Daniel Lukic

Neural Networks in Computational Chemistry: First tests of pattern recognition for the potential energy surface of butadiene

Bachelor Thesis

to achieve the university degree of

Bachelor of Science

Bachelor degree programme: Physics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Mag.phil. Dipl.-Ing. Dr.phil. Dr.techn. Andreas Hauser

Institute of Experimental Physics
Head: Univ.-Prof. Dipl.-Phys. Dr.rer.nat. Wolfgang Ernst

Graz, November 2017

Abstract

This Bachelor thesis studies the pattern recognition abilities of neural networks in the context of Molecular dynamics simulations. A neural network should be able to fit every continuous functions on the Euclidean space with a single hidden layer and a finite number of neurons according to the Universal approximation theorem. After training, the presented neural network should be able to predict.

The first two problems focus on the training of a neural network for binary pattern recognition. We investigate a simple XOR logical gate and a binary pattern problem where one of three inputs is dominating.

In the main part of this thesis a neural network is trained to be able to predict the energy of the molecule from structural data of this molecule. The structural data and the energies (Potential Energy Surface) of the molecule are obtained with Quasiclassical Ab-Initio Molecular Dynamics and Velocity Verlet integration.

Contents

List of Figures

1 Introduction

An artificial neural network can be described as a group of simple processing units which are connected with each other and send signals over a weighted connection.[1] This can be described as follows: An input with a numerical value from the neighbor or an external source reaches the processing unit. The processing unit multiplies the input with a certain weight and propagates it to the next node. The weights of the processing units are updated with respect to the calculated error and the learning algorithm.[2]

1.0.1 Feedforward neural neural network

A Feedforward neural network (FNN) consists of an input layer, at least one hidden layer and an output layer, where each layer is constructed by nodes as illustrated in figure 1.1. The number of the nodes in one layer depends on the kind of layer and given problem to solve. For the input layer, the number is determined by the attributes of the data set, and the number in the hidden layer(s) has to be determined first, which takes place during the training of the neural network (NN). The number of nodes in the output layer depends on the nature of the problem. For a regression problem the output layer consists of one node and for classification problem there are as many nodes as classes.[3]

The propagation of the data starts with the input layer, proceeds then to at least one hidden layer and then to the output layer as can be seen in figure 1.1. This propagation can also be described as a series of functional transformations.[4]

[1] Kröse et al., 1993.

[2] Kröse et al., 1993.

[3] Bishop, 2007.

[4] Bishop, 2007.

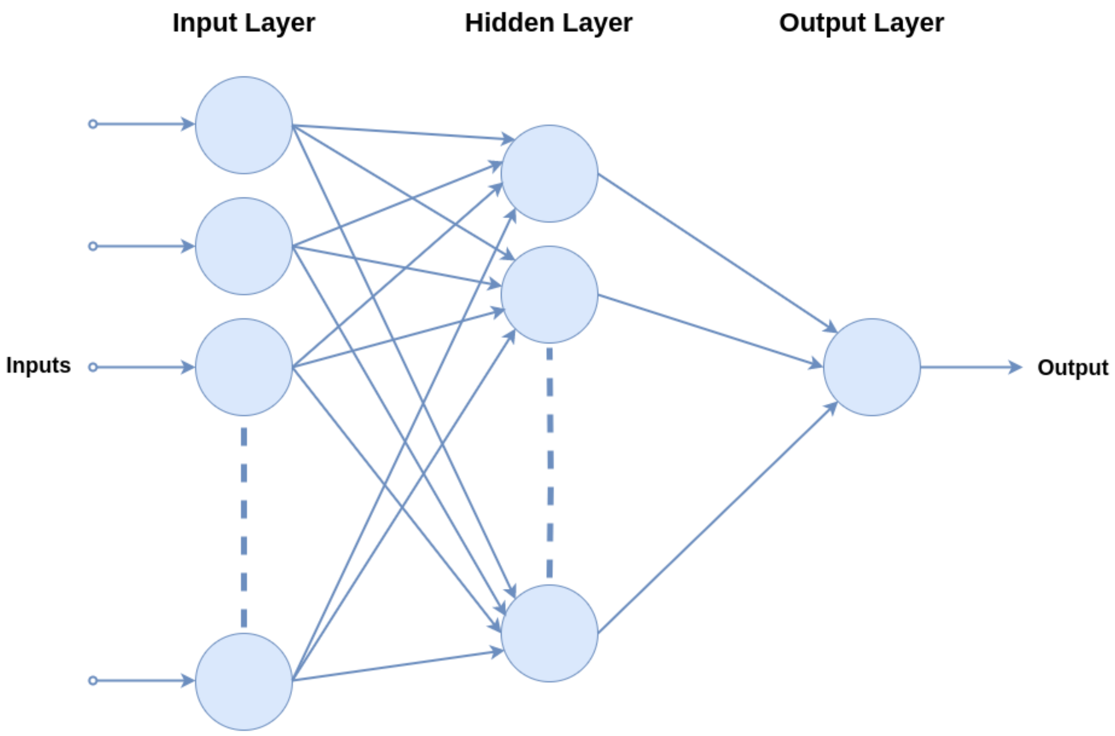

Figure 1.1: Schematic Figure of a Feedforward Network with one Hidden Layer

The numerical value[5] of a node i in layer j is

$$y_i^j = f_i^j\left(x_i^j\right) = f_i^j\left(b_i^j + \sum_{k=1}^{N_{j-1}} a_{k,i}^{j-1,k} \cdot y_k^{j-1}\right). \tag{1.1}$$

The weight $a_{k,i}^{j-1,k}$ connects the node k in layer $j-1$ with node i in layer j. N_{j-1} stands for the number of nodes in the previous layer and b_i^j is a bias which is added to each node to provide an adjustable shift. A non-linear function is applied to the resulting x_i^j. Typical functions for this purpose are shown in figure 1.2.[6]

[5] Bishop, 2007.

[6] Bishop, 2007.

