Neural Network Heuristic Functions: Taking Confidence into Account

Daniel Heller,*[1] Patrick Ferber,*[1, 2] Julian Bitterwolf,*[3] Matthias Hein,[3] Jörg Hoffmann[1,4]

[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2] University of Basel, Basel, Switzerland
[3] University of Tübingen, Tübingen, Germany
[4] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
patrick.ferber@unibas.ch, julian.bitterwolf@uni-tuebingen.de, matthias.hein@uni-tuebingen.de,
hoffmann@cs.uni-saarland.de

Abstract

Neural networks (NN) are increasingly investigated in AI Planning, and are used successfully to learn heuristic functions. NNs commonly not only predict a value, but also output a confidence in this prediction. From the perspective of heuristic search with NN heuristics, it is a natural idea to take this into account, e.g. falling back to a standard heuristic where confidence is low. We contribute an empirical study of this idea. We design search methods which prune nodes, or switch between search queues, based on the confidence of NNs. We furthermore explore the possibility of out-of-distribution (OOD) training, which tries to reduce the overconfidence of NNs on inputs different to the training distribution. In experiments on IPC benchmarks, we find that our search methods improve coverage over standard methods, and that OOD training has the desired effect in terms of prediction accuracy and confidence, though its impact on search seems marginal.

Introduction

Neural networks (NN) can learn powerful search guidance. Successes include the AlphaGo series (Silver et al. 2016, 2017, 2018), as well as heuristic search for single-agent games such as Rubik's Cube (Agostinelli et al. 2019). Given the prominence of heuristic search in AI Planning (Hoffmann and Nebel 2001; Helmert and Domshlak 2009; Richter and Westphal 2010; Helmert et al. 2014; Domshlak, Hoffmann, and Katz 2015), training NNs as heuristic functions is highly promising, and is actively pursued (Toyer et al. 2018; Garg, Bajpai, and Mausam 2019; Ferber, Helmert, and Hoffmann 2020; Shen, Trevizan, and Thiébaux 2020; Rivlin, Hazan, and Karpas 2020; Yu, Kuroiwa, and Fukunaga 2020; Karia and Srivastava 2021; Ferber et al. 2022). We contribute a new angle to this line of research, honing in on NN prediction confidence.

Commonly, a NN predicts not only a value, but also has a confidence in the prediction. From the perspective of heuristic search with NN heuristics (h^{NN}), it is a natural idea to take this into account. For example, if the NN's confidence in the estimate is low, then it might be beneficial to fall back to a standard heuristic like h^{FF} (Hoffmann and Nebel 2001).

We contribute an empirical study of this idea. We design two kinds of search methods which take decisions based on NN confidence: Pruning states with low confidence, and switching between search queues based on confidence. In the latter, we use dual-queue search (Röger and Helmert 2010), with h^{NN} in one search queue and h^{FF} in the other, and define a variety of switching schemes giving preference to h^{NN} on high-confidence states only.

We furthermore explore the possibility of out-of-distribution (OOD) training to avoiding the frequent problem of overconfidence on inputs differing from the training distribution (Nguyen, Yosinski, and Clune 2015; Amodei et al. 2016). Such overconfidence suggests knowledge the NN has not actually learned. In heuristic search with a NN heuristic h^{NN}, this may happen for states s dissimilar from the states that h^{NN} encountered during training.

A successful approach for *OOD training* is to train the NN additionally on dissimilar data where we teach it to be unconfident (Lee et al. 2018; Hein, Andriushchenko, and Bitterwolf 2019; Hendrycks, Mazeika, and Dietterich 2019). The OOD data does not have to be related to the actual OOD inputs encountered at execution time, so that the OOD training data can be designed in a generic manner. We transfer this idea to NN heuristic functions in planning. Starting from the supervised-learning approach by Ferber, Helmert, and Hoffmann (2020), we explore OOD training data based on "white noise", randomly generated states, as well as "weighted noise", which gives higher probability to facts that are frequently true and may thus be closer to OOD states encountered during search.

