[Re] Utilising Uncertainty for Efficient Learning of Likely-Admissible Heuristics

Lebohang Mosia
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
{2320396}@students.wits.ac.za

Scope of Reproducibility

The main claims of the original paper, "Utilising Uncertainty for Efficient Learning of Likely-Admissible Heuristics," focus on improving heuristic learning in planning algorithms by using epistemic and aleatoric uncertainty. The authors assert:

- Claim 1: Epistemic uncertainty can be utilized to explore task space and generate appropriately leveled training tasks.
- Claim 2: Combining epistemic and aleatoric uncertainty produces heuristics that are likely-admissible during planning, reducing compounding errors and high suboptimality.

Methodology

To reproduce the results, the study involved implementing the described methods using the Keras API to build and train a Weight Uncertainty Neural Network (WUNN). The domain used was the 15-puzzle, where various functions for state representation, neural network training, and heuristic calculation were implemented. The hardware used was an Acer Aspire A315-56 with an Intel Core i5-1035G1 CPU and 8GB RAM. The computational resources required included several hours of runtime for solving tasks and minutes for training the neural network.

Results

The reproduction attempt did not yield the same results as the original paper. The IDA* algorithm failed to converge to solutions for the set tasks within the specified 5000 iterations; hence, no valid results could be obtained in an effort to support the original claims.

What was easy

Implementing and training the WUNN using the information provided in the original paper was straightforward. The detailed descriptions of constants and parameters facilitated the transfer of information to code.

What was difficult

Several challenges arose during the reproduction: Equations and unfamiliar terms used in some parts of the original paper made them hard to understand. Associated with the paper is a repository on GitHub, which contains C# code. I tried to port this to Python, which took rather long and was full of errors; I chose to drop these files. Moreover, the implementation of multiple neural networks using the supplementary material posed a problem in that multiple errors cropped up. In addition, the IDA* algorithm was difficult to implement. The algorithm was supposed to solve a set of hard-coded six tasks, but it wasn't able to give a solution; these made the reproduction of the results obtained in the original study challenging.

Preprint. Under review.

1 Introduction

Admissible heuristics can produce excellent plans when A* and its variants are used however, it is very challenging to create strong admissible heuristics. This is because it requires a lot of domain knowledge or computational resources. Using machine learning methods to learn heuristics from data was explored, and the resulting heuristics require much less domain knowledge and memory. This approach has cons, as the heuristic is no longer guaranteed to be admissible because the optimality is given up.

Likely admissible heuristics were used to introduce the concept of admissibility using statistical machine learning. This method relied on having access to training data for optimal plans, which is expensive and hard to obtain.

A different approach that does not need access to training data for optimal plans is proposed. This approach develops a heuristic based on the cost of solving all training tasks. This method produces a strong heuristic. The limitations of this method are that it uses a lot of time and that the margin of error for the final heuristic is high because the data used to learn the heuristic is from non-optimal plans.

2 Scope of reproducibility

In this paper, the two main problems encountered by the authors are that the approach they used is not efficient and that the final heuristic has high suboptimality. The authors claim to be able to solve both of these problems by using epistemic and aleatoric uncertainty. The claims are described below.

- Claim 1: To explore task space to generate training tasks of the right level, epistemic uncertainty can be used. (See results in Table 1)
- Claim 2: Combining epistemic and aleatortic uncertainty produces a heuristic that will be likely-admissible during planning, and this reduces the compounding errors that cause high suboptimality. (See results in Table 2)

3 Background

3.1 Planning

A planning domain is defined by a tuple $\langle S, A, T, C, s\_0, s\_g \rangle$, where S is the set of states, A is the set of actions, T is the transition model, C is the cost function, s_0 is the initial state, and s_g is the goal state. In the paper, each domain has a different start state and that is because the assumption made by the authors is that each domain has a different goal state. Multiple planning algorithms additionally require a heuristic function that returns an estimate of the optimal cost-to-goal from a certain state within the state space. The authors note that planning algorithms such as the IDA* and A* ensure that the plan is optimal and cost-effective. This occurs if the heuristic function is an admissible heuristic so that the function is less than or equal to the optimal, and unknown, cost-to-goal from a certain state in the state space. Admissibility is a desired characteristic, but it can be challenging to ensure heuristics. A lesser alternative is to require the heuristic to be likely admissible [1].