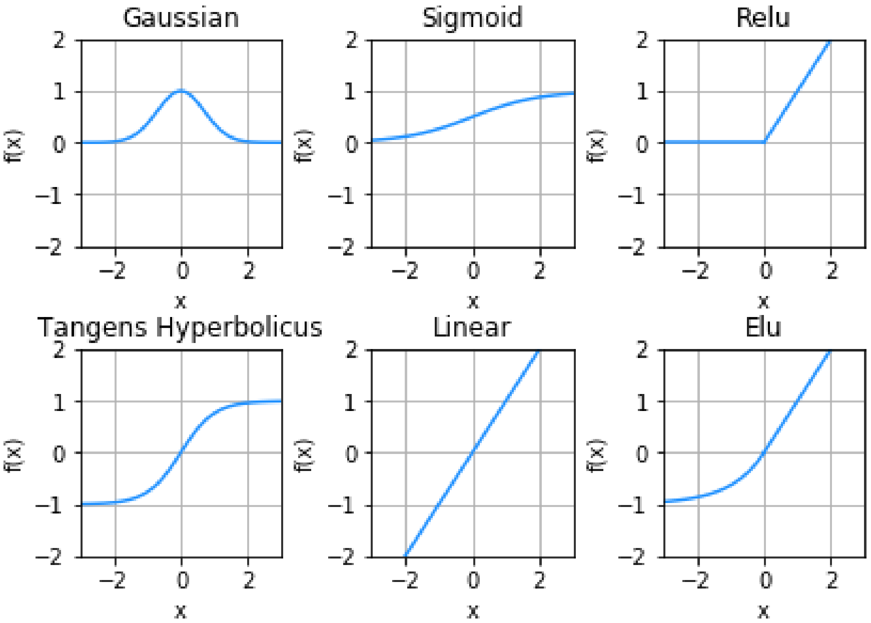

Figure 1.2: Common used Activation Functions for a Feedforward Network

1.0.2 Training of a neural network

The main goal of training a NN is to determine the parameters with which is then possible to provide a general function for a given problem. To determine these parameters, the error has to be evaluated during the training to estimate the performance of the NN. This evolution is based on the mean squared error (MSE),

$$E\left(\mathbf{w}\right) = \frac{1}{N}\sum_{n=1}^{N} \parallel \mathbf{y}\left(\mathbf{x}_n, \mathbf{w}\right) - \mathbf{t}_n \parallel^2 \tag{1.2}$$

.

$E(\mathbf{w})$	Error in dependence of $\mathbf{w}$
$\mathbf{w}$	Weights
$\mathbf{t}_n$	Target Vector
x_n	Input Vector
$\mathbf{y}\left(\mathbf{x}_n, \mathbf{w}\right)$	Prediction

The MSE then needs to be minimized by a learning algorithm during training, which will be discussed in the next section.

Overfitting/Underfitting

During the training of a NN **Overfitting/Underfitting** can occur.

Overfitting occurs when the NN predicts the training data accurately with a sufficiently small error but fails to predict the validation/testing data, which is visible in large test error. The trained model learns the training data in detail with the corresponding noise which leads to the consequence that it is not able to generalize anymore. This means that the random fluctuations and the noise of the training are further applied to new data and the model fails to predict the correct outcome.
Overfitting mostly appears in non-parametric and non-linear models because of their flexibility. This problem can be avoided if a small size of the NN is chosen in the beginning and if an early stopping condition is used. Early stopping conditions could be for example if the error of the validation data converges or that the error increases with the epoch.[7]

Underfitting occurs when the model neither fits the training data nor the validation data. The model fails to predict the correct outcome for training and validation data.
Underfitting occurs e.g. when the hyperparameters (see below) are not suitable. This can be avoided, e.g., by choosing a different learning algorithm or by varying the size of the NN.[8]

L2-Regularization

One method to prevent overfitting is the l2-regularization

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda||\mathbf{w}||^2 \tag{1.3}$$

which modifies the cost function 1.2 with an additional regularization to prevent high values for the weights. The numerical values of the weights are squared, multiplied with a coefficient λ and then added to the cost function, see equation 1.3.[9]

[7] Brownlee, 2016.
[8] Brownlee, 2016.
[9] Bishop, 2007.

Dropout

An additional method to prevent overfitting is to apply a dropout to the NN effectively reducing the number of nodes. This means that, e.g., 30% of the nodes are randomly selected and removed from the system entirely.[10]

Hyperparameters

For each problem a set of Hyperparameters has to be defined. The Hyperparameters[11] are:

- Learning Rate
- Network Size
- Regularization Parameters
- Learning Algorithm

The impact of these parameters on the performance has to be tested with a validation set of the available data. Only after the parameters have been determined a successful training of the NN for a problem can take place.
There are three main approaches with which these parameters can be found[12]:

- Manual tuning
  - The hyperparameters are chosen and validated manually based on experience.
- Grid search
  - A grid for the hyperparameters is defined and the neural NN performance is tested on each point.
- Random search
  - Each hyperparameter is defined with a probability distribution in a specified interval. The neural NN is then trained with a permutation of the random generated hyperparameters. The number of permutation is then lower than with the grid search. So a more efficient training takes place.

[10] Srivastava et al., 2014.
[11] Bishop, 2007.
[12] Bishop, 2007.

1.0.3 Gradient Descent Optimization Algorithm

The gradient descent in general is a method to minimize an objective function $J\left(\theta\right)$ parametrized by parameters $\theta \in \mathbb{R}^d$, by updating the parameters in the opposite direction of the gradient $\nabla_\theta J\left(\theta\right)$ with a predefined step size η called learning rate.[13] Typically the objective function is an error function, as for example equation 1.2.

Gradient Descent Optimizer

The simplest realization of a gradient descent is updating the weights

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right) \tag{1.4}$$

by taking a step η in the negative direction of the gradient.[14]

$\mathbf{w}^{(\tau+1)}$	Updated Weights
$\mathbf{w}^{(\tau)}$	Previous Weights
τ	Steps
η	Learning Rate
E	Error Function

Algorithm 2: Gradient Descent Optimizer

```
input  : Learning Rate $\eta$
initialization;
while stop criterion is not reached do
    Evaluate gradient of error function: $\mathbf{g} \leftarrow \nabla E\left(\mathbf{w}\right)$ ;
    Apply update on weights: $\mathbf{w} \leftarrow \mathbf{w} - \eta\mathbf{g}$
end
```

[13] Ruder, 2016.

