Uncertainty Quantification in Machine Learning and Nonlinear Least Squares Regression Models

Ni Zhan[1] and John Kitchin[1]

[1]Carnegie Mellon University

August 5, 2021

Abstract

Machine learning (ML) models are valuable research tools for making accurate predictions. However, ML models often unreliably extrapolate outside their training data. We propose an uncertainty quantification method for ML models (and generally for other nonlinear models) with parameters trained by least squares regression. The uncertainty measure is based on the multiparameter delta method from statistics, which gives the standard error of the prediction. The uncertainty measure requires the gradient of the model prediction and the Hessian of the loss function, both with respect to model parameters. Both the gradient and Hessian can be readily obtained from most ML software frameworks by automatic differentiation. We show that the uncertainty measure is larger for input space regions that are not part of the training data. Therefore this method can be used to identify extrapolation and to aid in selecting training data or assessing model reliability.

Introduction

Machine learning (ML) models are used in many fields of science and engineering. They can decrease computational time, make predictions and forecasts, and improve insights of complex and high dimensional datasets. Models are more useful when they provide a prediction with its uncertainty, and in some applications it may be critical to provide a reliable uncertainty estimate (*Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift*, 2019; Benjamin Kompa & Beam, 2021). The uncertainty estimate should also help identify input data regions that lead to extrapolation. Overall, ML models can be used more reliably when robust uncertainty quantification is available. In the simple case of low dimensional linear regression, an analytical prediction interval is available (Wasserman, 2004), and that can be used to calculate the uncertainty intervals in many statistical software packages. The analytical prediction interval for linear regression requires $(\mathbf{X}^T\mathbf{X})$ to be invertible, where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the design matrix with n data points and d feature dimensions. In general for more complex (nonlinear) models and higher dimension datasets, analytical prediction intervals do not exist and alternative methods are required.

Uncertainty quantification methods for ML models are an active area of research (*A Simple Baseline for Bayesian Uncertainty in Deep Learning*, 2019; *Subspace Inference for Bayesian Deep Learning*, 1169; *Simple and Principled Uncertainty Estimation with Deterministic Deep Learning Via Distance Awareness*, 2020). Some common methods are model ensembling and training models with built in uncertainty such as Gaussian process (GP) regression and quantile regression (*Distribution-Free Predictive Inference for Regression*, 2016; Yaniv Romano & Candès, 2019). If we already have a model with trained parameters, we may want to avoid training a different model type or additional model ensemble. In these cases, this paper presents a simple uncertainty quantification method for parameterized models trained on minimizing the summed squared error between a model and a dataset. The method exploits automatic differentiation to calculate the Hessian of the loss function based on summed squared errors, and provides an uncertainty estimate which depends on the prediction point, model training data, and model itself. We show how the uncertainty can identify if extrapolation is occurring and aids in dataset selection for an example application of molecular simulation. In the remaining sections, we review the background on uncertainty quantification and an application of ML models for molecular simulation, describe the method called the delta method, and show its use in an example neural network that predicts energies from atomic structures.

Uncertainty Quantification Methods

An ML model is defined by its mathematical definition and training dataset. In complex models, the mathematical definition may be best defined with a computing program (code) and includes the model structure, parameters, and hyperparameters. The training data is most likely preprocessed and modified to normalize and standardize them. The ML models are fitted to the training data, usually by minimizing an error function. In parameterized models, the training process determines the model parameters. We take the model parameters, the error function, exact training dataset with its modifications, and all aspects of the model's mathematical definition to be the ML model.

Here we describe the motivation and need for the uncertainty estimate. When we use a trained ML model to predict on another dataset, the prediction accuracy depends on the training data used. If the new data is in an extrapolation region, the accuracy is not expected to be good (Samad Hajinazar & Kolmogorov, 141AD). It is nonobvious when the model is predicting in an extrapolation region if the ML model has a high dimension input space. A motivation for an uncertainty measurement is to help determine when the model is extrapolating, or is simply not reliable. Knowledge of the uncertainty helps identify when a model is reliable and indicates confidence in a model prediction.

Uncertainty quantification methods for models include bootstrap, ensembling, using model-specific uncertainty, and the delta method. Bootstrap, ensembling, and the delta method can be used for parametric models and neural networks (NNs). Bootstrap uncertainty is based on statistical theory, has some different variations, and requires training multiple models on different bootstrap samples of the data or residuals (Tomohiro Endo & Yamamoto, 2015; *Calibrated Bootstrap for Uncertainty Quantification in Regression Models*, 2021; Hongfei Du & Jin, 1207). With the different models, the uncertainty on predictions can be estimated.

Ensemble methods require training multiple models on the entire dataset, and the different models give the uncertainty (*A Review of Uncertainty Quantification in Deep Learning: Techniques Applications and Challenges*, 2021; Florian Wenzel & Jenatton, 2020; Balaji Lakshminarayanan & Blundell, 2016). The uncertainty estimate quantitatively improves as the ensemble size increases, so the optimal number of ensemble models is unknown and must be user determined (Berger & Smith, 2019). Some studies used bootstrap uncertainty for NN models and showed that it gave more reliable uncertainties than the delta method (Tibshirani, 1996; Dybowski & Roberts, 2001), but training multiple models is computationally expensive and time consuming.

Model-specific uncertainty include Gaussian process regression (Rasmussen & Williams, 2006), dropout for NNs (Gal & Ghahramani, 2015), and Bayesian NNs (Mackay, 1992; Neal, 1996; Charles Blundell & Wierstra, 2015). Obtaining uncertainties in this way limits the possible mathematical forms of the model. Dropout

and Bayesian NNs are also more difficult to train than standard NNs. Section describes specific instances in which ensembling, bootstrap, and Gaussian process methods were used for molecular simulation. In another method, the posterior of model parameters can be approximated by the Laplace approximation, which is the second-order Taylor series expansion around the optimal model parameters (Hippolyt Ritter & Barber, 2018; Huijie Tian & Rangarajan, 2019). Hence the Hessian of the log likelihood contains relevant information about uncertainty.

This work focuses on the delta method, a method which also uses the Hessian of the log likelihood. The method is based on linearly approximating the model and uses an estimate of the standard error of model parameters assuming maximum likelihood estimation (MLE). Therefore, the method applies to models with parameters trained by minimizing squared error, and the model structure could be a simple linear regression to complex nonlinear regression including NNs. Different variants of the method use approximations of the Hessian, and experiments tested the delta method on NN models (Donaldson & Schnabel, 1987; Tibshirani, 1996; *Prediction Intervals for Neural Networks Via Nonlinear Regression*, 1998; G. Papadopoulos & Murray, 1278; Settles, 2009; A. Khosravi & Atiya, 1341). We further describe the method in Section with theoretical details in the Appendix (Section).

