On Using Admissible Bounds for Learning Forward Search Heuristics

Carlos Núñez-Molina¹, Masataro Asai², Pablo Mesejo¹ and Juan Fernández-Olivares¹

¹University of Granada, Spain
²MIT-IBM Watson AI Lab, USA

ccarlos@ugr.es, masataro.asai@ibm.com, pmesejo@ugr.es, faro@decsai.ugr.es

Abstract

In recent years, there has been growing interest in utilizing modern machine learning techniques to learn heuristic functions for forward search algorithms. Despite this, there has been little theoretical understanding of what they should learn, how to train them, and why we do so. This lack of understanding has resulted in the adoption of diverse training targets (suboptimal vs optimal costs vs admissible heuristics) and loss functions (e.g., square vs absolute errors) in the literature. In this work, we focus on how to effectively utilize the information provided by admissible heuristics in heuristic learning. We argue that learning from poly-time admissible heuristics by minimizing mean square errors (MSE) is not the correct approach, since its result is merely a noisy, inadmissible copy of an efficiently computable heuristic. Instead, we propose to model the learned heuristic as a truncated gaussian, where admissible heuristics are used not as training targets but as lower bounds of this distribution. This results in a different loss function from the MSE commonly employed in the literature, which implicitly models the learned heuristic as a gaussian distribution. We conduct experiments where both MSE and our novel loss function are applied to learning a heuristic from optimal plan costs. Results show that our proposed method converges faster during training and yields better heuristics.

1 Introduction

Motivated by the success of Machine Learning (ML) approaches in various decision making tasks [Mnih et al., 2015; Silver et al., 2016], an increasing number of papers are tackling the problem of learning a heuristic function for forward state space search in recent years. Despite this interest, there has been little theoretical understanding of what these systems should learn, how to train them and why we do so. As a result, heuristic learning literature has adopted many different training targets (corresponding to either admissible heuristics [Shen et al., 2020], suboptimal solution costs [Arfaee et al., 2011; Ferber et al., 2022; Marom and Rosman, 2020] or optimal solution costs [Ernandes and Gori, 2004; Shen et al., 2020]) and training losses (e.g., square errors [Shen et al., 2020], absolute errors [Ernandes and Gori, 2004] and piecewise absolute errors [Takahashi et al., 2019]).

In this work, we try to answer these questions from a statistical lens, focusing on how to effectively utilize admissible heuristics in the context of heuristic learning. We argue that learning from poly-time admissible heuristics, such as $h^{LMcut}$ [Helmert and Domshlak, 2009], by minimizing mean square errors (MSE) does not provide any practical benefits, since its result is merely a noisy, inadmissible copy of a heuristic that is already efficient to compute. Then, if admissible heuristics should not be used as training targets, how can we leverage them? In order to answer this question, we first analyze the statistical implications behind the commonly used loss function, the MSE, which implicitly models the learned heuristic as a Gaussian distribution. Nonetheless, we contend that a better modeling choice for heuristics is given by the Truncated Gaussian distribution (Fig. 1), due to the existence of bounds on the values a heuristic can take (e.g., heuristics never take on negative values).

The main contribution of this paper is a theoretically-motivated, statistical method for learning an inadmissible heuristic while exploiting an admissible heuristic.
pose to model the learned heuristic as a Truncated Gaussian, where an admissible heuristic provides the lower bound of this distribution, thus constraining heuristic predictions. This modeling choice results in a loss function to be minimized that is different from the standard MSE loss. We conduct extensive experimentation where both loss functions are applied to learning heuristics from optimal plan costs in several classical planning domains. Results show that those methods which model the learned heuristic as a Truncated Gaussian, i.e., which are trained with our novel loss function, learn faster and result in better heuristics than those which model it as an ordinary Gaussian, i.e., which are trained with the standard MSE loss. To the best of our knowledge, this is the first work that proposes the use of admissible heuristics to constrain heuristic predictions and improve learning.\footnote{Our full code and data can be found in github.com/pddl-heuristic-learning/pddls1. We provide the Appendix in the Arxiv version of our paper: arxiv.org/abs/2308.11905.}

2 Backgrounds

2.1 Classical Planning and Heuristics

We define a propositional STRIPS Planning problem as a 4-tuple \( (P, A, I, G) \) where \( P \) is a set of propositional variables, \( A \) is a set of actions, \( I \subseteq P \) is the initial state, and \( G \subseteq P \) is a goal condition. Each action \( a \in A \) is a 4-tuple \( (\text{PRE}(a), \text{ADD}(a), \text{DEL}(a), \text{COST}(a)) \) where \( \text{COST}(a) \in \mathbb{Z}^{0+} \) is a cost, \( \text{PRE}(a) \subseteq P \) is a precondition and \( \text{ADD}(a), \text{DEL}(a) \subseteq P \) are the add-effects and delete-effects, respectively. A state \( s \subseteq P \) is a set of true propositions (all of \( P \setminus s \) are false), an action \( a \) is applicable when \( s \supseteq \text{PRE}(a) \) (read: \( s \) satisfies \( \text{PRE}(a) \)), and applying action \( a \) to \( s \) yields a new successor state \( a(s) = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a) \).