We implemented all these methods in Neural Fast Downward (NFD, Ferber, Helmert, and Hoffmann 2020), which is an extension of Fast Downward (Helmert 2006) to incorporate NN. Running experiments on the NFD collection of IPC domains, we find that (1) our search methods can improve coverage over standard methods, in both the single-search-queue and the dual-search-queue setting; and that (2) our OOD training methods have the desired effect, preserving prediction-error performance while decreasing confidence on OOD inputs. Unfortunately, (3) the impact of OOD training on the actual search performance is marginal.

Overall, we contribute insights into the role of prediction confidence in search with neural heuristic functions, forming a basis for future research on this subject.

*These authors contributed equally.

Planning Background

We use the *FDR* planning framework (Bäckström and Nebel 1995). A planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, s_{\mathcal{I}}, \mathcal{G} \rangle$. $\mathcal{V}$ is a set of *variables*, $\mathcal{A}$ is a set of *actions*, $s_{\mathcal{I}}$ is the *initial state*, and $\mathcal{G}$ is the *goal*. Every variable has a domain $\mathcal{D}$. A *fact* is a variable-value pair $\langle v, d \rangle$ where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$. A state is a complete variable assignment, i.e. one fact per variable. The state space $\mathcal{S}$ is the set of all state of Π and the initial state is one of those states. Each *action* $a \in \mathcal{A}$ defines a *precondition* pre_a and an *effect* eff_a, both are partial variable assignments. An action a is applicable in a state s if $pre_a \subseteq s$. Applying a in s leads to the successor state $s' = \{\langle v, d \rangle \mid \langle v, d \rangle \in s, \neg \exists d' : \langle v, d' \rangle \in \mathit{eff}_a\} \cup \mathit{eff}_a$. The function $succ(s)$ generates all successor states of s, i.e. the set of states produced by applying the applicable actions of s to s. For simplicity, we consider unit action costs (all actions cost 1). A *plan* π is a sequence of actions $\langle a_1, \ldots, a_n \rangle$, such that starting from $s_{\mathcal{I}}$ and sequentially applying the action in π results in a state s^* with $s^* \subseteq \mathcal{G}$.

Satisficing planning tries to find for task any plan, not necessarily a shortest plan. Greedy Best-First Search (GBFS) is a wide-spread method for this. It maintains a single search queue of unexplored, reachable states. At every step, GBFS explores a state s with minimum heuristic value of this queue and adds the successor states $succ(s)$ to the queue. A heuristic is a value function $h : \mathcal{S} \to \mathbb{R}_0^+ \cup \{\infty\}$. A highly successful variant is dual-queue search (Röger and Helmert 2010), which maintains two queues differing in the used heuristic functions and/or pruning methods. Successor states generated by one queue are evaluated and inserted - unless pruned - in all queues. The search alternates between the queues. Often, this is a simple round-robin alternation. An important metric is *coverage*. Given a set of tasks T and a search method m, the coverage denotes for how many tasks of T, m found a plan.

To train neural network (NN) heuristic functions h^{NN}, we extend the method by Ferber, Helmert, and Hoffmann (2020), which trains h^{NN} generalizing over states in a fixed planning task. This amortizes if we encounter many different initial states for which we can use the offline-trained heuristic, as it is the case in many puzzles. Their method executes random walks - from some initial state - to generate seed states. From each seed state, it executes GBFS with h^{FF}. For every plan found by this search, all states along the plan are stored with their remaining cost on the plan. States are represented as Boolean vector s where each entry is associated with a fact. A feed-forward NN f is trained on those state-cost pairs. The output $f(s) \in [0, 1]^{H+1}$ is a probability distribution over the possible costs estimates i, which runs over all integers from 0 to the maximum observed cost H in the training data. The NN is trained with the cross-entropy loss, which is $\mathcal{L}(h^{\text{NN}}, s, c) = -\log(h^{\text{NN}}(s)_c)$ for a training state s with stored cost c. The predicted cost of the NN which we use as heuristic value is $h^{\text{NN}}(s) = \arg\max_i f(s)_i$.

