Aalto University
School of Science
Department of Applied Physics

Mathias Smeds

Hyperparameter optimisation for deep learning of spectroscopy

Special Assignment Report

PHYS-E0441 — Physics Special Assignment V

Report submitted for approval: May 31, 2019

Supervisor: Professor Patrick Rinke

Instructor(s): M.Sc. Kunal Ghosh, Dr. Milica Todorović

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of Physics Special Assignment

Author Mathias Smeds	
Title of the work Hyperparameter optimisation for deep learning of spectroscopy	
Degree programme Engineering Physics and Mathematics	
Major Engineering Physics	**Code of major** SCI3056
Supervisor Professor Patrick Rinke	
Instructor(s) M.Sc. Kunal Ghosh, Dr. Milica Todorović	
Date May 31, 2019 **Number of pages** iv+19+II	**Language** English

Abstract

Advances in computational power and algorithms have made computational driven science a part of materials science. Numerical approximations for solving quantum mechanical equations have laid the groundwork, and recently machine learning has produced promising results by adding value to previous research.

In this special assignment we focus on filling in gaps in knowledge from previous research on using convolutional neural networks for predicting photoemission spectra for small organic molecules. We explore the results of varying two parameters of the input data, the Gaussian broadening and datapoint density of the target spectra, as well as varying the loss function.

Keywords Spectroscopy Convolutional Neural Networks Materials Science

Contents

1 Introduction

Spectroscopy describes how materials react when perturbed. Depending on the kind of spectroscopy used, this perturbation can be caused either by mechanical force or electromagnetic radiation. The response of the material gives insight into its electron structure, and therefore also whether it would be suitable as a novel material for improving the performance of already available or even completely new applications.

Spectroscopy has a strong theoretical background that follows from the quantization of energy and the energy levels of electrons in the atom. A certain material will absorb and emit radiation at very specific energies, related to the energy levels of the electrons. These energies can be found by exposing the material to a spectrum of electromagnetic radiation and measuring which energies are absorbed or emitted. As these energies are specific to the material, a certain material can be identified by its energy spectrum.

These electron energy levels can also be numerically calculated from first principles, using quantum mechanical methods, such as density functional theory (DFT). Computational spectroscopy enables the materials scientist to cover a larger range of potential materials compared to experimental spectroscopy, without the need for expensive facilities and measurement devices. However, there is always a trade off between accuracy and computational cost. Despite advances in computational power, numerical quantum mechanics is still very much limited by the fact that the computational cost scales poorly with system size. Because of this scaling, numerical solutions that agree with experimental results are limited to small systems [1].

With global warming and an increased strain on the availability of natural resources, optimal materials are invaluable in addressing global challenges. One example is the need for breakthroughs in solar cell materials in order to reduce the utilization of fossil fuels [2]. As a way to make spectroscopy quicker to compute, scientists are aiming at developing methods based on advances in data science, more specifically machine learning and neural networks, to predict spectra. A well trained neural network should be able to quickly predict a spectra based on an input representation of the material. This quick spectra prediction would in turn enable scientists to quickly screen a database for suitable spectra, which would be useful when looking for novel materials for new applications, such as the solar cell example above.

Ghosh et al. [3] [4] have created a convolutional neural network (CNN) that can predict photoemission spectra that differ by 4.5% from the theoretical spectra. The predictions for 10000 molecules take less than a second, which is several orders of magnitude quicker than the time the theoretical spectra took to calculate on a computer cluster. In order learn to provide accurate predictions, a CNN has to be trained. The training of the CNN is based on the input data as well as network structure.

This project focuses on exploring how the training, and thus prediction accuracy, varies based on different choices of parameters in the input data and network

structure. For the input data we focus on the target spectra, using different values of Gaussian broadening (a measure for peak width) and different datapoint densities. For the CNN structure we explore the effect of using different loss functions during training. A loss function measures the difference between the predicted and target spectrum. Different loss functions emphasize certain differences between the prediction and target more than others, which is the reason for us wanting to find a loss function especially well suited for spectra. Some predicted spectra in [4] show negative intensities, which should not be present. A different loss function might perform better with regards to predicting strictly non-negative spectra.

The main objective is a deeper understanding of how the performance of the CNN changes when varying the Gaussian broadening, datapoint density and loss function. We expect a lower Gaussian broadening to be harder to train, due to the spectra having a higher amount of visible peaks, and thus essentially more information. The same theory applies to increasing the datapoint density, a higher datapoint density contains more information which should be harder to train. For the loss function we hope to find a loss well suited for the task of measuring the differences between two spectra.

Based on insights from this research, scientists can hopefully improve on the design of the CNN to gain better accuracy, and possibly also predict other properties than the photoemission spectrum.

2 Methodology

2.1 Datasets

2.1.1 Input data - Material representation

The molecular dataset used is based on the QM9 dataset by [5], which contains 133885 molecules built up of nine or fewer C, N, O and F atoms. This dataset was further preprocessed for neural networks by optimizing the geometries to obtain the most stable structure [4]. Molecules where the preprocessing didn't converge, or left fewer than 16 energy states for highest occupied electron orbitals (HOMO), were removed, leaving 132531 molecules. We are looking for the 16 highest occupied electron orbitals, which will be represented by the 16 peaks with the highest energies.

This dataset was further parsed into a matrix format better suited for machine learning, and especially convolutional neural networks. The material representation is called a Coulomb matrix (CM). For two atoms with atom numbers Z_i and Z_j as well as coordinates R_i and R_j the CM is given by equation 1. If there are less than 29 atoms in total, the matrix is zero-padded into a 29×29 matrix. An example CM for C_4H_9NO can be seen in figure 1.

$$C_{ij} = \begin{cases} \frac{1}{2} Z_i^{2.4}, & i = j \\ \frac{Z_i Z_j}{\|R_i - R_j\|_2}, & i \neq j \end{cases} \tag{1}$$