Addressing Uncertainty in Molecular Simulation

We examine uncertainty quantification using the example application of molecular simulation, an area which has benefited from ML. Here we describe the background of ML for molecular simulation. Simulations allow researchers to obtain materials' physical properties and quickly screen materials. Molecular dynamics (MD) and Monte Carlo simulations require a model of the potential energy surface (PES) which predicts energies and forces from atomic configurations. The options for the PES model include first principles methods such as density functional theory (DFT), physical potentials, and ML potentials. ML potentials aim to achieve the high accuracy of DFT at a significantly faster computational time. ML potentials are also more systematically improvable than physical potentials (Kitchin, 2018). Many studies have successfully used ML potentials in simulations (Boes & Kitchin, 2017; Khosrow Shakouri & Kroes, 2131; *Neural Network Potential for Al-Mg-Si Alloys*, 536AD; *Realistic Atomistic Structure of Amorphous Silicon from Machine-Learning-Driven Molecular Dynamics*, 2879).

Uncertainty quantification is useful for ML potentials. Commonly used ML potentials such as NNs will usually unreliably extrapolate on inputs much different from their training data. A consequence of extrapolation during a molecular simulation is that it likely gives wrong or unphysical results. The best ways to select enough of the relevant training space are nonobvious, since the space of atomic structures is often large, not

well understood, and not possible to enumerate. Further, atomic structures are translated into fingerprints which are high dimensional and less human-interpretable than the original atomic configurations. Hence, we require a method to determine the uncertainty of a prediction from a ML potential, and the quantitative uncertainty helps us avoid extrapolation and identify sparse regions in the training dataset.

Current methods developed to address uncertainty are ensemble of potentials, on-the-fly methods, and using ML models with built-in uncertainty. Ensemble methods independently train two or more ML potentials, and check for agreement between them. In Behler's approach, they trained NNs with different architectures, and atomic structures whose predictions' differ significantly across NNs are added to the training set (J Behler, 1830; Behler, 1032; Behler, 1282). Peterson et al. trained an ensemble of 50 NN potentials and found that ensemble spread was a good indicator for prediction error across the space (Andrew A. Peterson & Khorshidi, 1097). Smith et al. also used ensemble disagreement to approximate prediction error and select a small training set (*Less Is More: Sampling Chemical Space with Active Learning*, 2417; *Approaching Coupled Cluster Accuracy with a General-Purpose Neural Network Potential Through Transfer Learning*, 2019).

In MD simulations, on-the-fly methods use a ML potential augmented with quantum mechanical (QM) calculations (Zhenwei Li & Vita, 964AD). There is a query if the ML prediction can be used. If it fails, a QM calculation is run and added to a database, and the ML model can be retrained. A simple query is if the fingerprint is out of the minimum and maximum bounds in the current database (Botu & Ramprasad, 1074). This is a minimum requirement that the ML model is not extrapolating; however, guaranteeing that the fingerprint is within bounds of training data does not guarantee a low error (Artrith & Behler, 454AD).

Another approach is training Gaussian process regressions (Zhenwei Li & Vita, 964AD; Albert P. Bartok & Csanyi, 2018) or other ML models with built-in uncertainty estimates. Vandermause and Xie et al. used GP uncertainty to train potentials on-the-fly (*On-The-Fly Active Learning of Interpretable Bayesian Force Fields for Atomistic Rare Events*, 2020; *Bayesian Force Fields from Active Learning for Simulation of Inter-Dimensional Transformation of Stanene*, 2021). Many ML models, such as NNs, do not have theoretical guarantees for uncertainty of a prediction. Perturbation of NN weights could provide some range of uncertainty (Christensen, 2016). Other work used dropout in NN training as a Bayesian approximation and thereby calculating uncertainties for interatomic potentials (Wen & Tadmor, 2020). Janet et al. used the distance in values of the last layer of NNs (or latent space) as an uncertainty measure (*A Quantitative Uncertainty Metric Controls Error in Neural Network-Driven Chemical Discovery*, 2019). Tran et al. compared GP, Bayesian NN, dropout NN, and ensembles of different NN structures and found more conservative uncertainties for GP and overconfident uncertainties for Bayesian NN, dropout, and NN ensemble (*Methods for Comparing Uncertainty Quantifications for Material Property Predictions*, 2020).

Musil et al. compared GP, ensembling with random subsets of the data, and bootstrap methods for obtaining uncertainties of predicting formation energies on molecular datasets (Félix Musil & Ceriotti, 95AD). They found that random sampling was easier to implement than bootstrapping, computationally faster than GP for uncertainty estimates, and matches the true error and uncertainty from GP. Li et al. trained NN potentials with different NN structures (number of nodes), weight initialization, and learning rates, and compared the resulting prediction accuracies (Yumeng Li & Wang, 2018). Their work showed a quantitative uncertainty arising from some NN hyperparameters, but it required training several NN potentials for a new system, and does not provide confidence or prediction intervals. In an alternative approach, Botu et al. fitted an empirical function to an uncertainty estimate as a function of fingerprint distance between input and reference training fingerprints (V. Botu & R. Ramprasad, 2016). Their uncertainty estimation approach requires a larger training set size.

Overall, there is not a clear consensus on the best uncertainty quantification method, and its selection usually depends on the model form used, e.g. built-in uncertainty from GPR or ensembles when using NNs. The delta method provides a simple alternative for providing quantitative uncertainty when a pretrained model exists, without the necessity of training additional models. That is the focus of this work.

Methods

The delta method applies to regression problems of a model with parameters $g(\theta)$. The residuals of model prediction are assumed to be Gaussian distributed and centered around zero. We assume the model parameters $\hat{\theta}$ were obtained by minimizing a function of the summed squared errors, although the method can be extended to maximize a posteriori estimation and cross-entropy loss. We obtain an approximate standard error of a model prediction $g(\hat{\theta}, x)$ by using a Taylor series approximation and an approximate standard error of $\hat{\theta}$. Equation 1 shows the standard error $g(\hat{\theta}, x)$ for a point x, following the delta method.

$$\mathrm{se}(g(\hat{\theta}, x)) \approx \sqrt{\frac{\partial g(\hat{\theta}, x)}{\partial \hat{\theta}}^T I_n^{-1} \frac{\partial g(\hat{\theta}, x)}{\partial \hat{\theta}}} \tag{1}$$

where $\frac{\partial g(\hat{\theta}, x)}{\partial \hat{\theta}}$ is the gradient of the model with respect to parameters at the point x for which we are

calculating uncertainty, $\frac{\partial g(\hat{\theta},x)}{\partial \hat{\theta}}$ nonzero, and I_n is the Fisher information matrix, defined as the expectation of the Hessian of the negative log likelihood. The Fisher information is related to the Hessian of the loss, usually the sum of squared errors, by a scaling factor. Equation 1 shows that the model prediction of the standard error is a function of the training data, model, and point for which the uncertainty is calculated.