Search Methods

By standard definitions, the *confidence* of a NN is $c^{\text{NN}}(s) = \max_i f(s)_i$. For a NN trained with cross-entropy loss this is the maximum a posteriori probability for any of the classes (Richard and Lippmann 1991). Our idea is to take this confidence into account during search. Intuitively, the lower the confident of the NN is, the fewer we can rely on the predictions. Exploring an unconfident region means exploring a region without guidance and should be avoided. We introduce three methods along these lines. The first two methods use confidence for pruning, discarding low-confidence states. If used in a single queue search, the search is incomplete. Our third method uses the confidence to regulate the alternation in a dual-queue configuration.

Mean Threshold

The simplest way to incorporate NN confidence is to define a threshold t below which states are pruned. That is, for any newly generated state s, we check whether $c^{\text{NN}}(s) < t$, and if so we discard s. We thus focus the search on confident regions. Obviously, it can happen that all solutions pass through regions of low confidence, which would halt single-queue search.

Choosing the threshold t is a problem in itself. Good values of t are highly domain-dependent – even task-dependent – as the distribution of NN confidence may differ widely with the task. To counter-act this problem, we choose t as a function of NN confidence on the training data. We fix a value X (e.g. $X = 40$) and choose t such that $X\%$ of the training data are pruned.

This method can be used in any search configuration, including in particular single-queue search but also dual-queue search, by applying it to the h^{NN} search queue. In a dual-queue search where the other search queue is complete, we resolve the halting issue and recovers completeness.

Adaptive Threshold

The mean threshold method adapts the value of t to the task, but we observed that within a task, confidence often correlates negatively with the predicted heuristic value, since the exact heuristic value is harder to estimate far away from the goal. Thus, the mean-threshold technique tends to prune states with large heuristic values, an unwanted bias that may be highly detrimental.

To counteract this phenomenon, we introduce an *adaptive threshold* method. As before, we consider the NN confidences on all training states. But this time, we group samples by their observed cost. For each group G_i with $i \in \{0, \ldots, H\}$, we calculate an individual threshold t_i using the same method as before, i.e. we fix t_i such that $X\%$ of the states within G_i are pruned. As some groups are small, we combine groups with adjacent values i until each group contains at least 100 states.

As before: A state s on which the NN is unconfident, $c^{\text{NN}}(s) < t_i$ where $h^{\text{NN}}(s) = i$, is pruned during search.

Prioritizing Queue

Our third approach uses the same thresholds t or t_i as above, in a dual-queue setting where one queue uses h^{NN} and the other h^{FF}. But instead of pruning states in the h^{NN} queue, we employ confidence to control the alternation between the

two queues. We do not schedule the two queues in round-robin fashion, but we stick to the h^{NN} queue so long as it is confident. Specifically, whenever the h^{NN} queue expands a state s where $c^{\text{NN}}(s) \geq t$ (respectively $c^{\text{NN}}(s) \geq t_i$ where $h^{\text{NN}}(s) = i$), in the next search iteration we again expand a state from the h^{NN} queue. If the NN was unconfident, $c^{\text{NN}}(s) < t$, we switch to the h^{FF} queue, expand one state and switch back.

Training with OOD Inputs

Previous work (Nguyen, Yosinski, and Clune 2015; Hendrycks and Gimpel 2017; Hein, Andriushchenko, and Bitterwolf 2019) showed that a NN classifier which is trained only on its training distribution often is overconfident when evaluated on inputs that are very different from what has been encountered during training.

Consequently, confidence can be a bad indicator of prediction quality. The problem of detecting OOD inputs was approached with Bayesian neural networks (Kristiadi, Hein, and Hennig 2020; Wang and Aitchison 2022; Henning, D'Angelo, and Grewe 2021), distributional confidence predictors (Malinin and Gales 2018), density based methods (Nalisnick et al. 2019; Ren et al. 2019), and confidence calibrated classifier training (Lee et al. 2018). The latter, which uses classification NN, are among the most successful and established methods for OOD detection.