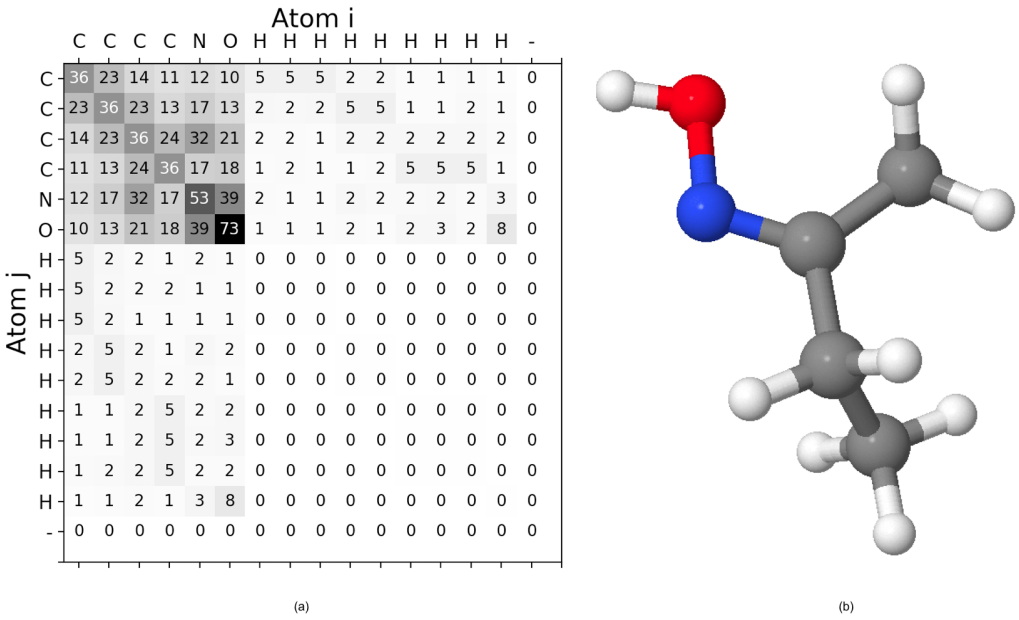

Figure 1: Representations for C_4H_9NO. (a) A 16×16 submatrix of the Coulomb matrix. Rows and columns 17-29 which are not shown have the value 0 everywhere. Atoms are indicated with symbols C, N, O and H. - indicates no atom and zero-padding. (b) Ball-and-stick model.

The Coulomb matrices are randomized before training. The randomization scheme proposed by [6] is used, which involves sorting by row norms with added noise. These randomized Coulomb matrices are the input used for the convolutional neural network, which is described in section 2.2.

2.1.2 Input data - Target spectra

With the help of large computational resources the electron energy levels and corresponding delta functions have been calculated for the preprocessed molecules in the QM9 dataset. These energy levels can be described by a spectrum, which is generated from the delta functions following the steps in figure 2.

For the work in [3] and this project the delta functions corresponding to the 16 HOMO energies are selected (2a, 2b) and Gaussian functions are superposed on the delta functions (2c). Finally the magnitudes of the Gaussians are summed to create the spectra (2d) that will be the output target for the convolutional neural network.

The smoothness of the spectra depends on both the Gaussian broadening used as well as the amount of datapoints that make up the spectra. The width of the Gaussian function is described by the parameter σ with unit eV. A smaller value of σ produces a narrower and sharper peak, which in turn causes a less smooth spectra. The spectra can be smoothed by increasing the amount of datapoints, Np, possibly at a cost of being harder for the network to predict. From figure 3 it is evident that a larger broadening gives a smoother spectra with lower peaks.

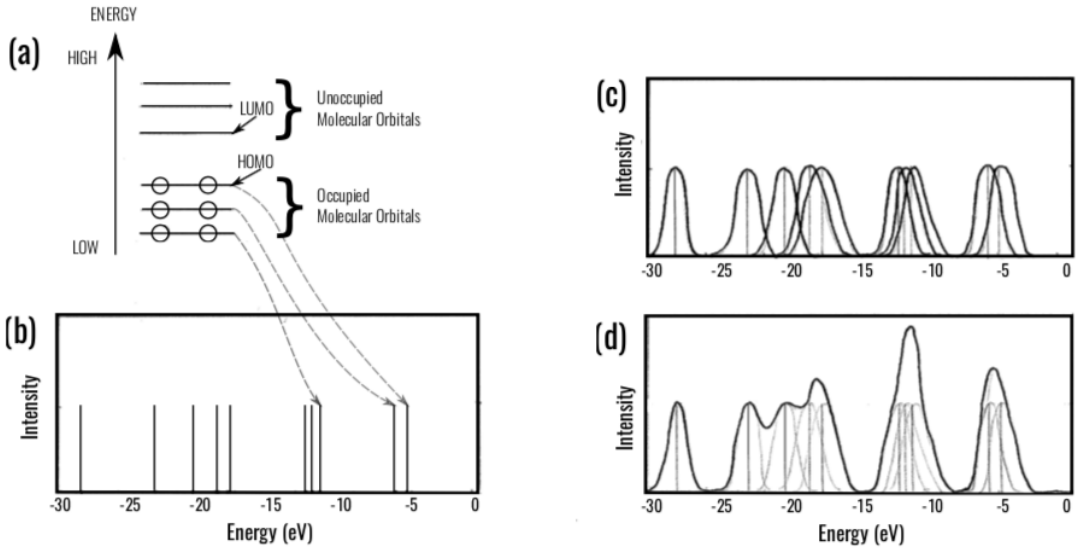

Figure 2: From energy levels to spectra. (a) Finding the 16 highest HOMO energies. (b) The delta functions corresponding to the HOMO energies. (c) Superposing Gaussian functions on the delta functions. (d) Summing the Gaussians. Image taken from [3]

Figure (4a) shows the lefttmost spectra from figure 3 plotted with different datapoint densities. Judging by eye there is little difference even when more than

