Superconduction qubit readout

Applied Machine Learning

Peter Rischel(rpv922), Anna Boye(jnc686),
Emil Fischer (gqz185)
12/06/2024

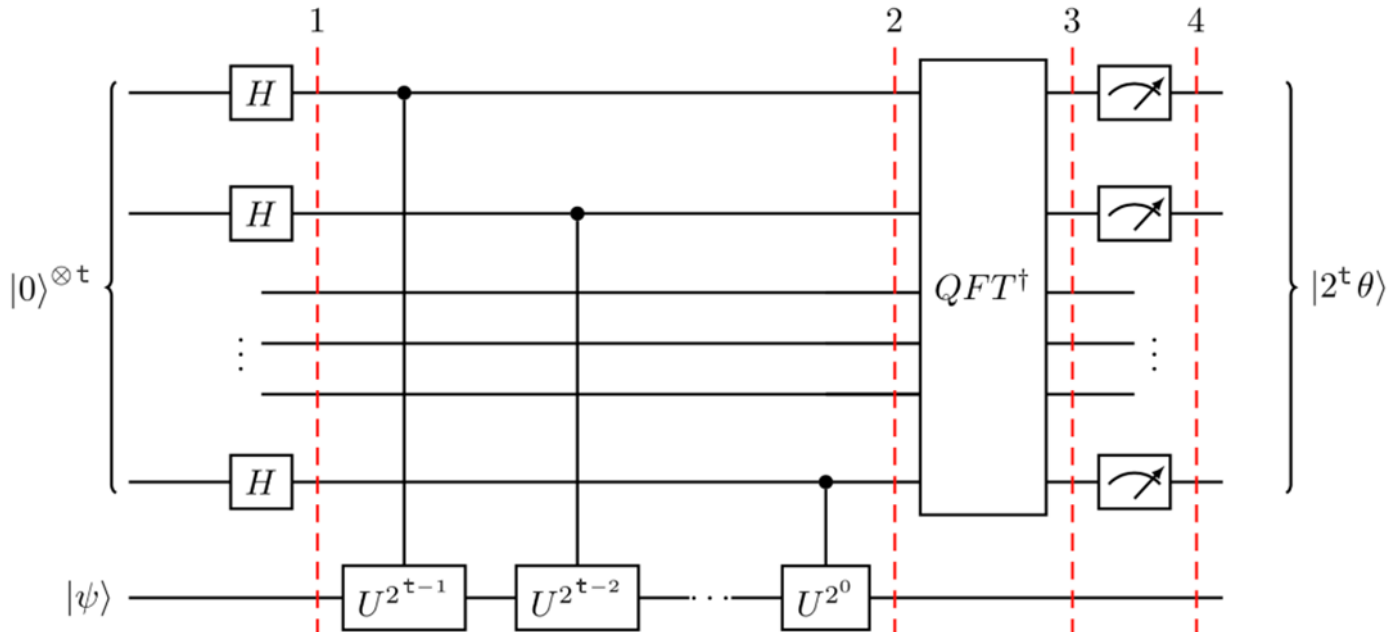
UNIVERSITY OF COPENHAGEN

Outline

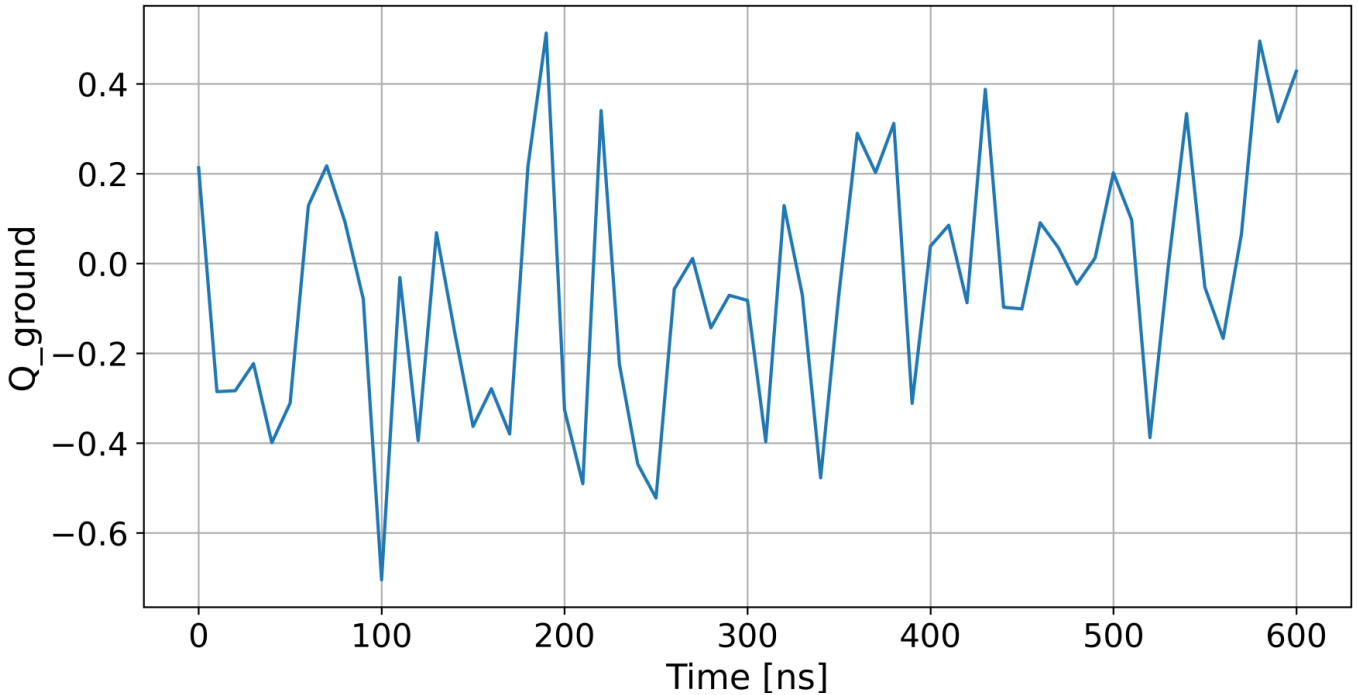
- Data and data preprocessing
- Vanilla Binary Classification NN (used on Aleph dataset)
- LDA for benchmarking
- Convolutional 1D NN
- Long Short-Term Memory NN
- Auto-encoder