3.2 Learning Heuristics from Data

Heuristics can be learned by using a training dataset that consists of plans from a domain using a supervised machine learning algorithm. The training dataset can be constructed by computing the cost-to-goal for the relevant states in each plan. Various regression-based supervised learning models can be used to train a heuristic on the training dataset. For each generated training task, a search is run to construct a plan to compute the total cost. We can then build a dataset of states-costs and train an ML model on this. This type of heuristic generally does not produce optimal plans because it is not admissible. Training on optimal or near-optimal plans leads to a high-quality heuristic with much less suboptimality, providing a close approximation of the optimal cost-to-goal for a certain state in the state space[1].

3.3 Bayesian Neural Networks

Neural networks can be viewed as probability distributions, each with a mean and variance. In standard regression with neural networks, the variance is assumed to be a known constant while the single output neuron learns the mean. [Added] By altering the neural network to have two output neurons, we can learn both the mean and variance, where the variance measures aleatoric uncertainty by accounting for noise in data. Weight uncertainty neural networks (WUNNs)

are used to compute the posterior distribution to model epistemic uncertainty. This approach leverages both types of uncertainty to generate training tasks and develop strong heuristics[1].

The method proposed in the paper involves generating training tasks by exploring the task space using epistemic uncertainty. For each training task generated, a search is conducted to construct a plan and compute the total cost, creating a dataset of states and their associated costs. A Bayesian neural network is then trained on this dataset, where the network accounts for aleatoric uncertainty by learning the variance and epistemic uncertainty through WUNNs. This approach aims to produce likely-admissible heuristics, reducing the compounding errors that lead to high suboptimality during planning.

4 Methodology

4.1 Model descriptions

A Weight Uncertainty Neural Network model using the Keras API is implemented and trained. This is a Sequential Neural Network, namely a feed-forward neural network. It consists of two layers: the first layer is a Dense layer composed of 64 units and with the ReLU, or Rectified Linear Unit, activation function, accepting the input shape of the number of features in input data, therefore the number of columns of memory_buffer minus one.

The model is compiled with an Adam optimizer and Mean Squared Error loss. It is trained for several iterations specified, which is here set to 5000. In every iteration, a random mini-batch is sampled with a size of 100 from memory_buffer. On each iteration, a mini-batch is used for one epoch of training, allowing the model weights to be updated iteratively. The memory_buffer represents a placeholder dataset assumed to be a NumPy array with 100 samples and 17 features, including the target variable. It is trained from scratch and is not pre-trained.

4.2 Environments

Only one domain was used, which is the 15-puzzle domain. The 15-puzzle is a sliding puzzle that consists of a 4x4 grid with 15 numbered tiles and one empty space. The goal is to arrange the tiles in numerical order by making sliding moves that use the empty space.

The state of the puzzle is represented as a list of length-16 integers, representing each tile, and the empty space is represented by 0. For example, the state [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 15] depicts the order of tiles with the empty space in the penultimate position. The goal state is represented as [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]; all the tiles are in order, and the empty space comes last.

The available actions involve moving the empty space within the grid, namely up, down, left, or right, while ensuring it remains within bounds. Each action incurs a uniform cost of 1, simplifying the cost function.

As for the implementation, a number of functions were written from scratch: one for generating neighbouring states, another for converting states into feature vectors for the input of the neural network, and one for the calculation of the Manhattan Distance Heuristic in the case of IDA* search. In addition, the implementation of the IDA* search algorithm to solve the puzzle efficiently and the integration of neural network training to enhance the estimation of heuristics have also been added.

4.3 Hyperparameters

The hyperparameter values were set according to the way they were set in the original paper. A hyperparameter search was not done.

4.4 Experimental setup and code

The experiments outlined in the provided code are designed to evaluate the performance of the NN-WUNN heuristic when applied to solving the 15-puzzle problem with the IDA* search algorithm. Several different components make up the setup.