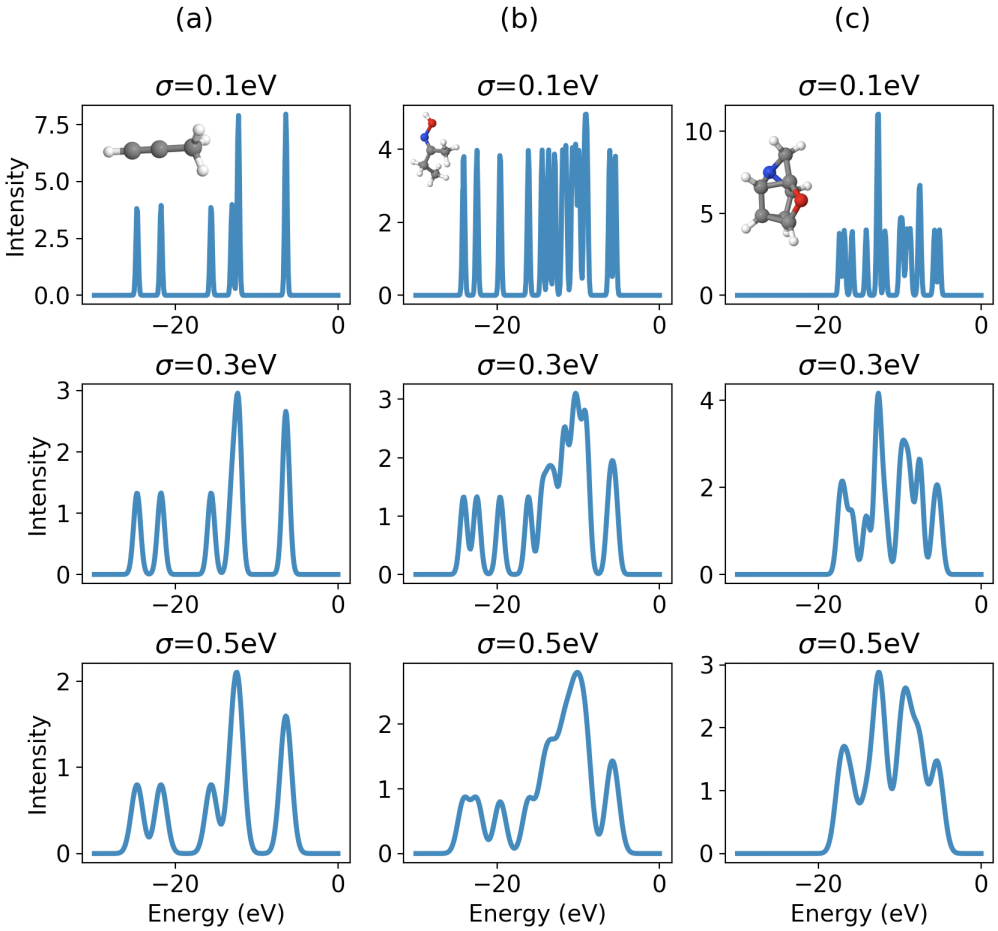

Figure 3: Comparison of three spectra with different Gaussian broadenings. Column (a), (b) and (c) represent the 3rd, 17th and last molecules of the QM9 dataset respectively.

doubling the datapoint density from 300 to 700. Zooming in on the rightmost peak in figure (4a) gives the figure (4b), which reveals a difference in smoothness at the peak, where the spectra changes most rapidly.

2.2 Convolutional neural network

Convolutional neural networks were shown to perform well for image recognition in the late 1990's by [7]. After 2012 they have become the state of the art for image recognition [8]. As the input CM can be thought of as a grayscale image of a molecule, it is reasonable to assume that CNN's might perform well for the task of predicting spectra.

The convolutional neural network used for this project is the same as in [3]. Its structure is described in figure 5 and it can be divided into 4 parts. The first three parts have three convolutional layers and one max pooling layer each. The convolutional layers use a 3×3 kernel and a zero-padding of dimension 1, while the max pooling uses a 2×2 kernel. Each layer, except for the fully connected layer,

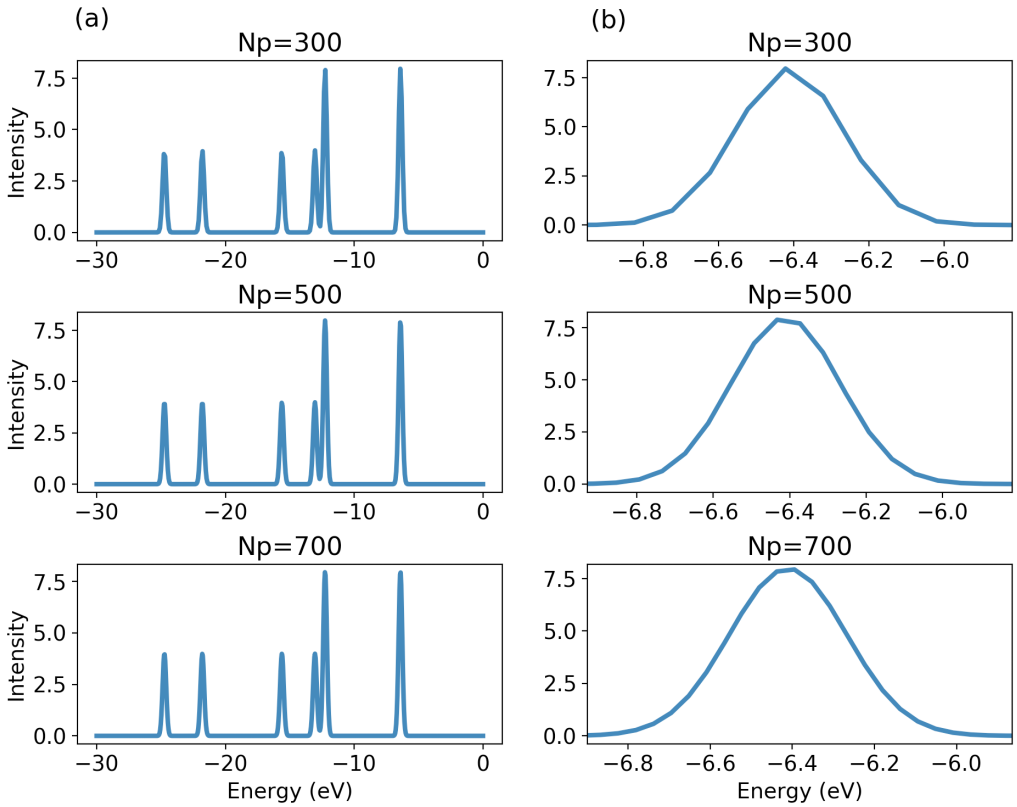

Figure 4: (a) Comparison of the same spectra with different datapoint densities. (b) Comparison of the smoothess of the rightmost peak

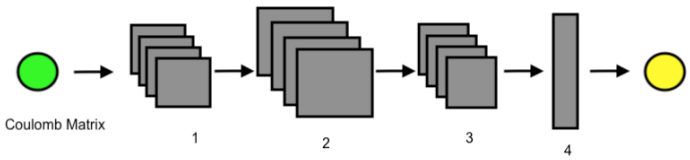

Figure 5: Structure of the CNN.

uses the rectified linear unit (ReLu) activation function before feeding values into the next layer. Part 1 has 22 filters, part 2 has 47 filters and part 3 has 42 filters. The fourth part is a fully connected linear layer that produces the final output for the Np-datapoint spectra.