Problem and data

- Classification problem



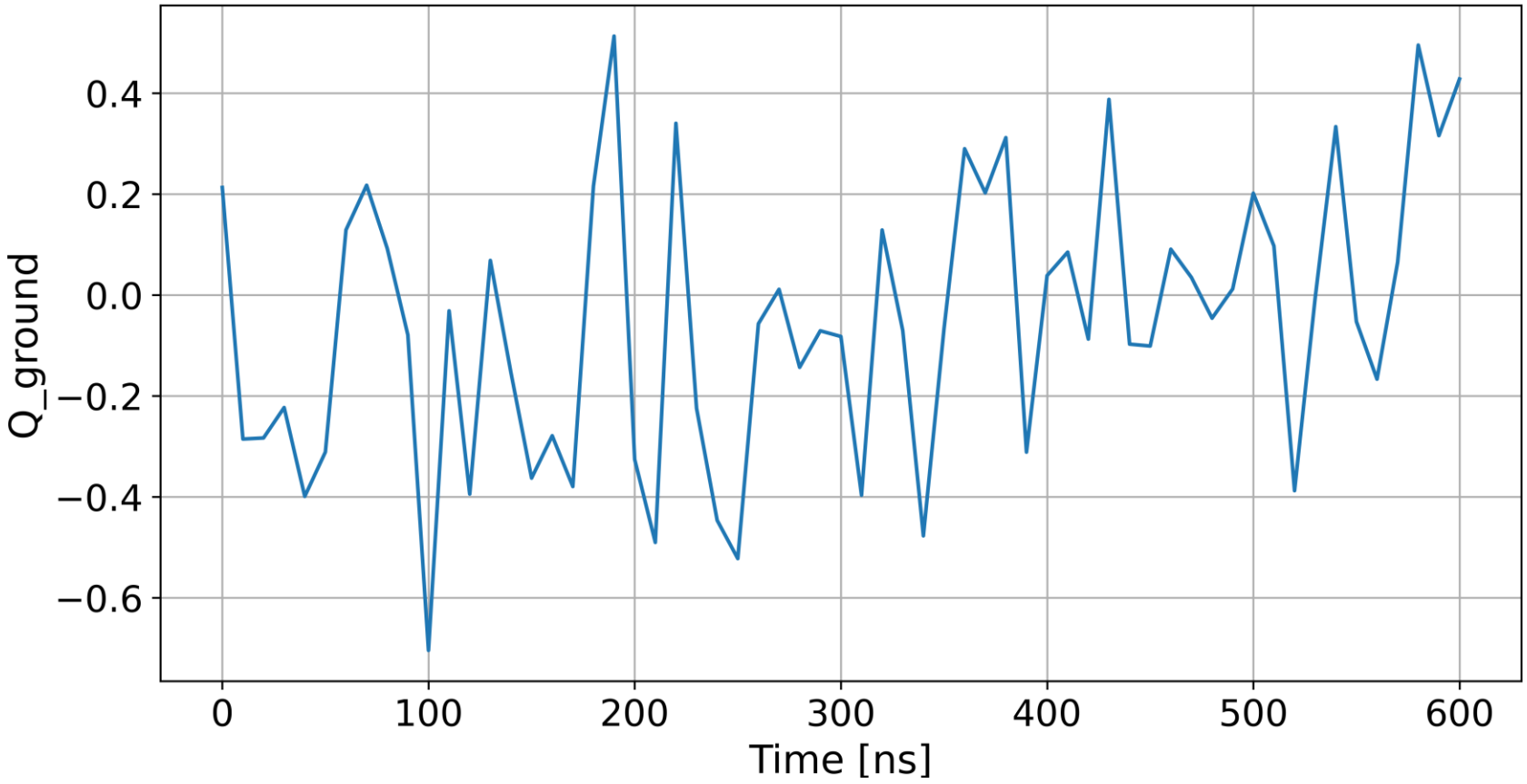
- Time series data



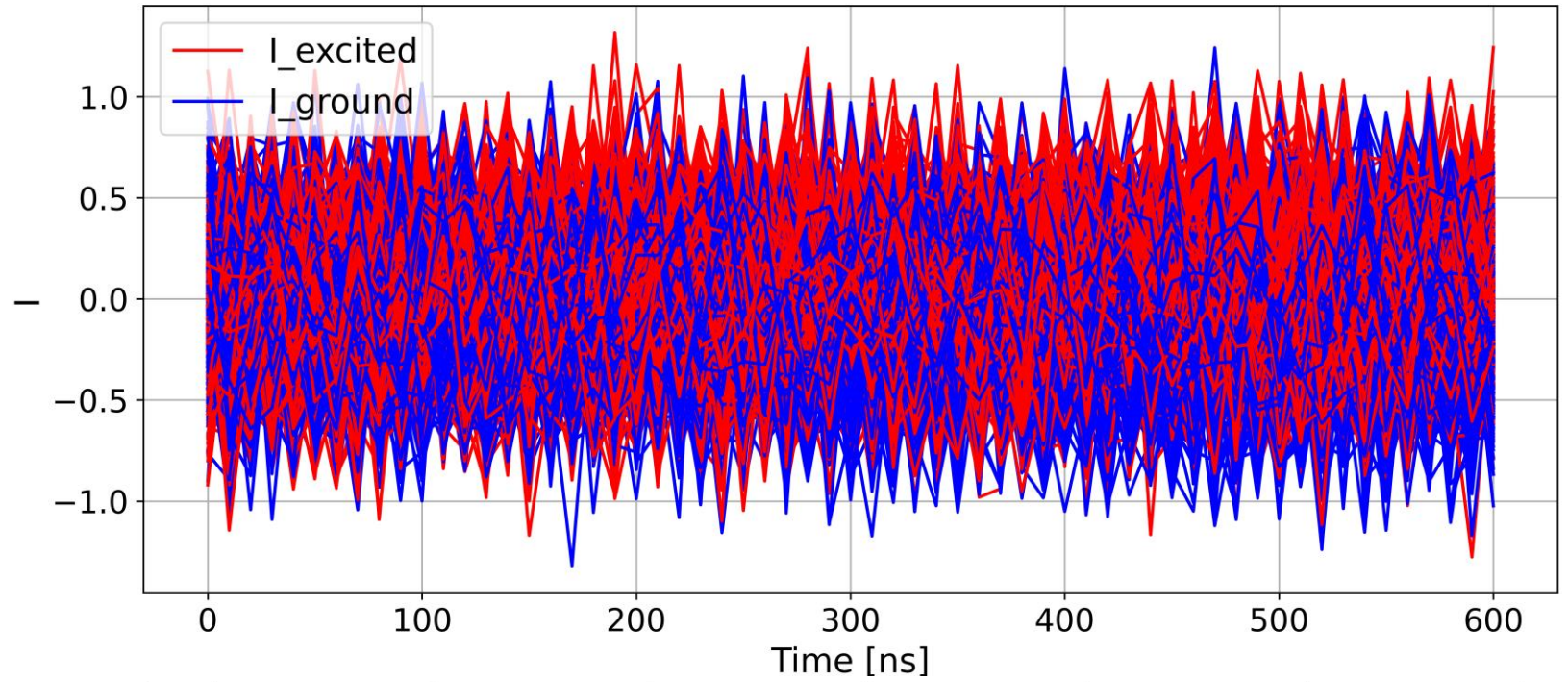
Simulated data and real data

- 2000 time series from experiment, and 1000 simulated time series
- Each time series has 61 time points and 2 features(Q and I).

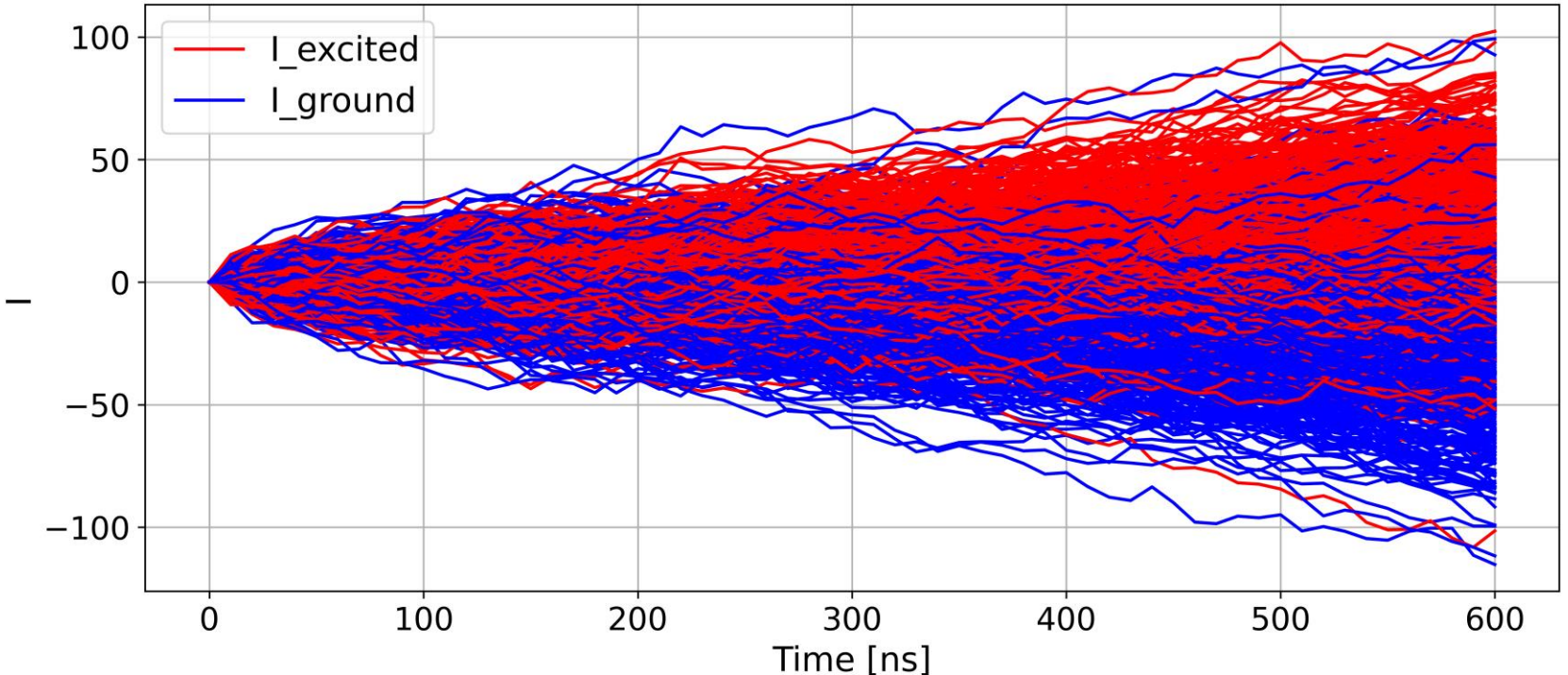
Example of one time series



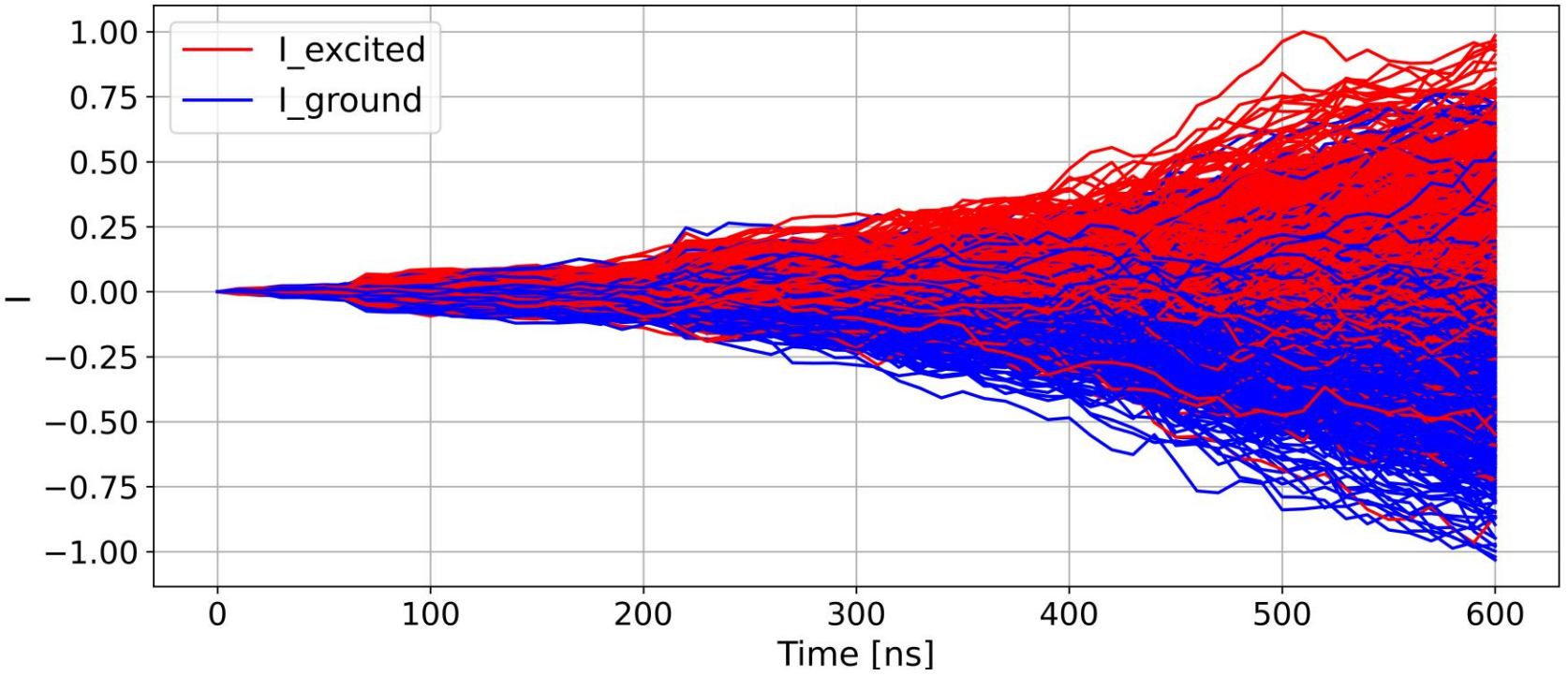
Regular time series data



Integrated time series data



Integrated time series data, weighted and scaled



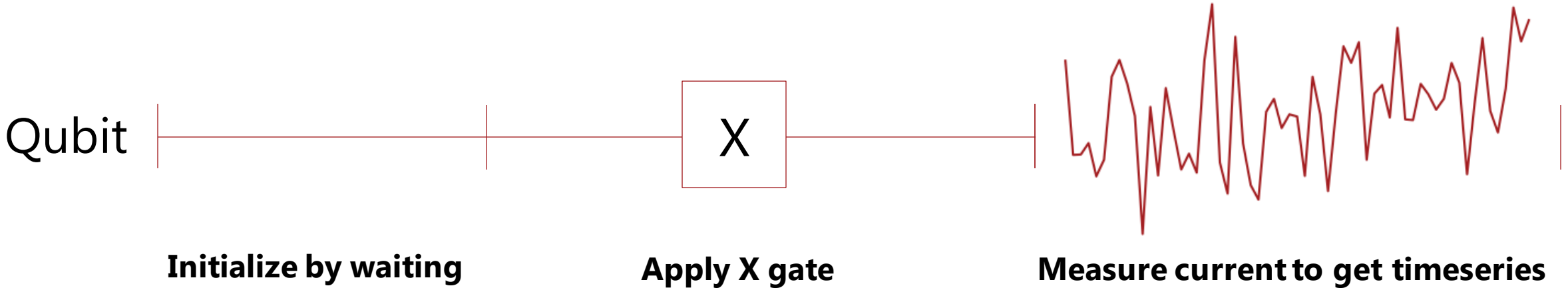
Fidelity vs accuracy

- Fidelity = (1 – FNR – FPR). Used because one want´s a better understanding of both error types when working with qubits.
- Fidelity and accuracy is correlated, but not equivalent.
- In the range we are working with, 0.60 fidelity ≈ 0.80 accuracy.