For small to medium models, the delta method is faster and easier to implement compared to ensembling, and the Hessian and gradients of the model are readily obtained with automatic differentiation that is included in most machine learning packages. To demonstrate the ease of use, we show a simple code example using the `autograd` (autograd, n.d.) package in Section . The delta method is limited by model size since the Hessian will be $m \times m$ where m is number of parameters, and the Hessian needs to be inverted. For very large models, approximations to the Hessian are required for the method, and for large NNs, Kronecker-factored Hessian is an option (Hippolyt Ritter & Barber, 2018).

The uncertainties are calculated after the model has finished training, and the Fisher information inverse only needs to be calculated once per model and training data set. In previous tests, a model with 861 parameters and 1900 training data points required around five minutes to calculate the inverse Fisher information with Intel Core i7-7820HQ CPU @ 2.9GHz using `autograd`. Using more modern automatic differentiation frameworks are expected to be faster. Calculating the uncertainties after obtaining the inverse Fisher information requires much less time. Theoretically, calculation of the Fisher information matrix scales quadratically with number of parameters and linearly with number of training data points. In fact, the Hessian of the loss can be linearly separated by training data since the loss is a sum over training data points.

The quality of standard errors calculated using the delta method depends on the fit of the model. We found that well fitted models have better uncertainty measures, and our assumptions required residuals to be independent, and identically distributed normal around zero. Poorly fitted models have uncertainty measures that are less quantitatively accurate.

Practical Modifications to the Inverse Fisher Matrix

There are a few steps or best practices to modify how the inverse Fisher information matrix is computed.

1. We start with H, the Hessian of the sum squared errors loss function. For some models, such as NNs, the Hessians of the loss functions with respect to parameters are often nearly singular with some eigenvalues much larger than the others (Levent Sagun & LeCun, 2016; Guy Gur-Ari & Dyer, 2018), and the optimal parameters may be at a saddle point.

2. Add a small number ϵ to Hessian diagonal. Adding ϵ to the diagonals makes the matrix better

conditioned for taking its inverse. ϵ should be larger in magnitude than the most negative eigenvalue. We used $\epsilon = \max(1e{-}5, 1.05 \cdot \text{abs}(\lambda_{\min}(H)))$, where $\lambda_{\min}(H)$ is the smallest eigenvalue of H. Modifying the Hessian of objective function with respect to NN parameters has been suggested in literature and is justified because the top eigenvalues are a few orders of magnitude larger than the other eigenvalues (Gill & King, 2004; Guy Gur-Ari & Dyer, 2018; Hippolyt Ritter & Barber, 2018). Also note the Hessian conditioning suggests that the number of parameters of the NN is much larger than the actual degrees of freedom of the NN.

3. We take the Moore-Penrose pseudoinverse $(H+\epsilon I)^{-1}$. If the inverse exists, which is most cases following step 2, it is the same as the true inverse.

4. Multiply $(H + \epsilon I)^{-1}$ by a scaling factor α. This is done to calibrate the uncertainties to be near the residuals. We set α to be mean squared error (MSE) in most cases. To select α, we suggest trying $n^{\beta} \cdot MSE$ where n is number of training data points and β is any nonnegative number, but usually in range [0, 1]. The α should be chosen so that uncertainties are the same order of magnitude as the residual errors for the training dataset.

5. Force the scaled inverse $P := \alpha(H + \epsilon I)^{-1}$ to be positive semi-definite. For eigendecomposition $P = Q\Lambda Q^{-1}$, the closest positive semi-definite matrix in terms of Frobenius norm is $Q \max(\Lambda, 0) Q^{-1}$, where max is element-wise max (Cheng & Higham, 1998).

The final inverse Fisher information I_n^{-1} used in Equation 1 is $Q \max(\Lambda, 0) Q^{-1}$.

Code Example

We show a simple code implementation of the delta method in Listing 47 using the `autograd` package (autograd, n.d.). Note that obtaining the required gradients is a single line for the Hessian (line 24) and gradients (line 30) , demonstrating the ease of automatic differentiation. Similar codes would apply in `PyTorch` (PyTorch: `http://pytorch.org/`, n.d.) or other machine learning packages. In this simple example we fit a quadratic function to some slightly noisy data, and show the resulting confidence intervals on the fit. The Hessian in this case was well-conditioned, so the modifications described above were not necessary.

```
import autograd.numpy as np
from autograd import elementwise_grad, hessian
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from scipy.stats.distributions import t
```

```
x = np.array([0.1, 0.3, 0.5, 0.7, 0.9])          # {x, y} data
y = np.array([0.0, 0.1, 0.3, 0.5, 0.8])

def g(theta, x):
    '''function with parameters thetha'''          #

    return theta[0] * x**2 + theta[1] * x + theta[2]

def sse(theta):
    '''Summed squared error objective function'''
    return np.sum((g(theta, x) - y)**2)

initial_guess = np.array([0.1, 0.5, 0.2])
sol = minimize(sse, initial_guess)                 # minimize sse
theta = sol.x
ypred = g(theta, x)

h = hessian(sse)(theta)                            # obtain Hessian of
                                                   # sse using autograd
p = sse(theta) / len(x) * np.linalg.pinv(h)        # inverse and scale
                                                   # Hessian
uncerts = []
for xi in x:
    gprime = elementwise_grad(g, 0)(theta, xi)     # obtain gradient
                                                   # using autograd
    uncerts += [np.sqrt(gprime @ p @ gprime)]      # delta method
uncerts = np.array(uncerts)

tval = t.ppf(0.975, len(x))                        # t-value

plt.plot(x, y, 'o')                                # plot the data, fit and
```

```
                                                    # confidence intervals
plt.plot(x, ypred)
plt.plot(x, ypred + tval * uncerts, '--r')
plt.plot(x, ypred - tval * uncerts, '--r')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['Data', 'Prediction', '95% confidence'])
plt.savefig('simple-code-ex.png')
```

Listing 1: Autograd example of the delta method.