More information on both the dataset and design of the neural network can be found in [4].

2.3 Training and hyperparameters

2.3.1 Training procedure

Neural networks consist of nodes and weights, also called model parameters. Each node is connected to other nodes via weights. The input to the network is copied to the input nodes, multiplied with the weights connected to the input nodes and propagated forward to the next nodes, which are again multiplied with new weights and so on. This is called forward propagation. Forward propagation is done until the output nodes are reached. The values of the output nodes is the output of the network. This procedure from input to output is called one pass through the network.

After one pass we can calculate the difference between the actual output of the network and the desired output. In our case the output is a predicted spectrum and the desired output is the target spectrum. The difference is calculated with a loss function. The choice of loss function determines how much various of differences add to the loss. One common loss function, root mean square error (RMSE), adds a larger loss for datapoints where the error is big, compared to another common loss function, mean absolute error (MAE).

The training procedure strives to tune the network by changing the weights to produce the most accurate output possible. The weights are tuned by making a pass through the network, calculating the loss with a loss function and finally making small changes to the weights depending on the partial derivative of the loss function with respect to each weight. The magnitude of the change is proportional to the learning rate used.

During training we iterate through the whole dataset multiple times, and update the weights after calculating the loss for a number of passes, instead of updating the weights after one pass. This is called training in batches. The number of passes before weight updates is called the batch size. One iteration through the whole dataset along with updating the weights is called an epoch.

In this project we assess the accuracy achieved by the CNN trained with different hyperparameter settings. We focus on the three sets of hyperparameters listed below.

- Gaussian broadening
- Datapoint density
- Loss function

Both Gaussian broadening and datapoint density affect the dataset, while the loss function is part of the CNN training procedure.

In order to have comparable results, we train the network varying one hyperparameter at a time, while running the training for 2500 epochs. Our default hyperparameter values are RMSE for loss function, broadening $\sigma = 0.5$ and datapoint density $Np = 300$. The default broadening of $\sigma = 0.5$ was chosen so that the spectra are close to experimental spectra, while $Np = 300$ was chosen based on a theory that a smaller output dimension is easier to predict.

We train the CNN using the Adam optimizer [9] with $\beta = (0.9, 0.999)$ and a learning rate of 10^{-4}. 90% of the dataset is used for training, 5% for testing and 5% for assessing the performance. Originally [3] used the RMSE as the loss function. For each epoch we measure the average loss for both the training and testing part of the dataset, calling them train loss and test loss respectively.

We assess the performance with four different measures.

- Accuracy
- RMSE
- MAE
- R^2

Accuracy is our main performance measure. It measures the total area between the predicted and target spectra, thus it takes into account both peak energy and intensity. RMSE and MAE both measure the difference in the intensity at each energy. RMSE takes the squared value of the difference, thus penalizing outliers more than MAE. R^2 is also called the coefficient of determination. It measures the goodness of fit by the proportion in the variance of the prediction that is predictable from the target output.

Accuracy and R^2 are independent of the absolute intensity of the spectrum. Thus they stay the same during scaling, for example when normalizing. RMSE and MAE are absolute measures, and thus change when the spectrum is scaled. This gives different results when looking at normalized versus unnormalized spectra, even though the accuracy itself doesn't change when normalizing.

The metrics for measuring performance are defined below, where T_i are the elements of the true spectra, P_i are the elements of the predicted spectra and dE is the energy step between two datapoints. $\bar{P}$ is the mean value of elements in P and n is the number of datapoints in T and P.

$$\text{Accuracy} = \frac{\sqrt{\sum_{i=1}^{n}(T_i - P_i)^2 dE}}{\sum_{i=1}^{n} P_i dE} \tag{2}$$

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n}(T_i - P_i)^2} \tag{3}$$

$$\text{MAE} = \frac{\sum_{i=1}^{n} |T_i - P_i|}{n} \tag{4}$$

$$\tag{5}$$

$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{n}(T_i - P_i)^2}{\sum_{i=1}^{n}(P_i - \bar{P})^2} \tag{6}$$

We also assess the negative intensities of the predicted spectra, where the predictions are made with the CNN trained by different loss functions. We measure this in two ways, the ratio of negative datapoints to total datapoints as well as total magnitude of the negative datapoints for the predicted spectrum. The total negative magnitude is calculated on normalized spectra. A spectrum should not show negative intensities, and ideally our CNN should not predict negative intensities either.

We implement the convolutional neural network in Python using PyTorch [10], which provides automatic backpropagation as well as functions for handling the data. The version used is 1.0.0. Supporting functions in the code also make use of the numpy and scikit-learn libraries.

We ran the training using Aalto University's computational resources, on computers with GPU's. Without significant other load on the computer the training took about 45 seconds per epoch on computers using an NVIDIA Quattro K220 with 4GB VRAM, Intel Xeon E3-1230 3.40 GHz CPU and 16GB RAM. This time per epoch did not vary when changing hyperparameters.

2.3.2 Training with different Gaussian broadenings

To compare how the Gaussian broadening affects the performance we train the network with three different sets of spectra, where the difference is in the broadening used. The different sets of spectra have broadenings with $\sigma = 0.1$, $\sigma = 0.3$ and $\sigma = 0.5$ respectively. The loss function is RMSE and datapoint density is $Np = 300$. We assess the performance with the measures described in section 2.3.1. The choice of $Np = 300$ and RMSE is based on the results being comparable with [3].