5-10 percent mislabeled data and decays

- Initialize the qubit in ground state by waiting
- Apply X-gate for 1000 of the 2000 trials to get excited states (one)
- Temperature fluctuations can cause the initialization to not be in ground state.

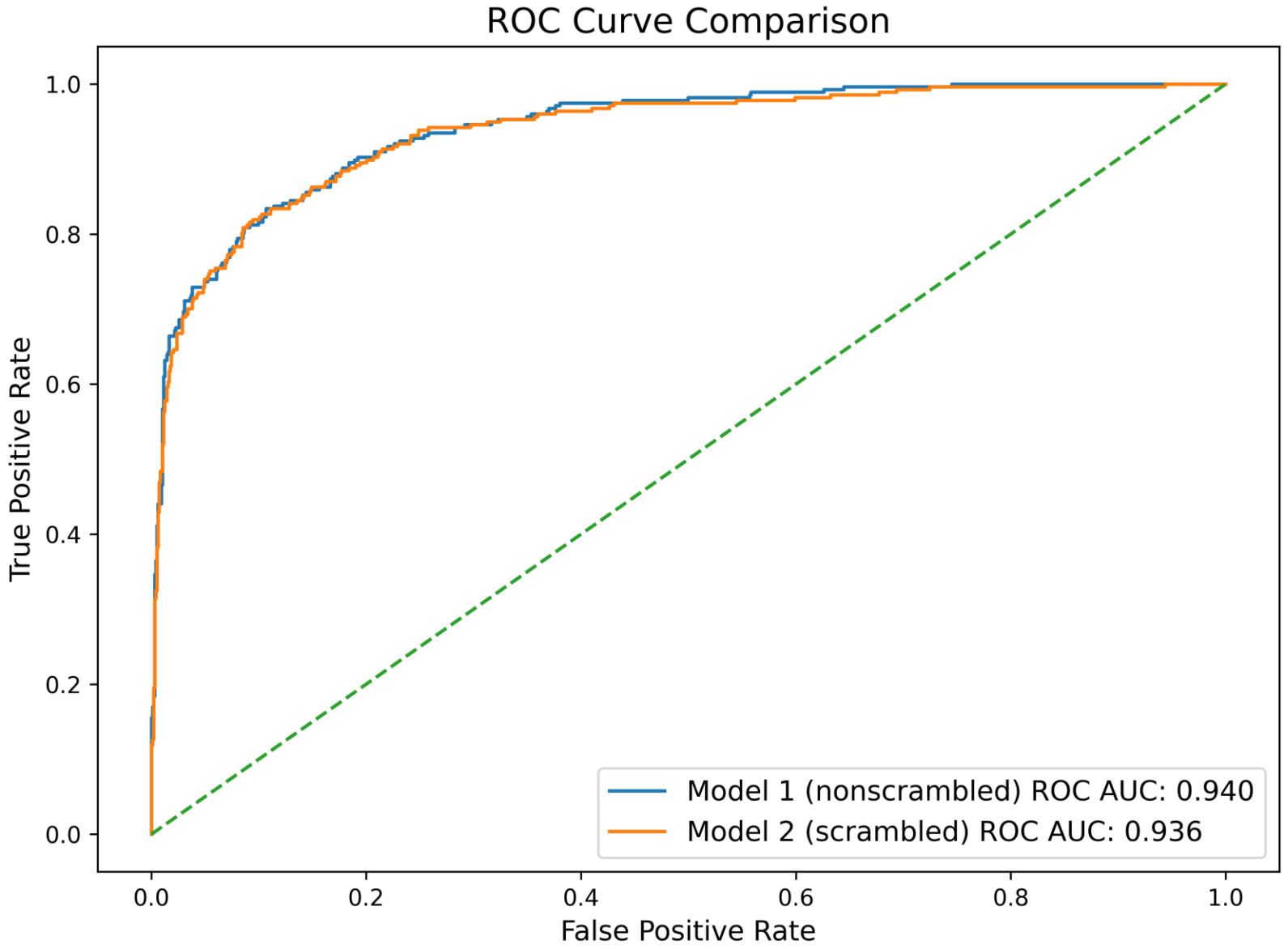
Example of one experiment



Model performance with 10 percent mislabels - Aleph

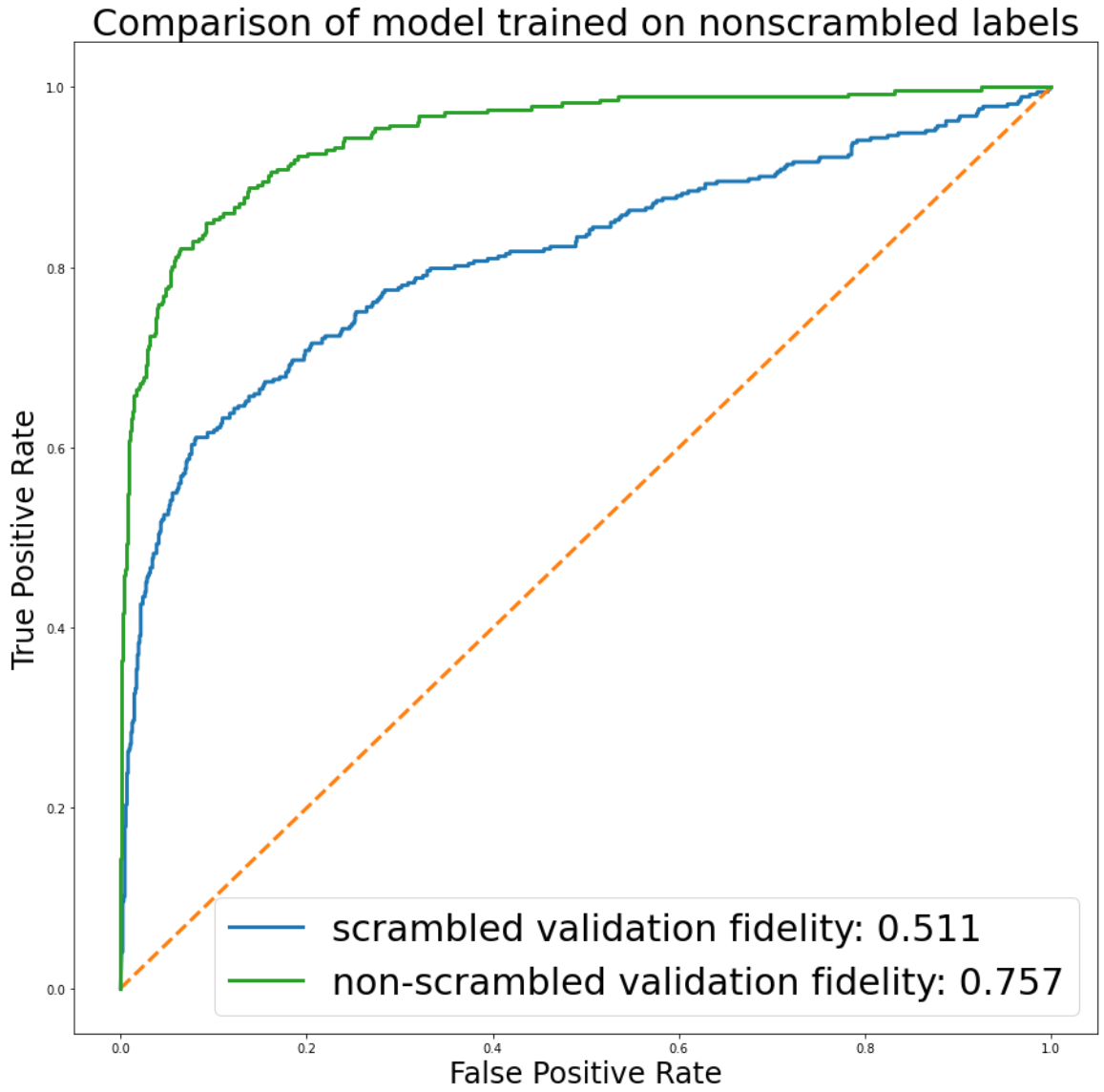
Fidelity for model trained on non-scrambled labels: 0.749