[14] Bishop, 2007.

Adam Optimizer

The advantage of the Adaptive Moment Estimation (Adam)[15] is that this algorithm computes an adaptive learning rate for each parameter and keeps an exponential decaying average of previous gradients. So, in comparison with the gradient descent optimizer for every weight in the neural net a learning rate is calculated and updated.[16]

Algorithm 3: Adam Algorithm

```
input  : Learning Rate $\eta$
input  : Exponential Decay Momentums $\beta_1$ and $\beta_2$
input  : A small constant for numerical stability $\epsilon$
initialization of 1st and 2nd momentum vector $\mathbf{m}_0 \leftarrow 0$, $\mathbf{v}_0 \leftarrow 0$;
initialization of weights $\mathbf{w}$;
initialization of time step $\mathbf{t} \leftarrow 0$;
while stop criterion is not reached do
    Take a minibatch of m from Training set ($\mathbf{x}$ input and $\mathbf{y}$ target);
    Evaluate gradient: $\mathbf{g} \leftarrow \nabla \sum_i L\left(f\left(\mathbf{x}^{(i)}, \mathbf{w}\right), \mathbf{y}^{(i)}\right)$ ;
    $t \leftarrow t + 1$ ;
    Update biased first moment: $\mathbf{m}_t \leftarrow \beta_1 \cdot m_t + (1 - \beta_1) \cdot g$ ;
    Update biased second moment: $\mathbf{v}_t \leftarrow \beta_2 \cdot v_t + (1 - \beta_2) \cdot g \odot g$ ;
    Correct bias in first moment: $\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\beta_1^t}$ ;
    Correct bias in second moment: $\hat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1-\beta_2^t}$ ;
    Compute update: $\Delta\mathbf{w} = \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}}$ ;
    Apply Update: $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
end
```

[15] Kingma and Ba, 2014.

[16] Ruder, 2016.

2 Pattern Recognition with Neural Networks

2.1 The XOR Problem

The XOR logical gate is a simple, well studied benchmark system for the application of machine learning techniques. The XOR Logic has two inputs (x and y) and one output (z). When the inputs are different from each other the output is 1, otherwise the Output is 0, as can be seen in table 2.1. Now, a NN should solve this XOR problem and deliver the correct output for every possible combination of the XOR problem.

Table 2.1: Rules for XOR Logic

Input		Output
x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

For the XOR problem a FNN is used, which has two inputs for x and y and one output for z. The number of hidden layers, nodes and the learning rate are to be determined during training.
The appropriate hyperparameters are determined via a grid search.

The library PyBrain [1] is used to set up and train the NN. The code for the solution of this problem is available under GitHub at XOR PyBrain [2]

[1] http://pybrain.org/

[2] https://github.com/danlukic/XOR_PyBrain

2.1.1 Training

All possible combinations of the following hyperparameter values are used to find the optimal hyperparameters for this problem: For learning rate $\eta = 0.05, 0.01, 0.005, 0.001$ are assumed and for the number of nodes 2 and 3. The number of hidden layer for this training is 1. The used learning algorithm is the **Gradient Descent Optimizer** 2 with the cost function 1.2.

The NN is then executed for the different hyperparameters. The results are summarized in figures 2.1 to 2.2 and tables 2.2 to 2.3.

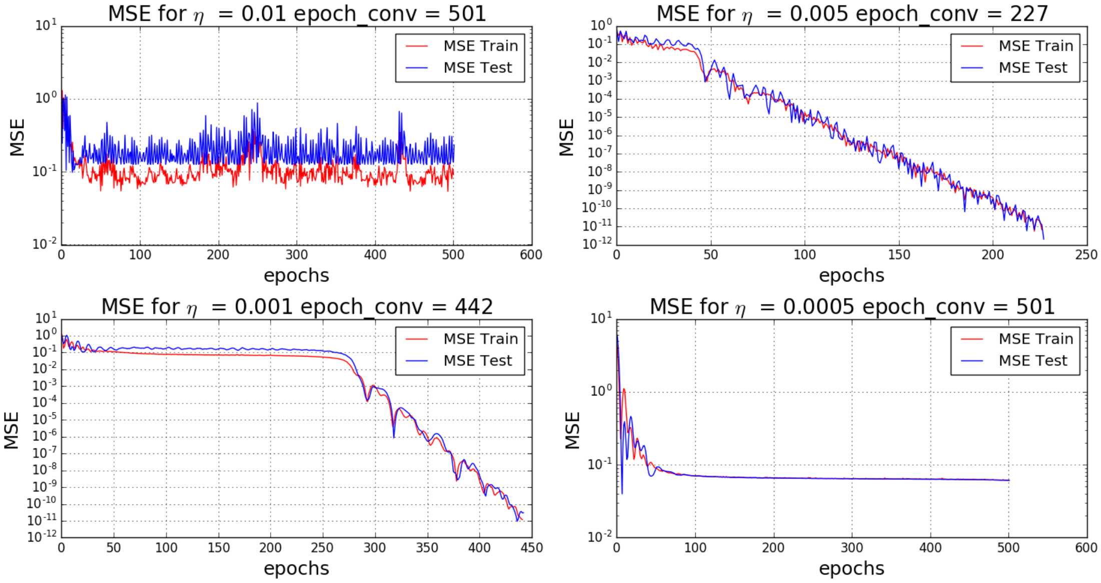

Figure 2.1: Training and Test Error for Neural Network with one Hidden Layer and two Nodes. η...Learning Rate, MSE...Mean Squared Error, epoch conv...Epoch where Neural Network reaches convergence criteria

