Project Machine Learning
SchNet: A continuous-filter convolutional neural network for modeling quantum interactions
— Milestone 2: Model Selection and Evaluation —

Ariane Oesch, Christian Kasim Loan, Trung Duc Ha

January 8, 2024

Abstract

In this second milestone report, we describe our implementation of SchNet [1]. We implement the model using PyTorch and resolve numerical issues with the proposed activation function. We employ k-fold cross-validation to estimate the generalization error and gain a confidence measure by using the cross-validation models as an ensemble neural network. We compare the model with both the baseline methods and the original paper, determining subpar energy predictions of our model. Finally, we suggest avenues for future research, particularly focusing on the model architecture and its applications in modeling quantum interactions.

1 Introduction

We continue the work from the first milestone by implementing the complete SchNet architecture [1]. The goal is to assess its performance, particularly in comparison with the baseline presented in the previous report.

SchNet was developed in response to specific modeling constraints in ML-driven quantum-chemical exploration. The challenges are diverse: the need for a representation for all types of molecular conformations, the need to respect the fundamental laws of quantum mechanics etc. To meet these issues, SchNet introduces a continuous-filter convolutional neural network to the landscape of existing solutions.

When implementing SchNet, we were confronted with some difficulties and uncertainties concerning certain definitions or results. The purpose of this report is also to discuss the ambiguities and find solutions to the problems we faced during the process of implementing and testing the model.

2 Approach

In this section, we detail the SchNet implementation process. First, we discuss the choice of this model over alternatives, then we explain our methodology. We analyze the model element by element, review the definitions, and identify the uncertainties and difficulties encountered.

2.1 Model Selection

The major contribution of SchNet lies in the use of continuous-filter convolutional layers. This innovation significantly enhances its ability to capture intricate molecular interactions. In the paper, three other significant works are put in comparison with SchNet, which are gradient domain machine learning (GDML [2]), deep tensor neural networks (DTNN [3]), and enn-s2s [4]. But there are many more, including variations and hybrid methods.

GDML [2] is a kernel-based approach. It enables reliable results to be obtained with a limited number of references, provided that the molecule is not too large (a few dozen atoms at most). Indeed, the size of the kernel matrix used increases quadratically with the number of atoms. The other drawback of GDML is transferability. A model trained on the conformational isomers of one molecule cannot be used to predict the energies and forces of another molecule. On the contrary, SchNet shows good transferability [5].

Another kernel-based method used in molecular machine learning is the SOAP (for Smooth Overlap of Atomic Positions) method, which consists of a kernel ridge regression [6]. Olsthoorn *et al.* [6] compare SchNet to the SOAP kernel to predict the electronic band gaps of crystalline materials. On a data set that contains band gaps for 12500 organic crystal structures, SchNet turns out to be slightly more accurate than SOAP.

Generally speaking, SchNet is more complex than kernel-based methods, but is often preferred for its greater flexibility, especially when it comes to handling diverse molecular structures.

DTNN [3] is a neural network architecture whose approach differs from SchNet. It uses interaction layers and radial basis functions to learn representations of atomic environments, while SchNet uses continuous convolutional layers. The latter, more sophisticated approach has proved more accurate in practice [1].

enn-s2s [4] (for Equivariant Neural Network Sequence to Sequence) is a variant of message-passing neural networks, and is designed to be invariant to molecular symmetries. Despite convincing results on the QM9 dataset, enn-s2s does not produce a continuous potential energy surface like SchNet, making it unusable for molecular dynamics.

Another advantage of SchNet is that the final prediction is computed as an average of individual atomic contributions. This enables us to identify structure-property relationships for individual atoms. SchNet is able to learn similarities between chemical elements as in the periodic table.

2.2 Methodology

We outline the methodology employed in our study, focusing on specific methods, feature engineering, hyperparameter choices, and improvements achieved. We implement the SchNet architecture [1] as PyTorch modules [7]. The model does not use feature preprocessing or selection methods like our baseline to preserve nuances in the original data structure [1]. It attempts to learn the feature extraction, employing a novel *continuous filter convolutions* (FConv) layer. In contrast to our baseline, SchNet requires additional feature engineering for its proposed interaction blocks, which we implement using the Atomic Simulation Environment (ASE) library [8].

2.2.1 Input Data Structure

The model accepts input data from the data loader function we implemented in the first milestone [9]. We feed the nuclear charges $Z = (z_1, ..., z_n)$ and atomic positions $R = (r_1, ..., r_m)$ to the model architecture. To handle variable size molecule compositions in a training batch, we introduce indexes to determine the neighboring atoms of the molecules. We precompute these indexes from a center atoms to their neighbors and feed them to the model. The neighbor indexes are computed using the ASE library [8].

We provide the indices of center atoms and its neighbors as engineered input features. However, the distances between the atoms must be computed within the network during training, allowing the gradient to flow through these features. We implemented a PyTorch module to compute the pairwise distances for every center atom and their neighbors.