2.3.3 Training with different datapoint densities

We train the network with three different sets of spectra, each set having a different datapoint density, $Np = 300$, $Np = 500$ and $Np = 700$ respectively. The Gaussian broadening is $\sigma = 0.1$ and the we use RMSE as the loss function. Again, we assess the performance with the measures described in section 2.3.1. The choice of $\sigma = 0.1$ is based on getting a worst-case estimate of the accuracy at different values of Np, as we assume $\sigma = 0.1$ will be the hardest to predict. As the output dimension increases, the number of model parameters to train also increase substantially. This is displayed in table 1.

Table 1: Model parameters as a function of datapoint density.

Np	Model parameters
300	221529
500	297329
700	373129

2.3.4 Training with different loss functions

In this project we try five different loss functions, listed below.

- RMSE
- Smooth L1 loss (Smooth)
- Log Cosh loss (Logcosh)
- 1-Cos loss (Cosloss)
- Pearson correlation coefficient (Pearson)

[11] has found the latter two loss functions to work well with spectra.

RMSE is defined in equation 3. Defining T as the true spectra, P as the predicted spectra, A_i as the elements of spectra A, $\bar{A}$ as the mean of A and n as the number of datapoints in the spectra, the loss functions are defined below.

$$\text{Smooth} = \frac{1}{n}\sum_{i=1}^{n} z_i \tag{7}$$

$$\text{where } z_i = \begin{cases} \frac{1}{2}(P_i - T_i)^2, & |P_i - T_1| < 1 \\ |P_i - T_i| - \frac{1}{2}, & \text{otherwise} \end{cases}$$

$$\text{Logcosh} = \sum_{i=1}^{n} \log(\cosh(P_i - T_i)) \tag{8}$$

$$\text{Cosloss} = 1 - \frac{\sum_{i=i}^{n}(P_i T_i)}{(\sum_{i=i}^{n}(T_i^2))^{1/2}(\sum_{i=i}^{n}(P_i^2))^{1/2}} \tag{9}$$

$$\text{Pearson} = 1 - \frac{\sum_{i=i}^{n}(P_i - \bar{P})(T_i - \bar{T})}{\sqrt{\sum_{i=i}^{n}(T_i - \bar{T})^2 \sum_{i=i}^{n}(P_i - \bar{P})^2}} \tag{10}$$

2.3.5 Normalization of the predictions

The loss functions RMSE, Smooth and Logcosh all measure both the peak energy and absolute intensity. Cosloss and Pearson in turn focus only on the Np-dimensional vector for the predicted spectra having the same direction as the Np-dimensional vector for the target spectra. Thus Cosloss and Pearson measure the peak energy and relative intensity. As the accuracy measure in formula (2) is dependent on both the peak energy and absolute intensity, the predicted spectrum needs to be scaled to the same scale as the target spectrum. We asses the performance for both normalized and unnormalized predictions with RMSE, Smooth and Logcosh, and normalized predictions for the Cosloss and Pearson.

3 Results

The network managed to learn with every single hyperparameter tested. We had a four-way tie for the best loss function in terms of prediction accuracy, with only the Pearson loss showing a lower performance. Varying the datapoint density made no difference in the performance, while a Gaussian broadening of $\sigma = 0.5$ showed the best performance.

From figure 6 it is evident that both the training and testing error keep decreasing throughout the training process. All of the plots have a very similar shape, evident of good training. It is expected that the error decreases quickly during the beginning and that the improvement approaches zero after training for enough epochs. The long flat portion of the plots suggest that no significant further performance gain would be achieved by running the tranining for more epochs. Plots for the train and test losses for the experiments varying σ and Np can be found in appendix A.

3.1 Prediction accuracies

The performances of the CNN trained with the different parameters can be seen in the tables below. The performance of loss functions are in table 2, the effect of different Gaussian broadening is in table 3 and the effect of datapoint density is in table 4.

We achieved an accuracy of close to 95% for all loss functions except for Pearson, which achieved an accuracy of 76%. Decreasing σ caused a decrease in performance. Different datapoint densities had no measurable difference in performance.

For the performance in table 2 we assess the performance for both normalized and unnormalized predictions with RMSE, Smooth and Logcosh. For the Cosloss and Pearson loss functions we only assess the performance for the normalized predictions. The reason for this is that the predictions for the CNN trained with Cosloss and Pearson are not necessarily in the same scale as the target spectra, due to the fact that Cosloss and Pearson are relative and not absolute loss functions.

It is worth to note that only the accuracy and R^2 are independent of scale, and thus comparable between normalized and unnormalized predictions. In some cases of worse accuracy we get a negative R^2 score.

The different loss functions, datapoint densities and Gaussian broadenings did not have a measurable effect on the training speed. As mentioned earlier, epochs took around 45 seconds, no matter what loss function or hyperparameters we used. The input file size scales with datapoint density, so only the initial loading of the data took longer when we increased the datapoint density.

Graphical representations of the prediction accuracies for the training with RMSE, $\sigma = 0.5$ and $Np = 300$ can be found in figure 7. Figure 7(a) and 7(b) shows the best and worst spectra respectively, while figure 7(c) shows a histogram of the accuracy distribution. The best prediction has an accuracy of 98.3% while the worst prediction has an accuracy of 82.2%. The histogram shows that the bulk of the predictions have accuracies of around 95%.