Fidelity for model trained on scrambled labels: 0.743

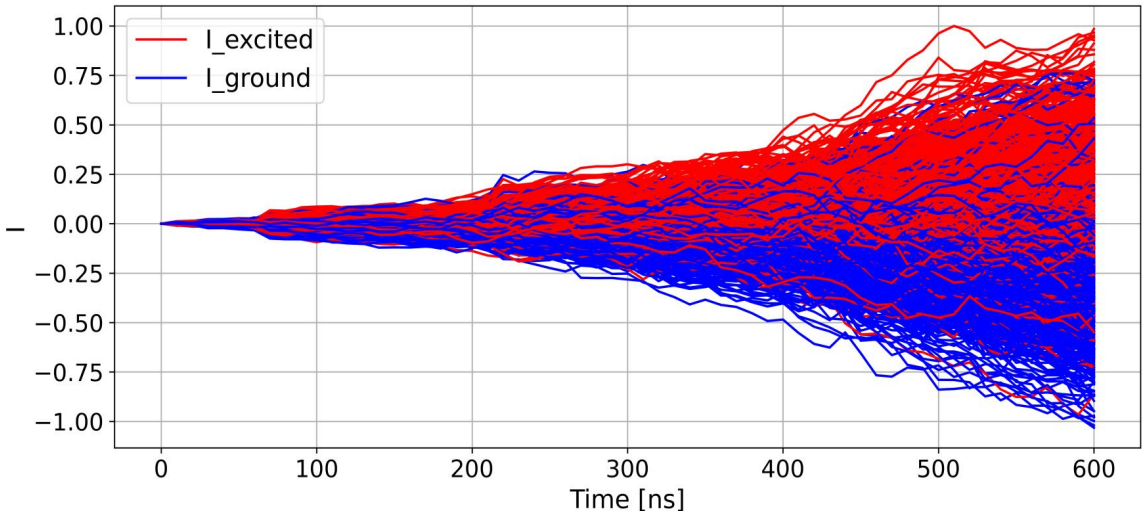


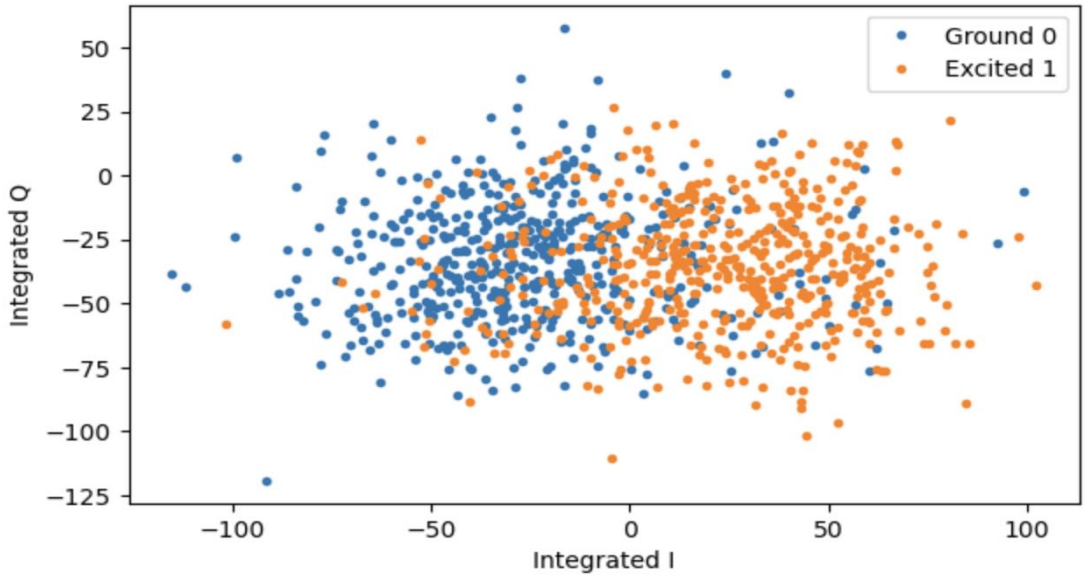
Fidelity decrease with scrambled labels

Validation of a model trained on perfect labels will lose ~33% fidelity when validated on scrambled labels compared to non-scrambled

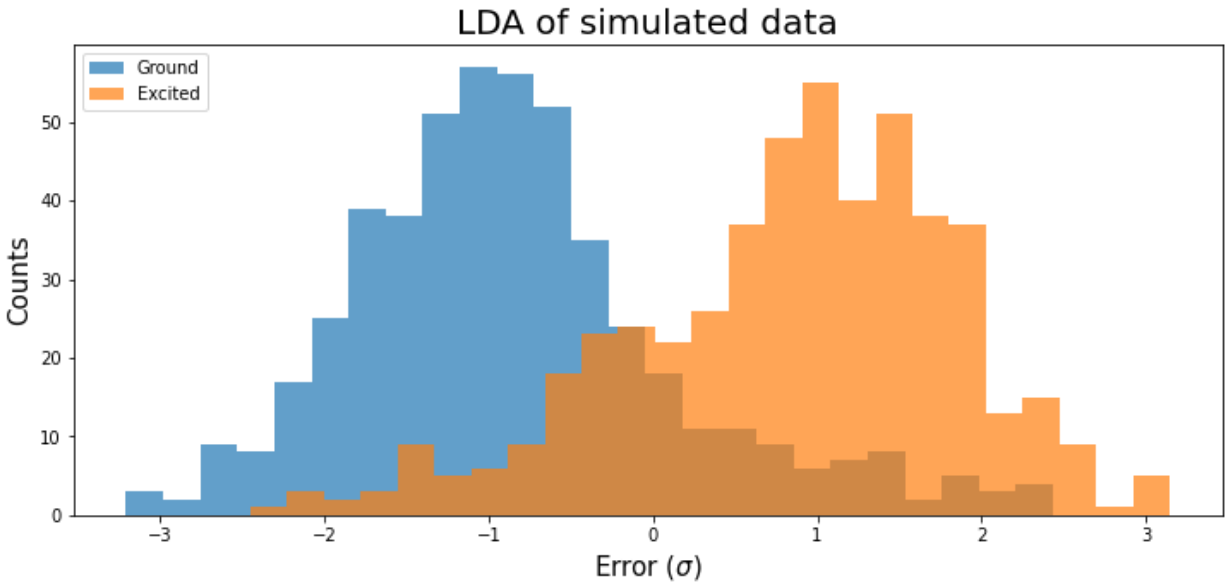


What to beat





- Simple LDA solution on integrated and weighted data
- Fidelity on sim data: 0.646
- Fidelity on real data: 0.618

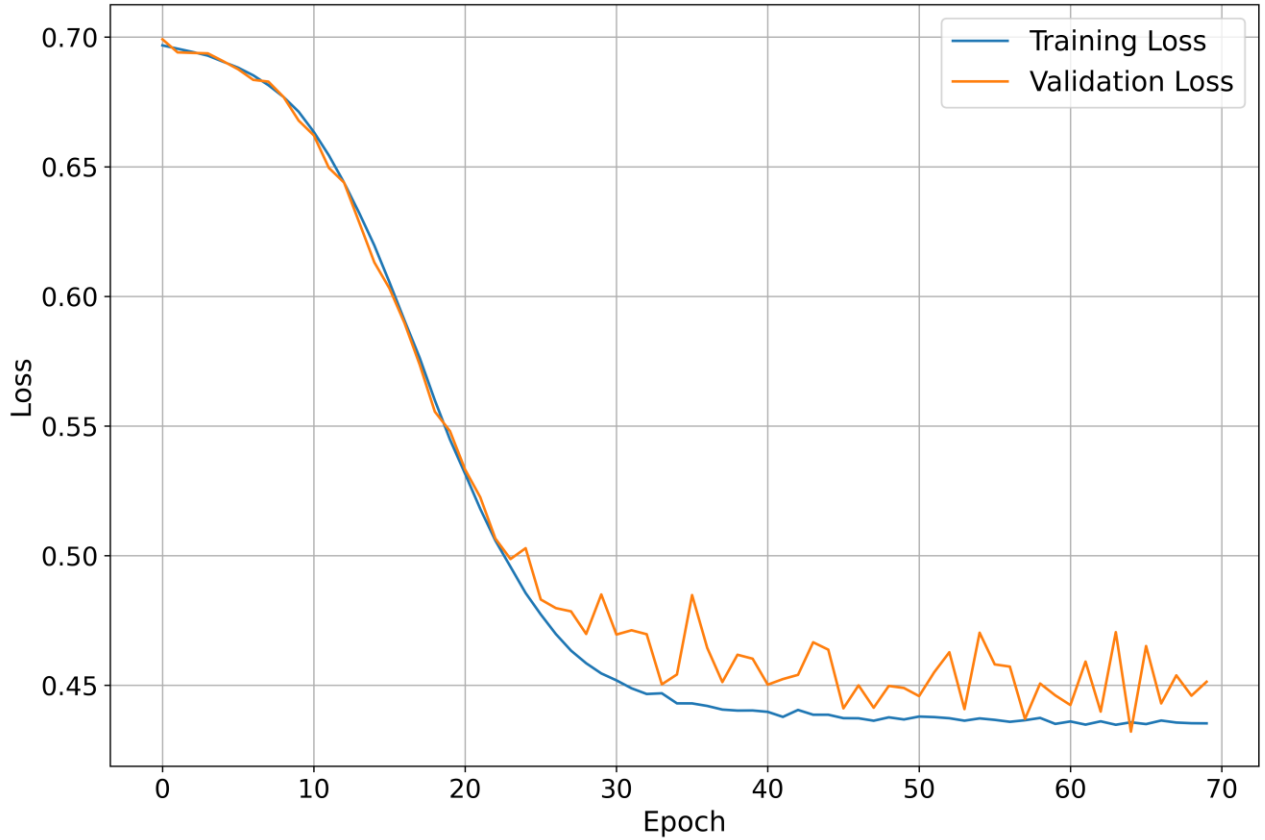


Convolutional 1d

- Changing the kernel(filter) one can modify a convolutional NN to handle time series data

- Can capture local trends in the data

Fidelity on simulated data: 0.640 pm 0.013
Fidelity on real data: 0.62 pm 0.017



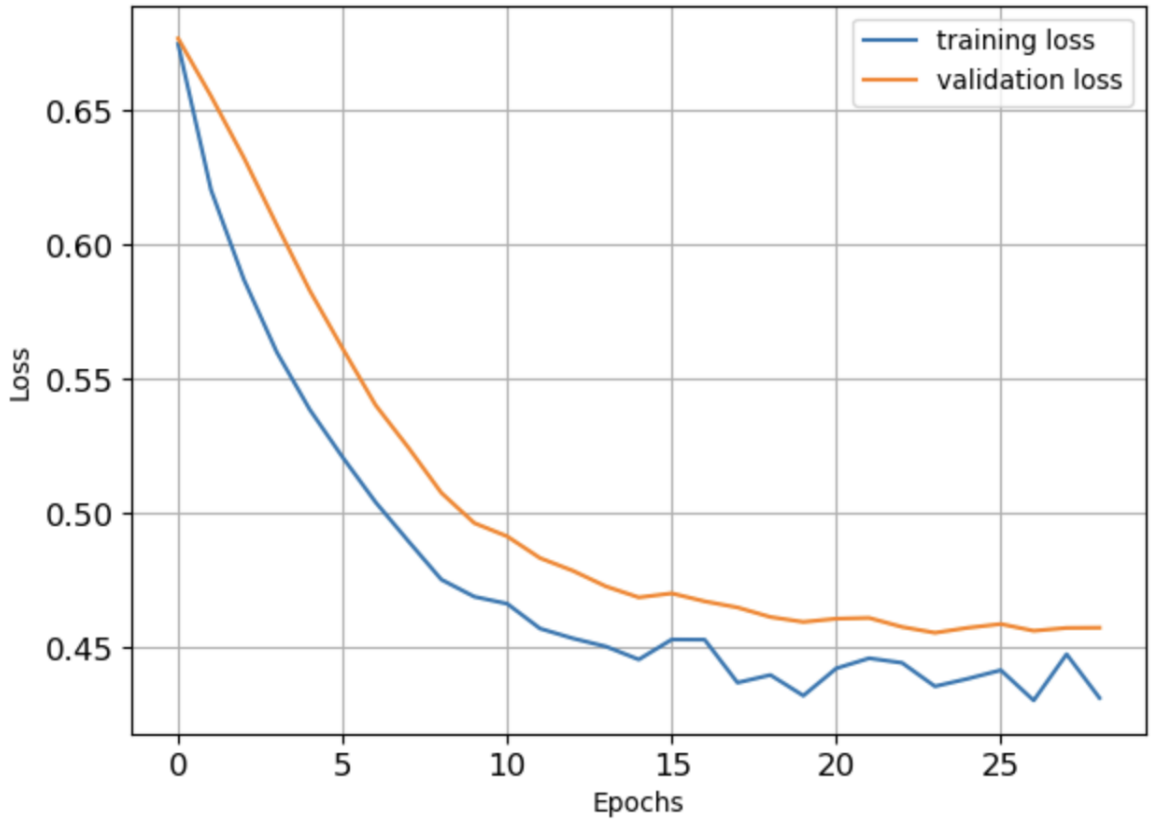
LSTM NN

- Can likely capture the general trends in the time series better than a convolutional network.