The model uses high-dimensional representations of each chemical element inside the network. We implement this using the `Embedding` module of PyTorch [7], mapping the unique nuclear charges of each element to one learned embedding.

2.2.2 Atom-wise layers

Atom-Wise layers are implemented as `Linear` layers in PyTorch [7], which are applied to the embedding representations of the molecules. Weights are shared across atoms to maintain scalability with respect to the size of the molecule. They are responsible for recombining feature maps and learning useful representations from atom representations in previous steps.

2.2.3 Continuous Filter Convolutions (FConv)

Convolutional layers in deep learning are used to learn functions that filter the input data for patterns, such that the following layers process distinct filtered areas of the data and overall loss is reduced. Due to the non-euclidean graph structure and geometry of molecule datasets, classical convolutions that are applied to images and similar grid data structures are not applicable [10].

Given the features of n data instances $X^l = \left(\mathbf{x}_1^l, \ldots, \mathbf{x}_n^l\right)$ with $\mathbf{x}_i^l \in \mathbb{R}^F$ at locations $R = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$ with $\mathbf{r}_i \in \mathbb{R}^D$, the FConv at layer l is a function $W^l : \mathbb{R}^D \to \mathbb{R}^F$ that filters the pairwise distances for statistically relevant patterns. This approach generalizes to an arbitrary number of positions and objects.

Schütt *et al.* [1] define the convolutional layer at the current layer l and the atom position $\mathbf{r}_i$ as

$$\mathbf{x}_i^{l+1} = \left(X^l * W^l\right)_i = \sum_j \mathbf{x}_j^l \circ W^l\left(\mathbf{r}_i - \mathbf{r}_j\right),$$

where $\circ$ is the element-wise multiplication.

The FConv layers are rotationally invariant in order to satisfy the requirement for modeling molecular energies. The rotational invariance is provided by using interatomic distances $d_{ij} = \|\mathbf{r}_i - \mathbf{r}_j\|$ as input to the network.

Schütt *et al.* [1] define the radial basis functions (RBF) e_k, which extracts spacial correlations between two atom positions, as

$$e_k\left(\mathbf{r}_i - \mathbf{r}_j\right) = \exp\left(-\gamma \left\|d_{ij} - \mu_k\right\|^2\right)$$

.

Each of the k functions has centers $0\text{Å} \leq \mu_k \leq 30\text{Å}$ every 0.1Å with $\gamma = 10\text{Å}$. They introduce additional non-linearity which initial filters less correlated and leads to faster training procedures. Choosing fewer centers is analogous to reducing the resolution of the filter. Restricting the range of centers is analogous to the filter size in a usual convolutional layer [1].

The minimum and maximum values of the filter convolutions are 0Å and 30Å, respectively. We assume that Schütt *et al.* chose these values to cover all possible distances and interactions between atoms of the same molecule within the datasets used. Indeed, atomic covalent bonds usually range between 1Å and 2Å for the atoms considered (C, N, O, F, H). Hydrogen bonds can occur within the range of about 1.5Å to 3.5Å, and Van der Waals interactions occur at distances typically ranging from 2Å to 4Å. As we also handle the benzene molecule and its aromatic ring, which also considers pi stacking interactions, usually around 3.5Å to 4.5Å.

2.2.4 Shifted Softplus (SSP)

Schütt *et al.* [1] propose a novel activation function in the form of the Shifted Softplus (SSP). They define it as

$$\operatorname{ssp}(x) = \log(0.5\exp(x) + 0.5)$$

.

The basis for this function is the Softplus activation function. However, due to the shift, it helps to obtain a "smooth potential energy surface", that is, continuous for all derivatives of all orders, while improving the convergence of the network. We discuss numerical instabilities with the implementation of this activation function in detail in Section 4.3.

2.2.5 Interaction layers

Interaction layers are responsible for updating the atomic representations based on the molecular geometry R. We implement the layer by creating a sequential module with an atom-wise layer, FConv layer and an atom-wise layer, after which a SSP is applied. A last atom-wise layer completes the module. Furthermore, interaction layers are also residual layers and computed as $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{v}_i^l$, where $\mathbf{x}_i^{l+1}$ is the output of the whole interaction layer, $\mathbf{x}_i^l$ is the input and $\mathbf{v}_i^l$ is the output of the final atom-wise layer of this module. This relationship incorporates interactions between atoms and feature maps [11].

2.2.6 Model Training

We train the models using the parameters suggested by Schütt *et al.* [1]. We divide the training sets into 50,000 data points in mini-batches of size 32. The initial learning rate is set to 0.001. Additionally, we decay the learning rate by 0.96 every 100,000 steps. However, we find that during our training procedures, the learning rate does not decay due to computing limitations, since we do not reach the decay threshold. Training for 5 epochs with these parameters and 50,000 data points results in approx. 15,000 steps, implying that the training in the original paper must have been considerably longer. We leave the tuning of this parameter for reduced computational time for future work.

