Deep Evidential Regression on PAINN Graph Neural Network

Paolo Federico
s212975@student.dtu.dk
Technical University of Denmark
Kgs. Lyngby, Denmark

Alexandra Polymenopoulou
s212558@student.dtu.dk
Technical University of Denmark
Kgs. Lyngby, Denmark

Melina Siskou
213158@student.dtu.dk
Technical University of Denmark
Kgs. Lyngby, Denmark

ABSTRACT

Evidential deep learning is a machine learning approach that combines deep learning techniques with probabilistic reasoning and decision theory. The goal of evidential deep learning is to provide a more robust and reliable approach to machine learning by accounting for uncertainty in the data and the model. When applied to graph neural networks (GNNs), evidential deep learning allows the GNN to not only make predictions, but also to assign a measure of uncertainty to those predictions. In this project, we applied Deep Evidential Regression to a polarizable atom interaction neural network (PAINN), a type of graph neural network, to predict tensorial properties and molecular spectra. The results of the model exhibited low uncertainty for in-distribution data and high uncertainty for out-of-distribution data, while maintaining accurate predictions, demonstrating its ability to account for uncertainty in the predictions and providing further evidence for the effectiveness of evidential deep learning in handling uncertainty in machine learning tasks.

KEYWORDS

Deep Learning, Evidential Learning, Evidential Regression, PAINN, GNN

1 INTRODUCTION

Evidential deep learning is a relatively new approach to machine learning that combines traditional deep learning techniques with probabilistic reasoning and decision theory. The goal of evidential deep learning is to provide a more robust and reliable approach to machine learning by accounting for uncertainty in the data, called 'aleatoric', as well as for uncertainty in the model itself, called 'epistemic'.

The evidential neural network is trained to output not just a prediction, but also a measure of the uncertainty associated with that prediction, which can include both epistemic and aleatoric components. This allows the model to better account for the inherent uncertainty in the data and the model, leading to more reliable and trustworthy predictions.

In this project, we present a novel application of Deep Evidential Regression on PAINN, a polarizable atom interaction neural network, which is a type of graph neural network (GNN), used for predicting tensorial properties and molecular spectra. The aim is to demonstrate that the model can assign high uncertainty to out-of-distribution samples and low uncertainty to in-distribution ones, while performing accurate predictions.

The source code of our implementation can be found in the **Github Repository**.

2 BACKGROUND

Our project was based on two key works in the field: the "Deep Evidential Regression"[2] authored by Alexander Amini, Wilko Schwarting, Ava Soleimany and Daniela Rus and the "Equivariant message passing for the prediction of tensorial properties and molecular spectra"[4] by Kristof T. Schütt, Oliver T. Unke and Michael Gastegger. Both of these papers have had a significant impact on our understanding of graph neural networks and evidential regression, and we have sought to build upon their implementations[3][1] to merge their ideas in our own research. In the following sections, we will describe the main ideas of these papers.

2.1 Evidential Learning

"Deep Evidential Regression" is a scientific paper that presents a novel method for training non-Bayesian NNs to estimate a continuous target as well as its associated evidence in order to learn both aleatoric and epistemic uncertainty.

Epistemic uncertainty refers to the uncertainty that arises from a lack of knowledge about the true underlying model that generated the data. This type of uncertainty can be reduced by gathering more data or by improving the model itself.

Aleatoric uncertainty, on the other hand, refers to uncertainty that is inherent in the data itself, due to random noise or other inherent sources of variability. This type of uncertainty cannot be reduced by gathering more data or improving the model, but must be accounted for in the model's predictions.

The NN that the authors propose is inspired by the theory of "evidential reasoning" in artificial intelligence. The main idea behind evidential reasoning is to represent uncertainty about the state of the world using probability distributions over possible outcomes. In the context of regression, this means representing the uncertainty about the predicted output value using a probability distribution.

In evidential deep learning, every training example adds support to a learned higher-order, evidential distribution (Figure 1). Sampling from this distribution yields instances of lower-order likelihood functions from which the data was drawn. Instead of placing probabilistic priors over network weights and using sampling to approximate output variance, which is how Bayeseian NN work, in this approach higher-order prior distributions are placed over the learned parameters governing the distribution from which our observations are drawn. By training a neural network to output the hyperparameters of the higher-order evidential distribution, a grounded representation of both epistemic and aleatoric uncertainty can then be learned without the need for sampling.