The task of classical planning is to find a sequence of actions called a plan \( (a_1, \cdots, a_n) \) where, for \( 1 \leq t \leq n, s_0 = I, s_t \supseteq \text{PRE}(a_{t+1}), s_{t+1} = a_{t+1}(s_t), \) and \( s_n \supseteq G \). A plan is optimal if there is no plan with lower cost-to-go: \( \sum \text{COST}(a_i) \). A plan is otherwise called satisficing. In this paper, we assume unit-cost: \( \forall a \in A; \text{COST}(a) = 1 \).

A domain-independent heuristic function \( h \) in classical planning is a function of a state \( s \) and the problem \( \langle P, A, I, G \rangle \). It returns an estimate of the shortest (optimal) path cost from \( s \) to one of the goal states (states that satisfy \( G \)), typically through a symbolic, non-statistical means such as delete-relaxation, a technique that ignores the delete-effects of actions in order to efficiently estimate the cost from \( s \) to \( G \). The optimal cost-to-go, or a perfect heuristic, is denoted by \( h^* \). A heuristic is called admissible if it never overestimates it, i.e., \( \forall s; 0 \leq h(s) \leq h^*(s) \), and inadmissible otherwise. Notable admissible heuristics include \( h^{\text{LMcut}}, h^{\text{max}} \) and \( h^* \) [Helmert and Domshlak, 2009; Bonet and Geffner, 2001; Betz and Helmert, 2009], whereas \( h^{\text{FF}}, h^{\text{add}} \) and \( h^{\text{GC}} \) [Hoffmann and Nebel, 2001; Bonet and Geffner, 2001; Fikes et al., 1972] are prominent examples of inadmissible heuristics.

2.2 Task: Supervised Learning for Heuristics

Let \( p^*(x) \) be the unknown ground-truth probability distribution of (an) observable random variable(s) \( x \) and let \( p(x) \) be our current estimate of it. Given a dataset \( \mathcal{X} = \{x^{(1)}, \ldots, x^{(N)}\} \) of \( N \) data points, we denote an empirical data distribution as \( q(x) \), which draws samples from \( \mathcal{X} \) uniformly. While often \( q(x) \) may also be informally called a ground-truth distribution, \( q(x) \) is entirely different from either \( p(x) \) or \( p^*(x) \) because it is a distribution over a finite set of points, i.e., a uniform mixture of dirac’s delta \( \delta \) distributions (Eq. 1). Our goal is to obtain an estimate \( p(x) \) that resembles \( p^*(x) \) as closely as possible. To do so, under the Maximum Likelihood Estimation (MLE) framework, we maximize the expectation of \( p(x) \) over \( q(x) \). In other words, MLE tries to maximize the expected probability \( p(x) \) of observing each data point \( x \sim q(x) \):

\[
q(x) = \sum_{i=1}^{N} q(x^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \delta(x = x^{(i)}) \quad (1)
\]

\[
p^*(x) = \arg\max_p \mathbb{E}_{q(x)}[p(x)]
= \arg\min_p \mathbb{E}_{q(x)}[-\log p(x)] \quad (2)
\]

Typically, we assume \( p^*(x) \) and \( p(x) \) are of the same family of functions parameterized by \( \theta \), such as a Gaussian distribution \( \theta \) such as a neural network weights or the trees in random forests, i.e., \( p^*(x) = p^0(\theta_0) \), \( p(x) = p_0(\theta) \). This makes MLE a problem of finding the \( \theta \) maximizing \( \mathbb{E}_{q(x)}[p_0(\theta)] \). Also, we typically minimize a loss such as the negative log likelihood (NLL) \( -\log p(x) \), since log is monotonic and preserves the optima \( \theta^* \) (Eq. 2). Furthermore, \( \mathbb{E}_{q(x)}[\cdot] \) is often estimated by Monte-Carlo sampling, e.g.,

\[
\mathbb{E}_{q(x)}[-\log p(x)] \approx \frac{1}{N} \sum_{i=1}^{N} -\log p(x_i), \text{ where each } x_i \text{ is sampled from } q(x).
\]