For the optimization process, we chose the Adam optimizer [12] due to its efficiency in handling sparse gradients and adaptive learning rates. This choice was further supported by its widespread and successful application in similar tasks.

Depending on the task, we will choose a different error metric. We describe this in Section 2.3.

2.2.7 Hyperparameter Selection

The model architecture in total has the following hyperparameters: number of interaction blocks, which controls the number of stacked interaction blocks in the model; atom embedding dimension, which specifies the dimension of learned atom embeddings; maximal value for the RBF, which controls the maximum radius of convolution functions; number of filters, which controls number of learned filter function.

We chose the values of the hyperparameters described by Schütt *et al.* [1] and attempt to explain their choices, which the authors do not justify. We attempt to empirically provide an answer to the rationale by training the model with different sets of hyperparameters. We use the same training procedure from Section 2.2.6, but adjust the training time roughly based on the complexity of the model. Table 1 shows the different configurations and the validation loss achieved.

Table 1: Hyperparameters and Test Metric.

Parameter	**Small**	**Default**	**Large**
Interaction blocks	1	3	6
Atom Embedding Dimension	32	64	128
Radial Basis Function Max Value	15	30	60
Number of Filters	150	300	600
Total Parameters	13,345	116,225	874,817
Epochs	1	2	4
Validation Loss	3.22	1.68	**1.32**

We observe, that the largest model outperforms the other models, even though we do not adjust the training time exactly for its complexity (double the epochs but about 7.5 times the complexity of the default model). This implies, that the increased model complexity still helps the model to reduce its bias and optimal complexity has not been reached. Because of computational limitations, we were not able to compute a large grid of parameters, but instead tested a small subset. We suggest in-depth exploration of the hyperparameter space for future work.

2.3 Error Metrics

To quantify the model's performance, we use established metrics such as the mean absolute error (MAE) and the newly proposed loss function by Schütt *et al.* [1], which incorporates energy as well as forces in the training loss to train the network. MAE is robust to outliers and has a gradient that is fast to compute [13]. We use it to showcase the effectiveness of the neural network trained with the joined energy-force loss.

Learning to predict both energy and forces with a model yields a more generalizing model overall, which seems to imply that the model learns general quantum properties at a computational cost. Schütt *et al.* [1] define the novel loss function as

$$\ell\left(\hat{E}, (E, \mathbf{F}_1, \ldots, \mathbf{F}_n)\right) = \rho|E - \hat{E}|^2 + \frac{1}{n}\sum_{i=0}^{n}\left\|\mathbf{F}_i - \left(-\frac{\partial \hat{E}}{\partial \mathbf{r}_i}\right)\right\|^2$$

, where n is the number of atoms of the molecule, E is the total energy and $\mathbf{F}_1, ..., \mathbf{F}_n$ are the atomic forces of each atom with $\rho = 0.01$ chosen empirically Schütt *et al.* [1] suggest. We faced challenges during implementing the energy-force loss which we outline in Section 2.4.

2.4 Uncertainties and Challenges

During the implementation process, we faced several challenges and uncertainties regarding the definitions and methodology of the original paper. First, we discuss the ambiguous definitions and implementation challenges of the loss function. Then we discuss problems with the proposed activation function.

We discuss the loss function for energy and force predictions in Section 2.3. We modify two aspects of the loss function. First, we observe that in the original loss function [1], the term $\|E - \hat{E}\|$ is ambiguous as it expresses the *norm* of a scalar. Conventionally, the norm of a one-dimensional normed vector space is defined as the absolute value, which we will interpret as such. Additionally, the term $\mathbf{R}_i$ is not defined. We interpret it as the atom positions $\mathbf{r}_i$, since this coincides with the definition of the force, that is, the partial derivative of the energy w.r.t. the atom positions.

This leads us to another challenge we faced, which is the computation of the force. PyTorch implements gradients with its `autograd` engine. The engine builds a computation graph to efficiently calculate gradients in a backward pass [7].

A novelty of SchNet is that the model incorporates gradient information of the atom positions, *i.e.*, forces into the optimization algorithm. The gradient of the atom positions needs to be computed separately to the gradient of the network parameters that the backward pass computes. By default, PyTorch does not record computations on gradients outside the backward pass. Failing to set necessary flags to the engine during force computation can lead to the force not being incorporated into the optimization, causing silent training failures with a high loss.

Lastly, as previously stated, the SSP activation function proposed by Schütt *et al.* [1] exhibits numerical instabilities. We discuss this problem in detail in Section 4.3.

3 Evaluation

In this section, we discuss the methods used to evaluate the performance of our implementation of the SchNet model. We use the metrics described in Section 2.3. We cover the results after training and compare it to the baseline methods from the first milestone and the original paper. Furthermore, we present an estimate for the generalization error of the model with k-fold cross-validation. We report all metrics for the energy and force predictions in kcal/mol and kcal/mol/Å respectively.