The aforementioned OOD methods have mostly been studied in the image domain and in some biological contexts. We transfer one of the most successful approaches to planning. To this end, we follow Lee et al. (2018), Hein, Andriushchenko, and Bitterwolf (2019) and Hendrycks, Mazeika, and Dietterich (2019). We simultaneously train the NN on the training states – the *in-distribution* – as well as on OOD inputs where the NN should have a low-confidence. We train the low confidence for OOD inputs $\bar{s}$ by minimizing the cross-entropy loss between $f(\bar{s})$ and the uniform distribution $(\frac{1}{H+1}, \ldots, \frac{1}{H+1})$ over all $H+1$ possible outputs. The resulting additional loss is $\mathcal{L}^{\text{OOD}}(h^{\text{NN}}, \bar{s}) = -\frac{1}{H+1}\sum_{i=0}^{H} \log(h^{\text{NN}}(\bar{s})_i)$ The training process mixes in-distribution and OOD inputs according to a fixed proportion, which we denote by $Y\%$ OOD inputs.

To generate the required OOD training inputs, we investigate two sampling methods:

- **Uniform Noise:** Hein, Andriushchenko, and Bitterwolf (2019) showed that data-agnostic synthetic noise can reduce confidence on OOD inputs. We adapt their approach to our discrete facts and set every entry of the input vector $\bar{s}$ to 0 or 1 with a 50% chance.
- **Weighted Noise:** Some facts are more frequently true in the in-distribution training data D than others. For every fact f, we calculate the empirical probability p_f that f is in a state of D. To generate weighted noise we sample each entry e (associated with a fact f) of $\bar{s}$ independently from the Bernoulli distribution with $P(\bar{s}_e = 1) = p_f$.

It may happen that our noises produce states from the in-distribution. As long as this happens rarely - which is the case - the NN still becomes less confidence on states outside the in-distribution and stays confident on states from the in-distribution.

Experiments

Next, we describe our setup, then we discuss our results. In the results, we start with the effect of OOD training on h^{NN} prediction error and confidence, as this starts the overall pipeline in our machinery. We then compare our confidence-aware search methods against their confidence-unaware counterparts, keeping a heuristic function fixed. We finally compare the impact of OOD training on search by fixing instead the search and varying the heuristic functions.

Setup

We build on Neural Fast Downward (NFD), changing Ferber, Helmert, and Hoffmann (2020)'s machinery only where needed. Hence we train feed-forward networks with 3 hidden layers using sigmoid activation function on the hidden layers, softmax on the final layer, the cross-entropy losses described above, and the *Adam* optimizer. Ferber, Helmert, and Hoffmann (2020) evaluate a single integer, a one-hot, and a unary encoding as output of the NN. We use only the one-hot encoding as output - thus a multi-class classification network - which enables us to build upon current OOD detection research.

The training is implemented in Keras (Chollet 2015) with TensorFlow (Abadi et al. 2015) as backend. We execute each training run on 4 cores of an Intel Xeon E5-2660 cpu with a memory limit of 15GB. For each configuration we use 10 fold cross-validation, i.e. we split the training data in 10 folds and train 10 NN. Each NN uses a different fold to monitor its training progress and trains on the remaining 9 folds. Once progress on that validation fold converges, the training stops. Hyper-parameters are not selected, but fixed for all runs.

We run experiments on the NFD collection of IPC domains, which encompasses Blocksworld, Depots, Grid,Pipesworld-NoTankage, Rovers, Scanalyzer, Storage, and Transport. We use the same categorization of tasks into *easy*, *moderate*, and *hard* difficulty as Ferber, Helmert, and Hoffmann (2020). In our experiments, we skip tasks categorized as easy; 49 tasks are categorized as moderate; and 63 tasks are categorized as difficult.

Every NN is trained for the state space of a specific planning task. For each task we generated 60 test instances, differing in the initial state. We evaluate our NNs by using them in search on these instances. To reduce the computational burden, we do not evaluate every test instance with each of the 10 NNs trained for its state space, but instead we run each NN on one-tenth of the test instances. Each search is executed on a single CPU core, with a 15GB memory limit, and with a time limit of 30 minutes for moderate tasks, and 120 minutes for hard tasks. Our code is online available (Heller et al. 2022).