First, the code defines functions for the representation and manipulation of the puzzle: 'get_neighbors' returns a list of all possible neighbours from a given state, function 'feature_representation' transforms a state into a

feature vector and function 'Manhattan_distance' calculates the Manhattan distance between a state and the goal state.

Next, in the neural network training phase, we use the function 'train_nn_wunn'. In this function, a neural network architecture is built using TensorFlow/Keras. It trains that on a memory buffer with state-action pairs. A neural network is further trained on a specified number of iterations using mini-batch gradient descent.

After training, the heuristic function is defined to calculate the heuristic value of any given state using the trained NN-WUNN model. This function uses the neural network to estimate heuristic values, which are later used within the IDA* search algorithm to direct the search process optimally.

The IDA* search algorithm itself is implemented in the 'ida_star' function. This function performs iterative, depth-limited searches until either a solution is found or the search space is exhausted. It uses the given heuristic function to pick the next states to explore, favouring those with lower heuristic values.

The code evaluates the performance of the NN-WUNN heuristic in the 'solve_and_update' function. This function applies the IDA* search algorithm over a set of predefined tasks of the 15-Puzzle; in addition, it updates a memory buffer with additional state-action pairs encountered during the search process.

Finally, 'collect_performance_data' gathers performance metrics for a set of test tasks. The metrics include the number of nodes generated in the course of the search, planning time in seconds, solution optimality, and the Manhattan distance from the starting state to the goal.

4.5 Computational requirements

To do this investigation, an Acer Aspire A315-56, with Windows 11 was used. The processor is Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz , 1.19 GHz. The RAM is 8.00 GB. The processor is a 64-bit processor.

For the Jupyter Notebook, the runtimes of the codes were less than a minute, and the training took about 2 minutes. The code for solving the tasks ran for about several hours before it stopped.

4.6 Results reproducing original paper

The experiments were performed, but unfortunately, we could not produce results as the IDA* algorithm ran for a long time without converging to solutions for any of the set tasks. The algorithm terminated prematurely after reaching the maximum specified number of iterations: 5000 iterations without producing any solutions that actually solve the given set of problems.

In the Jupyter Notebook, there is a code at the end that was meant to place all of the results in a table form. The form is shown below,

Task	Nodes Generated	Planning Time (s)	Optimal	Manhattan Distance

5 Discussion

One of the primary strengths of the experiment may lie in the utilization of a neural network-based heuristic, NN-WUNN, which has the potential to provide more accurate estimations of the distance to the goal state compared to traditional heuristic functions. In general, neural network-based heuristics are also flexible because they can be applied to various 15-puzzle tasks without the need for manual tuning or customization for specific instances.

One prominent shortcoming of the approach is its high computational cost, which had a significant influence on practical functionality and scalability. Training neural network-based heuristics and running IDA* search algorithms can be computationally expensive, particularly when the issue instance is large or complex. The complexity lead to significant runtimes and computing needs.

5.1 What was easy

Coding and training the Weight Uncertainty Neural Network did not give me a huge challenge and the code runs. The paper is structured in a way that it provides the reader with a lot of constants, and parameters in a paragraph, making it easier to transfer the information over to the code.

5.2 What was difficult

Some parts of the paper were quite challenging to understand because of the use of equations and unfamiliar terms. The paper is associated with a GitHub repository with a lot of code files. These were hard to read because they were coded in C#. I tried to convert those codes into Python codes, however, it was a very long process and I received a lot of errors. I ended up not using them.

I tried to use Algorithms 3 and 4 to produce my code as they provided more detail than Algorithms 1 and 2, but implementing more than one neural network as they did in the supplementary material gave me problems and a lot of errors.

Implementing the IDA* algorithm to solve tasks gave me a problem as well because I hardcoded a set of six tasks for the algorithm to solve but it could not solve any of them.

References

[1] Ofir Marom and Benjamin Rosman. Utilising uncertainty for efficient learning of likely-admissible heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 560–568, 2020.