- Can handle the smaller amount of data, compared to a transformer

Fidelity on simulated data: 0.664 pm 0.015

Fidelity on real data: 0.61 pm 0.03



Results - Classification

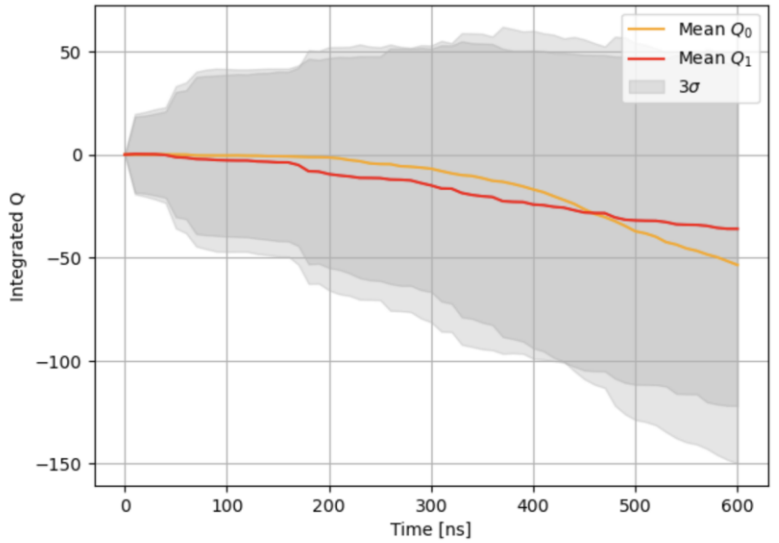
5 fold cross validation was used what evaluating fidelity.

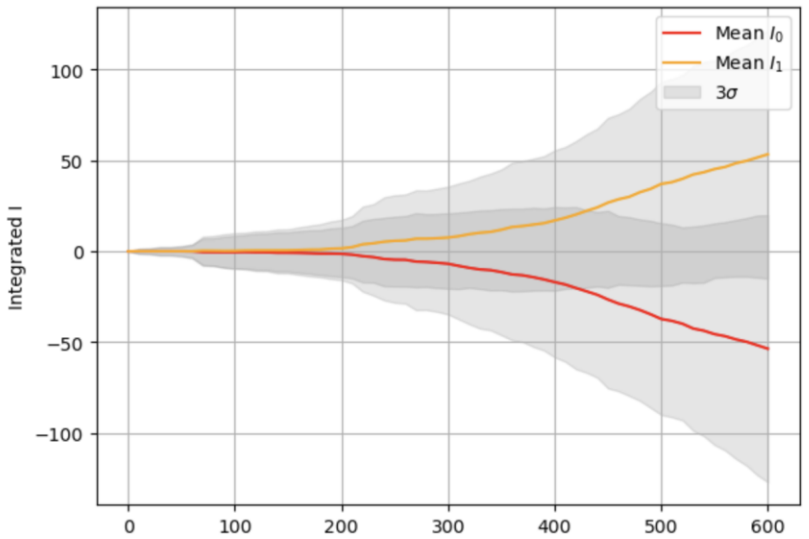
sim = simulated data
real = experiment data

Algorithm	Fidelity - sim*	Accuracy – sim*	Fidelity - real*	Accuracy - real*
LDA	0.646	0.822	0.618	0.804
LSTM	0.664 pm 0.015	0.816 pm 0.009	0.61 pm 0.03	0.80 pm 0.13
Convolutional1D	0.640 pm 0.013	0.819 pm 0.011	0.62 pm 0.017	0.81 pm 0.008

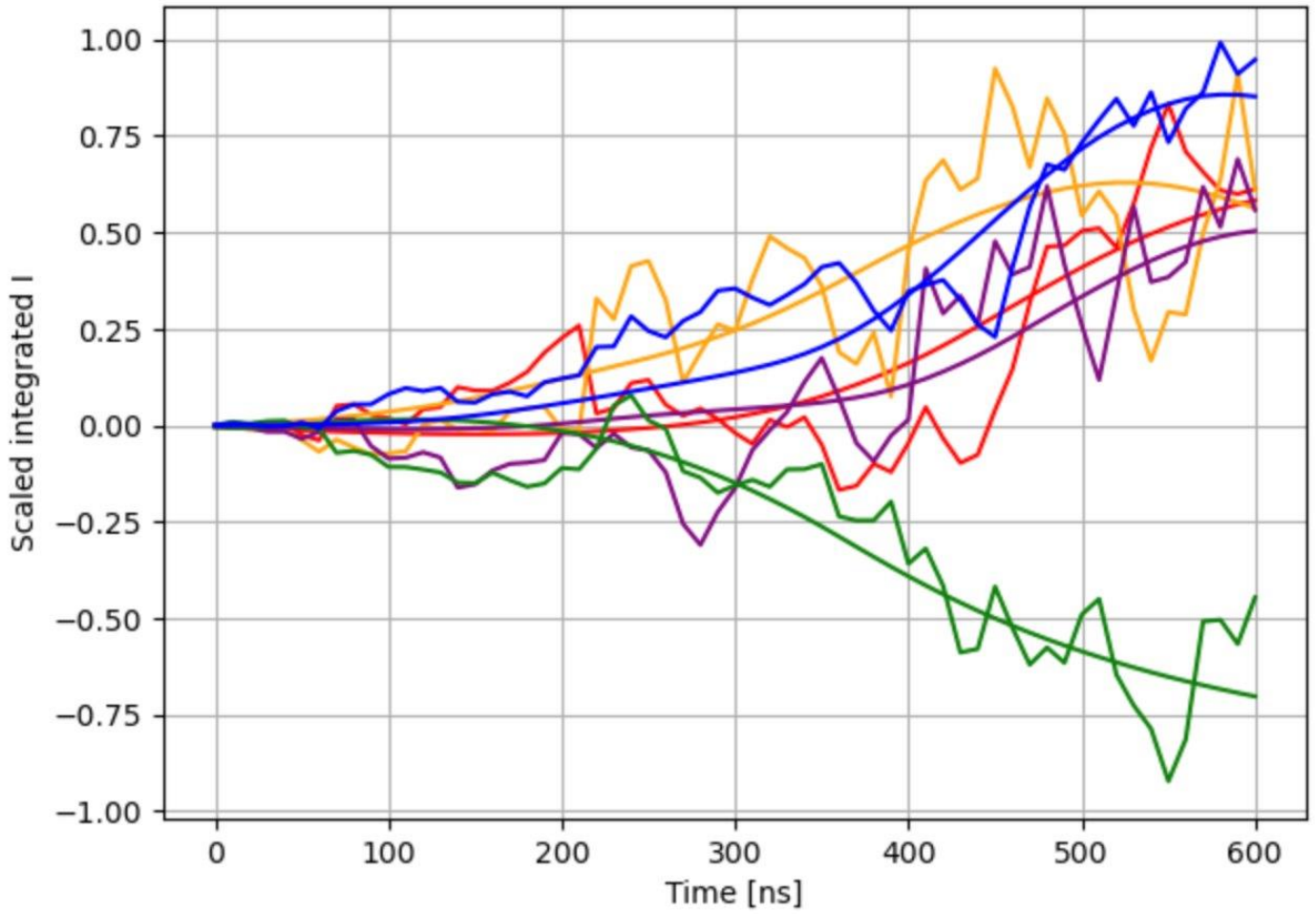
Auto Encoder for detecting decay trajectories

- Inspiration: The potato auto-encoder.
- Training data: Data labelled with high probability as either 0 or 1 by an NN, i.e. "perfect" trajectories.
- Test data: Data including some not "perfect" trajectories, i.e. perhaps decay trajectories
- Anomaly detection: prediction more than 3 sigma off
- Bayesian statistics to determine the probability of which distribution the trajectory belongs to

