Predicting the utility of search spaces for black-box optimization: a simple, budget-aware approach

Setareh Ariafar
Google Research

Justin Gilmer
Google Research

Zachary Nado
Google Research

Jasper Snoek
Google Research

Rodolphe Jenatton
Google Research

George E. Dahl
Google Research

Abstract

Black box optimization requires specifying a search space to explore for solutions, e.g. a d-dimensional compact space, and this choice is critical for getting the best results at a reasonable budget. Unfortunately, determining a high-quality search space can be challenging in many applications. For example, when tuning hyperparameters for machine learning pipelines on a new problem given a limited budget, one must strike a balance between excluding potentially promising regions and keeping the search space small enough to be tractable. The goal of this work is to motivate—through example applications in tuning deep neural networks—the problem of predicting the quality of search spaces conditioned on budgets, as well as to provide a simple scoring method based on a utility function applied to a probabilistic response surface model, similar to Bayesian optimization. We show that the method we present can compute meaningful budget-conditional scores in a variety of situations. We also provide experimental evidence that accurate scores can be useful in constructing and pruning search spaces. Ultimately, we believe scoring search spaces should become standard practice in the experimental workflow for deep learning.

1 Introduction and Motivation

Solving a black box optimization problem requires defining a search space to optimize over, but in important applications of black box optimization there is not an obvious best search space *a priori*. Selecting the hyperparameters[1] of machine learning (ML) pipelines serves as our exemplar for this sort of application, since the choice of search space can be critical for getting good results. If the search space is too broad, the search may fail to find a good solution, but if it is too narrow, then it may not contain the best points. To complicate matters more, the quality of a search space will also depend on the compute budget used to search, with larger budgets tending to favor broader search spaces.

The necessity of selecting good search spaces for black box optimization methods (e.g. Bayesian optimization) presents a barrier to fully automating hyperparameter tuning for ML, and presents challenges for the reproducibility of ML research. Accurately measuring progress in modern ML techniques is notoriously difficult as most new methods either present new hyperparameters to be tuned, or interact strongly with existing hyperparameters (e.g. the optimizer learning rate). As a result, the performance of any new method may depend more on the effects of implicit or explicit tuning rather than on any specific advantage it might hold. Worryingly, methods with initial promising results are often later found to offer no benefits over a baseline that was rigorously tuned in retrospect (Melis et al., 2018; Merity et al., 2018; Lucic et al., 2018; Narang et al., 2021). In fact, recently it has become commonplace for contradictory results in the literature to be due almost entirely to differing tuning protocols and search spaces (Pineau et al., 2020; Nado et al., 2021; Choi et al., 2019). Ideally, methods papers could provide recommended search spaces for any hyperparameters, but providing good recommendations is challenging without a way to select search spaces for different budgets.

---

[1]Although "hyperparameter" has a precise meaning in Bayesian statistics, it is often used informally to describe any configuration parameter of a ML pipeline. We adopt this usage here and depend on context to disambiguate.

When tuning ML hyperparameters, a search space typically corresponds to a set of hyperparameter dimensions (e.g. the parameters of the learning rate schedule, other optimizer parameters such as momentum, regularization parameters, etc.) and allowed ranges for each dimension. For modern neural networks, it should be possible to define, *a priori*, a maximum conceivable range for any given hyperparameter and whether it should be tuned on a log or linear scale. However, in general, researchers may not know exactly which hyperparameters should even be tuned, and will also have to guess appropriate ranges. Since training neural networks can be so expensive, when faced with small budgets, one should be cautious about adding dimensions or increasing the search space volume, whereas with larger budgets, broader, higher-dimensional search spaces might yield dramatically better results.

In order to address these issues, we argue that the community should consider the problem of creating *search space scoring methods* that predict the utility of conducting a search (with some black box optimization algorithm), in a particular search space, given a particular budget. Such a score could be informed by initial function observations from a broad, enclosing search space of the sort that can be specified easily *a priori*. A useful score should be able to accurately rank the quality of search spaces in a way that meaningfully depends on the budget used for the search and correlates with the outcomes of a typical search.

Predictive search space scores would have several applications in ML hyperparameter tuning workflows. First and foremost, accurate scores would lead to better decisions when tuning hyperparameters. Practitioners often find themselves interacting with tools such as Bayesian optimization or random search in an iterative loop, tweaking and refining the search space over multiple rounds of experiments until they find a suitable space. Anecdotally, this workflow has become especially common in recent years as larger budgets with higher degrees of parallelism have become available. A search space score could select the most promising and budget-appropriate of several candidate search spaces to try next, help prune bad regions of an initial broad search space, or help match the budget to the space.

As a second application, we can use search space scores to answer scientific questions about experimental results. Although tempting, it does not follow that just because the best point returned by a black box optimization method sets some hyperparameter to a particular value (e.g. uses a tanh activation function instead of ReLU) other values might not produce just as good results. To answer such a question, one would need to tune the various other hyperparameters for networks assuming a ReLU activation function, separately re-tune

them assuming a tanh activation function, and then compare the results. Alternatively, the same question can be answered by comparing the scores of two search spaces, one that fixed the activation function to tanh and one that fixed the activation function to ReLU. Budget-conditional scores also make it explicit that the answer depends on the tuning budget, since we might have a situation where, say, the tanh activation function performs just as well as ReLU when we can afford to carefully tune the learning rate schedule, but when that isn't possible, perhaps ReLU does better.

Finally, as we alluded to above, accurate, budget-conditional search space scores could be a useful tool to improve the reproducibility of ML research. Scores could be used to recommend search spaces given different budgets for the hyperparameters of a new method, removing some of the guesswork when trying to tune a ML method on a new problem or dataset.

In this paper, we develop a simple method for computing budget-conditional search space scores based on some initial seed data. Taking inspiration from Bayesian optimization, we define a general scoring method based on a utility function applied to a probabilistic response surface model conditioned on the seed data. Our method is agnostic to the specific probabilistic model and thus can make use of future improvements in modeling for Bayesian optimization. Specifically, in this work, we:

- Motivate the search space scoring problem for machine learning hyperparameter tuning and argue it is worthy of more study.
- Introduce a simple method for computing budget-conditional scores, show that it can be accurate in experiments on real neural network tuning problems, and demonstrate its failure modes.
- Provide empirical evidence that these scores could be useful in many tuning workflows.
- Present examples of how one might answer experimental questions using search space scores.

2   Related Work

Bayesian optimization has a deep literature with well established methodologies for experimental design (Shahriari et al., 2016b; Garnett, 2022). It has become the state-of-the-art methodology for the tuning of hyperparameters of ML algorithms (Bergstra et al., 2011; Snoek et al., 2012; Turner et al., 2021). While the literature is full of methodological improvements to Bayesian optimization, they generally assume a reasonably well chosen search space.

Although we do not create a new Bayesian optimization method in this work so much as present a different

problem formulation, methods that focus on performing Bayesian optimization on very large search spaces or identifying promising local regions within a larger space are related. These methods do, in some sense, dynamically refine the search space or switch between smaller spaces. Wang et al. (2016b) and Letham et al. (2020) perform a random projection into a smaller latent space to enable optimization on very large search spaces under the assumption that many dimensions are either strongly correlated or not useful. Eriksson et al. (2019) introduce TuRBO to optimize over large scale, high dimensional problems using a bandit-style algorithm to select between multiple local models. McLeod et al. (2018) developed a method called BLOSSOM, which switches between a global Bayesian optimization and more traditional local optimization methods. Assael et al. (2014) divide up the search space into several smaller search spaces in the context of a treed Gaussian process, so that local GPs are well-specified on each of the smaller regions. Wang et al. (2020) partition a search-space by means of Monte Carlo Tree Search.

Search-space *pruning* strategies (Wistuba et al., 2015), using sparsity-inducing tools (Cho and Hegde, 2019), or search-space *expansion* approaches (Shahriari et al., 2016a; Nguyen et al., 2019) dynamically adjust the search space during optimization. Moreover, when previous data about related tasks is available, Perrone et al. (2019) prune search spaces using transfer learning. Through theoretical analysis, Tran-The et al. (2020) develop a method to dynamically expand search spaces while incurring sub-linear regret. The idea of transforming the search space over the course of the optimization also appears within the context of adjustments made by developers of an ML system (Stoll et al., 2020). Constrained Bayesian optimization (Gelbart et al., 2014; Hernández-Lobato et al., 2014; Gardner et al., 2014; Hernández-Lobato et al., 2016) provides a framework for modeling infeasible regions of the search space through the framework of constraints.

Although several technically sophisticated and effective methods exist to operate on search spaces that are too small, too broad or ill-behaved, we are not aware of work that accomplishes our goal, namely developing a tool to score search spaces, primarily within the context of ML hyperparameter tuning.