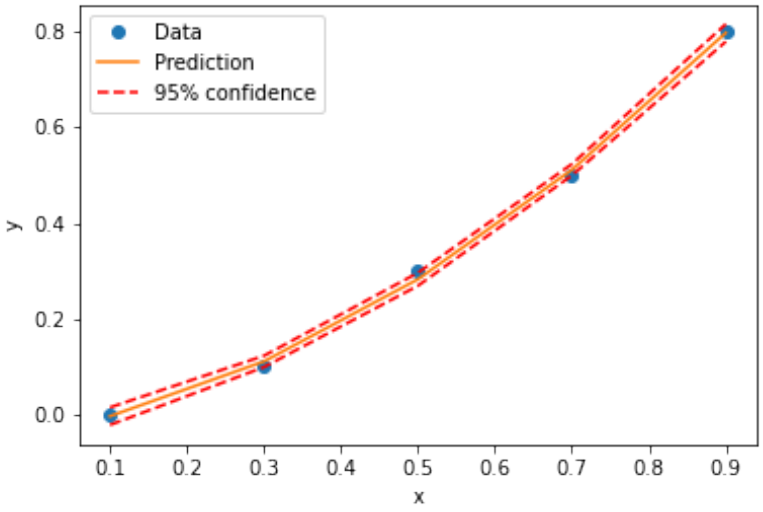

Figure 1: Result from Listing 47.

This simple example shows all the pieces of the delta method. There is data, and a function (line 11) with parameters that are fitted to the data. The regression here is done by optimization (line 20); this problem is linear and could be solved analytically, but we show the optimization approach for generality). We used automatic differentiation to obtain the Hessian (line 24) and gradient of the function (line 30) with respect to the parameters. The rest is conventional linear algebra.

In calculating the t-value (line 35), technically the degrees of freedom should be used instead of number of datapoints. However for large NNs, the model degrees of freedom is much smaller than the number of model parameters, and the effective degrees of freedom is unknown. For simplicity, we used the number of datapoints to estimate the t-value throughout our results.

Results

We show examples of using the delta method on different models to demonstrate how the uncertainty behaves. We begin with a simple 1-D NN, and build in complexity in subsequent examples.

One Dimension Input NN

This example is a one dimension input NN. We start with one dimension input for clearer intuition and visualization. We generated synthetic data from the one dimensional Lennard Jones (LJ) function and added some Gaussian noise. We fitted this data to a neural network with structure [1, 4, 1] (one input, one hidden layer with four nodes, and one output) using `scipy.optimize.minimize`. The NN had 13 total parameters.

We test how the standard error changes with different training data sets. We generated two sets of training data to fit the NN, and Fig. **??** shows the fits. These sets of training data were from the same LJ function and had the same variance of Gaussian noise added. We expect the true function to be within the confidence interval 95% of the time. In Fig. 2, the uncertainty increases for large and small x, which is desirable because we do not know how the NN will behave in those regions outside the training data. In Fig. **??**, there is a region of missing data in the middle, and the confidence interval expands in the region of missing data. These cases demonstrate that the uncertainty depends on the training data in a useful way. The uncertainty generally increases in regions with less data, which makes sense because we are less certain of our model in a space with less training data.

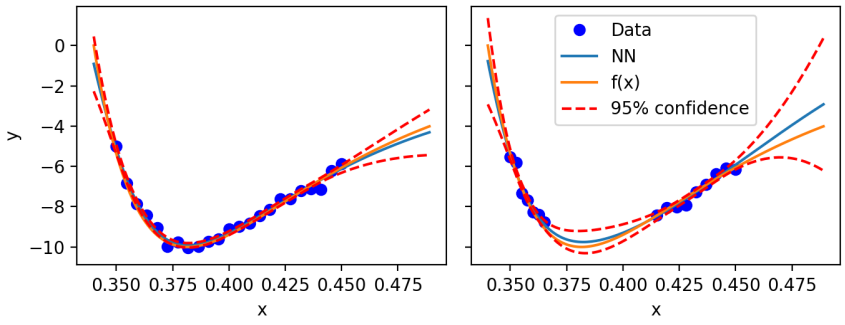

Figure 2:

High Dimensional NN Potential

Trained NN Potential

This example applies the delta uncertainty method to a high dimensional NN potential. We use the SingleNN (implemented in `PyTorch`) and weighted Behler-Parinello style symmetry functions (Behler, 741AD; *Wacsf-Weighted Atom-Centered Symmetry Functions As Descriptors in Machine Learning Potentials*, 2417; Liu & Kitchin, 1781). The data are DFT energy and force calculations based on atomic configurations, specifically the dataset used in Boes 2017 (Boes & Kitchin, 3479). The dataset contains 3907 unique AuPd slabs, and example configurations are shown in Fig. 3. The symmetry functions transform the atomic configuration information into a vector of numbers, or "fingerprint", and we used four weighted G_2 symmetry functions. For the NN, we used two hidden layers with 11 nodes each; thus the NN architecture is [4, 11, 11, 1], which is 211 total parameters.

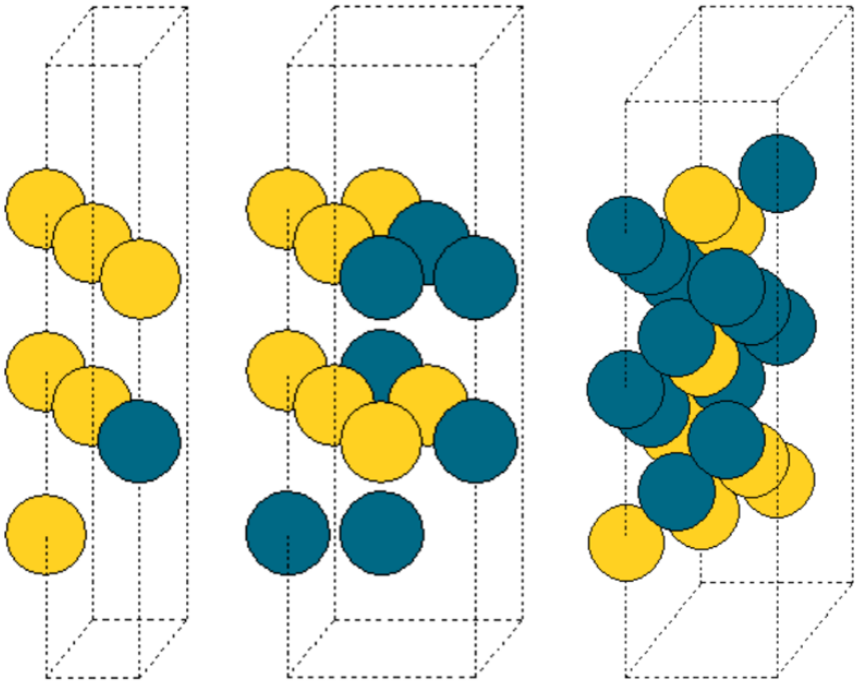

Figure 3: Three example atom configurations from dataset.

To demonstrate the usefulness of the uncertainty method, we start by training on a subset of the data. This mimics the iterative approach often used in training these models. We then check for extrapolation on the remaining data using the delta method. For this first potential, 572 configurations with a 3.934 Å lattice constant were randomly split into 64%, 16%, 20% train, validation, test sets, respectively. The NN was trained on energies and atomic forces using SingleNN, and uncertainties were calculated in the same

`PyTorch` framework.

