**Vor 19.01.:**

-reading 5 papers, 3 from CVL about (c)INNS and bayesflow, 2 from others about normalizing flows

-getting familiar with FrEIA and BayesFlow

-task from Stefan: implement RationalQuadraticSpline coupling layer in bayesflow

-understanding general FrEIA framework, then looked at RQSpline implementation

-wrote script to implement minimal example from FrEIA, primarily combined code snippets from FrEIA into single python file ("C:\Users\Fritz\Uni\Masterarbeit\Spline Module for BayesFlow\pytorch\_rqspline.py")

-beginning to write own basic tensorflow spline class ("C:\Users\Fritz\Uni\Masterarbeit\Spline Module for BayesFlow\tensorflow\_spline.py")

**Thursday, 19.01.**

-define 2 spline coupling nets instead of one, and more importantly figure out how the splines handle varying dimensionalities (one 2D-spline for each PARAMETER (always last dimension of x-dimensional target), so that spline is always R -> R)

-forward function framework is running now

-added condition into framework, not semantically yet

**Monday, 23.01.**

-trying to implement forward function correctly

-implemented basic versions of split\_params and constrain\_params

-stop: working on binning of spline, cant figure torch.searchsorted out

**…**

-biggest hold-up is that masking targets with in domain or not does not work in .forward()

**Monday, 06.02.**

-reading through analyzing invertible problems with INNs again

-starting to write own INN script, beginning with forward sampling functionality

-sampling works, starting to build basic INN model, basic coupling net also implemented, permutation also added

**…**

**Tuesday, 07.02.**

-“finished” spline-flow framework, works error-free forward, backward, with 2D and 3D input. Still returns a few NaNs, need to look at computations in more detail

-continued working on INN from scratch, implemented basic loss, gradient and optimizer framework. still have to deal with predicted NaNs

**…**

**Wednesday, 05.04.**

-worked on inn from scratch, which by now is trainable and predicts the forward process (angles -> end points) well. Implemented basic functionality of inverse process. Backward process produces NaNs though. Until now only MSE loss is implemented, not using the “real” invertible flow loss which also includes a regularization of the log jacobian. Have to find description of this in the papers, and implement it, that should help with the NANs

**Wednesday, 12.04.**

-noticed that I haven’t considered log jac dets in my loss function. Started to look at the required loss again more thoroughly, “analysing inverse problems with…” paper should help with that. It looks like its two independent losses for y (points) and z (latents) and that a loss for x (angles) is optional for improved performance but not necessary. Will look into it more tomorrow.

**Sunday, 16.04.2022**

-started working on making plehn´s group´s spline implementation compatible with 3d input also. First question: why does inside\_mask marginalize over each dimension of the inputs?

-3d functionality now works

**Thursday, 27.07.23**

-talked with ulli today, next steps are setting up GPU and trying to find old code from Lynton et al. which worked with data of multispectral cameras. Found code through lynton´s github (github repo of analyzing invertible…)

-there seems to have been quite a few changes to FrEIA, especially class names. so have to check if I can make code run easily or if I just use it for “inspiration”. Also, in the analyzing repo there is references to vislearn/datasets/dkfz but I can´t find it

**Saturday, 29.07.23**

-started manually adding the old Freia classes etc. referenced in analyzing… into extra python file (freia\_extras.py).

-fixing it bug by bug, currently stuck with AssertionError: Torch not compiled with CUDA enabled

**Friday, 4.8.23**

-working on my own FrEIA model now, using MNIST dataset. Not completely sure what to do with it yet, just want to get used to the framework now. Probably in x(image) -> y,z, y will be one-dimensional (info about which number), and z will contain styles or something like that. It seems to me, that with the DKFZ data the actual pictures were not accesses, as y in the code is only 8 dimensional. So maybe it’s a pixelwise mapping of absorption spectra values to the 5 params like so2 concentration etc. Have to ask ulli if it is sensible to pursue this topic, as I talked with Flo yesterday, and he has already been working on it and doesn’t think it can work. Was able to run the 8-modes gaussian toy model yesterday. Only a few code inconsistencies had to be adapted.