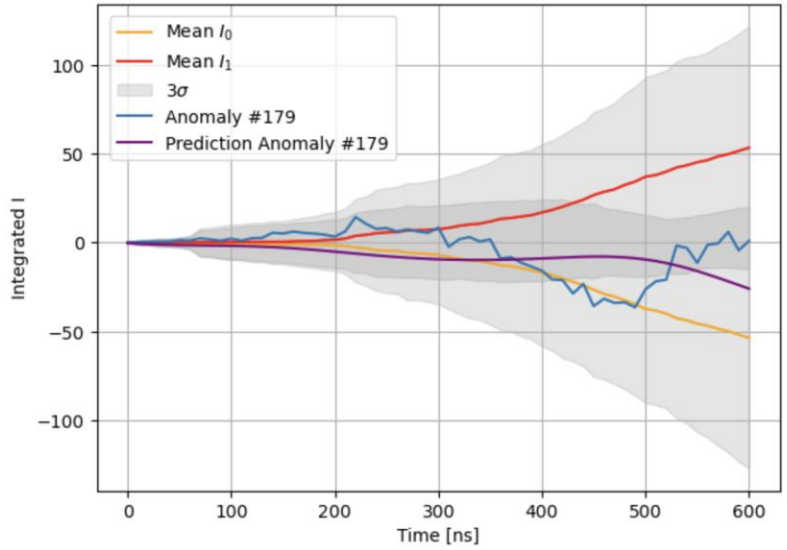


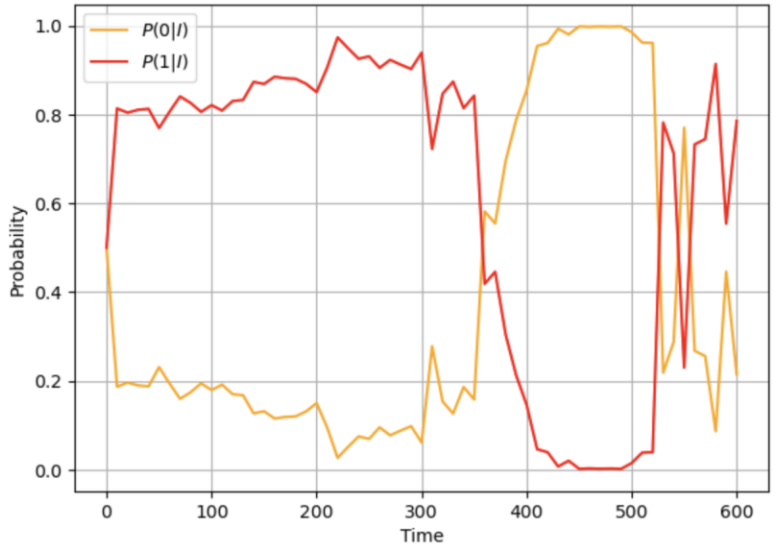


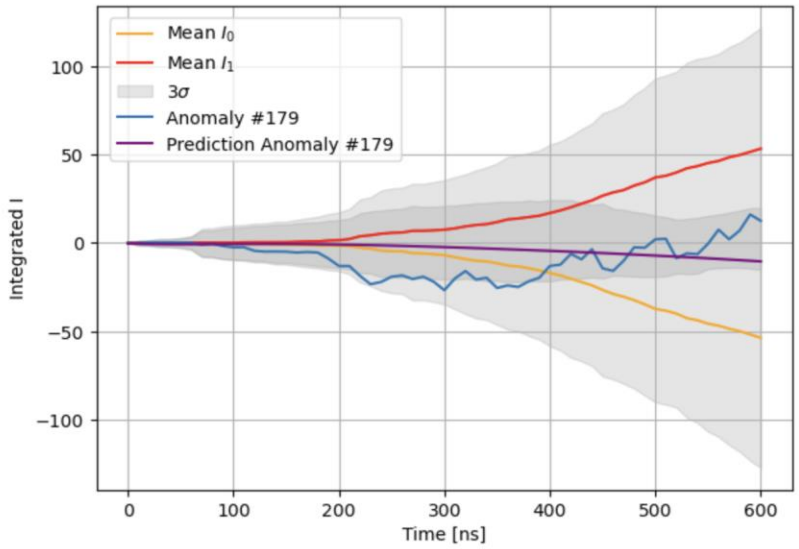
Auto encoder prediction examples, not anomalies

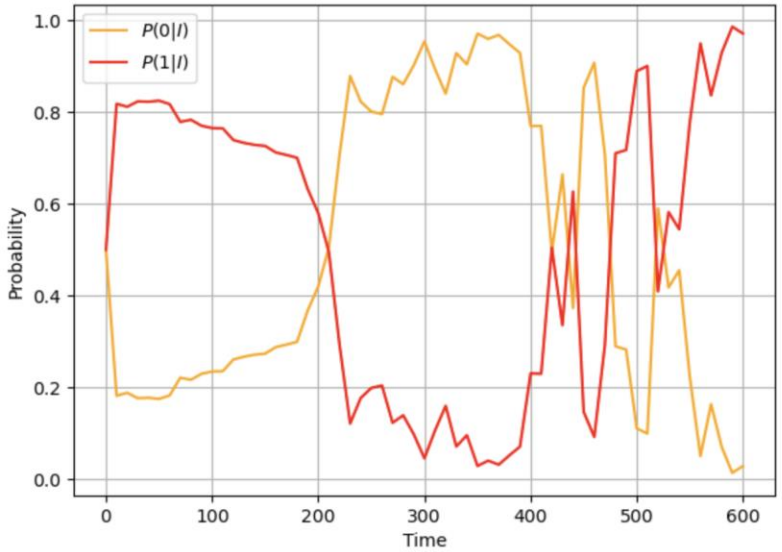


Auto-encoder anomaly detection









$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Example:

P(0, I): prob of state 0 given value I
P(I, 0): prob of measurement I given state 0
P(0): prior, prob 0. Determined from given label
P(I): P(I, 0)*P(0) + P(I,1)*P(1)

Conclusion and key takeaways

To conclude, the lack of data made it difficult to obtain meaningful results. But we learned a lot! Here are some key takeaways:

- Unsupervised learning to produce new labels does not work. If a clustering algorithm can separate data into two blobs, so can an NN…
- Maybe the evolution of the trajectories does not contain more information, than the final seperation, as LSTM & 1dConv did not really beat LDA…
- Initially not using Kfold to determine the final fidelity. Such a small data set is very (!) sensitive to the training and validation sets.
- A model trained on non-scrambled vs 10% scrambled data perform almost equally good with nearly identical fidelities and accuracies.
- Scrambling 10% of the labels on a validation set, reduces the fidelity of a model with 0.33.

If we had more time

- Become a PhD in Qdev and take more experimental data (or simulate more).
- If we got enough data, upgrade the LSTM to a transformer.
- More data would perhaps allow an NN to see trends in data with large variance.

Thanks to Johann!!

Appendix 0

- The project was evenly distributed

Appendix: Accuracy and Fidelity example

Model A:
TP = 40
TN = 900
FP = 50
FN = 10

Model B:
TP = 50
TN = 850
FP = 100
FN = 0

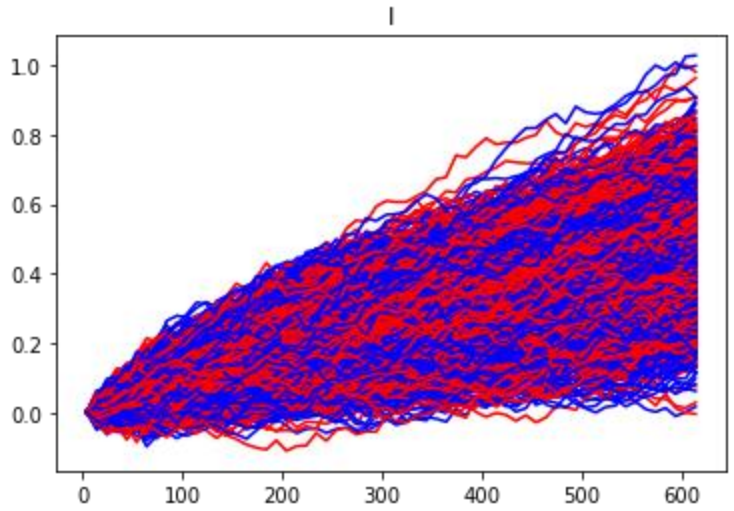
$$\text{Accuracy} = \frac{TP+TN}{(TP+TN+FP+FN)}$$

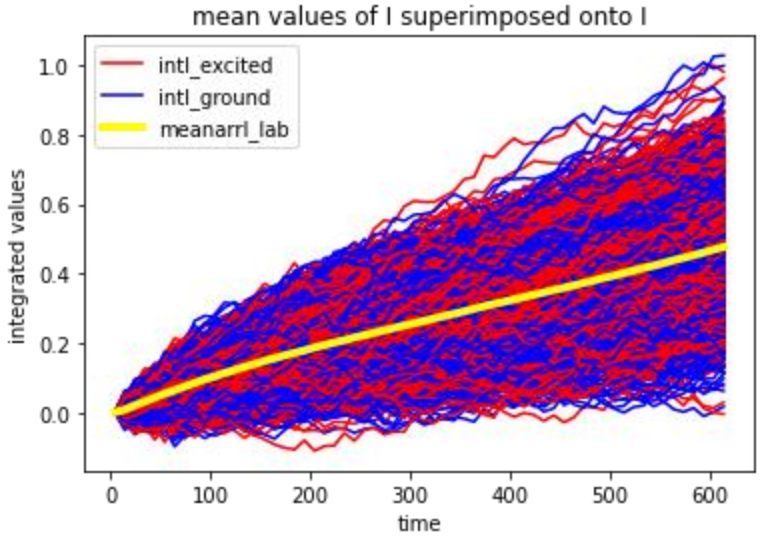
$$\text{Fidelity} = (1 - FPR - FNR)$$

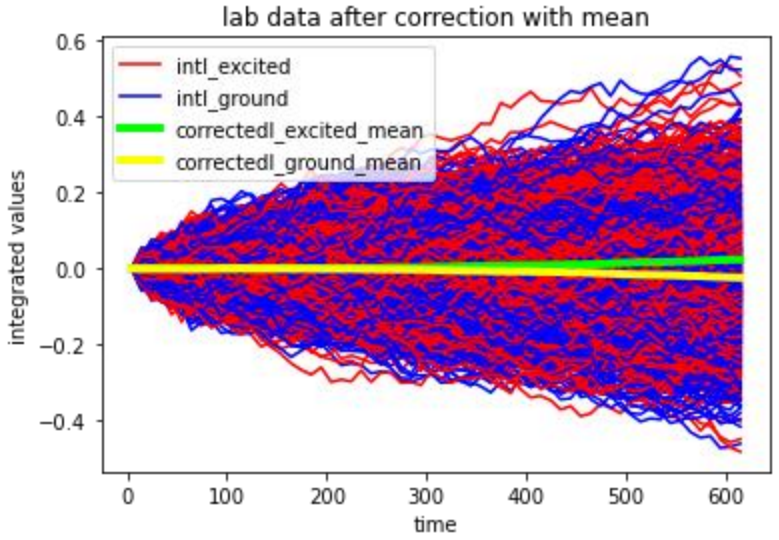
$$FPR = \frac{FP}{(FP+TN)} \qquad FNR = \frac{FN}{(FN+TP)}$$

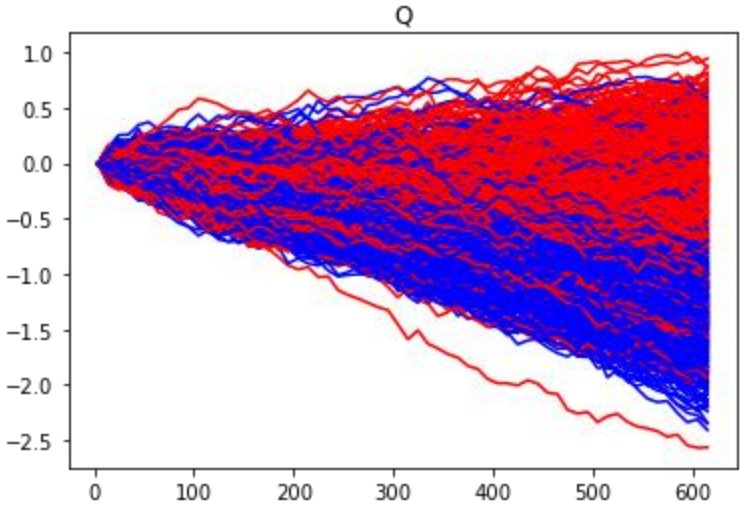
	Accuracy	Fidelity
Model A	0.94	0.74
Model B	0.90	0.89