3.1 Comparison to Baseline

We analyze the training results against the baseline results we presented in the first milestone report. We recall from the first milestone report, that force predictions are not applicable to the QM9 dataset [9]. We decide to train the SchNet model for an additional epoch, as the SchNet model has about a hundred times more parameters (1,160 and 116,225 respectively). Table 2 shows the results for each scenario.

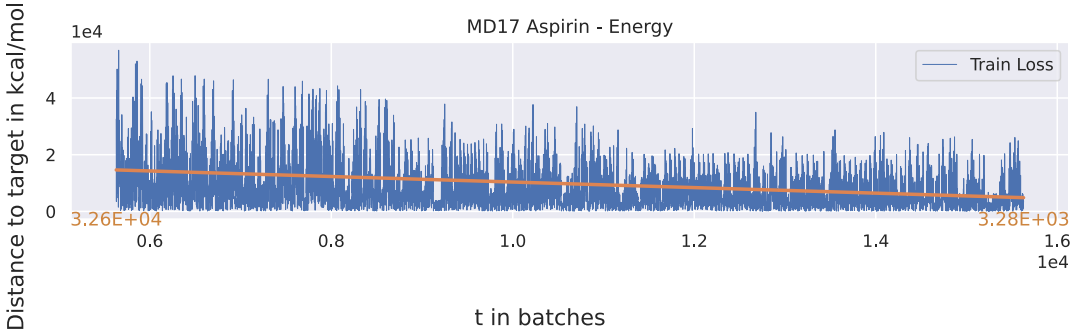
Table 2: Comparison of losses on test data for energy, force, and combined energy force tasks between Baseline and our trained SchNet models. The better performance (lower loss) is highlighted in bold. Both models were trained with 10,000 train and 1,000 test samples for 2 epochs. We report energy and force predictions in kcal/mol and kcal/mol/Å respectively.

Data Set	**Energy**		**Force**		**Energy+Force**	
	Baseline	*SchNet*	*Baseline*	*SchNet*	*Baseline*	*SchNet*
QM9	5.64	**3.01**	n.a.	n.a.	n.a.	n.a.
MD17						
Aspirin	$\mathbf{2.71 \times 10^3}$	6.16×10^3	19.53	**5.94**	2.58×10^7	$\mathbf{5.20 \times 10^3}$
Azobenzene	1.56×10^5	$\mathbf{1.11 \times 10^4}$	21.74	**3.55**	2.99×10^8	$\mathbf{3.33 \times 10^3}$
Benzene	1.26×10^5	$\mathbf{4.14 \times 10^3}$	10.94	**1.07**	4.99×10^8	$\mathbf{6.80 \times 10^2}$
Ethanol	2.50×10^4	$\mathbf{1.14 \times 10^3}$	18.02	**4.00**	5.86×10^7	$\mathbf{5.15 \times 10^2}$
Malonaldehyde	4.18×10^4	$\mathbf{3.69 \times 10^3}$	19.90	**3.72**	4.57×10^7	$\mathbf{1.43 \times 10^3}$
Naphthalene	7.66×10^4	$\mathbf{4.61 \times 10^3}$	19.52	**2.39**	7.73×10^7	$\mathbf{1.44 \times 10^3}$
Paracetamol	1.44×10^5	$\mathbf{9.76 \times 10^3}$	20.02	**4.89**	1.22×10^9	$\mathbf{4.15 \times 10^3}$
Salicylic Acid	3.97×10^4	$\mathbf{9.44 \times 10^3}$	20.26	**3.65**	4.36×10^7	$\mathbf{2.40 \times 10^3}$
Toluene	7.93×10^4	$\mathbf{1.37 \times 10^3}$	19.52	**2.78**	8.97×10^7	$\mathbf{1.18 \times 10^3}$
Uracil	7.69×10^4	$\mathbf{3.37 \times 10^3}$	20.05	**3.66**	1.15×10^8	$\mathbf{2.07 \times 10^3}$
ISO17	**129.48**	277.29	0.82	**0.25**	2.82×10^3	**7.33**

We observe that, in most cases, SchNet outperforms the baseline for every task. The exact relative improvements can be seen in Appendix A in Table 6. Only in the energy tasks, the baseline outperforms SchNet in 2 cases, namely the energy prediction task for MD17 with the aspirin molecule and ISO17. We explain this as follows.

Even with the models trained in Section 3.2, which were computed with far more training time, we observe signs of undertraining. We fit a linear regression on the training loss and determine that there is still a clear downward trend to the loss. This indicates that the model is undertrained and needs more training time to reach an optimal state. We observe this in detail for the MD17 aspirin model in Figure 1.

Figure 1: Training losses for the energy prediction of MD17 with aspirin, while training with 50,000 data points and 10 epochs. Each time step t is one batch with the train loss and a linear regression of it in orange. The first and last values of the linear regression is displayed. We include the last 10,000 batch steps, which are approx. 6 epochs.