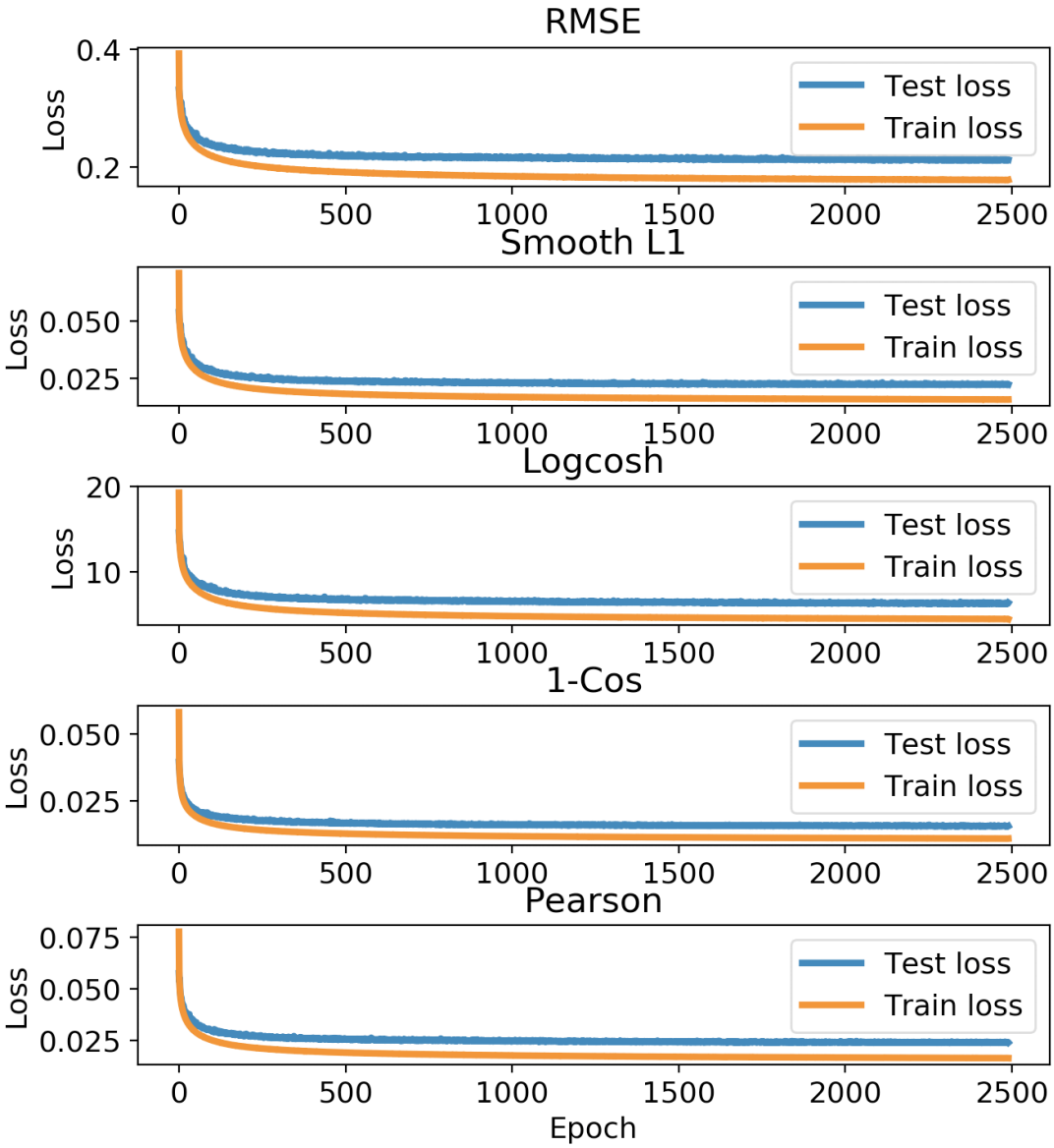

Figure 6: Train and test losses for training with different loss functions.

3.2 Negative intensities in spectra

The results for assessing negative intensities in the predicted spectra is presented in table 5. We observe that the CNN trained with the Pearson loss function performs significantly worse than the others both in terms of number of negative datapoints and magnitude of negative datapoints. The other outlier is Cosloss with one order of magnitude better performance in negative magnitude. As mentioned in section 2.3.1, we assess the negative magnitude for normalized spectra. Thus the negative magnitude can't be expressed in eV.

Table 2: Performance of different loss functions. Subindex n indicates that it is the measure for normalized predictions.

Loss function	σ	n	Accuracy	RMSE	MAE	R^2
RMSE	0.5	300	94.8%	0.209	0.111	0.948
Smooth	0.5	300	94.7%	0.210	0.112	0.948
Logcosh	0.5	300	94.8%	0.208	0.110	0.948
RMSE_n	0.5	300	94.8%	0.010	0.005	0.950
Smooth_n	0.5	300	94.7%	0.010	0.005	0.950
Logcosh_n	0.5	300	94.8%	0.010	0.005	0.950
Cosloss_n	0.5	300	94.7%	0.010	0.005	0.949
Pearson_n	0.5	300	76.3%	0.044	0.036	0.419

Table 3: Effect of Gaussian broadening on prediction accuracy.

σ	Loss function	n	Accuracy	RMSE	MAE	R^2
0.1	RMSE	300	74.1%	1.08	0.591	-0.0993
0.3	RMSE	300	90.6%	0.404	0.221	0.826
0.5	RMSE	300	94.8%	0.209	0.111	0.948

Table 4: Effect of datapoint density on prediction accuracy.

n	Loss function	σ	Accuracy	RMSE	MAE	R^2
300	RMSE	0.1	74.1%	1.08	0.591	-0.0993
500	RMSE	0.1	74.2%	1.08	0.592	-0.105
700	RMSE	0.1	74.1%	1.09	0.595	-0.1069

Table 5: Ratio of negative intensities and average negative magnitude in predicted spectra.

Loss function	Negative intensities	Negative magnitude (10^{-6})
RMSE	30.4%	138
Smooth L1	21.4%	112
Logcosh	30.9%	126
Cosloss	25.7%	6.23
Pearson	65.6%	280

3.3 Other output

The results and output of the project also include the code produced https://github.com/modjas/CNN_spectroscopy, trained PyTorch models and spectra predictions for further use by the CEST group at Aalto University.