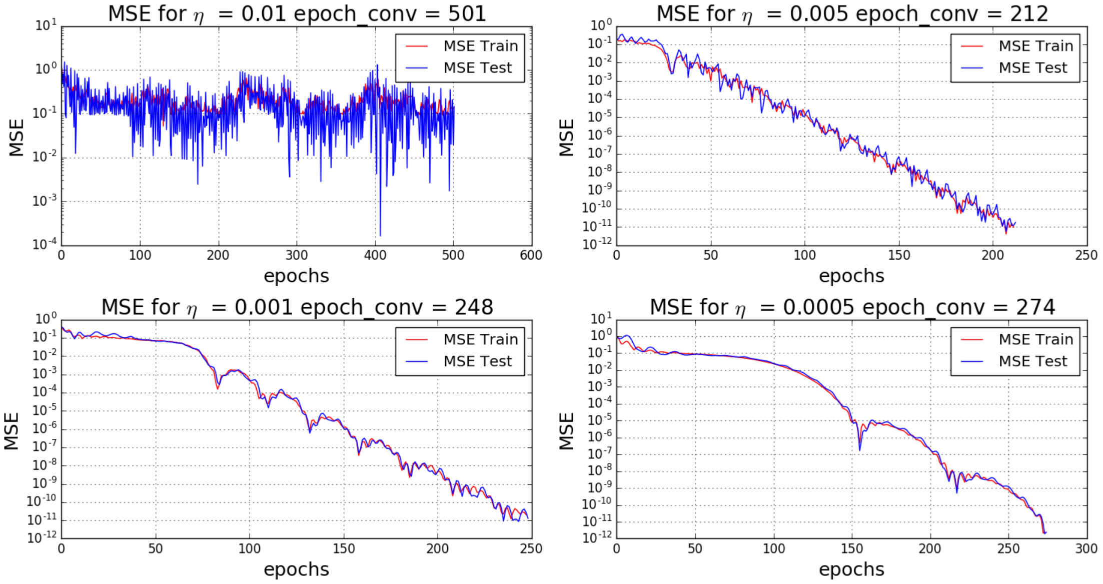

Figure 2.2: Training and Test error for of Neural Network with one Hidden Layer and three Nodes. η...Learning Rate, MSE...Mean Squared Error, epoch conv...Epoch where Neural Network reaches convergence criteria

Table 2.2: Test Results for Neural Network with one Hidden Layers and 2 Nodes

x, y . . . Inputs for Neural Network, Output . . . Prediction of Neural Network, η. . . Learning Rate

Input		Output			
x	y	$\eta = 0.01$	$\eta = 0.005$	$\eta = 0.001$	$\eta = 0.0005$
1	0	0.262	0.999	0.658	0.249
1	1	0.247	$3.460 \cdot 10^{-6}$	0.664	0.210
0	1	0.247	0.999	0.659	0.212
0	0	0.248	$1.732 \cdot 10^{-6}$	$-4.09 \cdot 10^{-5}$	0.244

Table 2.3: Test Results for Neural Network with one Hidden Layers and 3 Nodes

x, y . . . Inputs for Neural Network, Output . . . Prediction of Neural Network, η. . . Learning Rate

Input		Output			
x	y	$\eta = 0.01$	$\eta = 0.005$	$\eta = 0.001$	$\eta = 0.0005$
1	0	1.036	1.000	1.000	0.999
1	1	$-7.046 \cdot 10^{-5}$	$5.173 \cdot 10^{-6}$	$-3.558 \cdot 10^{-6}$	$2.078 \cdot 10^{-6}$
0	1	$-9.425 \cdot 10^{-5}$	1.000	1.000	0.999
0	0	$-2.453 \cdot 10^{-3}$	$3.995 \cdot 10^{-6}$	$-2.232 \cdot 10^{-5}$	$1.685 \cdot 10^{-6}$

2.1.2 Analysis

As can be seen in the tables 2.2, 2.3 and the figures 2.1, 2.2, it is evident that the configuration with a learning rate of $\eta = 0.005$ and one hidden layer with 3 nodes (figure 2.3 and table 2.2) delivers the best prediction for the output with the lowest epoch in respect to accuracy of the prediction.

In figure 2.1 and table 2.2 for the learning rate $\eta = 0.01$ it can be seen that the NN got locked in a local minimum and cannot leave this configuration. So, it delivers the same output for all inputs. It is also visible in the figure 2.1 that the error is of the same magnitude for all epochs and does not decrease.
For all other configurations propagation of the mean squared test error oscillates during training between local minima before it eventually converges.

It is also evident, as seen in table 2.2 and 2.3, that with 3 nodes in one hidden layer the NN is able to predict with three out of four learning rates ($\eta = 0.005, 0.001, 0.0005$) the output satisfyingly. In contrast, with 2 nodes in one hidden layer, only one configuration with learning rate $\eta = 0.005$ offers a satisfying prediction. This is due to the fact that with a higher amount of nodes in a hidden layer the NN net is more flexible and thus offers more solutions.

2.2 Binary Pattern Recognition

Here the idea is to see how well a NN can predict the output if it is tested with unknown situations after the training. In table 2.4 input and result data are illustrated for training and in 2.5 unknown input and result data for testing are illustrated. The NN should predict the correct output for the test data after the training.

Table 2.4: Pattern Training Data

Input			Output
x	y	z	out
0	0	1	0
1	1	1	1
1	0	1	1
0	1	1	0

Table 2.5: Pattern Test Data

Input			Output
x	y	z	out
1	0	0	1
1	1	0	1
0	1	0	0
0	0	0	0

2.2.1 Training

The NN is trained with different learning rates, different numbers of hidden layers and nodes to find the optimal hyperparameters for the problem. Here, the library TensorFlow [3] is used to set up and train the NN. The code for the solution of this problem is available under GitHub at Binary Pattern Recognition.[4].

The learning algorithm used is the **Adam** algorithm with the **MSE** as cost function, see equation 1.2. The convergence criteria for this problem is that if the mean squared test error reaches a convergence of $1 \cdot 10^{-7}$, the weights and the bias are randomly distributed truncatedly with a standard deviation of $std = 0.1$.

All possible combinations of the following hyperparameter values are used to find the optimal hyperparameters for this problem: For learning rate $\eta = 0.01, 0.001$ are assumed and for the number of nodes 2 and 4. The number of hidden layer for this training is 1. In tables 2.6 to 2.7 the prediction of the NN with the different configuration is visible. In figures 2.3 to 2.4 the cross entropy error is visible for the different combination of hyperparameters **??**.