OOD Training Effect on NN Confidence

Table 1 shows the effect of OOD training on confidence and prediction error. To measure the latter, we use the Kendall

Training Method	Confidence In	Out-U	Out-W	Ranking Coefficient
Standard	29.3	14.5	10.9	84.1
U50	30.0	0.5	12.0	84.0
U90	30.8	0.4	12.2	84.1
W50	29.0	10.9	1.1	84.6
W90	29.2	5.5	0.6	84.5

Table 1: Effect of OOD training on median confidence (in %) over all tasks and ranking coefficient (see text). In: in-distribution, Out-U: uniform out-distribution, Out-W: weighted out-distribution. Standard: NN trained without OOD, U: uniform noise, W: weighted noise, 50/90: proportion Y of noise used during training.

		depot	grid	pipes	block	rover	scan	stora	trans
Moderate Tasks	Single Q.	Base	0.85	0.89	0.80	**1.00**	0.51	**0.89**	**1.00**	**0.97**
		M5	-.09	-.42	-.20	-.23	-.05	-.11	-.04	-.33
		M40	-.79	-.89	-.75	-.91	-.50	-.91	-.86	-.96
		A5	**+.05**	**+.02**	**+.04**	**+.00**	**+.17**	-.35	**+.00**	-.04
		A40	-.29	-.78	-.45	-.62	-.09	-.59	-.41	-.50
	Dual Queue	Base	0.96	**1.00**	0.94	**1.00**	0.89	0.98	**1.00**	0.98
		M40	+.02	-.07	+.02	**+.00**	-.01	**+.02**	**+.00**	**+.02**
		M80	-.07	-.09	-.19	**+.00**	-.13	-.03	-.08	-.24
		A40	**+.03**	-.04	**+.05**	**+.00**	**+.01**	**+.02**	**+.00**	**+.02**
		A80	**+.03**	-.08	+.04	**+.00**	+.00	**+.02**	**+.00**	+.01
		P20	+.02	-.01	+.02	**+.00**	+.00	+.01	**+.00**	+.01
		P80	+.02	**+.00**	+.02	**+.00**	**+.01**	+.01	**+.00**	**+.02**
Hard Tasks	Dual Queue	Base	0.90	0.92	0.88	0.52	0.37	0.98		**0.01**
		A40	**+.05**	**+.03**	**+.01**	-.01	**+.07**	**+.01**		-.01
		A80	-.01	-.19	-.03	**+.01**	+.01	+.00		-.01
		P20	-.01	+.00	+.00	**+.01**	-.04	+.00		**+.00**
		P80	-.04	-.04	-.01	**+.01**	-.10	+.00		-.01

Table 2: Coverage (in %) and coverage change of search configurations. Base: confidence-unaware search, M: mean threshold, A: adaptive threshold, P: prioritizing queue. 5/20/40/80 fraction $X\%$ underlying the thresholds. Best configurations per block (top, middle, bottom) are bold.

rank correlation coefficient (Kendall and Gibbons 1990) between the state orders induced by the training data vs. the predicted heuristic values.

The data for in-distribution confidence and rank correlation are almost identical for the different heuristic functions. This is intended and expected – OOD training aims at preserving prediction confidence and quality, changing *only* the confidence on OOD data. The latter effect also clearly shows in Table 1: on their respective OOD distribution, the heuristics trained with OOD are far less confident than the other heuristics. For weighted-noise OOD training, this effect generalizes (to a lesser degree) also to uniform-noise OOD data. Overall, our OOD training methods have the desired effect.