We further assume \( p(x) \) to follow a specific distribution such as a Gaussian distribution \( \mathcal{N}(\mu, \sigma) \):

\[
p(x) = \mathcal{N}(x | \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (3)
\]

We emphasize that the choice of the distribution determines the loss. When the model designer assumes \( p(x) = \mathcal{N}(\mu, \sigma) \), then the NLL is a shifted and scaled squared error:

\[
-\log p(x) = \frac{(x-\mu)^2}{2\sigma^2} + \log \sqrt{2\pi\sigma^2}. \quad (4)
\]
Likewise, a Laplace distribution $L(x|\mu, b) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$ represents the absolute error because its NLL is $\frac{|x-\mu|}{b} + \log 2b$.

The NLL loss is thus more fundamental and theoretically grounded than losses such as the Mean Squared Error, although it is “more complicated” due to the division $\frac{1}{2\sigma^2}$ and the second term. A reader unfamiliar with statistics may rightfully question why such complications are necessary or why $\sigma$ is not commonly used by the existing literature. It is because many applications happen to require only a single prediction for a single input (point estimate): When we model the output distribution as a Gaussian $N(\mu, \sigma)$, we often predict $\mu$, which is simultaneously the mean and the mode of the distribution and does not depend on $\sigma$.

Moreover, the MSE is a special case of the NLL that can be derived from it. To derive the MSE, we first simplify the loss into the squared error $(x-\mu)^2$ by setting $\sigma$ to an arbitrary constant, such as $\sigma = \frac{1}{\sqrt{2}}$, because the variance/spread of the prediction does not matter in a point estimation of $\mu$. As a result, we can also ignore the second term which is now a constant. We then compute the expectation $E_{q(x)} (x - \mu)^2$ with a Monte-Carlo estimate that samples $N$ data points $x_1, \ldots, x_N \sim q(x)$, predict $\mu = \mu_0(x_i)$ for each $x_i$ using a machine learning model $\mu_0$, and compute the average: $\frac{1}{N} \sum_{i=1}^N (x_i - \mu_0(x_i))^2$. In other words, the MSE loss is nothing more than the Monte-Carlo estimate of the NLL loss of a Gaussian with a fixed $\sigma = \frac{1}{\sqrt{2}}$. In contrast, distributional estimates represent the entire $p(x)$: e.g., if $p(x) = N(\mu, \sigma)$, then the model predicts both $\mu$ and $\sigma$.

The MLE framework can be applied to the supervised heuristic learning setting as follows. Let $q(s, x)$ be the empirical data distribution, where $s$ is a random variable representing a state-goal pair (from now on, we will implicitly assume that states $s$ also contain goal information) and $x$ a random variable representing the cost-to-go (regardless of whether it corresponds to a heuristic estimate, optimal or suboptimal cost). Then, the goal is to learn $p^*(x|s)$ where:

$$p^*(x|s) = \arg\max_{\theta} E_{q(s,x)} p_\theta(x|s),$$

$$p_\theta(x|s) = N(x|\mu, \sigma_\theta(s), \sigma = \frac{1}{\sqrt{2}}),$$

and $\mu_\theta(s)$ is the main body of the learned model, such as a neural network parameterized by the weights $\theta$. Supervised heuristic learning with distributional estimates is formalized similarly, where the only difference is that an additional model (e.g. a neural network) with parameters $\theta_2$ predicts $\sigma$:

$$p_\theta(x|s) = N(x|\mu, \mu_\theta(s), \sigma = \sigma_\theta_2(s)).$$

### 3.2.3 The Principle of Maximum Entropy

The discussion above models $p(x)$ as a Gaussian distribution. While the assumption of normality (i.e. following a Gaussian) is ubiquitous, one must be able to justify such an assumption. The principle of maximum entropy [Jaynes, 1957] states that $p(x)$ should be modeled as the maximum entropy (max-ent) distribution among all those that satisfy our constraints or assumptions, where the entropy is defined as $E_{p(x)} (-\log p(x))$. A set of constraints defines its corresponding max-ent distribution which, being the most random among those that satisfy those constraints, minimizes assumptions other than those associated with the given constraints. Conversely, a non max-ent distribution implicitly encodes additional or different assumptions that can result in an accidental, potentially harmful bias. For example, if we believe that our random variable $x$ has a finite mean, a finite variance and a support/domain/range equal to $\mathbb{R}$, it must be modeled as a Gaussian distribution according to this principle because it is the max-ent distribution among all those that satisfy these three constraints.