[3] https://www.tensorflow.org/

[4] https://github.com/danlukic/Pattern_Recognition_TensorFlow

Table 2.6: Test Results for Neural Network with one Hidden Layers and 2 Nodes

x, y, z . . . Inputs for Neural Network, Output . . . Prediction of Neural Network, η. . . Learning Rate

Input			Output	
x	y	z	$\eta = 0.01$	$\eta = 0.001$
1	0	0	0.996	0.982
1	1	0	0.996	0.982
0	1	0	0.083	0.011
0	0	0	0.083	0.011

Table 2.7: Test Results for Neural Network with one hidden Layers and 4 nodes

x, y, z . . . Inputs for Neural Network, Output . . . Prediction of Neural Network, η. . . Learning Rate

Input			Output	
x	y	z	$\eta = 0.01$	$\eta = 0.001$
1	0	0	0.975	0.987
1	1	0	0.975	0.985
0	1	0	0.129	-0.023
0	0	0	0.129	-0.023

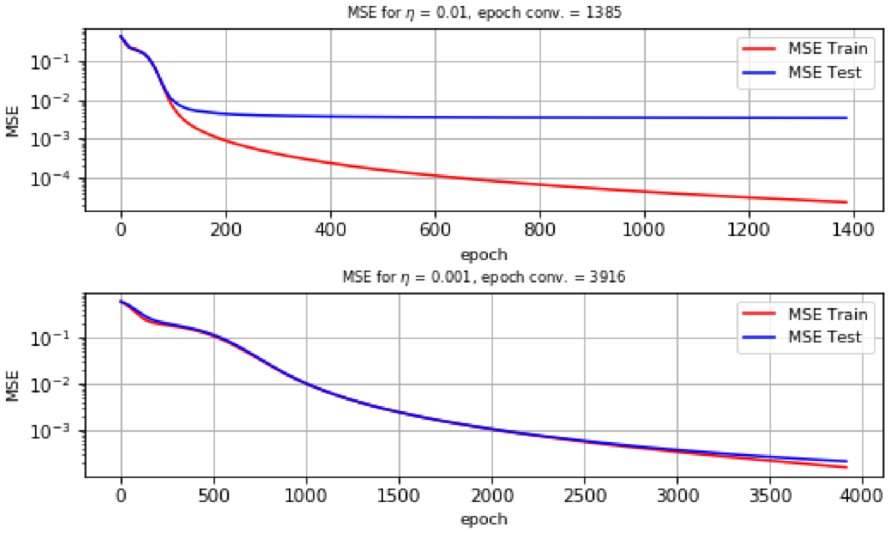

Figure 2.3: Training and Test Error for Neural Network with one Hidden Layers and two Nodes. η...Learning Rate, MSE...Mean Squared Error, Epoch Conv...Epoch where Neural Network reaches convergence criteria

The final MSE of a NN with 2 hidden nodes is $err_{MSE} = 3.451 \cdot 10^{-3}$ and $err_{MSE} = 2.13 \cdot 10^{-3}$ for learning rates of $\eta = 0.01$ and $\eta = 0.001$, respectively.

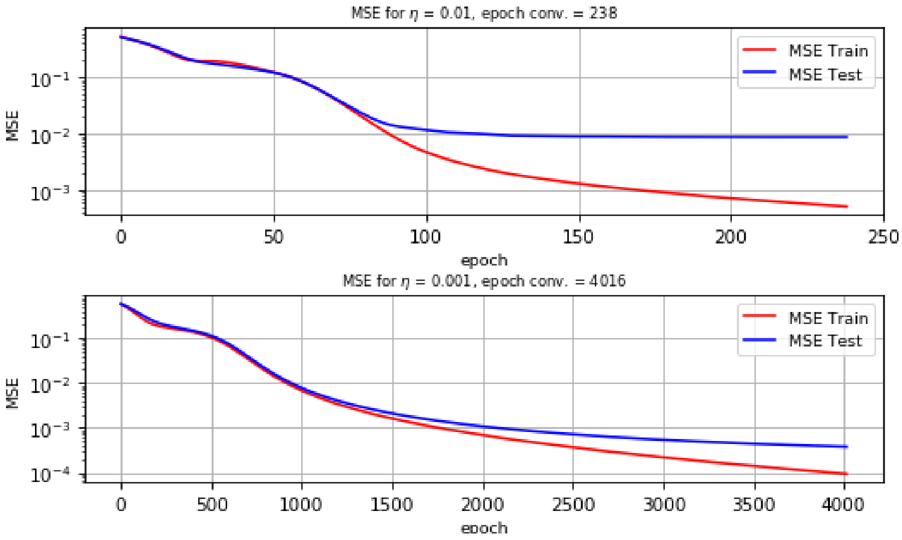

Figure 2.4: Training and Test Error for Neural Network with one Hidden Layers and four Nodes. η...Learning Rate, MSE...Mean Squared Error, Epoch Conv...Epoch where Neural Network reaches convergence criteria

The final MSE of a NN with 4 hidden nodes is $err_{MSE} = 8.72 \cdot 10^{-3}$ and $err_{MSE} = 0.000376052$ for learning rates of $\eta = 0.01$ and $\eta = 0.001$, respectively.

2.2.2 Analysis

As can be seen in tables 2.6, 2.7 and figures 2.3, 2.4, it is evident that the configuration with a learning rate of $\eta = 0.01$ and one hidden layer with 4 nodes (figure 2.3 and table 2.6) delivers the best prediction for the output with the lowest epoch in respect to accuracy of the prediction.

From figures 2.6 and 2.7 it is also evident that for all configurations the MSE for training and testing gets lower with the epochs. For the configuration with 4 nodes in one hidden layer it can be seen that it takes longer to converge than with 2 nodes. This can be explained by that fact that with more nodes the phase space for the global minimum is larger and this the search becomes larger.

It is also evident from both figures 2.3 and 2.4 that a smaller learning rate improves convergence because the scanning of the phase space happens in smaller steps. Finally, one can say that all configurations for this problem are offering a satisfying solution, as proven by the results in tables 2.6 and 2.7.