3   Scoring search spaces given budgets

Our goal is to score a search space $\mathcal{S}$, conditional on a budget b, such that drawing b points from a higher ranking search space is likely to lead to better (lower) function values from some function $f(x)$. In other words, our scores should reflect whether lower values of f reside inside a search space and how easy it will

be, on average, to find them with only b queries. In order to make this notion more precise, the following notation will be useful. Given a search space $\mathcal{S} \subseteq \mathcal{X}$, we denote a uniform sample from the set $\mathcal{S}$ with $x \sim U(\mathcal{S})$, and use $\mathbf{x} \sim U_b(\mathcal{S})$ to denote a uniform i.i.d. batch of b samples. Now we can write the expected minimal function value found after sampling b points from $\mathcal{S}$ as

$$\alpha(\mathcal{S}, b) = \mathbb{E}_{\mathbf{x} \sim U_b(\mathcal{S})} \left[\min_{x \in \mathbf{x}} f(x) \right].$$

Then, given two search spaces $\mathcal{S}_1$ and $\mathcal{S}_2$ together with a query budget b, $\alpha(\mathcal{S}_1, b) < \alpha(\mathcal{S}_2, b)$ would indicate that $\mathcal{S}_2$ should be preferred to $\mathcal{S}_1$ at budget b.

Unfortunately, there are two immediate challenges with this approach. First, the true function f is not known, and second, we can only access f through expensive, potentially noisy, evaluations. Therefore, if we assume access to some observed data $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^n$ where $\forall j \; x_j \in \mathcal{X}$ and $y_j = f(x_j) + \varepsilon = \tilde{f}(x_j)$, with ε denoting the observation noise, we can construct a probabilistic model of noise-corrupted f which we denote by $y = f(x) + \varepsilon \sim p(y|x, \mathcal{D})$. Our goal now is to leverage a learned probabilistic model $p(y|x, \mathcal{D})$ to design search space scoring functions $\hat{\alpha}(\cdot, b, p)$ that will be faithful to the original criterion $\alpha(\cdot, b)$. By "faithful", we mean that it should preserve the ranking that $\alpha(\cdot, b)$ induces, i.e. $\alpha(\mathcal{S}_1, b) < \alpha(\mathcal{S}_2, b) \;\Rightarrow\; \hat{\alpha}(\mathcal{S}_1, b, p) < \hat{\alpha}(\mathcal{S}_2, b, p)$.

In most cases, we actually want to improve upon some incumbent, best observed value y^+. In such cases, our scoring functions can target more general *utility functions* $u(y^+, f(\mathbf{x}))$. Similar batch improvement-based utilities exist in the Bayesian optimization literature as a part of batch acquisition functions, which we can directly employ here. For example, u might encode *whether* or *how much* the best f values of the x locations in $\mathbf{x}$ can improve upon y^+. However, because we do not have access to $f(\mathbf{x})$, we can instead estimate the expected utility $\mathbb{E}_{\hat{\mathbf{y}} \sim p(\mathbf{y}|\mathbf{x}, \mathcal{D})} [u(y^+, \hat{\mathbf{y}})]$ in place of $u(y^+, f(\mathbf{x}))$. If we adopt improvement over y^+ as our utility function, our expected utility expression becomes the multi-point expected improvement (b-EI) batch acquisition function (Wang et al., 2016a). Similarly, if we use binarized improvement as our utility, our expected utility will reproduce the multi-point probability of improvement (b-PI) batch acquisition function (Wang et al., 2016a). Of course unlike with the batch acquisition functions common in Bayesian optimization, we are not trying to find the location of the next batch of query points, but are instead predicting the score of a search space by averaging over batches of points within that search space.

Definition of the score: For concreteness, consider the case where we define the utility of a batch as the improvement over some incumbent value y^+. Ideally, we would directly compute the average batch expected improvement, i.e. the average b-EI, under $p(y|\mathcal{D}, x)$ for random batches of size b sampled from $\mathcal{S}$, namely

$$\hat{\alpha}(\mathcal{S}, b, p, y^+) =$$
$$\mathbb{E}_{\mathbf{x} \sim U_b(\mathcal{S}), \hat{\mathbf{y}} \sim p(\mathbf{y}|\mathbf{x}, \mathcal{D})} \left[\max \left(0, y^+ - \min(\hat{\mathbf{y}}) \right) \right], \quad (1)$$

or equivalently, $\mathbb{E}_{\mathbf{x} \sim U_b(\mathcal{S})} \left[\text{b-EI}(\mathbf{x}|y^+) \right]$. In addition to this score, we define an analogous score based on b-PI using binarized improvements. We refer to these scores as mean-b-EI and mean-b-PI in the remainder of the text. As a simple alternative, we can also replace the mean with respect to $\mathbf{x} \sim U_b(\mathcal{S})$ with the median. This change will lead to a slightly modified definition with hopefully less sensitivity to outlier utilities. We call the median-based versions of equation (1) median-b-EI and median-b-PI (with binarized improvements).

Computing search space scores in practice: In order to compute any scores in practice, we must first fix a probabilistic model. In this work, we assume a Gaussian Process (GP) model on f and evaluate the expected utility by taking the expectation of u with respect to $p(\mathbf{y}|\mathbf{x}, \mathcal{D})$, which is a multivariate Gaussian. See A.1 and (Rasmussen, 2003) for more details on the GP structure and inference. Assuming a GP model, the expected utilities b-PI and b-EI can be calculated using high-dimensional multivariate Gaussian CDFs, however, this is not practical especially for large values of b (Frazier, 2018).

As an alternative, we use a simple Monte Carlo approach to estimate the expectations in equation (1) ; we draw samples $\mathbf{x} = \{x_i\}_{i=1}^b$ from $U_b(\mathcal{S})$, then form the GP posterior $p(\mathbf{y}|\mathbf{x}, \mathcal{D})$ conditioned on the sampled input locations, and then finally compute the average. Similarly, we estimate the expectation over the different input locations by sampling a number of different batches of $\mathbf{x}$ and averaging. We refer to the result of this computation as the *predicted score* for $\mathcal{S}$ at budget b. In addition to the *ex-ante* predicted score, for the purposes of validation, we can also compute an *ex-post* score based on the empirical utility, that is $\mathbb{E}_{\mathbf{x} \sim U_b(\mathcal{S})} \left[u(y^+, \tilde{f}(\mathbf{x})) \right]$ using real, noise-corrupted evaluations of f at $\mathbf{x}$. We call this score the *empirical score* for $\mathcal{S}$ at budget b. As we will discuss later, the empirical scores are useful to validate the accuracy of predicted scores retrospectively.

3.1 Example applications of the scores

Our predicted scores have a variety of potential uses in the deep learning experimental workflow. We present three illustrative examples of how they could provide value, although there will inevitably be other cases too.

3.1.1 Ranking of search spaces

The simplest application, which we already have alluded to, is to rank a set of search spaces $\mathbb{S} = \{\mathcal{S}_t\}_{t=1}^T$ that are all subsets of an enclosing loose search space $\mathcal{X}$ based on their predicted score, at budget b, given some initial data $\mathcal{D}$ from $\mathcal{X}$. Ranking search spaces lets us pick what search space to explore next in workflows that construct multiple search spaces in sequence or can answer questions about results we have already posed as ranking questions. See Algorithm 1 for details. Note that the initial data $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^n$ could be obtained in a variety of ways, for example by uniformly sampling and evaluating n points from $\mathcal{X}$ or by n rounds of Bayesian optimization on $\mathcal{X}$. Although not useful for the applications we consider, we can also imagine a retrospective version of Algorithm 1 that replaces predicted scores with empirical scores that we can use to validate our predicted rankings.

Algorithm 1 Predicted search space ranking

Require: Budget b, search space set $\mathbb{S}$, initial data $\mathcal{D}$
1: $y^+ \leftarrow$ min objective in $\mathcal{D}$
2: Fit probabilistic model $p(y|x, \mathcal{D})$
3: **for** $\mathcal{S}$ in $\mathbb{S}$: **do**
4:      predicted-scores$[\mathcal{S}] = \hat{\alpha}(\mathcal{S}, b, p, y^+)$
5: **end for**
6: **return** predicted-scores

3.1.2 Pruning bad search space regions

With a modest amount of data from an overly broad initial search space $\mathcal{X}$, it should be possible to quickly remove bad regions of the space using search space scores, leaving a final pruned search space that is more efficient to search than the original. Although exploring more sophisticated pruning algorithms based on these scores would be an interesting avenue for future work, here we present a simple proof-of-concept algorithm that does one round of pruning based only on a single initial sample of observations. Our algorithm assumes access to a subroutine `propose_search_spaces`$(\mathcal{X}, T)$ that proposes candidate search spaces $\{\mathcal{S}_t\}_{t=1}^T$, where $\forall t, \mathcal{S}_t \subset \mathcal{X}$. We describe different possible strategies to implement such a function in A.2.2. The algorithm also makes use of a subroutine `sample_observations`$(\mathcal{S}, \tilde{f}, b)$ that selects b points in $\mathcal{S}$ and evaluates them using the objective function $\tilde{f}$, producing the resulting (x, y) pairs as output. The `sample_observations` subroutine could be implemented with uniform random search or a Bayesian optimization algorithm. Given a budget of $B = b_1 + b_2$, Algorithm 2 uses the first b_1 points as initial exploratory data to prune the starting search space and then sam-

ples the remaining b_2 points from the pruned space, finally returning the best point.