In other words, a person designing a loss function of a machine learning model must devise a reasonable set of constraints on the target variable $x$ to identify the max-ent distribution $p(x)$ of the constraints, which automatically determines the correct NLL loss for the model. This paper tries to follow this principle as faithfully as possible.

### 3 Utilizing Bounds for Learning

In the previous section, we provided some statistical background on heuristic learning. We now leverage this background to analyze many of the decisions taken in the existing literature, sometimes unknowingly, putting particular focus on how admissible heuristics are used during training. Based on this analysis, we argue that the proper way of utilizing the information provided by admissible heuristics is using them as the lower bound of a Truncated Gaussian distribution representing the learned heuristic.

We previously explained that the heuristic to be learned is modeled as a probability distribution (e.g., a Gaussian), instead of a single value: The ML model is unsure about the true heuristic value $h^*$ associated with a state $s$. When it predicts $\mu$, it believes not only that $\mu$ is the most likely value (the mode) for $h^*$, but also that other values are still possible. The uncertainty of this prediction is given by $\sigma$: The larger this parameter is, the more unsure the model is about its prediction. The commonly used MSE loss is derived from the ad-hoc assumption that $\sigma$ is fixed, i.e., independent from $s$, which means that the model is equally certain (or uncertain) about $h^*$ for every state $s$. This is unrealistic in most scenarios: it is generally more difficult to accurately predict $h^*$ for states that are further from the goal, for which the uncertainty should be larger. Therefore, the model should predict $\sigma$ in addition to $\mu$, i.e., it should output a distributional estimate of $h^*$ instead of a point estimate.

Another crucial decision involves selecting what to learn, i.e., the target function to use for the training. It is easy to see that training a model on a dataset containing a practical (i.e., computable in polynomial time) heuristic, admissible or otherwise, such as $h_{\text{Lin}}$ or $h_{\text{FF}}$, does not provide any practical benefits because, even if the training is successful, all we get is a noisy, lossy, slow copy of a heuristic that is already efficient to compute. Worse, trained models always lose admissibility if the target is admissible. To outperform existing poly-time heuristics, i.e., achieve a super-symbolic benefit from learning, it is imperative to train the model on data of a better quality, such as $h^+$ as proposed in [Shen et al., 2020] or optimal solution costs $h^*$. Although obtaining these datasets may prove computationally expensive in practice, e.g., $h^+$ is NP-hard, we can aspire to learn a heuristic
that outperforms the poly-time heuristic by training on these targets.

If poly-time admissible heuristics are not ideal training targets, are they completely useless for learning a heuristic? Intuitively this should not be the case, given the huge success of heuristic search where they provide a strong search guidance toward the goal. Our main question is then how we should exploit the information they provide. To answer this question, we must revise the assumption we made previously by using squared errors: That $x = h^*$ follows a Gaussian distribution $N(\mu, \sigma)$. The issue with this assumption is that $N(\mu, \sigma)$ assigns a non-zero probability $p(x)$ to every $x \in \mathbb{R}$, but we actually know that $h^*$ cannot take some values: Given some admissible heuristic like $h^{\text{LMcut}}$, we know that $h^{\text{LMcut}} < h^*$ holds for every state; therefore $p(x) = 0$ when $x < h^{\text{LMcut}}$. Analogously, if for some state $s$ we know the cost $h^{*s}$ of a satisficing (non-optimal) plan from $s$ to the goal, then $h^{*s}$ acts as an upper bound of $h^*$.

According to the principle of maximum entropy, which serves as our why, if we have a lower $l$ and upper $u$ bound for $h^*$, then we should model $h^*$ using the max-ent distribution with finite mean, finite variance, and a support equal to $(l, u)$, which is the Truncated Gaussian $T \mathcal{N}(x|\mu, \sigma, l, u)$ as proven in [Dowson and Wragg, 1973], formalized as Eq. 8:

$$T \mathcal{N}(x|\mu, \sigma, l, u) = \begin{cases} \frac{1}{\sqrt{2\pi} \sigma} \phi\left(\frac{x - l}{\sigma}\right) & l \leq x \leq u \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where $\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$, $\Phi(x) = \frac{1}{2}(1 + \text{ERF}(x))$,

$l$ is the lower bound, $u$ is the upper bound, $\mu$ is the pre-truncation mean, $\sigma$ is the pre-truncation standard deviation, and ERF is the error function. $T \mathcal{N}$ has the following NLL loss:

$$-\log T \mathcal{N}(x|\mu, \sigma, l, u) = \frac{(x - \mu)^2}{2\sigma^2} + \log \sqrt{2\pi \sigma^2} + \log \Phi\left(\frac{u - \mu}{\sigma}\right) - \Phi\left(\frac{l - \mu}{\sigma}\right) \quad (9)$$

Modeling $h^*$ as a $T \mathcal{N}$ instead of $N$ presents several advantages. Firstly, $T \mathcal{N}$ constrains heuristic predictions to lie in the range $(l, u)$ given by the bounds of the distribution. Secondly, $T \mathcal{N}$ generalizes $N$ as $T \mathcal{N}(x|\mu, \sigma, -\infty, \infty) = \mathcal{N}(x|\mu, \sigma)$ when no bounds are provided. Finally, $T \mathcal{N}$ opens the possibility for a variety of training scenarios for heuristic learning, with a sensible interpretation of each type of data, including the satisficing solution costs.