**Saturday, 5.8.23**

-started implementing MNIST inn with y being one-hot encoded class information (ydim=10). Implemented cross-entropy loss for y but it does not work yet. Code runs through but there seems to be no learning effect.

**Monday, 7.8.23**

-had put optimizer.step instead of optimizer.step(), which is why it didn’t learn. Learning to predict the classes with cross-entropy for y works well. Also included losses for z-norm and logjacdet, and learning works. But not sufficient performance yet. Am not sure if i´m doing model saving and loading in the most efficient way. Am playing with the lambda for the y\_loss now, because loss decreases but the quality of backward samples still is very bad. Dynamic plotting now also kind of works. Have to look a bit more at pyplot docu, for example how to mark where the learning rate decreases etc.

**Tuesday, 8.8.23**

-made a nice plotting script for the losses, and added plotting.py to the pack. For lambda\_Y=70, y\_norm just kept on rising, for 35 y\_norm stagnates at ca. 35, y\_loss at 0,5, ljd at 50. For lambda\_y=20: y: 0,8-1,1, z: 25-28, ljd: 60. Implemented saving the plots for different lambdas, and implemented a learning rate scheduler.

**Wednesday, 9.8.23**

Noticed that when I sample back, the first 10 values of x are exactly like the condition (y) given to the backward pass (0,0,0,0,1,0,…). Until now I performed x(64) -> [y(10),z(54)], not treating y specially. But freia offers extra condition methods, so I might have to restructure.

**Thursday, 10.8.23**

Noticed that in sample\_backward, I applied the inn twice to the sample, which means that it probably computed z ->x ->z, which could explain why the y values were not changed. But changing it and visualizing z -> x did not yield good results either. Will only work with the new conditioning I implemented, which computes x <-> z and adds y as input to each subnet that calculates s and t, as it is meant to be in freia implementation. This should mean however, that the classification functionality is lost, as y is only ever used as input and can not trivially be calculated as output. However it is possible to take an image x, and run it through the network separately with all possible conditions, and see which condition yields the smallest z-norm corresponding to highest probability (probability density estimation.). also tidied up my github

**Monday, 14.08.23**

Noticed that the initial learning rate has quite a significant influence on the learning behaviour. Am still using the reduceLRonplateau with factor of 0.6, and higher learning rates make ljd higher and z\_norm lower, but also make training rather instable. Didn’t do too much today, just compared learning behaviour for different lrs and reducefactors.

**Wednesday, 16.08.23**

So far, best training results with lr\_init=0.01 and reduce\_factor=0.6. ljd was at around 150 and z-norm at around 17, also on repeated test. Question is, if these metrics are good or if I should try to implement something like **FID**. Further questions are how to increase the performance. Options are hyper-parameter or model-based changes. \*Figured out that this limiting of s and t is what is meant with clamping, and is used in AllInOneBlock by default with alpha=2 (1.9 recommended in papers)

Hyper-parameters:

* Coupling net layers and sizes, activation functions of coupling net (limit s and t so that learning becomes less chaotic\*),
* N\_blocks
* Padding x and y
* Different optimizers
* Soft permutations

Architecture

-

Loss

* Should weight regularization be added for the coupling nets?

Other

* Add noise to x

Using soft\_permute makes loss more smooth (z-norm), ljd goes from 150 to 170, z-norm is a bit higher (worse).

Using more coupling blocks makes loss very instable, so that training doesn´t reach good values. Seems to be because z-norm diverges, not because of ljd.

All adaptations made z-norm higher on average!

Adding noise (0.05) made loss converge to worse value. Have to find out what the scaling of the images actually is.

Should continue with Haar wavelet downsampling possibly.

**Tuesday, 23.08.23**

Added an extra test.py file with argparse functionality. Right now it looks as if adding noise to x\_train only worsens performance. 5 coupling blocks, soft permutations, and larger coupling nets and lr\_init=1e-3 and reduce by 0,6 are now standard. Can now examine weight regularization for coupling nets, try different optimizers and add padding to x and y. Alternating training works in principle but does not give much improvement. Adding a factor >1 to the z-norm term reduces the z-loss by the same factor but does not seem to have good effects on the visual tests. The values from the visual test´s reconstruction are not in [0,1]. Ljd only reduces a bit (from around 230 to 180 for z-factor=7)