Furthermore from the first milestone report, we determined that the magnitude of values is very large [9]. Therefore, we examine the validation set for both tasks and analyze the standard deviation relative to the mean magnitude. For this, we calculate the coefficient of variation (CV) to analyze the standard deviation in relation to the mean. For the MD17 aspirin case, the CV is

0.001% (mean -4.07×10^5, std. 6.03) while for ISO17 it is 0.006% (mean -1.15×10^4, std. 0.7), requiring exceptionally low output variance. Perhaps the network also needs to be trained with a lower learning rate in later batches to capture these small relative changes, which we suggest for further study.

3.2 Comparison to the Original Paper

We observe that it outperforms our models in all benchmarks. We expected this, since we only trained for 10 epochs and Schütt *et al.* [1] did not specify the number of epochs or the training time. Getting close to the metrics of the original author shows that our implementation is correct. We also notice that Schütt *et al.* [1] has not provided any metric for the MD17 azobenzene and MD17 paracetamol data sets. We provide detailed results in Appendix B, which compares our trained SchNet model performance with the original SchNet authors.

However, similarly to the baseline case, the energy predictions seem to be far off. This is due to the models being undertrained, even with an increase in training time, as already described in Section 3.1. In Figure 2 in Appendix C we display all training losses for all models trained on MD17. Each model still exhibits a clear downward trend for training losses.

3.3 Empirical Estimate of the Generalization Error

We choose k-fold cross-validation to estimate the generalization error, which is simple to implement and is widely used in practice [14]. Furthermore, the large amount of data allows cross-validation to more accurately estimate the variance of the error, although this estimate is not generally well justified [15], [16]. We use the resulting k models as an ensemble model to estimate a confidence measure for the predictions, presented in Section 4.2.

We choose $k = 5$ as the number of folds. A lower number of folds results in higher bias estimates, and five or ten folds are generally recommended [14]. The model might fit better in the particular fold but may not generalize well, emphasizing the generalization capabilities of the model. Additionally, due to the large amount of data, this number of folds is more computationally efficient.

We set up the cross-validation method with the most challenging data set, ISO17. For each $i \in \{1, ..., k\}$, we use the fold i as a validation set and train the model on the remaining folds. Training hyperparameters are chosen as described in Section 2.2. The results for the cross-validation can be seen in Table 3.

Table 3: Results for 5-fold cross-validation, with minimum, maximum, mean loss and the standard deviation for the respective folds (rounded to two decimal places).

Fold	Minimum	Maximum	Mean	Std.
1	1.66	2.03	1.81	0.06
2	5.91	33.68	10.85	5.48
3	5.53	189.05	7.86	12.17
4	1.52	4.59	2.22	0.61
5	2.56	3.08	2.81	0.09

We find, that the model trained during Folds 1, 4, and 5 exhibit more stable and consistent performance. The loss metrics collected are low with an exceptionally low standard deviation. This means that the model has learned these folds well and has low bias and variance. However, the models of Folds 2 and 3 show high standard deviation. This suggests that the model performance is more sensitive to changes in the training data. For the case of the ISO17 data set, we believe that the model might not be able to generalize well to specific or unknown molecule configurations. We suggest further analysis of the behavior of the model on these unknown molecule/conformation configurations.

4 Discussion

In the following section, we examine the suitability of the solution proposed by SchNet, in terms of complexity and retrainability. We provide a confidence measure for the predictions and also address numerical problems with the activation function.

4.1 Model Complexity

We briefly analyze the complexity of SchNet with the related work, described in Section 2.1. We found a public repository for DTNN and use it to calculate the exact amount of parameters [17]. GDML and SOAP are kernel-based methods and are summarized as such. Unfortunately, we were not able to locating publicly available code for enn-s2s, leading to its exclusion from this analysis.

The default configurations for DTNN [3] and SchNet have a fixed size with 10,801 and 116,225 parameters, respectively. Therefore, SchNet has the potential to comprehend more intricate patterns, explaining the better benchmarks. Although generalization error can be an issue in complex models, SchNet still performs better than DTNN.

If we consider the scalability with the number of training examples, SchNet scales linearly, making it efficient for a larger training set. In comparison, the computation cost and memory size required for kernel-based methods increase rapidly with the number of data points: the training time scales with $\mathcal{O}(N^3)$ and the memory requirement for the kernel matrix with $\mathcal{O}(N^2)$ [18]. Therefore, when handling vast datasets (larger than 50k data points), neural network methods such as SchNet offer better computational scaling and lower memory requirements.

In addition, efficient retraining of classifiers is a critical aspect, especially in dynamic environments where new data become available over time. DTNN and SchNet are neural networks; hence, these models can be easily re-trained with new data due to the stochastic nature of the learning algorithms. In contrast, kernel matrices needs to be recomputed for new data, making this method prohibitively expensive.

4.2 Confidence Measure

We estimate a confidence measure for the results of the model by employing ensemble neural network models [19]. For this, we use the resulting models from k-fold cross-validation in Section 3.3. This means that we first perform cross validation on k-folds to generate k models. However, we keep another fold separate from the procedure. This will act as the validation set for the ensemble model. We combine the models as an ensemble to evaluate the predictions' statistics.