Fig. 4 shows the energy parity plots of the training, validation, and test sets. The parity is good in all cases, and root mean squared errors (RMSEs) are 0.003, 0.0023, 0.003 eV/atom for train, validation, and test, respectively. Fig. 5 shows the distributions of standard errors of confidence for the three datasets. These distributions are very similar and mostly overlapping. Fig. 6 shows the parity plot of the test-set with 95% prediction intervals. The true values are within the prediction intervals for 98% of the dataset, which is close to 95% and shows the delta method provides quantitatively reasonable uncertainties in this case. The main result is that similar datasets with the same accuracy using the model will have similar distributions of uncertainties.

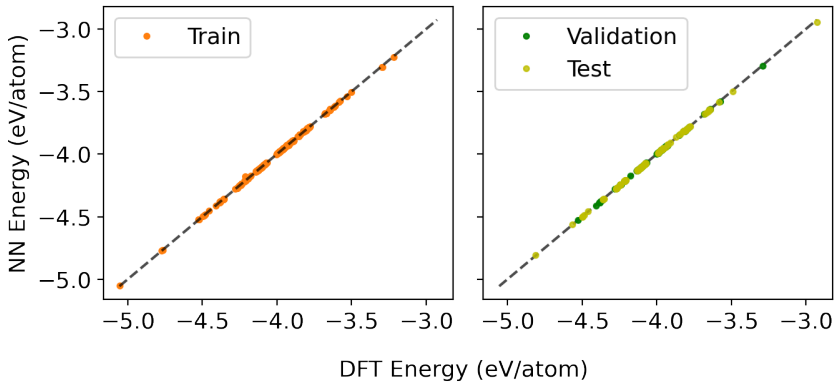

Figure 4: Parity plot of SingleNN.

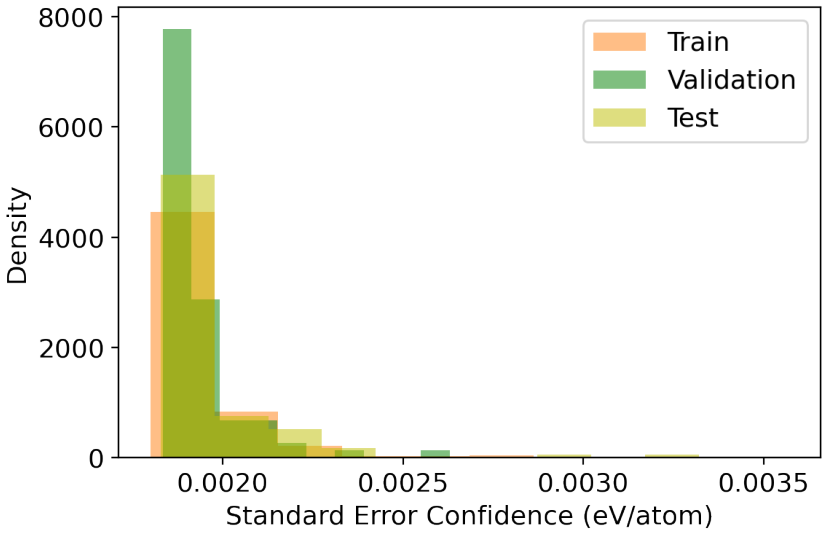

Figure 5: Distribution of uncertainties (standard error confidence).

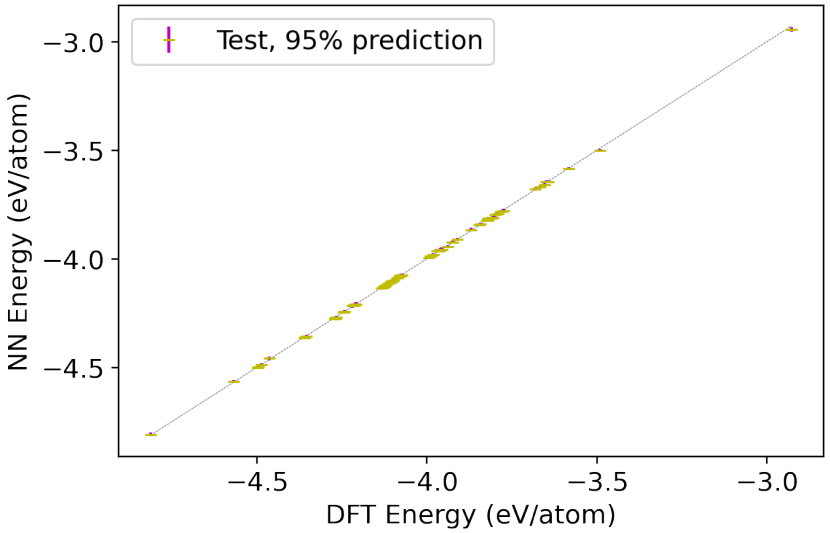

Figure 6: Parity plot with 95% prediction intervals for test set.

Next we use the same potential to predict on a new dataset. If the new dataset is dissimilar from the training data, we expect the uncertainties to be high. While the training-set all had 3.934 Å lattice constants, the new dataset has 4.034 and 4.134 Å lattice constants, which we will refer to as predict-4.0 and 4.1 datasets. As a result, we expect the fingerprints to differ from those of the train set, i.e. we know we are extrapolating here. Fig. 7 shows the energy parity plots for the predict-sets with 95% prediction intervals. The predictions are offset with an error, and the uncertainties are clearly much larger than those for the test-set from Fig. 6. Table 1 shows the average standard error of confidence/prediction for the datasets. Training and test-sets have around the same standard error confidence of 0.002 eV/atom, and predict-4.0 and 4.1 sets have higher uncertainties of 0.023 and 0.034 eV/atom, respectively, which are one order of magnitude larger than training and test. Since this uncertainty is much larger, it could indicate that the model is extrapolating on the predict-sets, and the parity plots (Fig. 7) seem to indicate this.

We examine the fingerprints, and Fig. 8 shows an example fingerprint for the train and predict datasets. There are regions where the predict-4.0 and 4.1 atoms' fingerprints are outside of the training distributions, which is suggestive of extrapolation. For predict-4.0, the true values are within the prediction intervals for 75% of the dataset, which is not that close to 95%, however for predict-4.1, the true values were within the prediction intervals for 0% of the dataset. This seems to indicate that the prediction interval becomes less quantitatively accurate as the extrapolation increases. However when the uncertainty is much larger than the training uncertainties, the model is likely extrapolating, and we should not trust the prediction. Therefore this uncertainty method helps identify the data regions where a model extrapolates. Fig. 9 shows

the standard error confidence vs. absolute energy error, and their distributions for test and predict datasets. Fig. 9 shows the general trend that uncertainty from the delta method increases when true error increases. The trend is most obvious in a heterogeneous dataset.