In this work, we focus on the scenario where an admissible heuristic $h$ is provided along with the optimal solution cost $h^*$ for each state, leaving other settings for future work. In this case, $h$ acts as the lower bound $l$ of $h^*$, which is modeled as a $T \mathcal{N}(x = h^*|\mu, h, \infty)$, where $\mu$ and $\sigma$ are predicted by an ML model. Note that we cannot use $h^*$ as $T \mathcal{N}(h^*|\mu, \sigma, h^*, h^*)$ since, during evaluation/test time, we do not have access to the optimal cost $h^*$. Also, this modeling decision is feasible even when no admissible heuristic is available (e.g., when the PDDL description of the environment is not known, as in Atari games [Bellemare et al., 2013]) since we can always resort to the blind heuristic $h_{\text{blind}}(s)$ or simply do $l = 0$, which still results in a tighter bound than the one provided by an untruncated Gaussian $T \mathcal{N}(x|\mu, \sigma, -\infty, \infty)$.

Finally, our setting is orthogonal and compatible with residual learning [Yoon et al., 2008], where the ML model does not directly predict $\mu$ but rather a residual or offset $\Delta \mu$ over a heuristic $h$, where $\mu = h + \Delta \mu$. Residual learning can be seen as initializing the model output $\mu$ around $h$ which, when $h$ is a good unbiased estimator of $h^*$, facilitates learning. This technique can be used regardless of whether $h^*$ is modeled as a $T \mathcal{N}$ or $N$ because it merely corresponds to a particular implementation of $\mu = \mu_0(s)$, which is used by both distributions. Residual learning is analogous to the data normalization commonly applied in standard regression tasks, where features are rescaled and shifted to have mean 0 and variance 1. However, residual learning is superior in the heuristic learning setting because target data (e.g., $h^*$) is skewed above 0 and because the heuristic used as the basis for the residual can handle out-of-distribution data due to its symbolic nature.

### 3.1 Planning with a Truncated Gaussian

At planning time, we must obtain a point estimate of the output distribution, which will be used as a heuristic to determine the ordering between search nodes. As a point estimate, we can use any statistic of central tendency, thus we choose the mean. It is important to note that the $\mu$ parameter of $T \mathcal{N}(x|\mu, \sigma, l, u)$ is not the mean of this distribution since $\mu$ corresponds to the mean of $N(\mu, \sigma)$ (i.e., the mean of the distribution before truncation) and does not necessarily lie in the interval $(l, u)$. The mean of a Truncated Gaussian is obtained according to Eq. 10. Note that a naive implementation of this formula results in rounding errors (See the Appendix for a numerically stable implementation).

$$\mathbb{E}[x] = \mu + \sigma \frac{\phi\left(\frac{l - \mu}{\sigma}\right) - \phi\left(\frac{u - \mu}{\sigma}\right)}{\Phi\left(\frac{u - \mu}{\sigma}\right) - \Phi\left(\frac{l - \mu}{\sigma}\right)} \quad (10)$$

Eq. 10 satisfies $l \leq \mathbb{E}[x] \leq u$. This means that, when a lower bound $l$ is provided (e.g., by an admissible heuristic), the heuristic prediction returned by the model will never be smaller than $l$. Analogously, when an upper bound $u$ is also provided (e.g., by a satisficing solution cost), the model will never predict a heuristic value larger than $u$. With this, we hope that the use of a $T \mathcal{N}$ during planning helps the model make predictions that are closer to $h^*$ than the bounds themselves, potentially helping it achieve a super-symbolic improvement over admissible heuristics.