For each batch in the validation set, the ensemble model evaluates the predictions of k energy and forces, each per model. We use the mean of the predictions as the final prediction of the ensemble model and calculate the standard deviation of the ensemble predictions. Finally, we compare the standard deviations for the entire validation set. We present the results in Table 4.

Table 4: Standard deviation of the 5-fold ensemble neural network model acquired from the 5-fold cross validation. The standard deviation is calculated on a separate validation set and is rounded to two decimal places.

Std. for	Minimum	Maximum	Mean
Energy	0.53	62.16	5.08
Forces (X)	0.02	212.24	0.78
Forces (Y)	0.01	36.01	0.64
Forces (Z)	0.01	90.03	0.63

We find the model architecture seems to produce results that are variable for the energy predictions, implying low confidence. This coincides with the analysis from Section 3.3. It seems like some models of the ensemble are better for certain types of molecule configurations than others. The models seem to be highly sensitive to the input data and might suffer from poor generalization capabilities for the energy predictions.

The exceptions are predictions for forces. First, we note that for some outlier points, the range of standard deviation is exceptionally high. However, for all three coordinates, the mean standard deviation is low, implying confidence between the model predictions. Therefore, predictions for the forces are well learned, even between different data set partitions, although some outliers persist.

4.3 Numerical Instability of the Shifted Softplus Activation Function

Initially, we used a naive implementation of the SSP described in Section 2.3, encountering issues where the model produced Not a Number (NaN) during training due to the unbounded use of the exponential function of the SSP. By default, PyTorch implements tensors using float32 [7]. When the exponential function receives a very large input, the output is not representable in float32, resulting in a buffer overflow (maximal input value is approx. $\log(3.4 \times 10^{38}) \approx 88.72$ [20]). Inputs exceeding this value will result in the exponential function evaluating to infinity (inf) and operations on it NaNs.

Switching to double precision would expand the representable range, but double the memory usage, without performance benefits. We explain this in Section 4.3.1 where SSP approximates the identity beyond a certain input threshold.

We set up the following experiment to analyze the numerical behavior of the SSP. We try to overfit on a single mini-batch of size 8 of the Aspirin data set of MD17 with the remaining hyperparameters staying identical as described in Section 2.2.6. During each iteration, we capture the maximum scalar values of the inputs to the SSP function in the interaction and output layers until the output of the network produces NaNs. In this particular example, this occurred after four epochs. The findings are presented in Table 5.

Table 5: Model layers with the maximum input to the SSP activation function at the fourth epoch.

Layer	Maximum Input x	$\exp(x)$
Interaction Layer 0	8.78	6502.87
Interaction Layer 1	56.92	5.24×10^{24}
Interaction Layer 2	300.01	inf
Atom Wise Output	NaN	NaN

At this point, the values have grown so large that the exponential function becomes numerically unstable. The exponential function produces inf and other arithmetic operations on tensors containing it will produce NaN. We present an approach to solve the issue.

4.3.1 Linear Approximation after a Threshold

As described in 2.2.4, the SSP is a modified version of the Softplus activation function. We observe the following identity for the SSP.

$$\begin{aligned} \operatorname{ssp}(x) &= \log(0.5 \exp(x) + 0.5) \\ &= \log(0.5(\exp(x) + 1)) \\ &= \log(0.5) + \log(\exp(x) + 1) \end{aligned}$$

The resulting expression is the sum of a constant term and the regular Softplus function. We observe that for larger x the expression $\log(\exp(x)+1)$ approximates $\log(\exp(x)) = x$, as the term $+1$ becomes negligible, suggesting an identity function. This is exactly how PyTorch achieves numerical stability in its implementation [7]. If the input is greater than the threshold value of 20, the Softplus function reverts to the identity and thus avoids the numerical problems of the exponential function.

Additionally, this approximation does not change the behavior of the gradient for the SSP. Let z be the output of a network layer preactivation and $\boldsymbol{\theta}$ be the parameters of the network. Then the gradient of z w.r.t. $\boldsymbol{\theta}$ is given as:

$$\nabla_{\boldsymbol{\theta}} \operatorname{ssp}(z) = \frac{\exp(z)}{\exp(z) + 1} \nabla_{\boldsymbol{\theta}} z = \frac{1}{1 + \exp(-z)} \nabla_{\boldsymbol{\theta}} z$$

We see for large z, that the expression $\frac{1}{1+\exp(-z)}\nabla_{\boldsymbol{\theta}} z$ approximates $\nabla_{\boldsymbol{\theta}} z$ as $\exp(-z)$ approaches zero. This is exactly the gradient of the identity function of z.

5 Conclusion

In conclusion, this milestone report details the implementation of SchNet addressing multiple issues found in the original paper. We use k-fold cross-validation to not only provide an estimate of the model's generalization error but also establish a confidence measure through an ensemble neural network approach. Comparative analysis against baseline methods and the original paper reveals subpar energy prediction performance in our implemented model. However, the model still vastly outperforms the baseline in most tasks.