Instead of assuming that μ and σ are known things that our network could predict:

$$(y_1, \ldots, y_N) \sim N(\mu, \sigma^2)$$

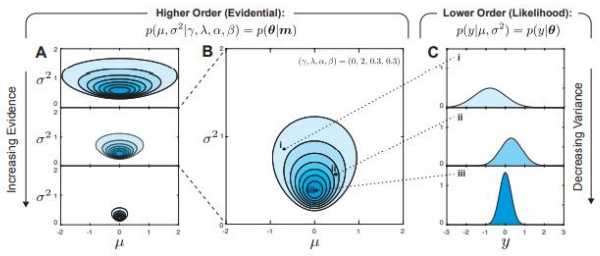

Figure 1: Normal Inverse-Gamma distribution

We consider the problem where the observed targets y are drawn from a Gaussian distribution as in maximum likelihood estimation, but now with unknown mean and variance. By placing evidential priors over the original likelihood function, the NN is trained to infer the hyperparameters of the evidential distribution, which is a Normal Inverse-Gamma (NIG) distribution:

$$\mu \sim N(\gamma, \sigma^2 \nu^{-1}), \sigma^2 \sim \Gamma^{-1}(\alpha, \beta),$$

where $\Gamma(\cdot)$ is the gamma function, $m = (\gamma, \nu, \alpha, \beta)$, and $\gamma \in \mathbb{R}, \nu > 0, \alpha > 1, \beta > 0$. Evidential learning tries to enable direct estimation of epistemic and aleatoric uncertainty by trying to learn these higher order distributions over the individual likelihood parameters.

prediction: $E[\mu] = \gamma$,
aleatoric: $E[\sigma^2] = \frac{\beta}{\alpha - 1}$,
epistemic: $\text{Var}[\mu] = \frac{\beta}{\nu(\alpha-1)}$

The approach for learning a model to output the parameters is maximizing the model fit by maximizing model evidence in support of the observations, while also minimizing evidence on errors by inflating uncertainty when the prediction is wrong. The total loss, $Li(w)$, consists of the two loss terms for maximizing and regularizing evidence, scaled by a regularization coefficient, λ:

$$\mathcal{L}(w) = \mathcal{NLL}(w) + \lambda \mathcal{R}(w)$$

Setting $\lambda = 0$ yields an over-confident estimate while setting λ too high results in over-inflation.

The authors demonstrate the effectiveness of the proposed approach on a number of benchmark datasets and show that the resulting models achieve state-of-the-art performance.

2.2 Equivariant message passing and PAINN

"Equivariant message passing for the prediction of tensorial properties and molecular spectra" is a paper that presents a machine learning approach called equivariant message passing (EMP) for the prediction of tensorial properties and molecular spectra. EMP is a variant of the popular message passing neural network (MPNN) architecture, which is a type of neural network that can process and reason about graph-structured data. It is based on the idea of using symmetries in the data to improve the prediction accuracy and computational efficiency of the model.

The authors of the paper propose the "Polarizable Atom Interaction Neural Network (PAINN)", in which the graph structure represents the molecular system, with atoms as nodes and chemical bonds as edges. To predict the polarizability of a molecular system, PAINN processes the graph structure of the molecule using message passing, which involves passing information between the atoms in the molecule. The information passed between atoms includes their local chemical environment, which is represented by their atomic features and the features of their nearest neighbors.

The PAINN model consists of several layers, which can be divided into two main categories: the message passing layers and the readout layers. The message passing layers are responsible for iteratively updating the states of the atoms in the molecule based on the states of their neighbors and the edge features connecting them. These layers involve a combination of linear and nonlinear transformations, such as weighted summation and activation functions. The readout layers are responsible for aggregating the information from the atoms and edges of the molecule to make a prediction, by simply summing the atom states.

The loss function used in the implementation of PAINN is the mean squared error (MSE) loss function, which is a common choice for regression tasks. The MSE loss function measures the average squared difference between the predicted and true polarizability tensors. The model is trained by minimizing the MSE loss function, which means that it is trying to reduce the difference between the predicted and true polarizability tensors as much as possible.