Table 1: Average standard errors confidence of datasets.

Dataset	Average standard error confidence (eV/atom)	Average standard error prediction (eV/atom)
Test	0.0020	0.0036
Predict 4.0	0.0234	0.0235
Predict 4.1	0.0336	0.0337

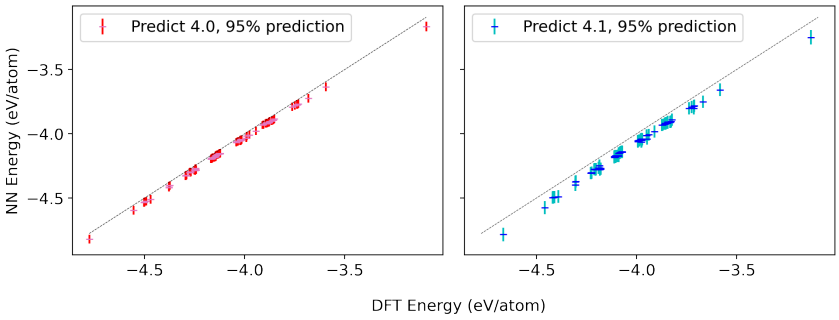

Figure 7: Prediction on new lattice datasets, uncertainty may be much larger in an extrapolation region.

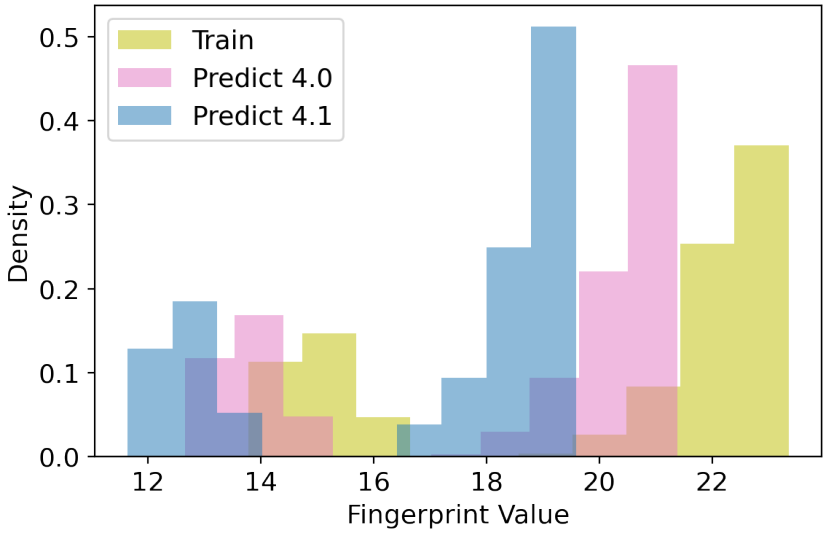

Figure 8: The predict-4.0 and 4.1 datasets have fingerprints outside of range of training distribution (fingerprint example shown is eta=0 with Pd center atoms).

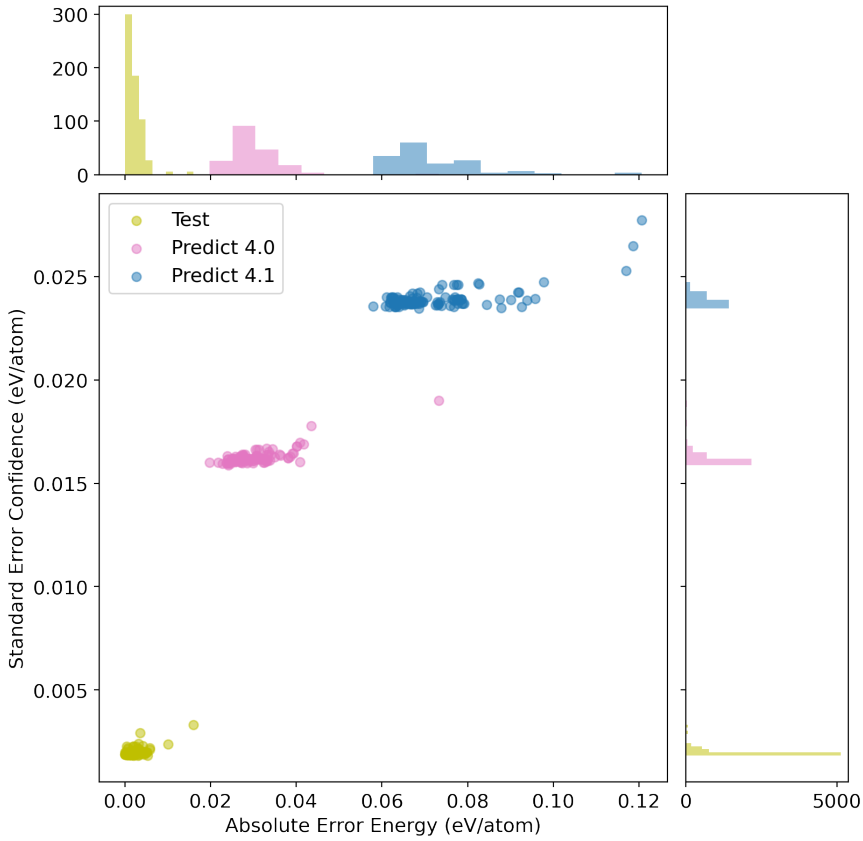

Figure 9: Standard error from delta method vs. absolute error and their distributions.

Uncertainties After Retraining

Next we retrain the potential with some of the predict-4.0 and 4.1 data and check how uncertainties are affected. We expect the uncertainties to decrease after retraining. We added 64% of each predict-4.0 and 4.1 dataset, or 365 datapoints each, and retrained. Fig. 10 shows the energy parity plots of the new training and predict sets. After retraining, the predict set is on parity and no longer offset. The true values are within the prediction intervals for 98.7% of the training data and 98.5% of the predict data, which are close to the theoretical 95% and show the uncertainties calculated from the delta method are quantitatively reasonable. Fig. 11 shows the updated standard error confidence vs. absolute energy error, and their distributions for test and predict datasets. After retraining, the standard errors across datasets are mostly overlapping, and the average standard errors are the same for the datasets. The average standard error confidence and predict are 0.002 and 0.003 eV/atom, respectively. Since we retrained on the predict-4.0 and 4.1 datasets, we are no longer extrapolating on that data and the uncertainties updated to reflect this: they are no longer an order of magnitude larger than the train-sets' as was the case before retraining. We can use this uncertainty method to iteratively retrain a potential by adding data with high uncertainties. This is sometimes called active learning.