We were able to reproduce some results of the paper regarding the force predictions. Based on this and compared to the related work, SchNet shows many strengths, indicating the capabilities of the architecture.

5.1 Future Work

In this section, we outline potential directions for future research and development to enhance the capabilities of our machine learning model, particularly focusing on SchNet and its application in modeling quantum interactions.

Our current implementation does not incorporate normalization techniques such as batch and layer normalization. Future work could explore the integration of these normalization methods to improve model performance. Normalization techniques have been shown to stabilize and accelerate training in deep learning models [21], [22].

Similarly, the model does not incorporate any regularization techniques, which would aid in its generalization capabilities. Common modern techniques such as weight decay and dropout could be incorporated to achieve a lower generalization error [15].

The current model configuration was selected based on a combination of literature review and empirical testing. Future research could employ more systematic approaches such as HyperGrid search or Bayesian optimization to identify optimal hyperparameters. This could potentially lead to improved accuracy and efficiency in the model [23]. Additionally, the activation function can also be seen as a hyperparameter. While the Softplus activation function and its derivative, SSP, are appropriate for certain tasks, more suitable activation functions might increase model performance [24].

Although SchNet has shown promising results, exploring other architectures designed for graph-structured data could yield further improvements. Graph Neural Networks (GNNs) and models incorporating attention mechanisms are potential candidates for experimentation [25], [26]. These architectures could offer enhanced capabilities in capturing complex relationships in molecular data.

Future iterations of the model could benefit from the integration of domain-specific knowledge in quantum chemistry. This could involve customizing the network architecture or loss functions to better reflect the underlying physical principles of molecular interactions [27].

Further research should also focus on enhancing the robustness and generalization ability of the model across diverse molecular datasets. This includes testing the model on unseen molecules and under different environmental conditions to ensure its applicability in real-world scenarios [28].

Finally, engaging in collaborative research with experts in quantum chemistry and contributing to open-source projects can accelerate advancements in this field. Sharing resources, datasets, and findings with the broader community can lead to more robust and innovative solutions [29].

References

[1] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, *et al.*, "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions," no. arXiv:1706.08566, Dec. 2017. arXiv: 1706.08566 [physics, stat].

[2] S. Chmiela, A. Tkatchenko, H. E. Sauceda, *et al.*, "Machine learning of accurate energy-conserving molecular force fields," *Science Advances*, vol. 3, no. 5, e1603015, 2017. DOI: 10.1126/sciadv.1603015.

[3] K. T. Schütt, F. Arbabzadah, S. Chmiela, *et al.*, "Quantum-chemical insights from deep tensor neural networks," *Nature Communications*, vol. 8, no. 13890, 2017.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, *et al.*, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 1263–1272.

[5] J. Airas, X. Ding, and B. Zhang, "Transferable implicit solvation via contrastive learning of graph neural networks," *ACS Central Science*, vol. 9, no. 12, pp. 2286–2297, 2023. DOI: 10.1021/acscentsci.3c01160. eprint: https://doi.org/10.1021/acscentsci.3c01160.

[6] B. Olsthoorn, R. M. Geilhufe, S. S. Borysov, *et al.*, "Band gap prediction for large organic crystal structures with machine learning," *Advanced Quantum Technologies*, vol. 2, no. 7–8, Jul. 2019, ISSN: 2511-9044. DOI: 10.1002/qute.201900023.

[7] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 721, Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 8026–8037.

[8] A. H. Larsen, J. J. Mortensen, J. Blomqvist, *et al.*, "The atomic simulation environment—a python library for working with atoms," *Journal of Physics: Condensed Matter*, vol. 29, no. 27, p. 273 002, 2017.

[9] C. K. Loan, T. D. Ha, and A. Oesch, *Project Machine Learning Milestone 1: Data Sets and Prototype*, Nov. 2023.

[10] M. M. Bronstein, J. Bruna, Y. LeCun, *et al.*, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[11] K. He, X. Zhang, S. Ren, *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, Jan. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs].

[13] Q. Wang, Y. Ma, K. Zhao, *et al.*, "A Comprehensive Survey of Loss Functions in Machine Learning," *Annals of Data Science*, vol. 9, no. 2, pp. 187–212, Apr. 2022, ISSN: 2198-5804, 2198-5812. DOI: 10.1007/s40745-020-00253-5.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics). New York, NY: Springer New York, 2009, ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] Y. Bengio and Y. Grandvalet, "No Unbiased Estimator of the Variance of K-Fold Cross-Validation," *The Journal of Machine Learning Research*, vol. 5, pp. 1089–1105, Dec. 2004, ISSN: 1532-4435.

[17] *Atomistic-machine-learning/dtnn: Deep Tensor Neural Network*, https://github.com/atomistic-machine-learning/dtnn.

[18] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (Adaptive Computation and Machine Learning). Cambridge, Mass: MIT Press, 2002, ISBN: 978-0-262-19475-4.