3 METHODS

The methods section of this report describes the approach that was taken to merge the PAINN model with the evidential model in order to make predictions about the Gibbs free energy of small organic molecules. In order to do this, we used the QM9 dataset, which provides quantum chemical properties for a relevant, consistent, and comprehensive chemical space of small organic molecules. While the standard PAINN model predicts the target value, when adding the evidential layer the prediction involves the four parameters of the inverse gamma distribution. These parameters allow us to both predict the target and the uncertainty levels (compare Figures 2 and 3). We accessed the data in the appropriate format and passed it to the model, which was trained using a subset of the data consisting of molecules with an enthalpy greater than -98 J and bigger than 16 atoms. The trained model was then used to make predictions on the whole dataset in order to analyze the different behavior of the evidential model with in-distribution data and out-of-distribution data. The specific details of the approach and the implementation of the model are described in more detail in the following sections.

Figure 2: PAINN model initial implementation

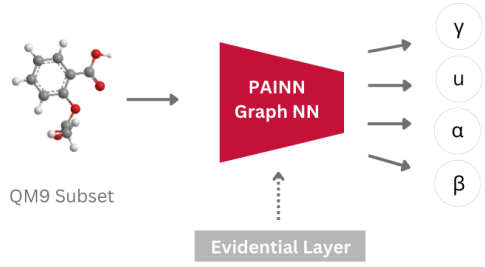

Figure 3: Integrating the evidential layer into PAINN

3.1 Model architecture

The proposed model takes as input a dictionary containing various information about the molecule, including the atomic numbers of each node (referred to as "nodes"), the positions of the nodes that are connected by edges ("edges") and the total number of nodes and edges in the molecule. The target variable for the model to predict in our experiments is the Gibbs free energy (G). The node values are embedded using an embedding size specified by the 'hidden_state' variable, which was set to 64 in our experiments. In graph neural networks (GNNs), message passing can be seen as a way to propagate information through the graph and make use of the graph's connectivity to make more informed predictions. We apply message passing through nodes using the following code.

```
class SchnetMessageFunction(nn.Module):
    """Message function"""

    def __init__(self, node_size, edge_size):
        """
        Args:
            node_size (int): Size of node state
            edge_size (int): Size of edge state
        """
        super().__init__()
        self.msg_function_edge = nn.Sequential(
            nn.Linear(edge_size, node_size),
            ShiftedSoftplus(),
            nn.Linear(node_size, node_size),
        )
        self.msg_function_node = nn.Sequential(
            nn.Linear(node_size, node_size),
            ShiftedSoftplus(),
            nn.Linear(node_size, node_size),
        )

    def forward(self, node_state, edge_state):
        """
        Args:
            node_state (tensor): State of each sender node (num_edges, node_size)
            edge_state (tensor): Edge states (num_edges, edge_size)

        Returns:
            (num_edges, node_size) tensor
        """
        gates = self.msg_function_edge(edge_state)
        nodes = self.msg_function_node(node_state)
        return nodes * gates

class Interaction(nn.Module):
    """Interaction network"""

    def __init__(self, node_size, edge_size):
        """
        Args:
            node_size (int): Size of node state
            edge_size (int): Size of edge state
        """
        super().__init__()

        self.message_function = SchnetMessageFunction(node_size, edge_size)

        self.state_transition_function = nn.Sequential(
            nn.Linear(node_size, node_size),
            ShiftedSoftplus(),
            nn.Linear(node_size, node_size),
        )

    def forward(self, node_state, edges, edge_state):
        """
        Args:
            node_state (tensor): Node states (num_nodes, node_size)
            edges (tensor): Directed edges with node indices (num_edges, 2)
            edge_state (tensor): Edge states (num_edges, edge_size)

        Returns:
            (num_nodes, node_size) tensor
        """
        # Compute all messages
        nodes = node_state[edges[:, 0]]  # Only include sender in messages
        messages = self.message_function(nodes, edge_state)

        # Sum messages
        message_sum = torch.zeros_like(node_state)
        message_sum.index_add_(0, edges[:, 1], messages)

        # State transition
        new_state = node_state + self.state_transition_function(message_sum)

        return new_state
```

The code defines two PyTorch modules: the SchnetMessageFunction and the Interaction class. The SchnetMessageFunction class represents a message function, which is a function that is used to compute messages that are passed between nodes in a graph neural network. The message function takes as input the state of each sender node (node_state) and the state of each edge (edge_state), and produces an output tensor of the same shape as node_state. The message function is implemented as a neural network with two sequential blocks: one for the edge state and one for the node state. Both blocks consist of a linear layer followed by the ShiftedSoftplus activation function and another linear layer.