2.3 Butadiene Pattern Recognition

This problem deals with the pattern recognition of structural data from the butadiene molecule. The NN should be able to predict the energy of the molecule from structural data. The latter consists of the distance, angle between atoms and the dihedral angle of the atoms, as illustrated in figure 2.6. Variations of the structural data or "internal coordinates" and the energies (Potential Energy Surface) of the molecule are obtained via Quasiclassical Ab-Initio Molecular Dynamics and Velocity Verlet integration with a timestep of $ts = 0.484$ fs at a temperature of $T = 300$ K.

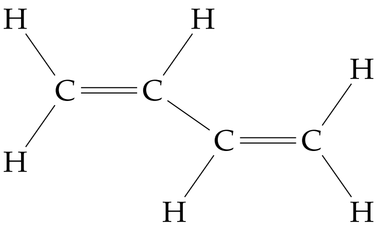

Figure 2.5: Structure of 1,3 Butadien

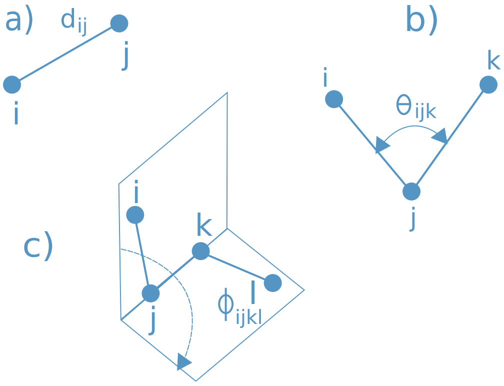

Figure 2.6: Coordinates of Molecule: a) bond length, b) angle between two atoms, c) dihedral angle

For this problem there are 24 structural inputs and one output for the energy. The data file consists of 10000 energy evolution for different geometries. The variation in the internal coordinates data are plotted not normalized and normalized for the first 200 data, as can be seen in figure 2.7. The input data are normalized with

$$\mathbf{X_{norm}} = \frac{\mathbf{X} - \mu}{\sigma} \tag{2.1}$$

to smooth the data for the NN.
In the figure 2.7 the not normalized and the normalized input data are plotted for the first 200 points to illustrate the visual difference of normalizing and not normalizing. A NN can cope better with input data which are in the same range and smooth.

With,

$\mathbf{X_{norm}}$	Normalized Data
$\mathbf{X}$	Not Normalized Data
μ	Mean of not Normalized Data
σ	Standard Deviation of not Normalized Data

Also, the target data are normalized with

$$\mathbf{X_{norm}} = 2 \cdot \frac{\mathbf{X} - min\left(\mathbf{X}\right)}{max\left(\mathbf{X}\right) - min\left(\mathbf{X}\right)} - 1 \tag{2.2}$$

between 1 and −1.

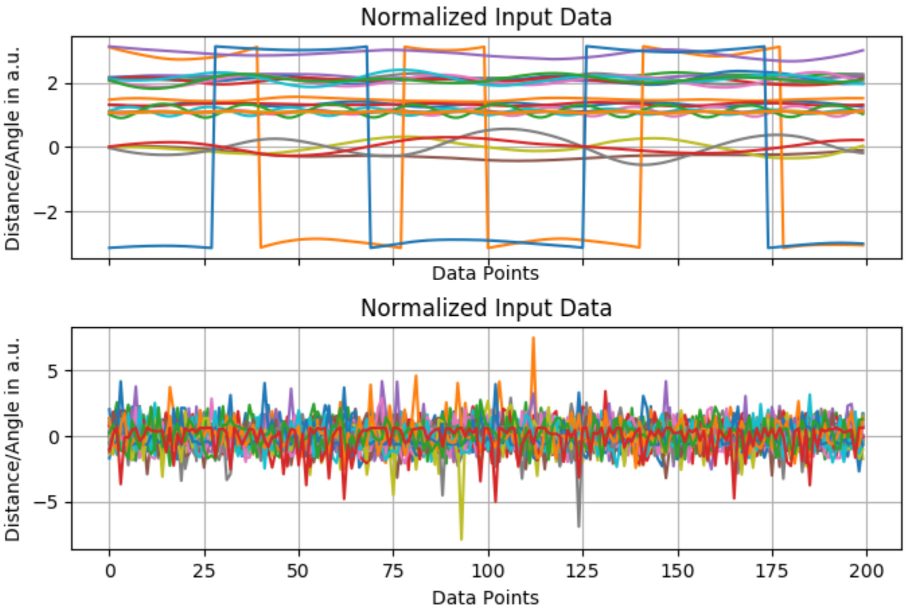

Figure 2.7: Structural Data plotted Not Normalized and Normalized for the first 200 points