[19] K. M. R. Alam, N. Siddique, and H. Adeli, "A dynamic ensemble learning algorithm for neural networks," *Neural Computing and Applications*, vol. 32, no. 12, pp. 8675–8690, Jun. 2020, ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-019-04359-7.

[20] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul. 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[21] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, Jun. 2015, pp. 448–456.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[23] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[24] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, Sep. 2022, ISSN: 09252312. DOI: 10.1016/j.neucom.2022.06.111.

[25] F. Scarselli, M. Gori, A. C. Tsoi, *et al.*, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[26] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[27] J. Gilmer, S. S. Schoenholz, P. F. Riley, *et al.*, "Neural message passing for quantum chemistry," *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272, 2017.

[28] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014. arXiv: 1412.6572 [cs.LG].

[29] E. S. Raymond, *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media, Inc., 2001.

A Appendix: Relative Improvement of SchNet over Baseline

Table 6: Relative improvement of the loss of SchNet over the baseline for different data sets and categories, rounded to two decimal places. A negative value indicates that the loss for SchNet is lower than the baseline, thus performing better, written in bold. We report energy and force predictions in kcal/mol and kcal/mol/Å respectively.

Data Set	Energy	Force	Energy+Force
QM9	$\mathbf{-2.63}$	n.a.	n.a.
MD17			
Aspirin	3.45×10^3	$\mathbf{-13.6}$	$\mathbf{-2.58 \times 10^7}$
Azobenzene	$\mathbf{-1.45 \times 10^5}$	$\mathbf{-18.2}$	$\mathbf{-2.99 \times 10^8}$
Benzene	$\mathbf{-1.22 \times 10^5}$	$\mathbf{-9.87}$	$\mathbf{-4.99 \times 10^8}$
Ethanol	$\mathbf{-2.39 \times 10^4}$	$\mathbf{-14.0}$	$\mathbf{-5.86 \times 10^7}$
Malonaldehyde	$\mathbf{-3.81 \times 10^4}$	$\mathbf{-16.2}$	$\mathbf{-4.57 \times 10^7}$
Naphthalene	$\mathbf{-7.20 \times 10^4}$	$\mathbf{-17.1}$	$\mathbf{-7.73 \times 10^7}$
Paracetamol	$\mathbf{-1.34 \times 10^5}$	$\mathbf{-15.1}$	$\mathbf{-1.22 \times 10^9}$
Salicylic Acid	$\mathbf{-3.03 \times 10^4}$	$\mathbf{-16.6}$	$\mathbf{-4.36 \times 10^7}$
Toluene	$\mathbf{-7.79 \times 10^4}$	$\mathbf{-16.7}$	$\mathbf{-8.97 \times 10^7}$
Uracil	$\mathbf{-7.35 \times 10^4}$	$\mathbf{-16.4}$	$\mathbf{-1.15 \times 10^8}$
ISO17	1.48×10^2	$\mathbf{-0.57}$	$\mathbf{-2.81 \times 10^3}$

B Appendix: Our SchNet vs Authors SchNet

Table 7: Comparison between the authors SchNet (A-SchNet) models and our trained SchNet models (O-SchNet) on test data for energy, force, combined energy force tasks, rounded to two decimal places. Better performance (lower loss) is highlighted in bold. Both models were trained with 50,000 train and 1,000 test samples for 10 epochs. We report energy and force predictions in kcal/mol and kcal/mol/Å respectively.

Data Set	Energy		Force		Energy+Force	
	O-Schnet	*A-SchNet*	*O-Schnet*	*A-SchNet*	*O-Schnet*	*A-SchNet*
QM9	0.60	**0.59**	n.a.	n.a.	n.a.	n.a.
MD17						
Aspirin	1.44×10^3	**0.25**	1.25	**0.33**	868	**0.33**
Azobenzene	1.02×10^3	n.a	1.18	n.a	330	n.a
Benzene	1.74×10^2	**0.08**	0.38	**0.17**	22	**0.17**
Ethanol	1.14×10^2	**0.07**	0.86	**0.05**	344	**0.05**
Malonaldehyde	4.33×10^2	**0.13**	1.03	**0.08**	300	**0.08**
Naphthalene	2.26×10^3	**0.20**	0.80	**0.11**	184	**0.11**
Paracetamol	9.80×10^2	n.a	1.00	n.a	531	**n.a**
Salicylic Acid	2.18×10^3	**0.25**	1.09	**0.19**	678	**0.19**
Toluene	9.36×10^2	**0.16**	0.87	**0.09**	159	**0.09**
Uracil	1.65×10^3	**0.13**	1.16	**0.11**	380	**0.11**
ISO17	304	**0.52**	9.62	**4.13**	1.44	**1.00**

C Appendix: Train Losses For MD17

Figure 2: Training losses for the energy prediction of MD17 for the long training. Each time step t is one batch with the train loss and a linear regression of it. The first and last value of the linear regression is displayed. We only display the last 10,000 batch steps, which are approx. 6 epochs.