3.1.3 Deciding whether or not to tune a particular hyperparameter

Given infinite resources, tuning any given hyperparameter can never do worse than setting it to a fixed value. However, in practice, when constrained by a finite budget, in some cases fixing certain hyperparameters will outperform tuning them. Although there are many ways to determine the relative importance of different hyperparameters (e.g. by examining learned length scales of an ARD kernel or approaches such as van Rijn and Hutter (2018)), determining when to tune or fix specific hyperparameters is a slightly different question that inherently involves reasoning about the available tuning budget. Thankfully, with budget-conditional search space scores, this question is quite straightforward to answer since we can reduce it to a question about ranking search spaces. Specifically, let $\mathcal{X}^d$ be the dth hyperparameter and suppose the search space $\mathcal{X}$ constrains $\mathcal{X}^d \in [l^d, u^d]$. Let $\mathcal{Z}$ be a copy of search space $\mathcal{X}$ where $\mathcal{X}^d = \delta$, for some value $\delta \in [l^d, u^d]$. To decide whether, given a budget of b evaluations, we can afford to tune $\mathcal{X}^d \in [l^d, u^d]$ or we should instead fix $\mathcal{X}^d = \delta$, we can apply Algorithm 1 to score the search spaces $\mathcal{X}$ and $\mathcal{Z}$ and see which one obtains the better score. In practice, we might be interested in considering a δ that is either some standard default value of the hyperparameter or the value it takes on at the best point observed so far.

Algorithm 2 One-shot search space pruning

Require: Budget split $B = b_1 + b_2$, objective $\tilde{f}$, T, `sample_observations`, `propose_search_spaces`
1: $\mathcal{D}_1 \leftarrow$ `sample_observations`$(\mathcal{X}, \tilde{f}, b_1)$
2: $y^+ \leftarrow$ minimum objective in $\mathcal{D}_1$
3: Build probabilistic model $p(y|x, \mathcal{D}_1)$
4: $\{\mathcal{S}_1, \ldots, \mathcal{S}_T\} \leftarrow$ `propose_search_spaces`$(\mathcal{X}, T)$
5: $\mathcal{S}^\star = \arg\max_{\mathcal{S}_t} \hat{\alpha}(\mathcal{S}_t, b_2, p, y^+)$
6: $\mathcal{D}_2 \leftarrow$ `sample_observations`$(\mathcal{S}^\star, \tilde{f}, b_2)$
7: **return** The best value $(x_{\min}, y_{\min})$ from $\mathcal{D}_1 \cup \mathcal{D}_2$

---

4 Experiments

In addition to synthetic functions to build intuition, given our focus on machine learning applications, we use several realistic problems from hyperparameter tuning for deep neural networks to demonstrate our search space scoring method. Our experiments generally follow the three example applications in Section 3.1, with the details of the synthetic functions and hyperparameter tuning problems we consider, below.

4.1 Experiment & optimization task details

We instantiate our GPs following the standard design choices of Eriksson et al. (2019); see A.2.1 for details. Initial observations to train the GP are uniformly sampled from the base search space $\mathcal{X}$. Unless otherwise specified, scores are mean-b-EI and computed with with 1000 sampled x-locations and 1000 samples from the GP posterior at each x-location. We observed that other variations including mean-b-PI, median-b-EI and median-b-PI produce generally similar results. See A.3.2 for more details. All neural network experiments used a polynomial learning rate decay schedule parameterized by an initial learning rate, a power, and the fraction of steps before decay stops; the learning rate reaches a final constant value 1e-3 time the initial learning rate (see A.2.3 for details). The broad search space for all neural net tuning problems has seven hyperparameters: the initial learning rate η, one minus momentum $1 - \beta$, the decay schedule power p, decay step factor τ, dropout rate r, l_2 decay factor λ, and label smoothing γ. See Table 1 for the scaling and ranges. We collected our neural network tuning data on CIFAR100, ImageNet and LM1B using open source code from Gilmer et al. (2021). An implementation of our score function and its applications will be available at https://github.com/google-research/google-research/tree/master/spaceopt.

Synthetic tasks: We use two synthetic functions in our experiments, the two-dimensional Branin function and the six-dimensional Hartman function (Eggensperger et al., 2013). The base search space for Branin is $\mathcal{X} = [[-5., 10.], [0., 15.]]$. The base search space for the Hartman function is $[0, 1]^6$. For Branin, in addition to the base search space $\mathcal{X}$, we make use of two additional search spaces, $\mathcal{S}_1$ and $\mathcal{S}_2$, that both have roughly 10% of the volume of $\mathcal{X}$, with $\mathcal{S}_1$ centered around $x_{\text{best}} = \arg\min_{y_j \in \mathcal{D}} y_j$ and $\mathcal{S}_2$ centered around $x_{\text{worst}} = \arg\max_{y_j \in \mathcal{D}} y_j$ (see A.2.2 for additional search space construction details).

CIFAR100: On CIFAR100 (Krizhevsky, 2009) we tune a WideResNet (Zagoruyko and Komodakis, 2016) using two hundred training epochs. In addition to the base search space $\mathcal{X}$, we make use of two additional search spaces $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_1$ is generated following A.2.2 around $x_{\text{median}} = \arg\text{median}_{y_j \in \mathcal{D}} y_j$ with roughly 25% of the volume of $\mathcal{X}$ and $\mathcal{S}_2$ is generated similarly around x_{median} with roughly 50% of the volume of $\mathcal{X}$. Note that since both $\mathcal{S}_1$ and $\mathcal{S}_2$ are centered around the same point, $\mathcal{S}_2$ encloses $\mathcal{S}_1$ and, as always, $\mathcal{X}$ encloses both. For details, see A.2.3.

ImageNet: On ImageNet (Russakovsky et al., 2015) we tune ResNet50 (He et al., 2016), trained using ninety epochs. In addition to $\mathcal{X}$, we make use of two additional
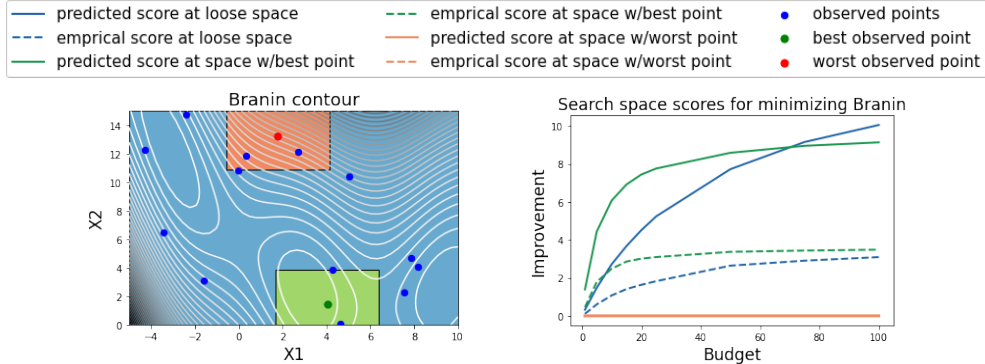
Figure 1: Scoring Branin function search spaces: given fifteen random observations (left), we compare three search spaces: the enclosing broad search space (blue), and two equal-volume spaces centered around the best and the worst observations (green and orange). We report the predicted mean-b-EI scores (right: solid lines) and compare with empirical scores observed from random sampling true values within each search space (right: dashed lines). The predicted scores correctly show the orange search spaces is worse, and that the gap between the others shrinks at larger budgets.
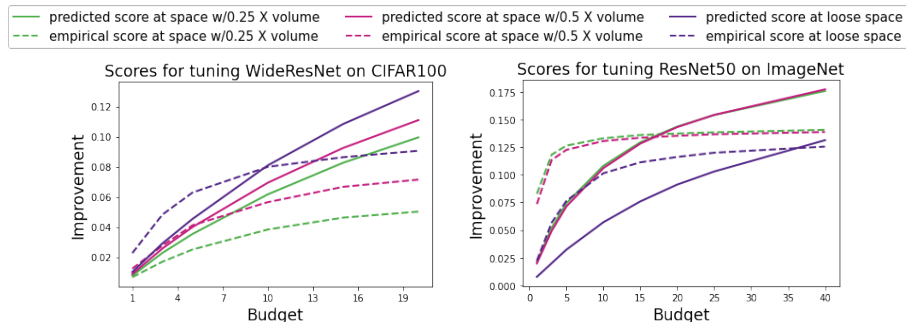

Figure 2: Comparing search spaces across budgets for tuning a WideResNet on CIFAR100 (left) and a ResNet50 on ImageNet (right). For both, the predicted scores successfully preserve the search space ranking of the empirical scores.

search spaces $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_1$ is generated following A.2.2 centered at $x_{\text{best}} = \arg\min_{y_j \in \mathcal{D}} \; y_j$ with roughly 25% of the volume of $\mathcal{X}$ and $\mathcal{S}_2$ is also centered on x_{best}, but with roughly 50% of the volume of $\mathcal{X}$. Once again $\mathcal{S}_1 \subset \mathcal{S}_2 \subset \mathcal{X}$. See A.2.3 for additional details.