In contrast, the mode $\arg \max_p p(x)$ of $T \mathcal{N}$ is uninteresting: While we could use it as another point estimate, it is the same as the untruncated mean $\mu$ when the predicted $\mu$ is within the bounds, and equal to one of the upper/lower bounds otherwise (see Fig. 1). However, this inspires a naive alternative that is applicable even to $N$, which is to clip the heuristic prediction $\mathbb{E}[x]$ (equal to $\mu$ for $N$) to the interval $[l, u]$. We expect only a marginal gain from this trick because it only improves really bad predictions, i.e., those which would lie outside $[l, u]$ otherwise, and does not affect predictions that correctly lie inside $[l, u]$. In our experiments, we show that this approach is inferior to our first method.
We re-emphasize that despite the use of admissible heuristics during training the learned heuristic is inadmissible, just like any learning-based heuristics proposed so far. In case a distributional estimate is used, i.e., when the ML model also learns to predict σ, we could discuss likely-admissibility [Erennades and Gori, 2004; Marom and Rosman, 2020]. However, this extension is left for future work.

4 Experimental Evaluation

We evaluate the effectiveness of our new loss function under the domain-specific generalization setting, where the learned heuristic function is required to generalize across different problems of a single domain. Due to space limitations, we focus on the high-level descriptions and describe the detailed configurations in the Appendix.

Data Generation. We trained our system on four classical planning domains: blocksworld-4ops, ferry, gripper, and vistatll. Using PDDL domains as benchmarks for evaluating planning performance is a standard practice, as exemplified by the International Planning Competitions (IPCs) [Vallati et al., 2015]. For each domain, we generated three sets of problem instances (train, validation, test) with parameterized generators used in the IPCs. We provided between 456 and 1536 instances for training (the variation is due to the difference in the number of generator parameters in each domain), between 132 and 384 instances for validation and testing (as separate sets), and 100 instances sampled from the test set for planning. The Appendix describes the domains and generator parameters. Notably, the test instances are generated with larger parameters in order to assess the generalization capability to generate the dataset from these instances, we optimally solved each instance with A* [Hart et al., 1968] and h\textsubscript{LMcut} in Fast Downward [Helmert, 2006] under 5min runtime / 8GB memory (train, val) and 30min runtime / 8GB memory (test). Whenever it failed to solve a instance within the limits, we retried generation with a different random seed for a maximum memory (train,val) and 30min runtime / 8GB memory (test).

We evaluated three different ML models configurations.

Model Configurations. We evaluated three different ML methods to show that our statistical model is implementation-agnostic. Neural Logic Machine (NLM) [Dong et al., 2019] is an architecture designed for inductive learning and reasoning over symbolic data which has been successfully applied to classical planning domains for learning heuristic functions [Gehringer et al., 2022] with Reinforcement Learning (RL) [Sutton and Barto, 2018]. STRIPS-HGN [Shen et al., 2020], HGN for short) is another architecture based on the notion of hypergraphs. Lastly, we used linear regression with the hand-crafted features proposed in [Gomoluch et al., 2017], which comprise the goal-count [Fikes et al., 1972] and FF [Hoffmann and Nebel, 2001] heuristics, along with the total and mean number of effects ignored by FF’s relaxed plan.

We analyze our learning & planning system from several orthogonal axes. Gaussian vs. Truncated: Using μ(s) as the parameter of a Gaussian \(N(\mu(s), \sigma(s))\) or Truncated Gaussian \(TN(\mu(s), \sigma(s), l, \infty)\) distribution. Learned vs. fixed sigma: Predicting \(\sigma(s)\) or using a constant value \(\sigma(s) = \frac{1}{\text{c}}\), as it is done for the MSE loss. Lower bounds: Computing the lower bound \(l\) with the \(h_{\text{LMcut}}\) heuristic. When we use a Gaussian distribution, \(l\) is used to clip the heuristic prediction \(\mathbb{E}[x] = \mu(s)\) to the interval \([l, \infty)\). Ablation studies with \(l = h_{\text{max}}(s)\) [Bonet and Geffner, 2001] and \(l = h_{\text{blind}}(s)\) are included in the Appendix. Residual learning: Either using the model to directly predict \(\mu(s)\) or to predict an offset \(\Delta \mu(s)\) over a heuristic \(h(s)\), so that \(\mu(s) = \Delta \mu(s) + h(s)\). We use \(h = h_{\text{FF}}\) as our unbiased estimator of \(h^*\), as proposed in [Yoon et al., 2008]. In the Appendix, we conduct experiments with \(h_{\text{LMcut}}\) as the basis of the residual.

