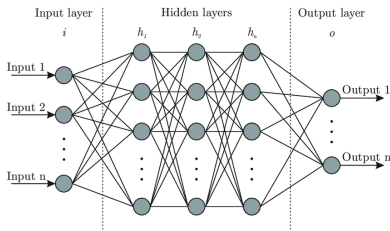
# Learning Bijections

Can Duyar

Advisor: Dr. Zafeirakis ZAFEIRAKOPOULOS

8 December 2021



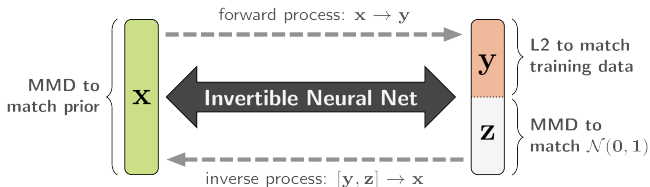


My project is related with training a neural network to act as a bijection. I am going to pick two sets of combinatorial objects that are known to have the same cardinality and train a network to act as a bijection. In this case, I am going to have different output for each input.

The goal is to investigate best practices to develop a general framework in the direction of learning bijections.



## My Solution: Invertible Neural Network Architecture

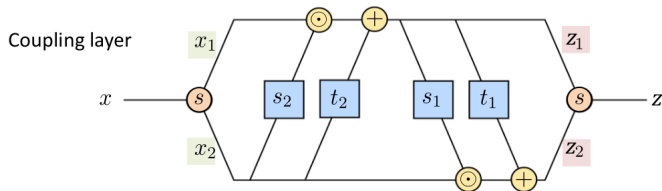


In my previous presentations, I mentioned that I plan to use an invertible network structure to learn bijections, because invertible networks have the feature of reconstructing the input from the output as well as learning the output from the input.

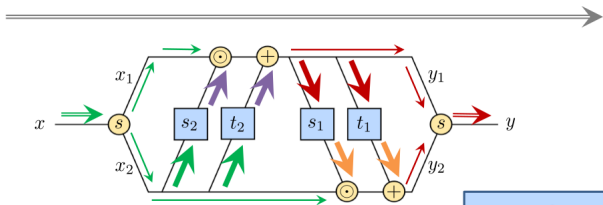


Forward computation:  $z_1 = x_1 \odot s_2(x_2) + t_2(x_2)$ ,  $z_2 = x_2 \odot s_1(z_1) + t_1(z_1)$

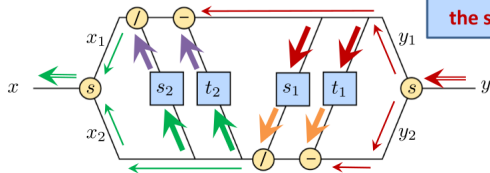
Inverse computation:  $x_2 = z_2 \oslash s_1(z_1) - t_1(z_1)$ ,  $x_1 = z_1 \oslash s_2(x_2) - t_2(x_2)$



forward

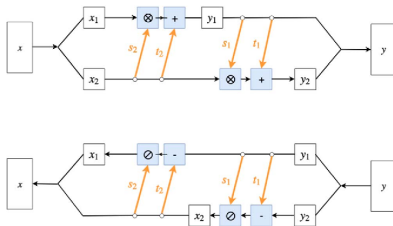


backward



nested networks  
 $s_{1/2}$  and  $t_{1/2}$  are  
always executed in  
the same direction

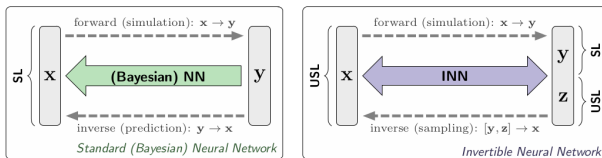




Invertible networks offer the opportunity to simultaneously optimize for losses on both the input and output domains, which allows for more effective training. Hereby, we perform forward and backward iterations in an alternating fashion, accumulating gradients from both directions before performing a parameter update.



## Abstract Comparison of Standard and Invertible Neural Networks



The standard direct approach requires a discriminative, supervised loss(SL) term between predicted and true  $x$ , causing problems when  $y \rightarrow x$  is ambiguous. Our network uses a supervised loss only for the well-defined forward process  $x \rightarrow y$ . Generated  $x$  are required to follow the prior  $p(x)$  by an unsupervised loss(USL), while the latent variables  $z$  are made to follow a Gaussian distribution, also by an unsupervised loss.



## InvertibleNetworks.jl

Documentation		Build Status	
docs	<a href="#">stable</a>	docs	<a href="#">dev</a>
		CI	<a href="#">failing</a>
		DOI: <a href="https://doi.org/10.5281/zenodo.5752164">10.5281/zenodo.5752164</a>	

Building blocks for invertible neural networks in the [Julia](#) programming language.

- ▶ Julia is a new language and implementing invertible networks has a different structure than normal networks, so I did some framework research on its implementation. As a result of my research, I found a framework developed for Invertible Networks. This framework provides some building blocks to implement the invertible neural networks.





```
# Activation normalization
AN = ActNorm(k; logdet=true)

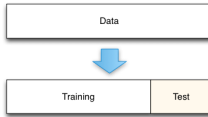
# Forward-inverse
Y = AN.forward(X)
X = AN.inverse(Y)

# Backprop
ΔX, X = AN.backward(ΔY, Y)
ΔY, Y = AN.backward_inverse(ΔX, X)

# Jacobian
J = Jacobian(AN, X; io_mode="θ→Y")
ΔY = J*Δθ
Δθ = J'*ΔY
```

- ▶ Inverse blocks of the Invertible Network package





- ▶ My dataset was already ready and I was thinking how to divide this dataset into train and test splits when I will start model training. I did some research on this and found a library called "Lathe". I wrote a short script for my dataset to see how I can split train and test on the dataset with this library.



```
using CSV
using DataFrames
using Lathe.preprocess: TrainTestSplit

df = CSV.read("learning_bijections_dataset.csv",DataFrame)
train, test = TrainTestSplit(df,.75)





println(size(train))# (75093, 4)
println(size(test)) # (25273, 4)
```

Script for reading the CSV and split it for train and test operations with using Lathe library.



- ▶ I am trying to make simple model implementations using K-net and invertibleNetwork frameworks. In some cases (especially during implementation with K-net) I wanted to try testing it on GPU. I had some driver problems with my computer's GPU and that's why I'm working on Google Colab for now because Google Colab provides free GPU support.



-  Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W. Pellegrini, Ralf S. Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe, *Analyzing inverse problems with invertible neural networks*, 2019.
-  Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen, *Invertible residual networks*, 2019.
-  Philipp A. Witte, Mathias Louboutin, Ali Siahkoohi, and Felix J. Herrmann Gabrio Rizzuti, *Invertiblenetworks.jl – memory efficient deep learning in julia*, 2021.
-  Han Zhang, *Invariance and invertibility in deep neural networks*, 2020.