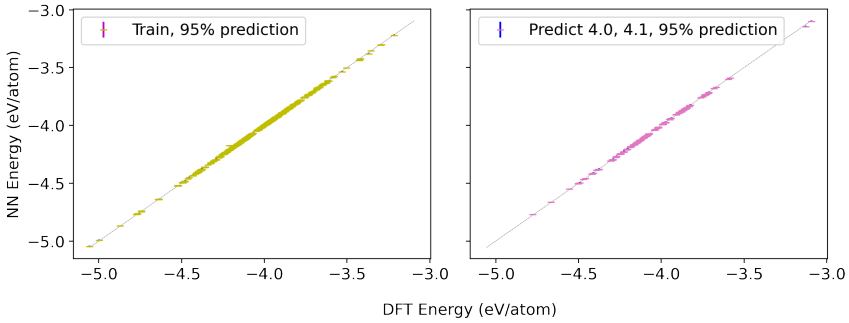

Figure 10: Parity plot after retraining.

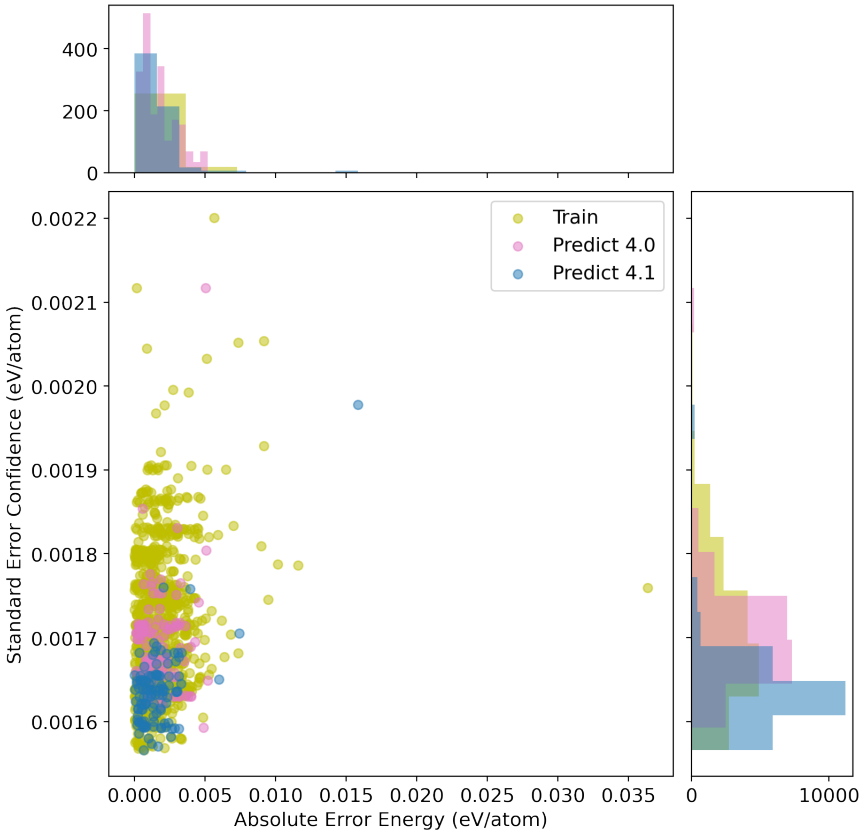

Figure 11: Distribution of uncertainties after retraining.

In the calculation of the Fisher information matrix, we used the errors of energies only, although we trained on energies and forces. From a theoretical perspective, the Fisher information should include some information about force errors, but exactly how much to include is nonobvious. By using only loss of energies, we save computational time for calculating the Fisher information, and the uncertainty measurement still accomplishes the objective and is quantitatively reasonable. Therefore in practice, using only the loss of energies for the Fisher information works well.

We can also extend uncertainty to other properties such as forces. For this case, in Equation 1, $g(\hat{\theta})$ is force, which is $-\frac{\partial E}{\partial \text{position}}$, where E represents energy. We obtain $g'(\hat{\theta})$ through automatic differentiation by taking the derivative of $-\frac{\partial E}{\partial \text{position}}$ with respect to model parameters. In this way, we use the delta method to calculate uncertainties for other quantities of interest. Further work can be done to investigate the quality and methods for force uncertainties of NN potentials.

There is a possibility for fast approximations of the Fisher information after retraining. If we retrain by adding one or a few new training points, we may want a cheaper calculation of the Fisher information matrix. If the parameters of the model did not change from retraining, then the new Fisher information is

the summation of the original Fisher information and the Fisher information for the new training points, because the loss is a sum over training points. Since retraining likely alters the model parameters, the previous Fisher information from old model parameters is an approximation. If only a few training points are added and model parameters do not change much, taking the Fisher information of the new training points and adding it to the original can be a fast approximation of the true Fisher information. Further work is required to determine when this approximation is adequate.

Conclusions

The delta method is a fast and easy way to estimate uncertainty. It requires the Hessian of the loss and gradient with respect to model parameters, and these are obtainable with most machine learning packages using automatic differentiation. The delta method is applicable to most models that are parametric and have nonzero gradients with respect to parameters. The uncertainty estimate will depend on the training data, model, and input (point) for which the uncertainty is calculated. The delta method is an alternative to ensemble or bootstrapping methods for obtaining uncertainty estimates, and uncertainty estimates are important because they can help determine when a model is extrapolating and increase model reliability.

We showed an application of the delta method to a high-dimensional NN potential in molecular simulation. We illustrated how we can iteratively retrain a model by adding data with high uncertainties to improve it. This could also be done on the fly, e.g., while running an MD simulation with a ML potential. The uncertainty can determine the longest timescale MD simulation that is valid for a potential, or to identify when additional data should be added to the training data to improve it. The utility of the delta method shown here extends far beyond molecular simulation, and it can also be applied to many other applications that rely on linear or nonlinear regression models.

Appendix

The delta method is based on regression, and gives a standard error of prediction by linearly approximating the model. We are doing a regression with data $\{x_i, y_i\}$. Our model predicts $y\left(x_i \mid \theta\right)$, and the theory of the delta method assumes that the data output is the sum of the model prediction and some Gaussian error.

$$y_i = y\left(x_i \mid \theta\right) + \epsilon_i$$

with $\epsilon_i \sim N(0, \sigma_i)$, y_i as data output, x_i as data input, and θ as model parameters.

The log likelihood of the data given the model, l_n, is

$$l_n = \log\, P(\{y_i\} \mid \theta)$$

Since we assumed ϵ_i was Gaussian,

$$l_n \propto -\frac{1}{2} \sum_i \left(\frac{y_i - y(x_i \mid \theta)}{\sigma_i} \right)^2$$

The above term includes the sum of squared errors which is common as the loss or regression objective function during training. In least squares regression, we minimize the sum squared errors to get the maximum likelihood estimate of parameters, $\hat{\theta}$.