Training. We trained each configuration with 5 different random seeds on a training dataset that consists of 400 problem instances subsampled from the entire training problem set (456-1536 instances, depending on the domain). Due to the nature of the dataset, these 400 problem instances can result in a different number of data points depending on the length of the optimal plan of each instance. We performed \(4 \times 10^4\) weight updates (training steps) using AdamW [Loshchilov and Hutter, 2017] with batch size 256, weight decay \(10^{-2}\) to avoid overfitting, gradient clip 0.1, learning rate of \(10^{-2}\) for the linear regression and NLM, and \(10^{-3}\) for HGN. All models use the NLL loss for training, motivated by the theory, but note that the NLL of \(N(\mu, \sigma = 1/\sqrt{2})\) matches the MSE up to a constant, as previously noted. For each model, we saved the weights that resulted in the best validation MSE metric during the training. On a single NVIDIA Tesla V100, each NLM training took \(\approx 0.5\) hrs except in vistatll (\(\approx 2\) hrs). HGN was much slower (\(\approx 3\) hrs except \(\approx 15\) hrs in blocksworld). Linear models trained much faster (12-20 minutes).

Evaluation Scheme. We first report two different metrics on the test set: “MSE” and “MSE+clip”. Here, MSE is the mean square error between \(h^*(s_i)\) and \(h(s_i) = \mathbb{E}[x]\), i.e., \(\frac{1}{N} \sum_{i=1}^{N} (h(s_i) - h^*(s_i))^2\), for i-th state \(s_i\) of \(N\) states in the test dataset. \(\mathbb{E}[x]\) of \(TN\) is given by Eq. 10 while \(\mathbb{E}[x]\) of \(N\) is simply \(\mu\). “+clip” variants are exclusive to \(N\) and they clip \(\mu\) to \(l\), i.e., use \(\max(\mu, l)\) in place of \(\mu\) to compute the MSE. We also obtained the MSE for \(h = h_{\text{FF}}\) and \(h = h_{\text{LMcut}}\).

We then evaluate the planning performance using the point estimate provided by each model as a heuristic function to guide a search algorithm. Since the learned heuristic is inadmissible, we evaluate our heuristics in an agile search setting, where Greedy Best-First Search [Bonet and Geffner, 1999, GBFS] is the standard algorithm. We do not use A* because it does not guarantee finding the optimal (shortest) plan [Russell and Norvig, 2010] with inadmissible heuristics and it is slower than GBFS in the agile search as it must explore all nodes below the current best \(f = g + h\) value, which is unnecessary for finding a satisficing solution. In our experiments, we evaluate search performance as the combination of the number of solved instances and the number of heuristic evaluations required to solve each instance, with a limit of 10000 evaluations per problem. We do not use runtime as our metric so that results are independent of the hardware and software configuration. Additionally, we evaluated GBFS with the off-the-shelf
We compared the search performance of GBFS using heuristic as a baseline. The planning component is based on Pyperplan [Alkhazraj et al., 2020].

4.1 Heuristic Accuracy Evaluations

We focus on the results obtained by the NLM models, as our conclusions from the Linear and the HGN models (See Appendix) were not substantially different. Table 1 shows the MSE metric of the heuristics obtained by different configurations evaluated on the test instances (which are significantly larger than the training instances). Compared to the models trained with the NLL loss of $N$, those trained with our proposed $T_N$ loss often result in significantly more accurate heuristics. For example, in ferry and gripper, some $N$ models completely fail to learn a useful heuristic, as shown by the large heuristic errors (e.g., the base $N$/fixed/none model on ferry obtains an MSE of 118.59). In these situations, the clipping trick often reduces errors significantly (e.g., the $N$ + clip/fixed/none model on the same domain obtains an MSE of 10.50). However, this simply indicates that the $N$ models are falling back to the $h_{\text{LMcut}}$ heuristic for those (many) predictions which are smaller than $h_{\text{LMcut}}^*$. This is why, even with clipping, $N$ models fail to match the accuracy of $T_N$ models in many cases: For example, the MSE of $N$+clip/learn/none on gripper is 7.7 points larger than the one of $T_N$/learn/none. This confirms our hypothesis that admissible heuristics such as $h_{\text{LMcut}}^*$ should be used as the lower bound of $T_N$, instead of simply to perform post-hoc clipping of heuristic predictions.

Additional detailed observations follow. First, $T_N$ tends to converge faster during training, as shown in Fig. 2. Second, residual learning often improves accuracy considerably, thus proving to be an effective way of utilizing inadmissible heuristics. Third, we observed that the trained heuristics, including those that use residual learning from $h_{\text{FF}}$, tend to be more accurate than $h_{\text{FF}}$. This rejects the hypothesis that residual learning is simply copying $h_{\text{FF}}$ values. Fourth, learning $\sigma$ helps $T_N$ exclusively. For every $N$ and $T_N$ model, Table 1 contains 2 comparisons related to $\sigma$ (learn/none vs. fixed/none and learn/$h_{\text{FF}}$ vs. fixed/$h_{\text{FF}}$) across 4 domains, resulting in a total of 8 comparisons. Out of 8, learning $\sigma$ degrades the MSE of $N$ in 5 cases, while it improves the MSE of $T_N$ in 7 cases. This happens because $\sigma$ affects the expected value $E[x]$ of $T_N$ used as the heuristic prediction but it does not for $N$. In other words, $T_N$ models require both $\mu$ and $\sigma$ in order to achieve good heuristic accuracy. This explains why $T_N$/fixed/$h_{\text{FF}}$ is not as competitive as $N$/fixed/$h_{\text{FF}}$: fixed/$h_{\text{FF}}$ is an ill-defined configuration for $T_N$.