The Interaction class represents an interaction network, which is a type of graph neural network that operates by iteratively passing messages between nodes in the graph. The interaction network takes as input the node states (node_state), the directed edges connecting the nodes (edges), and the edge states (edge_state), and produces an output tensor of the same shape as node_state. The interaction network uses the message function defined in the SchnetMessageFunction class to compute the messages that are passed between nodes, and then sums these messages to obtain an updated node state using a state transition function. The state transition function is a neural network with a linear layer followed by the ShiftedSoftplus activation function and another linear layer. The message passing process is repeated three times. After this, each node passes through a dense layer with 64 neurons, both as input and output, and the shifted softplus activation function ($softplus(x) - log(2.0)$) is applied. The output of each node in the molecule is then combined to obtain the overall "graph output", which is normalized. The graph output is then passed through the evidential layer, which consists of a dense layer with 64 input neurons and 4 output neurons. These four output neurons represent the four parameters of the inverse gamma function: γ, $\log(\upsilon)$, $\log(\alpha)$, and $\log(\beta)$. The softplus function is applied to all of these parameters except for γ, and α is increased by 1 to ensure that it

is greater than 1. The gamma parameter represents our prediction of the Gibbs free energy, while the other parameters are used to estimate uncertainty. Code of the evidential layer:

```
**class DenseNormalGamma(Module):
    def __init__(self, n_input, n_out_tasks=1):
        super(DenseNormalGamma, self).__init__()
        self.units = n_input
        self.n_out = 4 * n_out_tasks
        self.n_tasks = n_out_tasks
        self.dense = nn.Linear(self.units, self.n_out)

    def forward(self, x):
        x = self.dense(x)
        if len(x.shape) == 1:
            gamma, lognu, logalpha, logbeta = torch.split(x, self.n_tasks, dim=0)
        else:
            gamma, lognu, logalpha, logbeta = torch.split(x, self.n_tasks, dim=1)

        nu = F.softplus(lognu)
        alpha = F.softplus(logalpha) + 1.
        beta = F.softplus(logbeta)

        return torch.cat([gamma, nu, alpha, beta], axis=-1).to(x.device)**
```

3.2 Training

Training a complex model such as this requires a powerful computer to handle the computational demands. To take advantage of the processing capabilities of a graphics processing unit (GPU), we used DTU high performance computing (HPC) and cuda. By utilizing these resources, we were able to significantly improve the speed and efficiency of the training process. In addition to using HPC and cuda, we also employed the standard PAINN architecture and split the dataset into training and validation sets in a 9:1 ratio to ensure that the model was being trained on a diverse and representative set of data. We processed the training and validation sets in batches of 100 and 32 randomized samples, respectively, to improve the model's efficiency and generalizability. We used the Adam optimizer to optimize the model's parameters and fine-tune its performance.

During training, we observed that the weights of our neural network were becoming NaN. This can occur due to a variety of reasons, such as vanishing/exploding gradients or an overly large/small weight decay parameter. To address this issue, we experimented with several values for these hyperparameters and monitored the training process to ensure that the activations and gradients were stable. We ran several combinations of learning rate and λ values (used in loss calculaion) in HPC and concluded that the best and most stable results were achieved by setting the learning rate to 1e-3 and λ to 1e-6. To further improve the model's accuracy and robustness, we also implemented a regularization approach by setting the weight decay to 1e-6. All these allowed us to successfully train the model and mitigate the occurrence of nan values.

3.3 Data

We used the QM9 dataset, which consists of 133,885 small organic molecules with various quantum chemical properties. The data is provided in the Atomic Simulation Environment (ASE) database format and can be accessed using SQLite commands. The dataset includes features for each molecule such as the unique identifier, mass, Gibbs free energy, charges, dipole, and atomic numbers of the atoms. To process the data and input it into the model, we used functions that extracted the graph-level information from the molecules and their Gibbs free energy. This data was then passed to the model for training. Our goal was to evaluate the behavior of the evidential model on in-distribution and out-of-distribution data. To do this, we trained the model using a subset of the molecules with enthalpies greater than -98 J and a size of more than 16 atoms. Finally, we used the trained model to make predictions on the entire dataset.

4 DISCUSSION

The results of the model evaluation indicate that the NN has a good fit to the data (Figure 4). The model achieved training loss 0.101586 and validation loss 0.387 at 800000 steps, suggesting that it is able to accurately predict Gibbs free energy.