Search Method Performance

We now evaluate the confidence-aware search methods (cf. Table 2). We fix the h^{NN} heuristic function here, to use weighted noise in a 1:1 ratio to the in-distribution data. The

		depot	grid	pipes	block	rover	scan	stora	trans
Moderate	Standard	0.98	0.94	0.98	**1.00**	**0.90**	**1.00**	**1.00**	**1.00**
	W50	+.01	+.02	**+.01**	**+.00**	**+.00**	**+.00**	**+.00**	**+.00**
	W90	**+.02**	**+.03**	**+.01**	**+.00**	**+.00**	-.01	**+.00**	**+.00**
	U50	+.00	+.01	+.00	**+.00**	**+.00**	**+.00**	**+.00**	**+.00**
	U90	-.01	**+.03**	-.01	**+.00**	**+.00**	**+.00**	**+.00**	**+.00**
Hard	Standard	0.93	**0.96**	**0.90**	0.51	**0.44**	**1.00**		**0.02**
	W50	**+.02**	-.01	-.01	+.00	**+.00**	-.01		-.02
	W90	-.03	-.02	**+.00**	+.00	-.01	-.01		-.02
	U50	**+.02**	**+.00**	-.02	**+.01**	-.13	**+.00**		-.02
	U90	+.00	-.02	-.04	+.00	-.10	**+.00**		-.02

Table 3: Coverage (in %) and coverage change of search with differently trained NNs. Standard: no OOD training, U: uniform noise, W: weighted noise, 50/90: proportion Y of noise used during training. Best configurations per block (top, bottom) are bold.

thresholds 5%, 20% 40%, and 80% are chosen based on preliminary experiments identifying the interesting ranges.

Consider first the single-queue search configurations (Table 2 Top) The mean-threshold variant consistently decreases coverage. This is expected given our previously discussed observations: states far away from the goal tend to have low confidence, and these are aggressively pruned here. Choosing the threshold adaptively instead performs much better, indeed this method dominates mean-threshold almost consistently given the same pruning fraction X. Ignoring states where h^{NN} is very unconfident (A5) improves coverage in 5 domains and harms in only 2. As before however, pruning too aggressively (e.g. 40%) is detrimental.

Consider now the dual-queue search configurations (Table 2 Middle/Bottom). We run these configurations also on the hard tasks (the largest instances) as they are more effective and coverage is near perfect on the moderate tasks already for the baseline. We include mean-threshold only on the moderate tasks as it is, again, dominated by adaptive-threshold. Beyond these two methods, we include the third method, using NN confidence information not for pruning but to prioritize search queues.

As the data shows, mean threshold is less detrimental here than in single-queue search – in fact can be beneficial – which is expected given the dual-queue "safety net" against too aggressive pruning. However, adaptive-threshold still works better consistently. Our best setting M40 for that method improves coverage in 4 domains on the moderate tasks and in 5 domains on the hard tasks, while deteriorating coverage only on 1 domain and 2 domains respectively.

For the prioritization variants P20 and P80, keep in mind that higher values of X here mean a weaker prioritization and hence more similarity to the baseline, as the h^{NN} queue continues while its confidence is *above* the threshold. The empirical benefits from this method are higher for more aggressive prioritization with $X = 20$. With few exceptions, the method is dominated by adaptive-threshold.

OOD Training Effect on Search Performance

Let us finally turn to the impact of OOD NN training methods on search performance (cf. Table 3). Here we fix the search configuration to the best-performing one, dual queue A40.

Somewhat surprisingly, as the data clearly shows, OOD training in whichever form has little effect on search performance. The strongest beneficial case is that of W90 in Depots and Grid on moderate tasks, but the improvements there are counter-balanced by similar losses for the same domains and configuration on hard tasks. Given our observations in Section showing that the OOD training does have the desired effect of decreasing prediction confidence on OOD data while preserving similar prediction accuracy, this raises the question whether the particular OOD data we used in OOD training are at fault. Possibly, these OOD data are too far away from the state distribution encountered during search, so that h^{NN} confidence does not allow us to distinguish search states s that are similar to the training data from ones that aren't. Whether this is the case and how it can be improved remains a question for future research.

Conclusion

We have contributed a study of NN confidence in the context of NN heuristic functions in planning, including confidence-aware search methods and an exploration of out-of-distribution training. The empirical advantages so far are moderate, but they show that these issues can be of interest when using learned heuristics for search. We see our contribution as an initial study yielding first methods and insights, forming a basis for future research on this subject.