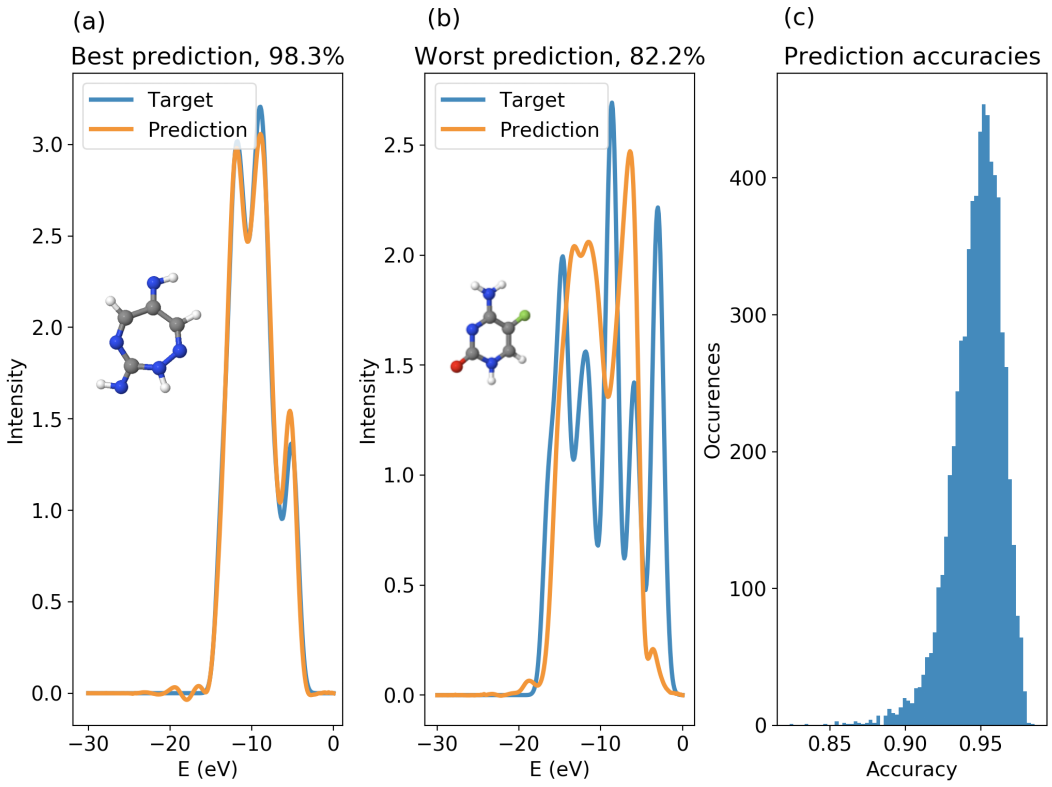

Figure 7: Graphical representation of prediction accuracies when training with RMSE $\sigma = 0.5$ and $Np = 300$. (a) Prediction and target for the best prediction. (b) Prediction and target for the worst prediction. (c) Histogram of accuracy distribution. The bin size is 0.25%.

4 Discussion

From the figures showing plots of train and test losses it is evident that the network did in fact learn to predict spectra with all hyperparameters. Looking at the results in tables 2, 3 and 4 we observe that we were not able to find better hyperparameters in terms of prediction accuracy than what [4] used. RMSE, $\sigma = 0.5$ and $Np = 300$ still produce the best results. We observe that different choices of datapoint densities did not change the performance at all, and that the Gaussian broadening value $\sigma = 0.5$ proved to be the best.

Our experiments revealed that the predictions are sensitive to the Gaussian broadening used, which is in line with our expectations. We achieved the best performance with the value of $\sigma = 0.5$. As previously mentioned, the value of $\sigma = 0.5$ was chosen in order to have the spectra be as close to typical experimental spectra as possible. This value is already blurring the underlying delta functions quite a bit. The original delta functions are less obvious when looking at a spectrum with $\sigma = 0.5$ compared to a spectrum with $\sigma = 0.1$. We are, in a sense, loosing information when increasing σ. With this in mind, it is not unexpected that we achieve a better accuracy when predicting spectra with less information density. Based on the work in this project, it is unclear which value of σ produces spectra which contain the most accurate information about the underlying delta-functions, and thus electron energy levels. There might be an optimal σ to maximise the information content in the final predicted spectra, where the tradeoff is between the information content in the target spectra and the accuracy of the CNN. This optimization is left for further research.

Even though the model parameters increased significantly, the epoch time and prediction accuracy stayed essentially constant when varying the datapoint density. This is surprising, and indicates a redundancy somewhere that could be exploited. Perhaps the CNN could be used to predict other outputs in the addition to the spectrum at the same time. This is possibly the most interesting result of the project, with the most potential for further research. The theory of a higher information density being harder to predict does not apply here, as a higher datapoint density obviously contains more information.

The accuracy performance of all different loss functions except for one was almost exactly the same, with the accuracy being constant 95%. The outlier, Pearson loss, had an accuracy of 76%. This similarity in accuracy was not expected, since we believed the loss functions to be fundamentally different. As explained previously, RMSE, Logcosh and Smooth L1 loss all measure the absolute difference between prediction and target, while Cosloss and Pearson measure the similarity of the Np-dimensional vector.

It turns out that the loss functions performing at the same level are not so different after all. RMSE is obviously quadratic and Smooth L1 loss is quadratic at small values per definition. From figure 7 we observe that even for the worst prediction the error is almost always under the threshold for Smooth L1 loss to go into the linear territory. Expanding the Taylor series for Logcosh and Cosloss reveal that the first term for both is quadratic, with the second term being proportional to

the fourth power of the error. Thus for small errors, the loss function is essentially a quadratic function for both Logcosh and Cosloss.

We have thus unintentionally retrained the network with a quadratic loss function over and over again. For the Pearson loss we don't have a quadratic approximation at small differences, which is the probable reason for differences in performance.

It is surprising that even though the accuracy was essentially the same for all loss functions except for Pearson, there are differences when looking at the negative intensities predicted. Smooth performs the best in terms of minimizing amount of datapoints with negative intensities, while Cosloss is significantly outperforming the rest in terms of magnitude of the negative values by one order of magnitude. The order of magnitude for amount of negative datapoints is the same over all loss functions, with Pearson performing significantly worse than the rest. We were not able to find a loss function completely eliminating negative values, but both Smooth and Cosloss show better results than RMSE used by [4]. As Cosloss and Pearson strive to keep the direction of the Np-dimensional vector for the predicted spectrum and target spectrum the same, they were good candidates for minimizing the negative intensities and magnitude. Cosloss turned out to perform well as expected, but Pearson showed worse performance.

The Smooth L1 loss is a special case of Huber loss. Huber loss takes the threshold between the linear and quadratic parts as an input parameter. Retraining with Huber loss, we could decrease the quadratic interval to be the less dominant one for the scale of our spectra. This way we could avoid retraining with a quadratic loss function. A similar understanding of using a quadratic versus linear loss function could also be achieved by training with MAE as the loss function. It is possible that spectra prediction would perform differently with a linear loss function. This is, however, left for further research.

5 Conclusion

In this project we present results for optimizing three hyperparameters for predicting spectra using convolutional neural networks. The hyperparameters we optimized are the loss function, Gaussian broadening and datapoint density.