LM1B: For LM1B (Chelba et al., 2013) we train Transformer model from Vaswani et al. (2017), also referred to as the post-layer-norm Transformer, for 75 epochs. We tune the same seven hyperparameters as above for CIFAR100 and ImageNet.

4.2 Ranking search spaces across budgets

When ranking search spaces, our scores should exhibit three qualitative behaviors that are sometimes in tension. First, scores should prefer search spaces where $p(y|x, \mathcal{D})$ predicts better function values. Second, as the budget increases, scores should be less risk averse. In other words, the score will be less sensitive to how hard good points are to locate *within* the search space, as long as good points exist (according to $p(y|x, \mathcal{D})$). Third, predicted scores should generally agree with the empirical ranking. Our goal here is to illustrate the different types of behavior the predicted scores can have and show that, although our desiderata are

sometimes in conflict, nonetheless on these example problems we see sensible behaviors (we discuss failure modes in Section 4.6). We use GPs conditioned on a modest number of observed points: $|\mathcal{D}| = 15$ for Branin and tuning WideResNet on CIFAR100, and $|\mathcal{D}| = 20$ for tuning ResNet50 on ImageNet.

We can see several examples of mean-b-EI preferring search spaces where the GP predicts there are good function values or, equivalently, penalizing search spaces where the GP predicts there are no good function values. On Branin (Fig. 1), $\mathcal{S}_1$ (green) and $\mathcal{S}_2$ (orange) are disjoint and $\mathcal{S}_1$ holds the incumbent y^+ whereas $\mathcal{S}_2$ contains worse observations. And indeed we see $\mathcal{S}_2$ decisively ranked worse (in agreement with the empirical score ranking). For CIFAR100 (Fig. 2, left), $\mathcal{S}_1 \subset \mathcal{S}_2 \subset \mathcal{X}$, but $\mathcal{S}_1$ and $\mathcal{S}_2$ are both centered on a mediocre point and the best point we measured is outside of them both. Presumably the GP posterior assigns a relatively high probability to good points being found outside of $\mathcal{S}_1$ and $\mathcal{S}_2$ and thus we see $\mathcal{X}$ ranked above $\mathcal{S}_2$ ranked above $\mathcal{S}_1$, once again agreeing with the empirical score ranking. For ImageNet (Fig. 2, right), $\mathcal{S}_1 \subset \mathcal{S}_2 \subset \mathcal{X}$ and both the inner search spaces are centered on the same very good observation that the GP also conditions on, so the more interesting
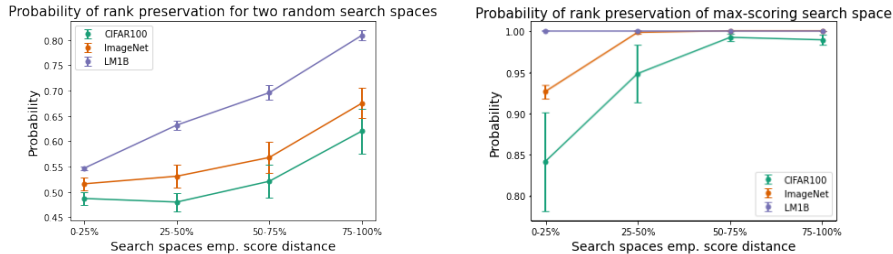
Figure 3: Probability of preserving the rank of **(1)** two random search spaces (left) and **(2)** a random search space and the best (max-scoring) search space (right). We consider three tuning problems including a WideResNet on CIFAR100, a ResNet50 on ImageNet and a Transformer model on LM1B. We report rank accuracy as a function of how close the empirical scores of the search spaces are. The closer the search spaces, the harder their correct ranking will be to preserve. In both cases, we see that the predicted-score rank accuracy improves as the search spaces differ more.

question is whether the predicted ranking agrees with the empirical score ranking, which it does.

We can also see how the predicted scores become more risk tolerant (care less about how unlikely random search is to find the good points in a search space) in our three example problems. On Branin, $\mathcal{S}_1 \subset \mathcal{X}$ and as the budget increases, eventually the predicted score ranks $\mathcal{X}$ above $\mathcal{S}_1$, which is reasonable *even though it disagree with the empirical score ranking*. Both $\mathcal{X}$ and $\mathcal{S}_1$ contain optimal points, but the GP has no way of knowing $\mathcal{S}_1$ contains an optimum, so eventually the score prefers the larger, enclosing search space. For CIFAR100, although the outermost search space is always preferred, the gap widens as the budget increases, as we would hope. For ImageNet, the outermost enclosing search space happens to be ranked worse for the budgets we plotted. However, although we don't have enough data to extend the empirical scores to a high enough budget, eventually, around $b = 1000$, the predicted score for the $\mathcal{X}$ actually overtakes the others.

4.3 Ranking randomly generated search spaces

In 4.2, we showed that our scores correctly rank a set of manually defined search spaces. Here, we estimate the probability of preserving the rank of pairs of randomly generated search spaces. We consider three tuning problems including a WideResNet on CIFAR100, a ResNet50 on ImageNet and a Transformer model on LM1B, all scored with $\mathcal{D} = 20$ and budget 15. We generate search spaces from the distribution described in Appendix A.2.2 with volume reduction rates $\rho \in \{0.1, 0.2, \ldots, 0.9\}$ with 50 search spaces per rate. Then, for two random search spaces $\mathcal{S}_1$ and $\mathcal{S}_2$, we compute the probability of having $\texttt{emp\_score}(\mathcal{S}_1) > \texttt{emp\_score}(\mathcal{S}_2) \Rightarrow \texttt{pred\_score}(\mathcal{S}_1) > \texttt{pred\_score}(\mathcal{S}_2)$. Clearly, the *closer* $\mathcal{S}_1$ and $\mathcal{S}_2$ are in empirical score, the more likely it is for noise in both scores to cause a benign disagreement. Furthermore, the particular distribution of search spaces we consider will induce an

arbitrary distribution over the distance between pairs of search spaces. Therefore, we report the probability of correctly ranking as a function of how close $\mathcal{S}_1$ and $\mathcal{S}_2$ are, based on $|\texttt{emp\_score}(\mathcal{S}_1) - \texttt{emp\_score}(\mathcal{S}_2)|$ quantiles (e.g., "0-25 %." contains all the smallest distances until the first quartile). We estimate the probability using 2000 samples, where each sample selects two search spaces at random from the pool of 450 we generated; Fig. 3 reports the mean and standard error over 10 runs (each run has a different random $\mathcal{D}$). We also replicate the experiment for random search spaces versus the max-scoring (according to the empirical score) search space, $\mathcal{S}^*$.

Note that the first case, i.e., comparing two random search spaces is more challenging than the second case, comparing a random search space with the max-scoring search space. Regardless, we see in Fig. 3 that our scores correctly rank well-separated search spaces with high probability in both cases. Intuitively, the ranking accuracy increases as the search spaces differ more significantly.