Acknowledgments

This work was funded by DFG (German Research Foundation) Grant 389792660 as part of TRR 248 (CPEC, https://perspicuous-computing.science). The authors acknowledge support from the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A), from the DFG under Germany's Excellence Strategy (EXC number 2064/1, Project number 390727645), and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (TAILOR, no. 952215 and BDE, no. 817639).

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Fern; a Viégas; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/. Accessed: 2022-05-31.

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1: 356–363.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. arXiv:1606.06565 [cs.AI].

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11(4): 625–655.

Chollet, F. 2015. Keras. https://keras.io. Accessed: 2022-05-31.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A New Systematic Approach to Partial Delete Relaxation. *AIJ*, 221: 73–114.

Ferber, P.; Geißer, F.; Trevizan, F.; Helmert, M.; and Hoffmann, J. 2022. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In *Proc. ICAPS 2022*.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proc. ECAI 2020*, 2346–2353.

Garg, S.; Bajpai, A.; and Mausam. 2019. Size Independent Neural Transfer for RDDL Planning. In *Proc. ICAPS 2019*, 631–636.

Hein, M.; Andriushchenko, M.; and Bitterwolf, J. 2019. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proc. CVPR 2019*.

Heller, D.; Ferber, P.; Bitterwolf, J.; Hein, M.; and Hoffmann, J. 2022. Code for the SoCS 2022 paper "Neural Network Heuristic Functions: Taking Confidence into Account". https://doi.org/10.5281/zenodo.6553254.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *JACM*, 61(3): 16:1–63.

Hendrycks, D.; and Gimpel, K. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *Proc. ICLR 2017*.

Hendrycks, D.; Mazeika, M.; and Dietterich, T. 2019. Deep Anomaly Detection with Outlier Exposure. In *Proc. ICLR 2019*.

Henning, C.; D'Angelo, F.; and Grewe, B. F. 2021. Are Bayesian neural networks intrinsically good at out-of-distribution detection? In *ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning (UDL)*.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14: 253–302.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In *Proc. AAAI 2021*, 8064–8073.

Kendall, M.; and Gibbons, J. D. 1990. *Rank Correlation Methods*. A Charles Griffin Title, 5th edition.

Kristiadi, A.; Hein, M.; and Hennig, P. 2020. Being bayesian, even just a bit, fixes overconfidence in relu networks. In *Proc. ICML 2020*, 5436–5446.

Lee, K.; Lee, H.; Lee, K.; and Shin, J. 2018. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *Proc. ICLR 2018*.

Malinin, A.; and Gales, M. 2018. Predictive uncertainty estimation via prior networks. In *Proc. NeurIPS 2018*.

Nalisnick, E.; Matsukawa, A.; Teh, Y. W.; Gorur, D.; and Lakshminarayanan, B. 2019. Do deep generative models know what they don't know? In *Proc. ICLR 2019*.

Nguyen, A.; Yosinski, J.; and Clune, J. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. CVPR 2015*, 427–436.

Ren, J.; Liu, P. J.; Fertig, E.; Snoek, J.; Poplin, R.; Depristo, M.; Dillon, J.; and Lakshminarayanan, B. 2019. Likelihood ratios for out-of-distribution detection. In *Proc. NeurIPS 2019*.

Richard, M. D.; and Lippmann, R. P. 1991. Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural computation*, 3(4): 461–483.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39: 127–177.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. In *ICAPS Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 16–24.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proc. ICAPS 2010*, 246–249.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proc. ICAPS 2020*, 574–584.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587): 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the Game of Go Without Human Knowledge. *Nature*, 550(7676): 354–359.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies with Deep Learning. In *Proc. AAAI 2018*, 6294–6301.

Wang, X.; and Aitchison, L. 2022. Bayesian OOD detection with aleatoric uncertainty and outlier exposure. In *Proc. AABI 2022*.

Yu, L.; Kuroiwa, R.; and Fukunaga, A. 2020. Learning Search-Space Specific Heuristics Using Neural Network. In *ICAPS Workshop on Heuristics and Search for Domain-independent Planning*, 1–8.