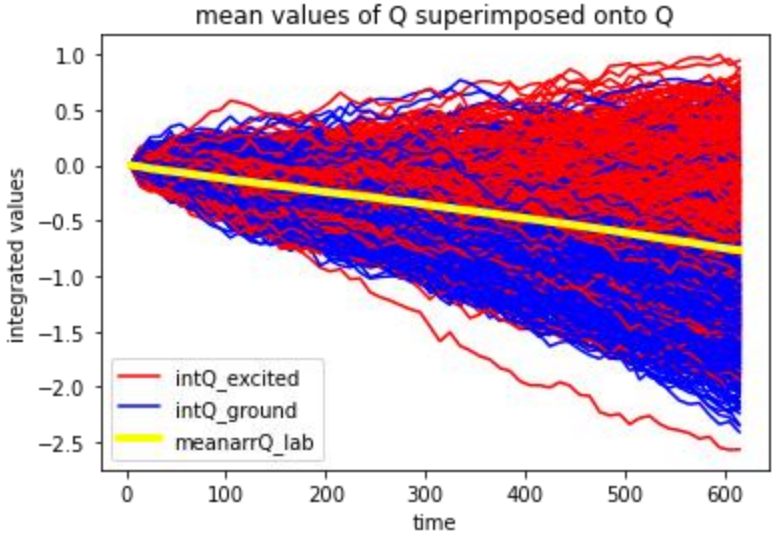
Scaling and Mean

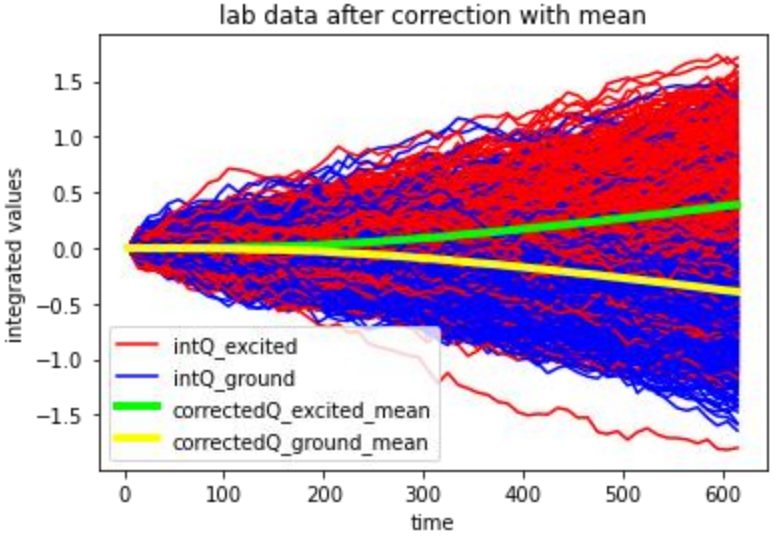












Weights – From Johann´s thesis

$$k_{x,t} = \frac{\langle X_0 - X_1 \rangle}{\mathrm{Var}(\Delta S)}$$

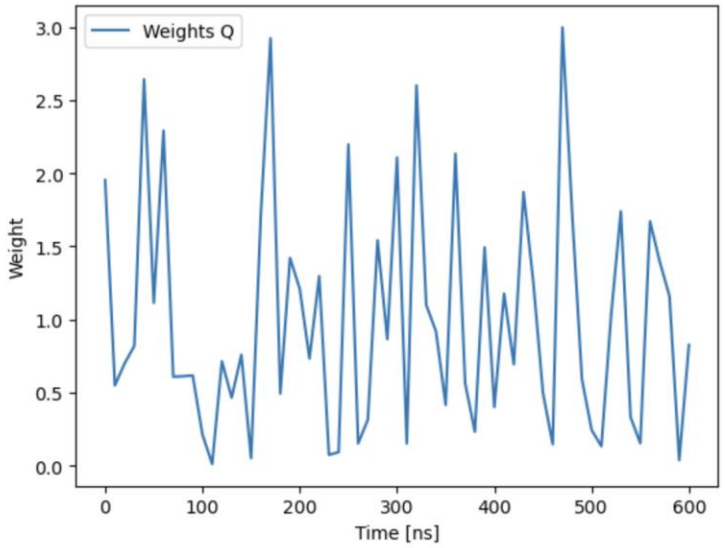
These weights maximizes the signal to noise ratio, and essentially seperates the Gaussians that I,Q values are drawn from the most.

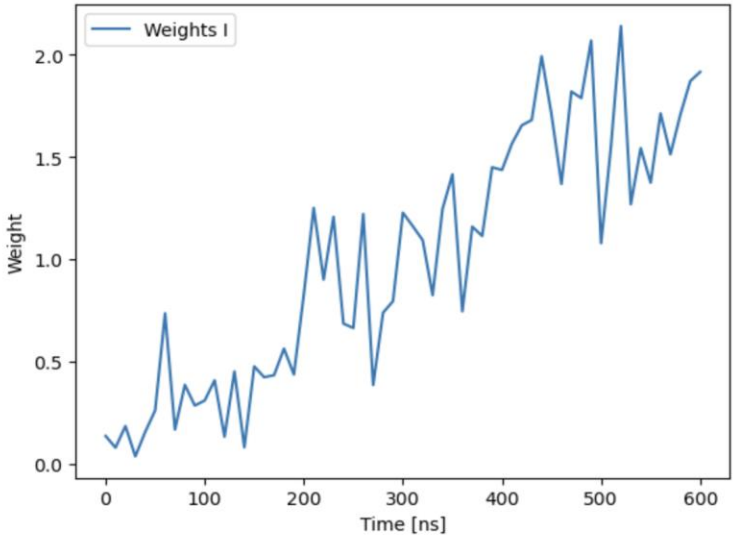
<X_0 – X_1> = np.mean(X_0) – np.mean(X_1)
Var(DeltaS) = np.var(np.mean(X_0) – np.mean(X_1))

Here X can be I and Q, so weights for both Q and I are calculated for each timestep.

Integrating, weighting, scaling, and shifting data

- I and Q values are integrated in time, then scales with the analytical weights from Johanns master thesis. See calculated weights:





Example of data preprocessing

- Fig 1->2: Sklearns TimeSeriesMinMaxScaler() is used to scale the trajectories.
- Fig 2->3: Each trajectory is then shifted so it begins in 0