4.4 Pruning bad search space regions

We consider three problems including the Hartman function, a WideResNet on CIFAR100 and a ResNet50 on ImageNet, using budgets of $B = 60$, 40 and 40 respectively. For each problem, we set $b_1 = b_2 = B/2$ and let $\texttt{sample\_observations}$ be random search and batch Bayesian optimization (batch-BO) with batch size $B/2$. For batch-BO, we use the batch Thompson sampling acquisition function (Hernández-Lobato et al., 2017). By design, both random search and batch-BO coincide on b_1, since the initial points of batch-BO are random. We implemented $\texttt{propose\_search\_spaces}$ as described in Appendix A.2.2 with volume reduction rates $\rho \in \{0.1, 0.2, \ldots, 0.9\}$ with 500 search spaces per rate. We ran Alg. 2 for each problem for 100 rounds for Hartman and 50 rounds for the rest and report the mean and standard error (see Fig. 4 and Fig. 8). For all problems, random search/batch-BO in the max-scoring
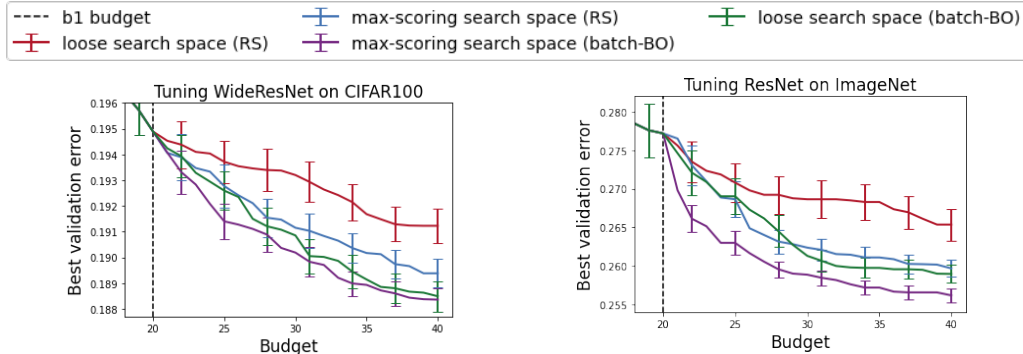
Figure 4: Random search and batch Bayesian optimization (BO) applied to the loose search space (red and green) and the max-scoring pruned search space (blue and purple). The score-guided random search (blue) and score-guided batch-BO (purple) split and spend the budget as explained in Alg. 2. Before the b_1 split (on the left of the vertical dashed lines; not fully displayed), all approaches coincide by design. For both WideResNet on CIFAR100 (left) and ResNet50 on ImageNet (right), the pruning provides a clear advantage by the time the budget is exhausted for both sampling methods.
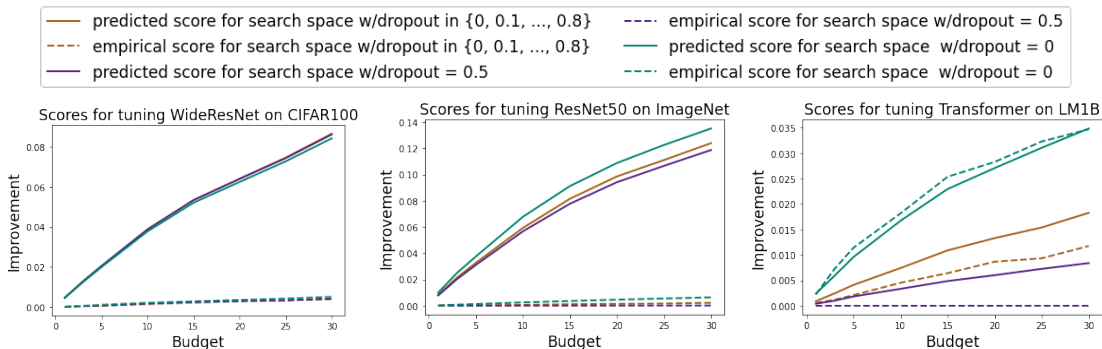

Figure 5: Deciding whether to tune over dropout, fix it to 0, or fix it to 0.5 for WideResNet on CIFAR100, ResNet50 on ImageNet and Transformer on LM1B.

space $\mathcal{S}^*$ outperforms random search/batch-BO over the broad search space $\mathcal{X}$. Interestingly, for ImageNet, the final performance of random search on $\mathcal{S}^*$ is close to that of bath-BO on $\mathcal{X}$. See A.3.3 for the performance of CIFAR100 under different budget splits $B = b_1 + b_2$ where we show our method is robust to the choice of split threshold.

4.5 Deciding whether to tune dropout

Consider the choice to tune or fix dropout on CIFAR100, ImageNet, and LM1B given $|\mathcal{D}| = 35$ points of initial data, with various budgets of remaining evaluations. We score search spaces with dropout tuned ($\mathcal{X}$), fixed to $r = 0$ ($\mathcal{Z}_1$), and fixed to $r = 0.5$ ($\mathcal{Z}_2$). For CIFAR100 (Fig. 5, left), both the predicted and empirical scores are nearly equal for all three search spaces, and indeed for WideResNet on CIFAR100 in our setup tuning dropout or fixing it to 0 or 0.5 makes no difference. For ImageNet (Fig. 5, middle), the predicted scores are moderately higher for $\mathcal{Z}_1$ ($r = 0$) and agree with the empirical ranking.

For LM1B (Fig. 5, right), we see that the predicted scores successfully detect that at these budgets dropout should be fixed to zero and not tuned and that this

choice matters a lot. Interestingly, the predicted scores have a similar relative difference as the empirical scores. On all three problems, the predicted scores succeed in deciding whether to tune dropout and correctly indicate when the choice matters, suggesting that search space scores can be a useful tool in the deep learning tuning workflow.

4.6 Limitations & failure modes

Our search space scores are only as good as the probabilistic model they depend on; if $p(y|x, \mathcal{D})$ is poorly calibrated or otherwise inaccurate they will produce nonsensical results. A variety of things can cause a poor GP fit, but we encountered issues most frequently when using small numbers of observations, especially when the observations only included relatively bad points. See Fig. 6 for a demonstration on Branin. In addition to harming the GP fit, observed data with consistently large objective values results in an unrealistically large incumbent value that especially harms the b-PI-based scores. Similar issues can infect our downstream applications. Revisiting deciding whether to tune dropout on LM1B, this time with $|\mathcal{D}| = 5$ with the best validation error a disappointing $y^+ = 0.769$ (see Fig. 7). the
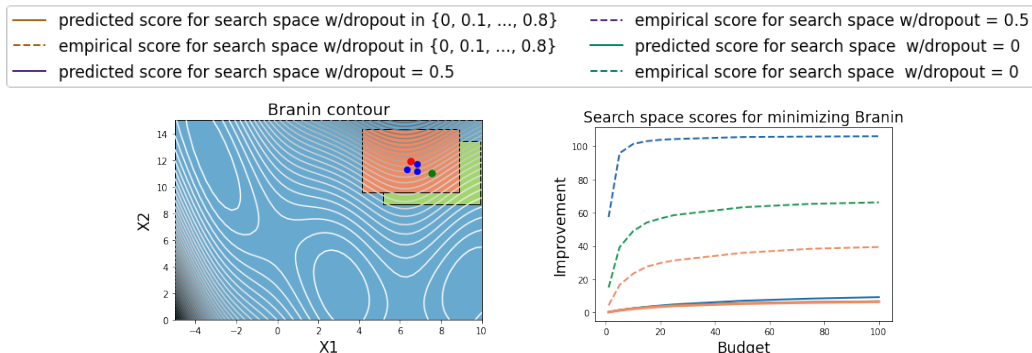
Figure 6: A failure mode of the scoring method. We revisit the Branin experiment from 4.2 but instead pick five observed points intentionally from a bad region. Although the search spaces including the best and worst observations overlap (left), the empirical scores suggest a clear ranking of the three search spaces, with the loose search space ranked highest. However, the predicted scores view the search spaces as nearly equivalent due to the poor GP fit.
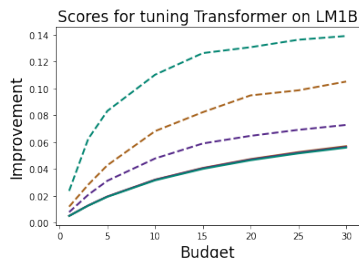

Figure 7: Reproducing Fig. 5, right, with a much worse GP that only observes five points, all with relatively poor performance causes the predicted scores to be nearly useless.

predicted scores can no longer distinguish the search spaces. Finally, for pruning bad regions of a search space in 4.4, although our method is fairly robust to the value of splitting threshold (see A.2.2), extreme values (e.g. $b_1 = 1$ or $b_1 = B - 1$) make sampling in the pruned space $\mathcal{S}^*$ no better than using $\mathcal{X}$.

5 Conclusions and Future Work

In this work, we motivated the problem of scoring search spaces conditioned on a budget, and demonstrated a simple method for computing such scores given any probabilistic model of the response surface. Our experiments demonstrate the proof of concept that useful scoring models will allow practitioners to answer tuning questions currently unaddressed in the literature, and greatly benefit their workflow. Applications of accurate search space scores will go beyond simply pruning bad search spaces in iterated studies, they have the potential to help mitigate the ML reproducibility crisis by allowing researchers to provide guidelines on how to best reproduce their results conditioned on a budget. Hopefully, this will enable the community to develop robust methods that can be deployed broadly and reduce effort adapting methods to new settings.

Our search space scoring functions are based on a general expected utility framework and thus can be

extended in a variety of ways. They are agnostic to the choice of model for $p(\mathbf{y}|\mathbf{x}, \mathcal{D})$ and alternative models such as random forests or Bayesian neural networks might, e.g., scale better. We can also generalize the improvement-based utility $u(y^+, \mathbf{y})$ to $u_g(\theta, \mathcal{D}, \mathbf{x}, \mathbf{y})$ where $\mathcal{D}$ is the observed data and θ denotes additional parameters of the generalized utility function. For example, inspired by batch Bayesian optimization, $u_g(\theta, \mathbf{x}, \mathbf{y})$ could be the amount of information gained about the location of the minimum of f from observing $\mathbf{x}, \mathbf{y}$ (Shah and Ghahramani, 2015). Such a choice of utility function would remove the direct dependence of the scores on the incumbent y^+, but it might come with new challenges, namely search spaces that produce the most information about the location of the minimum are not necessarily the same ones that will result in better points. Information-theoretic utility functions also might be more difficult to estimate than our simple improvement-based utility functions. Finally, we could explore scores based on non-uniform sampling of $\mathbf{x}$ locations, that is to say $\mathbf{x} \sim \Delta_b(\mathcal{S}, v)$ where v denotes additional parameters of the sampling distribution.