4.2 Search Performance Evaluations

We compared the search performance of GBFS using heuristic functions obtained by the different models as well as the State-of-the-Art off-the-shelf $h_{\text{FF}}$ heuristic. We included our proposed learn/$h_{\text{FF}}$ configuration and the baseline fixed/none configuration. Results for learn/none and fixed/$h_{\text{FF}}$ can be found in the Appendix. Table 2 shows the average±stdev of the ratio of problem instances solved (i.e., coverage), where a value of 1 means all instances are solved, and the average number of node evaluations per problem over 5 seeds. The second metric is introduced to differentiate between methods that solve most or all of the instances.

We observed that, with our proposed learn/$h_{\text{FF}}$ configuration, the learned heuristics significantly outperform the off-the-shelf $h_{\text{FF}}$ heuristic. Additionally, $T_N$ outperforms $N$ and $N$+clip in every domain when both the ratio of solved instances and number of node evaluations are considered (the second metric is used to break ties in the first one).

Conversely, with the traditional but less ideal fixed/none configuration, several learned heuristics are surpassed by $h_{\text{FF}}$ and, also, $T_N$ is outperformed by $N$ or $N$+clip in some cases. These results align with those shown in Table 1. Firstly, $N$ models which do not use clipping sometimes learn dismal heuristics (e.g., in gripper, $N$/fixed/none fails to solve any instance). Secondly, $T_N$ models need to predict $\sigma$ (in addition to $\mu$) in order to learn heuristics of good quality.
we argued that this heuristic should instead be modeled as a Gaussian distribution. Then, explained how the commonly used MSE loss implicitly mod-

Table 2: Planning results on NLM weights saved according to the best validation MSE metric, comparing the average stdev of the ratio of solved instances under $10^4$ node evaluations and the average number of evaluated nodes across problems. The number of evaluated nodes is counted as $10^4$ on instances the planner failed to solve. For each configuration (learn/$h_{FF}$ or fixed/none), we highlight the best results in bold.

6 Conclusion and Future Work

In this paper, we studied the problem of supervised heuristic learning under a statistical lens, focusing on how to effectively utilize the information provided by admissible heuristics. Firstly, we provided some statistical background on heuristic learning which was later leveraged to analyze the decisions made (sometimes unknowingly) in the literature. We explained how the commonly used MSE loss implicitly models the heuristic to be learned as a Gaussian distribution. Then, we argued that this heuristic should instead be modeled as a Truncated Gaussian, where admissible heuristics are used as the lower bound of the distribution. We conducted extensive experimentation, comparing the heuristics learned with our truncated-based statistical model versus those learned by minimizing squared errors. Results show that our proposed method improves convergence speed during training and yields more accurate heuristics that result in better planning performance, thus confirming that it is the correct approach for utilizing admissible bounds in heuristic learning.

Our findings serve to answer the three important questions we raised in the introduction: What should the model learn? To achieve super-symbolic benefits, we should use expensive metrics such as $h^*$, not poly-time heuristics or sub-optimal plan costs. How should we train the model? We maximize the likelihood of the observed $h^*$ assuming a Truncated Gaussian distribution lower bounded by an admissible heuristic. Why so? The principle of maximum entropy: the Truncated Gaussian distribution encodes our prior knowledge without any extra assumptions that may cause harmful bias.

In future work, we will extend our proposed method to other learning settings. One interesting scenario is given by iterative search algorithms [Richter et al., 2010; Richter et al., 2011], where the cost of the best solution found so far could be used as the upper bound of a Truncated Gaussian. Another avenue for future work is to explore the RL setting where a value function is learned instead of a heuristic, extending the work on residual learning for RL [Gehring et al., 2022].
Acknowledgements

This work has been partially funded by the Grant PID2022-142976OB-I00, funded by MICIU/AEI/10.13039/501100011033 and by “ERDF/EU”, as well as the Andalusian Regional predoctoral grant no. 21-111-PREDOC-0039 and by “ESF Investing in your future”.

References