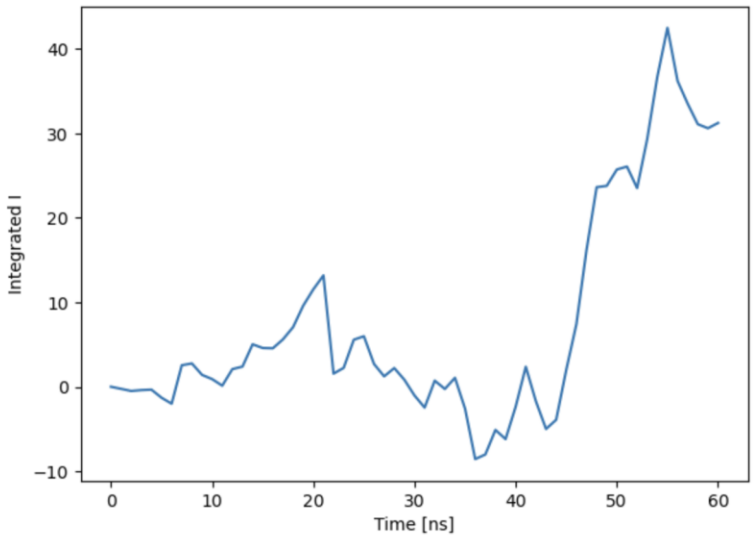


Fig 1

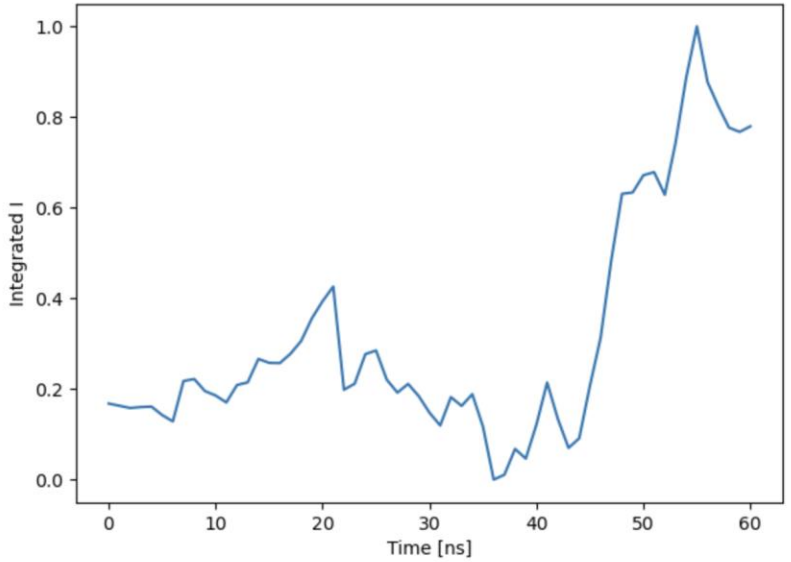


Fig 2

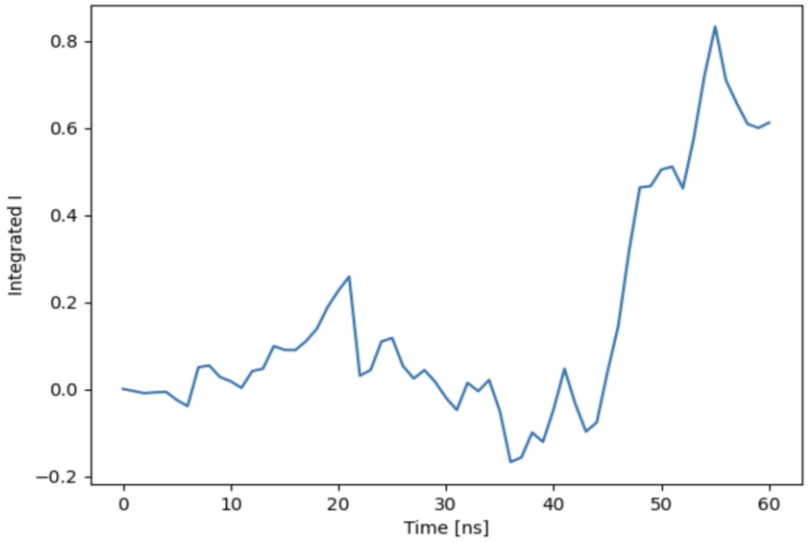
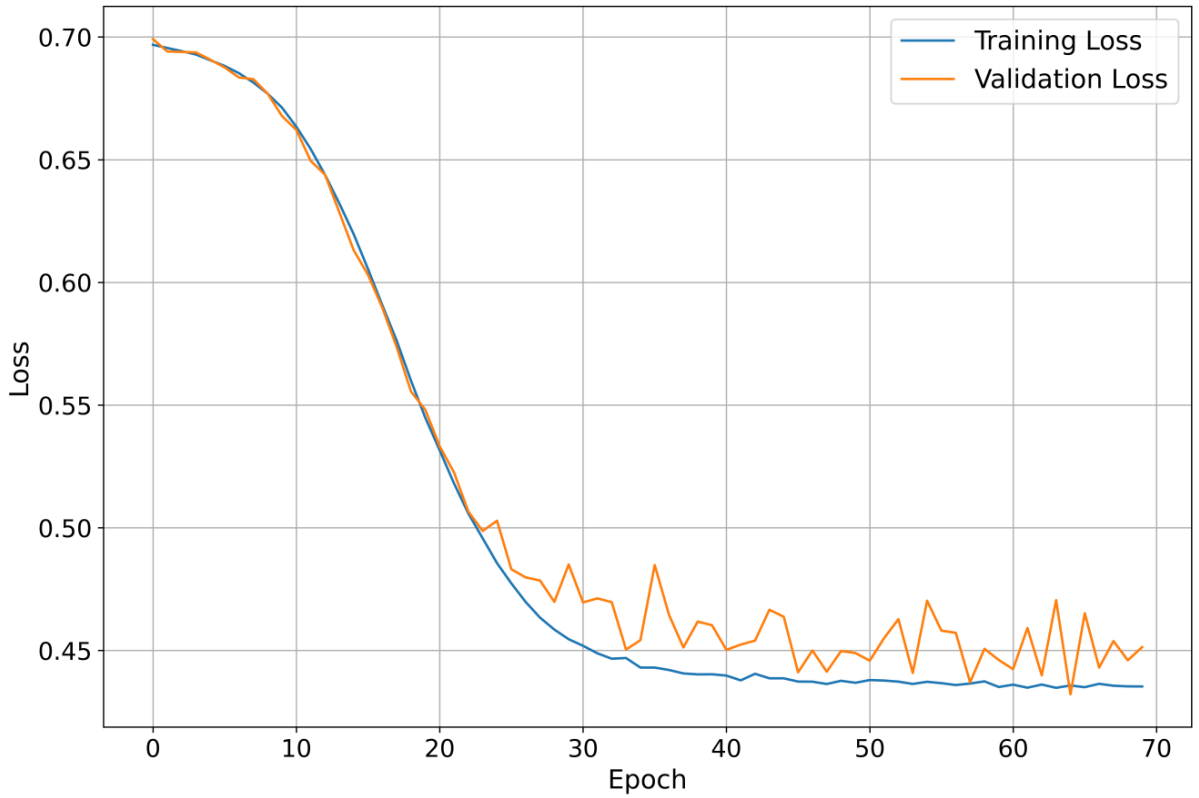


Fig 3

Convolutional1D classifier

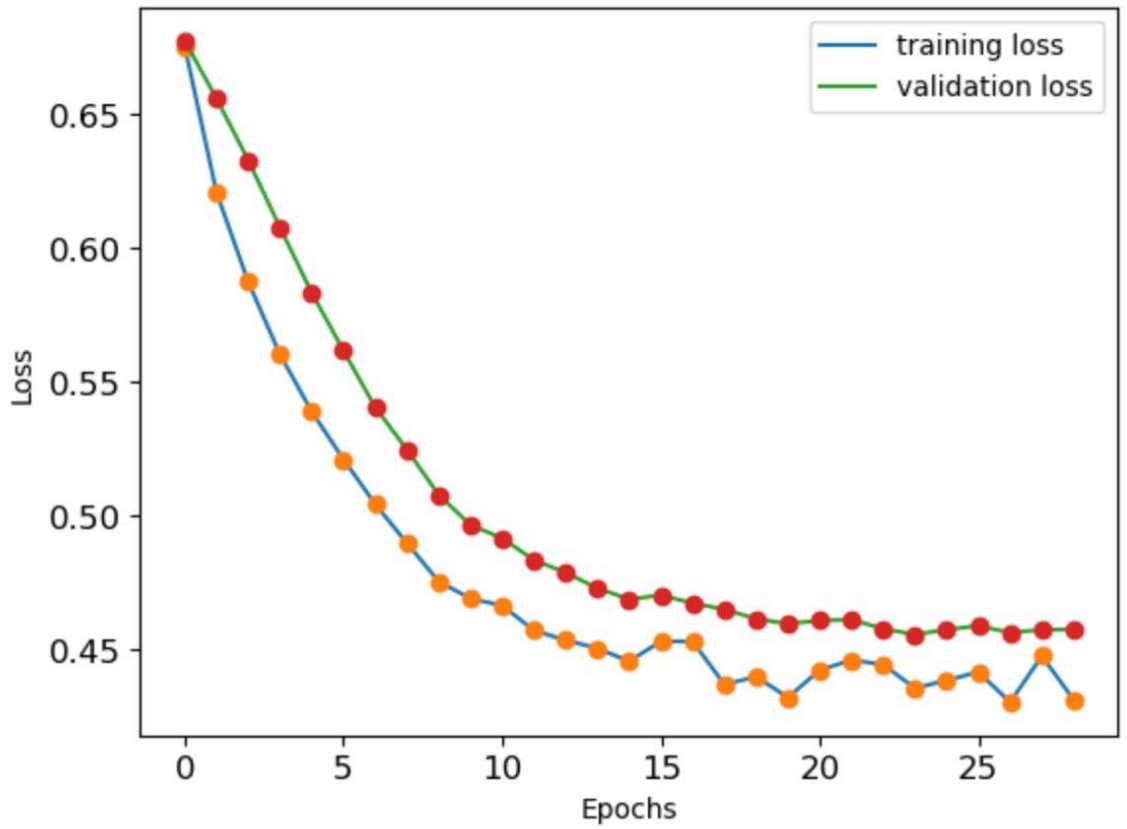
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv1d: 1-1                            [-1, 16, 61]              112
├─MaxPool1d: 1-2                         [-1, 16, 30]              --
├─Conv1d: 1-3                            [-1, 32, 30]              1,568
├─MaxPool1d: 1-4                         [-1, 32, 15]              --
├─Linear: 1-5                            [-1, 32]                  15,392
├─Linear: 1-6                            [-1, 1]                   33
├─Sigmoid: 1-7                           [-1, 1]                   --
==========================================================================================
Total params: 17,105
Trainable params: 17,105
Non-trainable params: 0
Total mult-adds (M): 0.07
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.07
Estimated Total Size (MB): 0.08
==========================================================================================
```



LSTM classifier

Layer (type)	Output Shape	Param #
batch_normalization_542 (BatchNormalization)	(None, 61, 2)	8
lstm_385 (LSTM)	(None, 16)	1216
dense_889 (Dense)	(None, 4)	68
dropout_233 (Dropout)	(None, 4)	0
batch_normalization_543 (BatchNormalization)	(None, 4)	16
dense_890 (Dense)	(None, 4)	20
dense_891 (Dense)	(None, 1)	5

Total params: 1,333
Trainable params: 1,321
Non-trainable params: 12

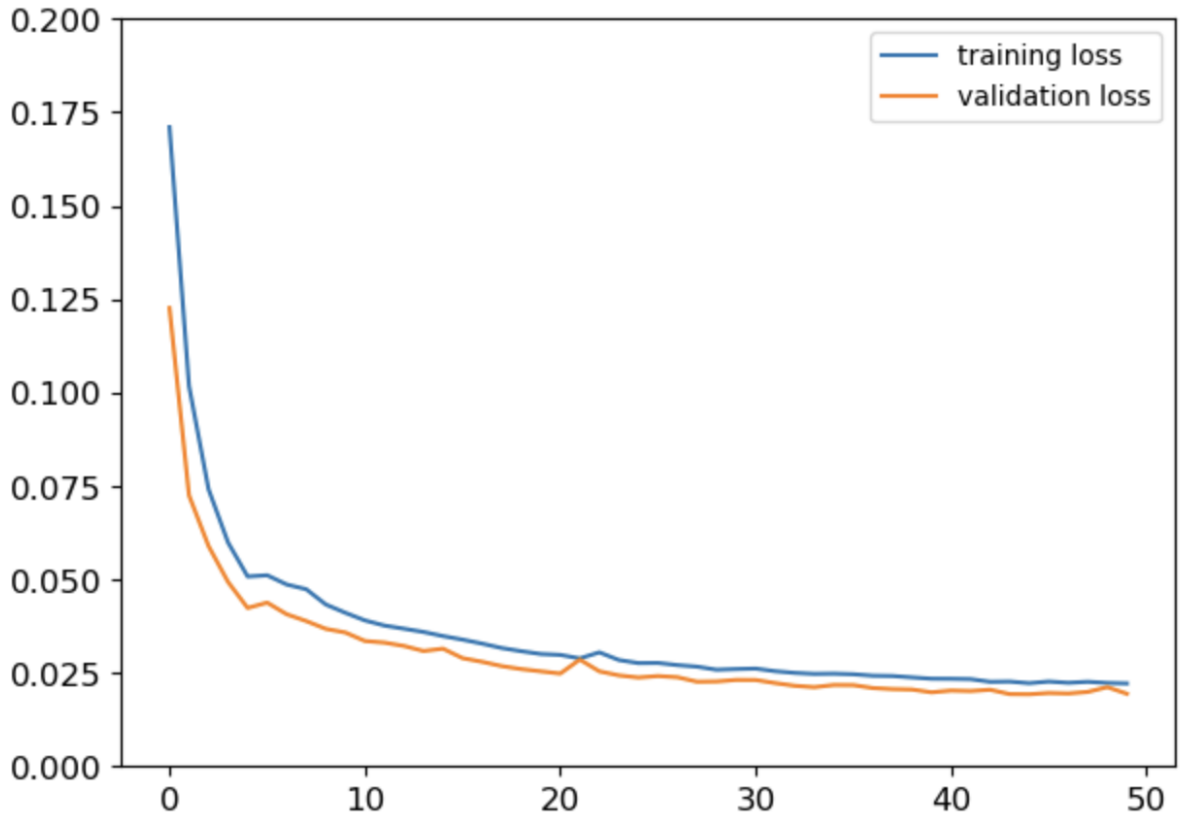


LSTM auto-encoder

Contains a lot more trainable parameters than the classifiers, as we want to predict entire trajectories

Layer (type)	Output Shape	Param #
input_103 (InputLayer)	[(None, 61, 2)]	0
lstm_408 (LSTM)	(None, 61, 64)	17152
lstm_409 (LSTM)	(None, 32)	12416
repeat_vector_98 (RepeatVector)	(None, 61, 32)	0
lstm_410 (LSTM)	(None, 61, 32)	8320
lstm_411 (LSTM)	(None, 61, 64)	24832
dropout_38 (Dropout)	(None, 61, 64)	0
time_distributed_97 (TimeDistributed)	(None, 61, 2)	130

Total params: 62,850
Trainable params: 62,850
Non-trainable params: 0



Clustering used to train and validate our "perfect" model. Note this