Moving forward, there are many potential directions to build on our search space scores. Potential extensions to our framework include: more general sampling strategies, more general budget constraints (such as time), and using a score function to propose optimal search space bounds for a given budget. Beyond extending the framework, there is more to be done on the modeling side. While our scores usually were rank preserving, they sometimes overestimated the magnitude of improvement, and performed poorly when conditioned on little data. Better model specification, e.g. through bespoke or data-dependent priors, could mitigate these issues. Overall, developing better budget-aware search space scores would unlock new applications for Bayesian optimization and we hope our method serves as a useful starting point for the community to tackle this challenging problem.

References

J.-A. M. Assael, Z. Wang, B. Shahriari, and N. de Freitas. Heteroscedastic treed bayesian optimisation, 2014. arXiv:1410.7172.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, 2011.

C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013. URL http://arxiv.org/abs/1312.3005.

M. Cho and C. Hegde. Reducing the search space for hyperparameter optimization using group sparsity. In *International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2019.

D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown, et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, 2013.

D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in Neural Information Processing Systems*, 2019.

P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

J. Gardner, M. Kusner, Zhixiang, K. Weinberger, and J. Cunningham. Bayesian optimization with inequality constraints. In *International Conference on Machine Learning*, 2014.

R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. in preparation.

M. Gelbart, J. Snoek, and R. Adams. Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence*, 2014.

J. M. Gilmer, G. E. Dahl, and Z. Nado. init2winit: a jax codebase for initialization, optimization, and tuning research, 2021. URL http://github.com/google/init2winit.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems*, 2014.

J. M. Hernández-Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman, and Z. Ghahramani. A general framework for constrained bayesian optimization using information-based search. *Journal of Machine Learning Research*, 2016.

J. M. Hernández-Lobato, J. Requeima, E. O. Pyzer-Knapp, and A. Aspuru-Guzik. Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *International conference on machine learning*, pages 1470–1479. PMLR, 2017.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

B. Letham, R. Calandra, A. Rai, and E. Bakshy. Re-examining linear embeddings for high-dimensional bayesian optimization. In *Advances in Neural Information Processing Systems*, 2020.

D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly. Are gans created equal? a large-scale study. In *Advances in Neural Information Processing Systems*, 2018.

M. McLeod, S. Roberts, and M. A. Osborne. Optimization, fast and slow: optimally switching between local and bayesian optimization. In *International Conference on Machine Learning*, 2018.

G. Melis, C. Dyer, and P. Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018.

S. Merity, N. S. Keskar, and R. Socher. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018.

J. Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

Z. Nado, J. Gilmer, C. J. Shallue, R. Anil, and G. E. Dahl. A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes. *arXiv preprint arXiv:2102.06356*, 2021.

S. Narang, H. W. Chung, Y. Tay, W. Fedus, T. Févry, M. Matena, K. Malkan, N. Fiedel, N. Shazeer, Z. Lan, Y. Zhou, W. Li, N. Ding, J. Marcus, A. Roberts, and C. Raffel. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

V. Nguyen, S. Gupta, S. Rana, C. Li, and S. Venkatesh. Filtering bayesian optimization approach in weakly specified search space. *Knowledge and Information Systems*, 60(1):385–413, 2019.

V. Perrone, H. Shen, M. W. Seeger, C. Archambeau, and R. Jenatton. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, 2019.

J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. B. Fox, and H. Larochelle. Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program). *ArXiv preprint arXiv:2003.12206*, 2020.

C. E. Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

A. Shah and Z. Ghahramani. Parallel predictive entropy search for batch global optimization of expensive objective functions. *arXiv preprint arXiv:1511.07130*, 2015.

B. Shahriari, A. Bouchard-Côté, and N. Freitas. Unbounded bayesian optimization via regularization. In *Artificial intelligence and statistics*, 2016a.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016b.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

D. Stoll, J. K. H. Franke, D. Wagner, S. Selg, and F. Hutter. Hyperparameter transfer across developer adjustments. *NeurIPS 4th Workshop on Meta-Learning*, 2020.

H. Tran-The, S. Gupta, S. Rana, H. Ha, and S. Venkatesh. Sub-linear regret bounds for bayesian optimisation in unknown search spaces. *arXiv preprint arXiv:2009.02539*, 2020.

R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, 2021.

J. N. van Rijn and F. Hutter. Hyperparameter importance across datasets. *SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2018)*, 2018.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

J. Wang, S. C. Clark, E. Liu, and P. I. Frazier. Parallel bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*, 2016a.

L. Wang, R. Fonseca, and Y. Tian. Learning search space partition for black-box optimization using monte carlo tree search. *arXiv preprint arXiv:2007.00708*, 2020.

Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Freitas. Bayesian optimization in a billion dimensions via random embeddings. *J. Artif. Int. Res.*, 55(1):361–387, Jan. 2016b. ISSN 1076-9757.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 104–119. Springer, 2015.

S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A   Appendix

A.1   Background: Bayesian Optimization

Bayesian Optimization (BO) is an automated method for optimizing expensive black-box objective functions. Consider a black-box (expensive to query) objective function f over domain $\mathcal{X}$ that we wish to minimize. The observed values of f are potentially corrupted with noise. Given a set of observed data $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^n$ where $y_i \sim \text{Normal}(f(x_j), \sigma_{\text{noise}}^2)$, BO builds a response surface model for f, typically assumed to be a Gaussian Process (GP). GPs are probabilistic non-parametric models and a popular choice for modelling smooth functions because they allow closed-form inference and can naturally accommodate prior knowledge (Rasmussen, 2003).

Let $x_{(1:n)}$ and $y_{(1:n)}$ denote the observed data in $\mathcal{D}$. Given a batch of points $\mathbf{x} = \{x_i\}_{i=1}^m$, the posterior distribution over the values of f at $\mathbf{x}$ is a multivariate Gaussian with mean

$$\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x}, x_{(1:n)})\Big(\mathbf{K} + \sigma_{\text{noise}}^2 I\Big)^{-1} y_{(1:n)}$$

and covariance

$$\Sigma(\mathbf{x}) = \mathbf{k}(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, x_{(1:n)})\Big(\mathbf{K} + \sigma_{\text{noise}}^2 I\Big)^{-1}\mathbf{k}(x_{(1:n)}, \mathbf{x}),$$

where $\mathbf{K} = [k(x_j, x_j')]_{j,j'=1}^n$, $\mathbf{k}(\mathbf{x}, \mathbf{x}) = [k(x_i, x_{i'})]_{i,i'=1}^m$, and $\mathbf{k}(\mathbf{x}, x_{(1:n)}) = [k(x_i, x_j)]_{i,j=1}^{m,n}$. Bayesian optimization selects the x-location of the next batch of points to query f by maximizing an acquisition function over a sample batch of points $\mathbf{x}$. The value of the acquisition function for a batch $\mathbf{x}$ is the expected *utility* of the batch, where the expectation is taken with respect to the objective values at $\mathbf{x}$, that is $\mathbf{y}|\mathbf{x}, \mathcal{D}$. Two popular choices of utility functions are the indicator and the magnitude of the improvement over the best value of $\mathbf{y}$, also known as the incumbent value (Močkus, 1975; Jones et al., 1998). See (Frazier, 2018) for different choices of acquisition functions and details of BO algorithm.

A.2   Experimental Setup

In this section, we describe the details of the specific design and modeling choices we made for our experiments in Section 4. In Section A.2.1, we explain the GP implementation details, including the choice of the mean and the covariance functions, the hyperpriors, and the reparameterization we applied to the GP hyperparameters. Next, in Section A.2.2, we discuss the procedures we followed to generate search spaces that we score in Section 4.2 and Section 4.4, although many other alternative procedures are possible. We make no claim to have found any sort of optimal procedure.

Finally, in Section A.2.3, we include the specific bounds of the search spaces we scored in Section 4.1 for tuning a WideResNet on CIFAR100 and for tuning a ResNet50 on ImageNet.

A.2.1   GP implementation details

In our experiments, we used GPs with constant, zero mean and an ARD Matérn-5/2 kernel. For the purposes of optimization, we reparameterized amplitude, lengthscales and noise variance using a softplus function. We place standard zero-mean, unit variance log-Normal priors on the amplitude, and inverse-lengthscales, and a zero-mean 0.1-variance normal on the noise term. The GP hyperparameters are fitted by optimizing the log of the marginal likelihood function via 3000 steps of LBFGS (Liu and Nocedal, 1989).