We found that the performance in terms of accuracy did not change with output dimension, and was constant for several different loss functions. The performance was the best with output spectra using a Gaussian broadening parameter that reflects experimental spectra well. Narrower broadening turned out to be significantly harder to predict.

We suggest further work based on insights from the computational time and accuracy being constant for varied datapoint densities. Furthermore, the research on different loss functions is not complete yet, especially in terms of non-quadratic loss functions and loss functions that could possibly produce non-negative spectra.

References

[1] M. Rupp, "Machine learning for quantum mechanics in a nutshell," *International Journal of Quantum Chemistry*, vol. 115, no. 16, pp. 1058–1073, 2015.

[2] S. Chu and A. Majumdar, "Opportunities and challenges for a sustainable energy future," *nature*, vol. 488, no. 7411, p. 294, 2012.

[3] K. G. et al., "Deep learning spectroscopy: Neural networks for molecular excitation spectra," *Advanced Science*, vol. 6, no. 9, 2019.

[4] K. Ghosh, "Deep learning for predicting molecular electronic properties." (Unpublished Master's Thesis), 2017.

[5] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, p. 140022, 2014.

[6] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. Von Lilienfeld, "Machine learning of molecular electronic properties in chemical compound space," *New Journal of Physics*, vol. 15, no. 9, p. 095003, 2013.

[7] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[11] Y. Iwasaki, A. G. Kusne, and I. Takeuchi, "Comparison of dissimilarity measures for cluster analysis of x-ray diffraction data from combinatorial libraries," *npj Computational Materials*, vol. 3, no. 1, p. 4, 2017.

A Traintest plots

A.1 Convergence of train and test losses

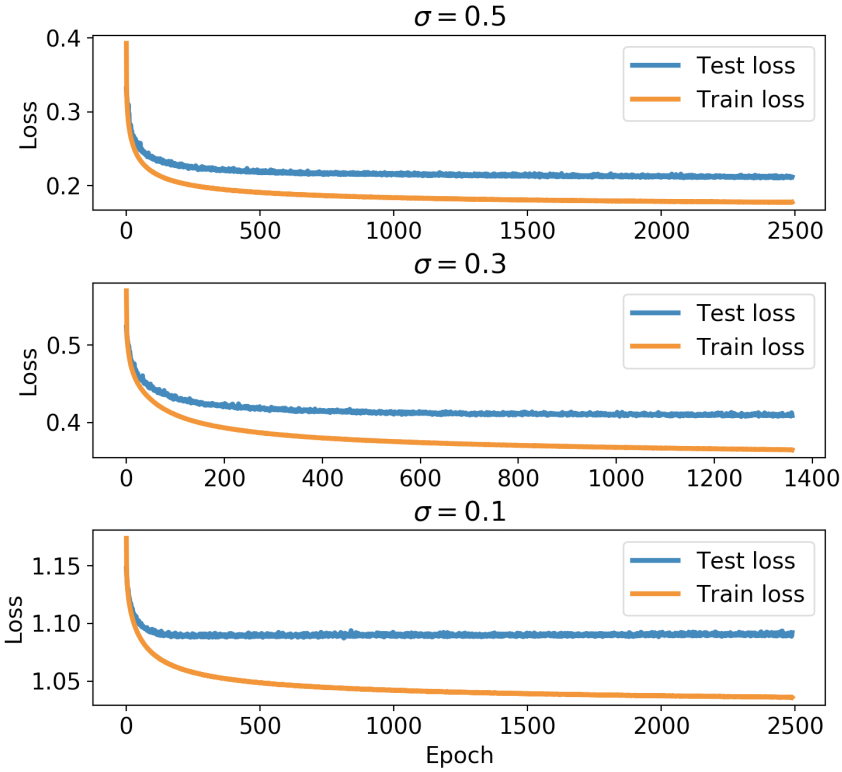

Figure 8: Train and test losses for training with different broadenings.

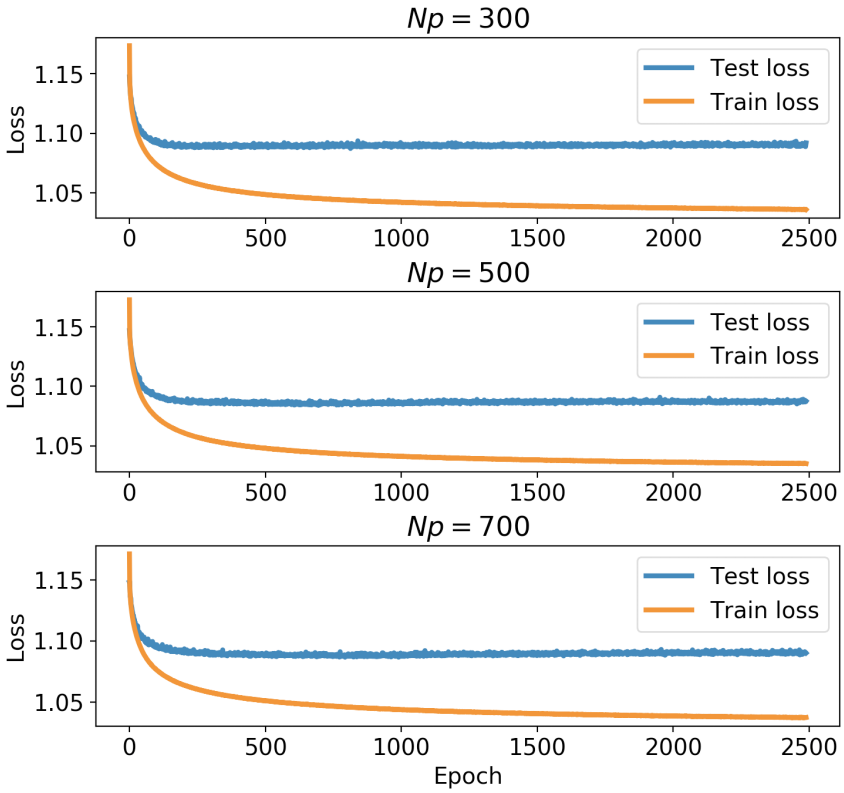

Figure 9: Train and test losses for training with different datapoint densities.