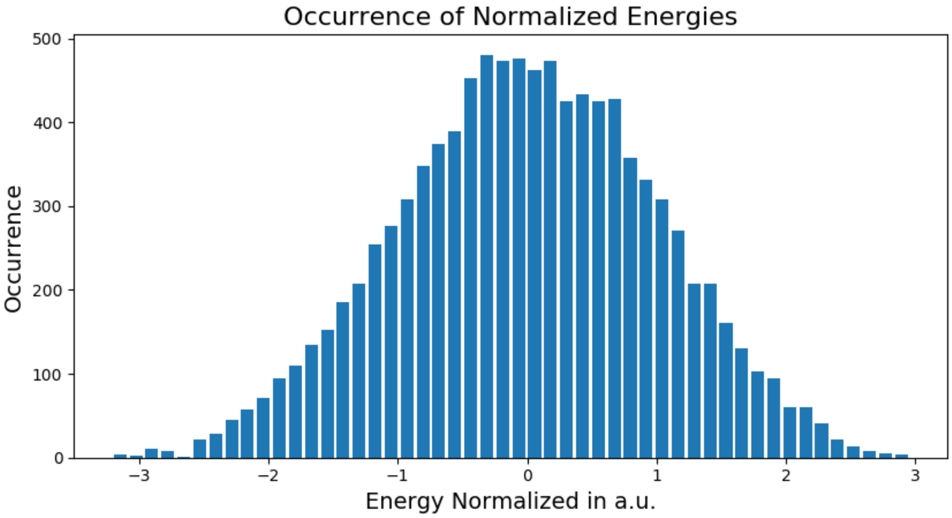

Figure 2.8: Histogram of Target Data with Normalized Energies

2.3.1 Training

For this NN an exponential decaying learning rate was used, see equation 2.3, with a standard starting learning rate of $\eta = 0.001$. The algorithm the Adam Algorithm 3 was used with standard values for $\text{beta}_1 = 0.9$ and $\text{beta}_2 = 0.999$. For the activation function Exponential Linear Unit (Elu) is used, see figure 1.2. The activation function Elu has an exponential form where the y values are in the range of -1 to ∞ for x values from $-\infty$ to ∞. Also, the order of the data was randomized at the beginning and during the training. The NN was trained with a batch size of 100. For the cost function, analog to the two previous examples, a MSE 1.2 and an additional l2-regularization with a constant $\lambda = 1 \cdot 10^{-7}$ 1.3 was used. The given data set is separated in 90% training data and 10% testing data. The convergence criteria for this problem arises when the mean squared test error reaches a convergence of $1 \cdot 10^{-8}$. The weights and the bias are randomly distributed truncatedly with a standard deviation of $std = 0.1$.

$$\delta\eta = \eta \cdot dr^{gls/ds} \tag{2.3}$$

With,

$\delta\eta$	Decayed Learning Rate
η	Starting Learning Rate
gls	Global Step
ds	Decay Steps

To set up and train the NN, the library TensorFlow [5] is used. To read and prepare the data set for the NN and training, the library pyQChem [6] is used. The code for the solution of this problem is available under GitHub at Butadien Patter Recognition.[7] The NN was trained with several different configurations, without and with decaying learning rates and dropouts, as can be seen in tables 2.8 and 2.9 to find the configuration where the prediction is the most accurate with the lowest mean squared test error and in respect to the accuracy the lowest epoch, see table 2.9.

[5] https://www.tensorflow.org/

[6] https://github.com/awhauser/pyQChem

[7] https://github.com/danlukic/Butadien_Pattern_Recognition

Table 2.8: Mean Squared Error for several Configurations without Decaying Learning Rate and without Dropout

HL. . . Hidden Layer, MSE Test Data. . . Mean Squared Error Test Data, Max Epoch . . . Maximal Epoch for Training

Number Of Nodes							
HL 1	HL 2	HL 3	HL 4	HL 5	HL 6	MSE Test Data	Max Epoch
50	50	10	-	-	-	0.00975413	1000
70	70	15	-	-	-	0.0080178	1000
100	100	30	-	-	-	0.00554385	1000
100	100	100	30	-	-	0.000708318	1000
200	200	200	50	-	-	0.000424553	1000
300	300	300	50	-	-	0.00060632	1000
300	300	300	300	50	-	0.0015613	1000
400	400	400	400	50	-	0.00315422	1000
300	300	300	300	300	50	0.00144361	1000

Table 2.9: Mean Squared Error for several Configurations with Decaying Learning Rate and with Dropout

HL. . . Hidden Layer, MSE Test Data. . . Mean Squared Error Test Data, Max Epoch . . . Maximal Epoch for Training

Number Of Nodes							
HL 1	HL 2	HL 3	HL 4	HL 5	HL 5	MSE Test Data	Conv. Epoch
50	50	10	-	-	-	0.0101364	717
70	70	15	-	-	-	0.00812312	907
100	100	30	-	-	-	0.00596537	897
100	100	100	30	-	-	0.00123002	369
200	200	200	50	-	-	0.000285369	594
300	300	300	50	-	-	0.000194999	652
300	300	300	300	50	-	0.000159884	545
400	400	400	400	50	-	0.000238753	448
300	300	300	300	300	50	0.00018395	493

For the configuration with the lowest MSE the prediction and reference are plotted, as can be seen in 2.9 and in figure 2.10 the prediction and the reference are plotted against each other.

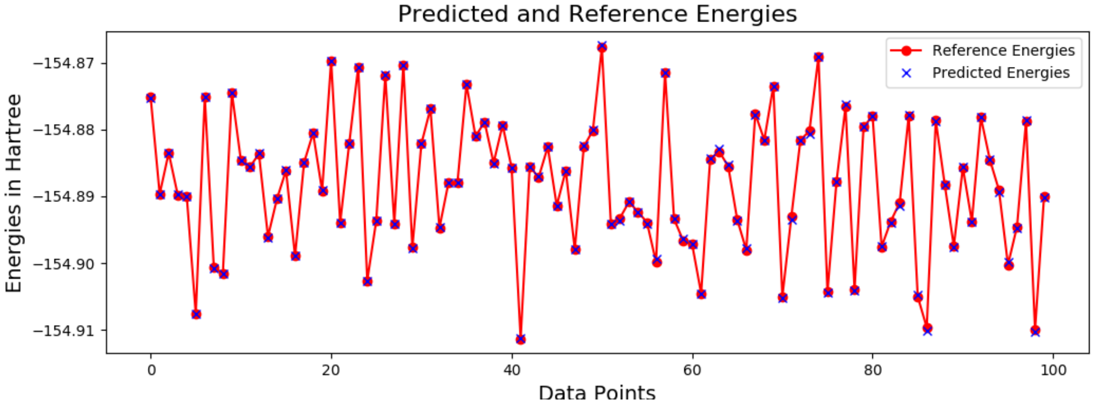

Figure 2.9: Reference Energies and Predicted Energies. First 50 Data Points are Training Data and last 50 Data Points are Testing Data