A.2.2   Search space generation

Throughout our experiments, we followed a simple volume-constrained procedure to generate search spaces, depending on the situation, either centering them on a particular point or placing them randomly with a base search space. Given a volume reduction rate $\rho \in (0, 1]$ and a d-dimensional base hyperrectangular search space $\mathcal{X}$, we consider two scenarios. First, building a random search space in $\mathcal{X}$ with volume ρ times the volume of $\mathcal{X}$ and second, building a search space centered at a point $x \in \mathcal{X}$ with approximate volume ρ times the volume of $\mathcal{X}$. In both cases the generated search space is a subset of the base search space $\mathcal{X}$.

Random search space generation   Let $\mathcal{S}$ be the search space we want to generate where $\mathcal{S} \subset \mathcal{X}$. We denote the ith dimension of $\mathcal{X}$ and $\mathcal{S}$ with $\mathcal{X}^i$ and $\mathcal{S}^i$ accordingly. For each dimension i, $i = 1, \ldots, d$, we compute the dimension-wise reduction rate as $\rho_i = \rho^{1/d}$. The length of $\mathcal{S}^i$ is denoted by $l_i = \rho_i \times \big(\max(\mathcal{X}^i) - \min(\mathcal{X}^i)\big)$. We generate a random value in $\big[\min(\mathcal{X}^i), l_i\big]$ and set it as $\min(\mathcal{S}^i)$. We have $\max(\mathcal{S}^i) = \min(\mathcal{S}^i) + l_i$ and $\mathcal{S}^i = \big[\min(\mathcal{S}^i), \max(\mathcal{S}^i)\big]$ and we repeat this procedure for each i and stack the intervals $\mathcal{S}^i$ to generate $\mathcal{S}$.

Search space generation centered at a point   Let $x \in \mathcal{X}$ denote the point we want to generate search space $\mathcal{S} \subset \mathcal{X}$ around it. For each dimension i, we calculate the interval length l_i as explained in Section A.2.2. We set $\min(\mathcal{S}^i) = \max\big(\min(\mathcal{X}^i), x^i - \frac{1}{2}l_i\big)$ and $\max(\mathcal{S}^i) = \min\big(\max(\mathcal{X}^i), x^i + \frac{1}{2}l_i\big)$. As before, $\mathcal{S}^i = \big[\min(\mathcal{S}^i), \max(\mathcal{S}^i)\big]$ and we repeat this procedure for each i and stack the intervals $\mathcal{S}^i$ which generates $\mathcal{S}$.

Note that because of the clipping of the bounds, the

volume constraint might not be precisely met given the location of x, for example when x is relatively close to the boundaries of $\mathcal{X}$. Although this might affect the volume of some of the generated search spaces in our experiments, it does not affect our experimental conclusions. Hence, with a slight misuse of terminology, we still refer to a generated search space by its *targeted* volume reduction rate.

A.2.3 Tuning details & Search Spaces

Here we specify the search spaces we have scored and ranked in the experiments Sections 4.2, 4.4 and 4.5. For all tuning problems, including tuning a WideResNet on CIFAR100, tuning a ResNet50 on ImageNet, and tuning a transformer on LM1B, we have used the same broad, seven-dimensional, base hyperparameter search space. This search space includes the base learning rate η, momentum β, learning rate decay power p, decay step factor τ, dropout rate r, l_2 decay factor λ, and label smoothing γ. See Table 1 for specific lower and upper bounds on each hyperparameter. Note that following Nado et al. (2021), we tune one minus momentum instead of momentum. Additionally, we tune the base learning rate, one minus momentum and l_2 decay factor λ over a $\log_{10}$ scale. For each hyperparameter except for the dropout rate that was sampled from a discrete set, the reparameterized hyperparameter locations were sampled uniformly at random within the scaled specified bounds. The dropout locations were sampled uniformly at random over $\{0, 0.1, 0.2, \ldots, 0.7, 0.8\}$. All the concrete search spaces we used in experiments used the same hyperparameter scaling and reparameterization choices.

Tuning a WideResNet on CIFAR100 In Section 4.2, we score three search spaces for tuning a WideResNet on CIFAR100, including the base loose search space, defined in Table 1, and two volume-constrained search spaces generated following the second search space generation strategy in Section A.2.2 with volume reduce rates $\rho \in \{0.25, 0.5\}$. Both reduced search spaces are centered around the observed hyperparameter point in $\mathcal{D}$ with the median validation error in $\mathcal{D}$. See bounds of these search spaces in Table 2 and Table 3. In Sections 4.4 and 4.5, the base loose search space is as defined in Table 1. In Section 4.5, in addition to the base loose search space, we consider two other search spaces. For each search space, all the hyperparameters except for the dropout rate r have similar bounds, scaling and reparameterization as in the base loose search space stated in Table 1. For the dropout rate, we fix it to zero for one search space and to 0.5 for another one, that is, following the notation in Section 4.5, $r \in \{0, 0.5\}$.

Tuning a ResNet50 on ImageNet In Section 4.2, we score three search spaces for tuning a ResNet50 on ImageNet, including the base loose search space, defined in Table 1, and two volume-constrained search spaces generated following the second search space generation strategy in Section A.2.2 with volume reduction factors $\rho \in \{0.25, 0.5\}$. Both reduced search spaces are centered around the observed hyperparameter point in $\mathcal{D}$ with the best (minimum) validation error amongst all validation errors in $\mathcal{D}$. Bounds of these search spaces can be found in Table 4 and Table 5. In Sections 4.4 and 4.5, the base loose search space is as defined in Table 1. In 4.5, in addition to the base loose search space, we consider two other search spaces. For each search space, all the hyperparameters except for the dropout rate r have similar bounds, scaling and reparameterization as in the base loose search space stated in Table 1. For the dropout rate, we fix it to zero for one search space and to 0.5 for another one, that is, following the notation in Section 4.5, $r \in \{0, 0.5\}$.

Tuning a Transformer on LM1B In Section 4.5, we score three search spaces for tuning a Transformer on LM1B, including the base loose search space, defined in Table 1, and two other search spaces. For each search space, all the hyperparameters except for the dropout rate r have similar bounds, scaling and reparameterization as in the base loose search space. For the dropout rate, we fix it to zero for one search space and to 0.5 for another one, that is, following the notation in Section 4.5, $rb \in \{0, 0.5\}$.

Table 1: The base loose search space including the base learning rate η, momentum β, power p, decay step factor τ, dropout rate r, l_2 decay factor λ and label smoothing γ.

HYPERPARAMETER	LOG_{10}	MIN	MAX
η	✓	-5	1
$1 - \beta$	✓	-3	0
p	-	0.1	2
τ	-	0.01	0.99
r	-	0.1	0.8
λ	✓	-6	-0.69
γ	-	0	0.4

Decay Schedule for Tuning For all the tuning problems, we used the following polynomial decay schedule for the learning rate

$$\eta_t = \frac{\eta}{1000} + \left(\eta - \frac{\eta}{1000}\right)\left(1 - \frac{\min(t, \tau T)}{\tau T}\right)^p,$$

where t denotes the current training step and T is the total number of training steps.

Table 2: The search space with approximately 0.25 of the volume of the base search space for CIFAR100 experiment in Section 4.2.

HYPERPARAMETER	LOG$_{10}$	MIN	MAX
η	✓	-5	-1.18
$1 - \beta$	✓	-3	-0.96
p	-	0.98	2
τ	-	0.01	0.73
r	-	0.1	0.53
λ	✓	-6	-3.33
γ	-	0	0.3

Table 3: The search space with approximately 0.5 of the volume of the base search space for CIFAR100 experiment in Section 4.2.

HYPERPARAMETER	LOG$_{10}$	MIN	MAX
η	✓	-5	-0.92
$1 - \beta$	✓	-3	-0.83
p	-	0.9	2
τ	-	0.01	0.78
r	-	0.1	0.56
λ	✓	-6	-2.99
γ	-	0	0.2

Table 4: The search space with approximately 0.25 of the volume of the base search space for ImageNet experiment in Section 4.2.

HYPERPARAMETER	LOG$_{10}$	MIN	MAX
η	✓	-1.81	1
$1 - \beta$	✓	-2.58	-0.11
p	-	0.1	1.37
τ	-	0.42	0.99
r	-	0	0.53
λ	✓	-6	-3.65
γ	-	0.11	0.4

Table 5: The search space with approximately 0.5 of the volume of the base search space for ImageNet experiment in Section 4.2.