The standard error of $\hat{\theta}$

$$\text{se}(\hat{\theta}) \approx \frac{1}{\sqrt{I_n(\theta)}}$$

where $I_n(\theta)$ is the Fisher information matrix defined as

$$I_n(\theta) = -\mathbb{E}_\theta \left[\frac{\partial^2\, l_n(\{y_i\} \mid \theta)}{\partial \theta^2} \right]$$

The standard error of $\hat{\theta}$ is obtained from doing a Taylor's series expansion around $l_n'(\theta)$ (Wasserman, 2004). We are able to obtain this standard error by assuming $\hat{\theta}$ is centered and Gaussian around the true parameters θ.

In the Fisher information, note that l_n is the same log likelihood defined earlier, so the Fisher information is proportional to the Hessian of the loss with respect to model parameters, and thus can be readily obtained.

Now we will obtain the standard error of model prediction. For some function $g(\hat{\theta})$,

$$\text{se}(g(\hat{\theta})) \approx \sqrt{(g')^T I_n^{-1} g'}$$

and $g'(\hat{\theta})$ is nonzero. The standard error of $g(\hat{\theta})$ is obtained by doing a Taylor's series around $g(\theta)$ and using $\text{se}(\hat{\theta})$ obtained previously (Wasserman, 2004).

The standard error depends on the training data because the Fisher information depends on the training data. The standard error also depends on the model, its parameters, and the point we are predicting, because these determine g'.

In this work, we assume the error ϵ_i is independent of the data point x_i. This allows the simplification

$$l_n \propto -\frac{1}{2} \sum_i \left(\frac{y_i - y(x_i \mid \theta)}{\sigma_i} \right)^2 = -\frac{1}{2\sigma^2} \sum_i \left(y_i - y(x_i \mid \theta) \right)^2$$

We estimate σ^2 as

$$\sigma^2 \approx \frac{1}{n} \sum_i^n \left(y_i - y(x_i \mid \theta) \right)^2$$

Once obtaining standard errors for a prediction, we can construct confidence intervals. We use $t_{\frac{\alpha}{2}} \cdot \text{se}(g(\hat{\theta}))$ for $(1-\alpha)\%$ confidence intervals. The confidence interval indicates confidence of fit. The prediction standard error has an additional term

$$\text{prediction se}(g(\hat{\theta})) = \sqrt{(g')^T I_n^{-1} g' + \sigma_r^2}$$

where σ_r^2 is residual variance and approximated by

$$\sigma_r^2 \approx \frac{1}{n} \sum_i^n \left(g_i - g(x_i \mid \theta) \right)^2$$

A $(1-\alpha)\%$ prediction interval is then $t_{\frac{\alpha}{2}} \cdot (\text{pred. se}(g(\hat{\theta})))$. The prediction interval represents how often a new point would fall in the interval.

References

(2019).

Second Opinion Needed: Communicating Uncertainty in Medical Machine Learning. (2021).

Em All of Statistics. (2004).

(2019).

(1169).

(2020).

(2016).

Conformalized Quantile Regression. (2019).

Stratified Construction of Neural Network Based Interatomic Models for Multicomponent Materials. (141AD).

Confidence Interval Estimation by Bootstrap Method for Uncertainty Quantification Using Random Sampling Method. (2015).

(2021).

Uncertainty Quantification in Cnn Through the Bootstrap of Convex Neural Networks. (1207).

(2021).

Hyperparameter Ensembles for Robustness and Uncertainty Quantification. (2020).

Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. (2016).

On the Statistical Formalism of Uncertainty Quantification. (2019).

A Comparison of Some Error Estimates for Neural Network Models. (1996).

Confidence Intervals and Prediction Intervals for Feed-Forward Neural Networks. (2001).

Em Gaussian Processes for Machine Learning. (2006).

Dropout As a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. (2015).

Em Bayesian Methods for Adaptive Models. (1992).

Em Priors for Infinite Networks , Pages 29–53. (1996).

Weight Uncertainty in Neural Networks. (2015).

A Scalable Laplace Approximation for Neural Networks. (2018).

Estimating Vibrational and Thermodynamic Properties of Adsorbates with Uncertainty Using Data Driven Surrogates. (2019).

Computational Experience with Confidence Regions and Confidence Intervals for Nonlinear Least Squares. (1987).

(1998).

Confidence Estimation Methods for Neural Networks: a Practical Comparison. (1278).

Active Learning Literature Survey. (2009).

Comprehensive Review of Neural Network-Based Prediction Intervals and New Advances. (1341).

Machine Learning in Catalysis. (2018).

Neural Network Predictions of Oxygen Interactions on a Dynamic Pd Surface. (2017).

Accurate Neural Network Description of Surface Phonons in Reactive Gas-Surface Dynamics: N2 + Ru. (2131).

(536AD).

(2879).

Representing Potential Energy Surfaces by High-Dimensional Neural Network Potentials. (1830).

Constructing High-Dimensional Neural Network Potentials: A Tutorial Review. (1032).

First Principles Neural Network Potentials for Reactive Simulations of Large Molecular and Condensed Systems. (1282).

Addressing Uncertainty in Atomistic Machine Learning. (1097).

(2417).

(2019).

Molecular Dynamics with On-The-Fly Machine Learning of Quantum-Mechanical Forces. (964AD).

Adaptive Machine Learning Framework to Accelerate Ab Initio Molecular Dynamics. (1074).

High-Dimensional Neural Network Potentials for Metal Surfaces: A Prototype Study for Copper. (454AD).

Machine Learning a General Purpose Interatomic Potential for Silicon. (2018).

(2020).

(2021).

Em Error Mitigation in Computational Design of Sustainable Energy Materials. (2016).

Uncertainty Quantification in Molecular Simulations with Dropout Neural Network Potentials. (2020).

(2019).

(2020).

Fast and Accurate Uncertainty Estimation in Chemical Machine Learning. (95AD).

Uncertainty Quantification of Artificial Neural Network Based Machine Learning Potentials. (2018).

Machine Learning Force Fields: Construction, Validation, and Outlook. (2016).

Eigenvalues of the Hessian in Deep Learning: Singularity and Beyond. (2016).

Gradient Descent Happens in a Tiny Subspace. (2018).

What to Do When Your Hessian Is Not Invertible. (2004).

A Modified Cholesky Algorithm Based on a Symmetric Indefinite Factorization. (1998).

Atom-Centered Symmetry Functions for Constructing High-Dimensional Neural Network Potentials. (741AD).

(2417).

Singlenn: Modified Behler-Parrinello Neural Network with Shared Weights for Atomistic Simulations with Transferability. (1781).

Modeling Segregation on Aupd(111) Surfaces with Density Functional Theory and Monte Carlo Simulations. (3479).