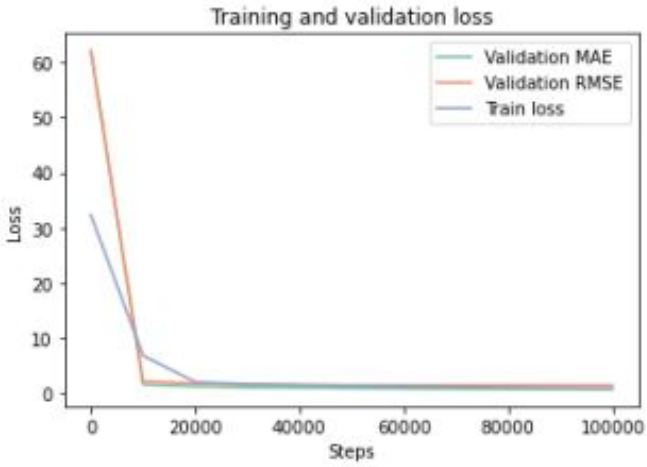

Figure 4: Training and Validation loss

What we expected to see in the prediction, where the whole QM9 dataset was used as input, is the model predicting low uncertainty on the data we consider to be in -distribution, which are the data we used for training, and high uncertainty on the out-of-distribution data. Indeed, in the plot (Figure 5) where the epistemic uncertainty of the predictions is displayed, in the upper right part, which contains the in -distribution data, the uncertainty is low, but as far as we move, it is getting higher. Specifically, it is very clear how higher it is getting for molecules with small number of atoms. For those with small enthalpy value the high uncertainty cannot be seen. A potential reason of this behavior is the small values of enthalpy (smaller than -98), as well as the fact that the range of the enthalpy in these data is not significant: the enthalpy of these OOD data is close to the enthalpy of in-distribution data.

The loss of the predictions was found to be relatively low. Specifically, it is 0.103 for in-distribution data, and 1.3 for OOD data. The fact that the loss of out-of-distribution predictions is almost 13 times bigger than the corresponding one for in-distribution predictions, can prove what we aimed to show on this project: the model has the ability to infer the uncertainty or in other words, understand when it cannot be trusted.

In future research, we could investigate the effects of using different thresholds for filtering the training dataset on the model's

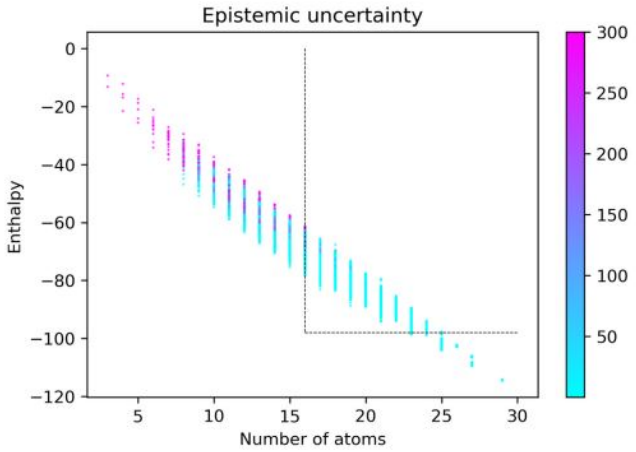

Figure 5: Epistemic uncertainty

uncertainty estimates for in-distribution and out-of-distribution data. Additionally, we could examine the model's ability to infer uncertainty by testing different input features for the Gibbs free energy prediction, or by predicting different features of the dataset. Another potential avenue for exploration would be to modify the message passing method by applying it through edges rather than just nodes, in order to understand if this could affect our model differently.

REFERENCES

[1] DTU. 2020. Implementation of a graph neural network in Pytorch. *Computer software* (2020). https://gitlab.gbar.dtu.dk/sure/graphnn

[2] Amini Alexander et al. 2020. Deep evidential regression. *Advances in Neural Information Processing Systems* 33 (2020), 14927–14937.

[3] Amini Alexander et al. 2020. Deep evidential regression. *Computer software* (2020). https://github.com/aamini/evidential-deep-learning

[4] Oliver Unke Schütt, Kristof and Michael Gastegger. 2021. Equivariant message passing for the prediction of tensorial properties and molecular spectra. *International Conference on Machine Learning* (2021).