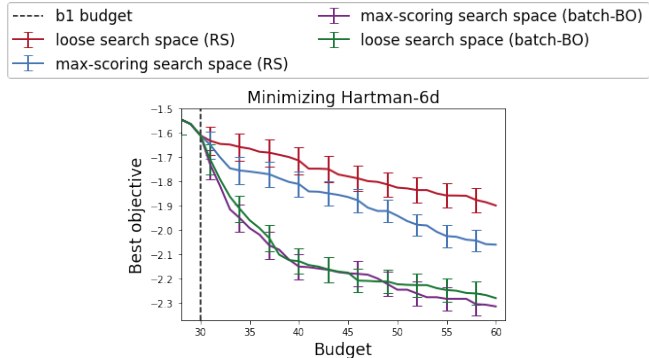| HYPERPARAMETER	LOG$_{10}$	MIN	MAX
η	✓	-2.06	1
$1 - \beta$	✓	-2.7	0
p	-	0.1	1.45
τ	-	0.37	0.99
r	-	0	0.56
λ	✓	-6	-3.42
γ	-	0.09	0.4

Figure 8: We show that minimizing the six-dimensional synthetic Hartman function over max-scoring search space versus the enclosing loose search space can achieve better results, faster for both random search and batch-BO (see Section 4.4 for more details).

A.3 Additional Results & Sensitivity Analysis for pruning bad regions of the search space

Here, we report some additional results that were omitted from the main manuscript to save space. In Section A.3.1 we report the result of pruning bad regions of the search space for the six-dimensional Hartman function. Next, in Section A.3.2 we report on the performance of variations of the scores including median-b-EI, mean-b-PI, and median-b-PI and compare them to our baseline score mean-b-EI for tuning a WideResNet on CIFAR100 and for minimizing Hartman; ultimately the different variation are quite correlated. Finally, in Section A.3.3, we analyze the sensitivity of search space pruning results for tuning WideResNet on CIFAR100 to the budget split $B = b_1 + b_2$. We show that regardless of the specific budget split (within reason), our strategy of pruning bad regions a base loose search space and running random search on the pruned space consistently outperforms running random search in the base search space.

A.3.1 Pruning bad search space regions for Hartman

Here, we report on the effect of pruning bad regions of the base loose search space, i.e. $[0, 1]^6$ for minimizing the Hartman function. Following Section A.2.2, we have generated 500 random search spaces for each volume reduction rate $\rho \in \{0.1, 0.2, \ldots, 0.9\}$. After spending b_1 budget uniformly at random within the base search space, we score all the generated search spaces and the base search space using the model trained on the b_1 points and spend the remaining b_2 in the max-scoring search space with random search/batch Bayesian optimization (See Section 4.4 for more details). We compare the performance of this procedure to two baselines, first spending the entire budget $B = b_1 + b_2$
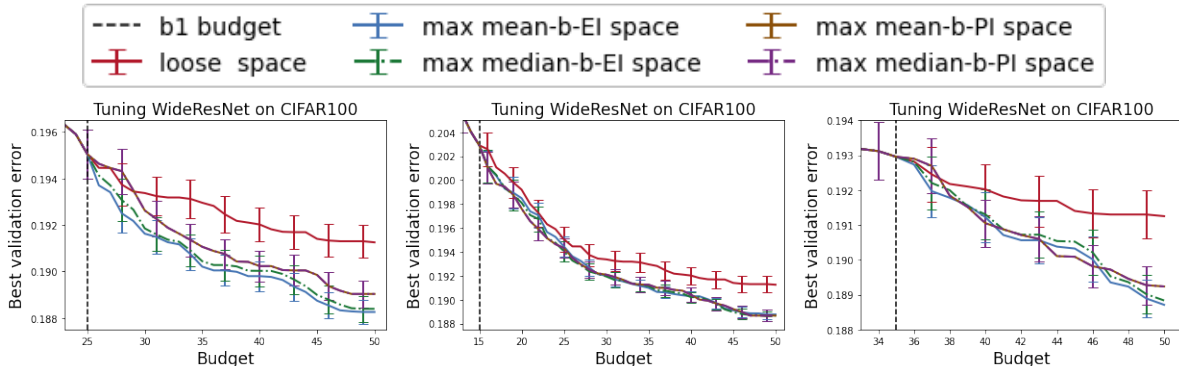
Figure 9: Comparison of the effect of different splitting budget thresholds $B = b_1 + b_2$ in Alg. 2 on tuning a WideResNet on CIFAR100. Interestingly, all budget split choices result in improvement upon the baseline. For all splitting choices, the larger the budget, the more significant the improvement becomes.

within the base search space with random search and second, spending b_1 with random search and b_2 with batch-BO over the base search space. We repeat this for 100 random initialization and report the mean performance with standard errors in Fig. 8. As portrayed in Fig. 8, pruning bad regions of the base search space improves the performance of random search/batch-BO for minimizing the Hartman function.
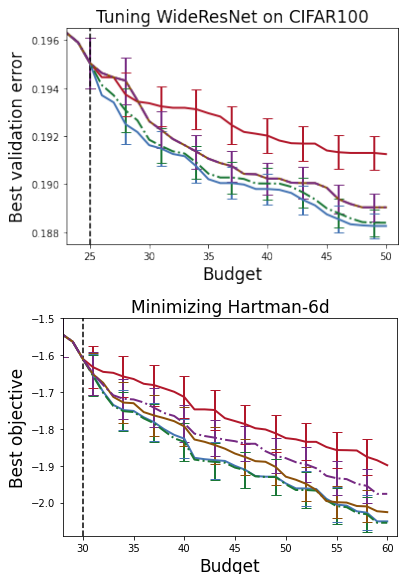

Figure 10: Comparison of random search within a max-scoring (according to different score variants) search space and the enclosing, loose search space. See Fig. 9 for the legend. Mean-b-EI and median-b-EI show almost identical performance for Hartman and very similar performance for CIFAR100. For Hartman, median-b-PI outperforms mean-b-PI while both scores perform identically for CIFAR100. For both Hartman and CIFAR100, the b-EI scores slightly outperform the b-PI scores. Overall, all the score variations outperform the baseline which shows that pruning the bad regions of the loose search space can improve the results, regardless of the pruning method details.

A.3.2 Performance of different score variants

Here, we compare the performance of different score variants including mean-b-EI, median-b-EI, mean-b-PI, median-b-PI when used for pruning bad regions of the base loose search space. We consider two problems including tuning a WideResNet on CIFAR100 and minimizing the six-dimensional Hartman function. See Section 4.4 for more details. As mentioned before, all the scores perform similarly. Naturally, given an expected utility (e.g. either b-EI or b-PI), the mean and the median scores are often strongly correlated (mean-b-EI and median-b-EI, or, similarly, mean-b-PI and median-b-PI). Given a centrality measure over the x-locations (e.g. mean or median), the scores are still correlated but the b-EI vs b-PI distinction seems to matter a bit more than the mean vs median distinction. Overall, pruning bad regions of the base loose search space using any of the scores and running random search over the max-scoring search space results in improvement upon the baseline of random search on the enclosing base search space. See Fig. 9 and Fig. 10 for an illustration.

A.3.3 The effect of budget splitting threshold for pruning bad search space regions

Finally, we show that for tuning a WideResNet on CIFAR100, our score-based pruning method is robust not only to the particular score function variant, but to the choice of how to split the total budget $B = b_1 + b_2$ between phases. Recall that Alg. 2 splits the entire available budget $B = b_1 + b_2$, spends the initial budget b_1 in the loose search space, and the remaining budget b_2 on the max-scoring search space. As shown in Fig. 9, our scoring method for the CIFAR100 tuning problem is relatively insensitive to the choice of splitting threshold in the sense that it can consistently achieve better results compared to the baseline of expending the entire budget in the enclosing, loose search space. We consider $B = 50$ and report the results over $b_1 \in \{15, 25, 35\}$,
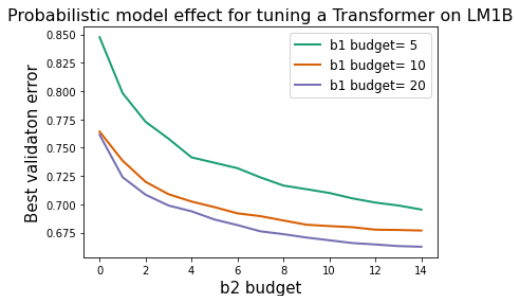
Figure 11: Comparing the impact of budget b_1 on the fitted distribution $p(y|x, \mathcal{D})$ and pruning bad regions of the search space. Larger b_1 values result in better fitted models, hence more promising max-scoring search spaces.

trying all four score variants (mean-b-EI, median-b-E, mean-b-PI and median-b-PI). Consistently, for all the scores across all the splitting thresholds, the score-guided, two-phase tuning outperforms tuning over the loose search space. This result shows the robustness of these scoring functions and their consistent capability to improve upon the baseline.

Additionally, we investigated the effect the quality of $p(y|x, \mathcal{D})$ has on the search space pruning results for tuning a Transformer model on LM1B. By varying the amount of training data for $p(y|x, \mathcal{D})$ we can explore probabilistic models of different qualities. Specifically, we vary b_1 while keeping b_2 fixed in our search space pruning experiment. As shown in Fig. 11, the larger b_1, the better the fitted distribution $p(y|x, \mathcal{D})$ is and the better the max-scoring search space will perform after the additional b_2 evaluations.