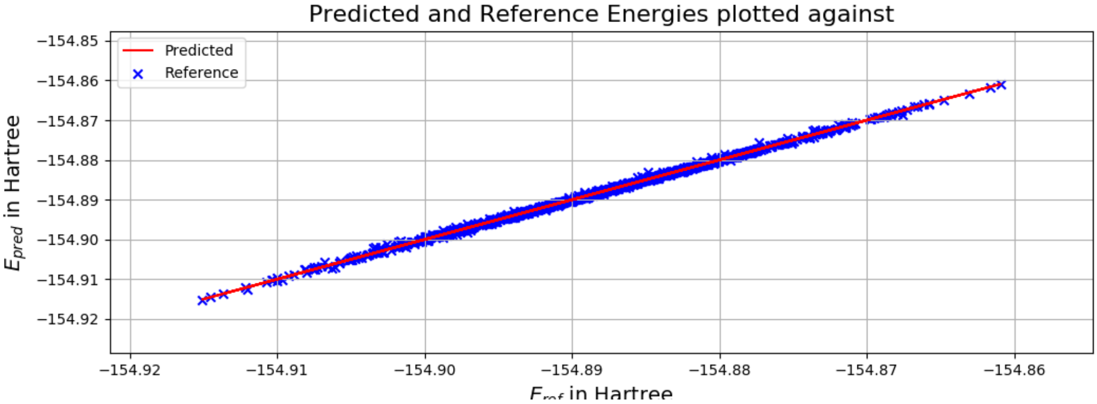

Figure 2.10: Predicted and Reference Energies plotted against each other.

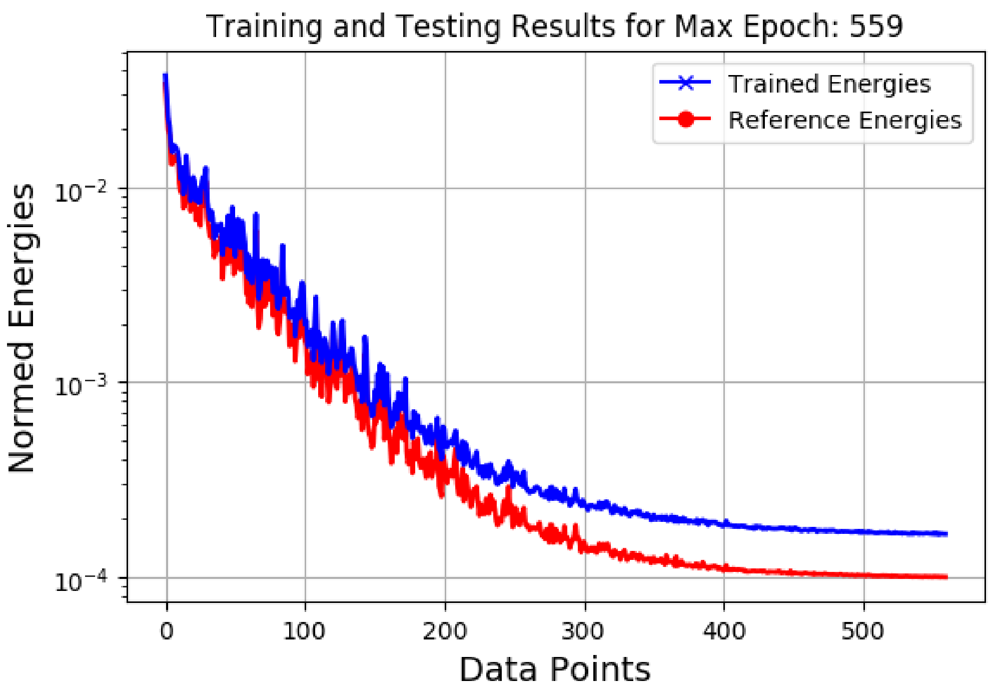

Figure 2.11: Training and Test Error of a Neural Network for Butadien. MSE...Mean Squared Error

In table 2.10 five energies from the reference data set and the prediction data set were picked randomly. The data were back calculated because of the normalization 2.1.

Table 2.10: Prediction and Reference Energies randomly picked from Testing Data Set

E_{ref} . . . Reference Energy in Hartree, E_{pred} . . . Prediction Energy in Hartree

E_{ref}	E_{pred}
−154.90321612	−154.90325928
−154.88227844	−154.88233193
−154.88153134	−154.88159180
−154.88245343	−154.88243103
−154.88180891	−154.88171387

3 Conclusion

As can be seen in figure 2.9, 2.10 and table 2.8, 2.9, the NN with four hidden layers and each 300 nodes per layer yields the prediction with the lowest MSE. From table 2.10 it is visible that the prediction agrees with the reference with a MSE of 0.101 kcal/mol.

Also, as can be seen in tables 2.8 and 2.9, the decaying learning rate and the l2-regularization makes significant change in the training of the NN. For the first three configurations in table 2.8 and table 2.9 the MSE for testing data is lower than the error obtained with methods while actively prevent overfitting; however they do not converge. The mean squared test error then increases for the further configurations and also does not converge and has to be stopped with a maximal epoch setting.

The constant λ for the l2-regularization with a rather low value of $1 \cdot 10^{-7}$ is used for the NN. This means that the numerical values of the quadratic weights is multiplied by $1 \cdot 10^{-7}$ and the product is added to the cost function. A standard value for λ is $1 \cdot 10^{-3}$. Yet, the prediction is more accurate than without regularizations. The conclusion of this is that with the l2-regularization overfitting is prevented and more flexibility is added to the NN. Note that the MSE for the testing data set increases for the first four configurations This is due to the fact that l2 regularization only works with NN with a higher number of nodes and hidden layers.

Finally, one can say that the NN with the configuration four hidden layers and 300 nodes each is able to predict the energies of the molecule for a training data size of 90% and testing data size of 10% and with a MSE of 0.101 kcal/mol.

Bibliography

Bishop, Joseph (2007). *Pattern recognition and machine learning*. Fith. Springer (cit. on pp. 1, 2, 4–6).

Brownlee, Jason (2016). *Overfitting and Underfitting With Machine Learning Algorithms*. URL: http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/ (visited on 11/26/2017) (cit. on p. 4).

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 7).

Kröse, Ben et al. (1993). *An introduction to Neural Networks* (cit. on p. 1).

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms." In: *CoRR* abs/1609.04747. arXiv: 1609.04747 (cit. on pp. 6, 7).

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15, pp. 1929–1958 (cit. on p. 5).