A Sub-sampling Noise Analysis

As mentioned in Section 2, many multi-task BO methods, including Fabolas, treat the training data size as a an additional hyperparameter and use the performance of one subset of training data to extrapolate to the performance over the entire dataset. To prevent bias, these methods shuffle the data before each sub-sampling step. This shuffling, as well as the fact that these methods rely on *one* subset of data for each round of training, adds further *sub-sampling noise* to the observations; this noise is non-stationary relative to the subset size, as mentioned in Klein et al. [2016a,b].

Typically, GPs in the context of BO consider a homogeneous noise which is added to the diagonal of the Gram matrix (see Eq. (1)) and is estimated either with maximum likelihood or MCMC. Klein et al. [2016b] show that the sub-sampling noise is negligible relative to this homogeneous noise for an SVM on MNIST. Here, we further investigate this for neural networks. We consider a ResNet on CIFAR-10 with four hyperparameter (see 4.2 for details). We drew 10 hyperparameter sets from a Latin hypercube design and divided the training size $\in [128, 50000]$ into 5 linearly spaced intervals.

Given a hyperparameter set x_i and the subset size s_j, we trained on x_i with 5 different randomly chosen training subsets with size s_j. Let $y_k(x_j, s_i)$ denote the validation error of each such run. We estimated the sub-sampling noise as

$$\sigma^2_{\text{subsampling}}(x_i, s_j) = \frac{1}{5}\sum_{k=1}^{5}(y_k(x_i, s_j) - \mu_{i,j})^2,$$

where $\mu_{i,j} = 1/5 \sum_k y_k(x_i, s_j)$. Moreover, using the collected data, we have estimated the noise of GP using maximum likelihood and MCMC assuming a horseshoe prior with length scale 0.1. Fig. 4 (right) shows the mean and the standard deviation (over all hyperparameter set) of the estimated noises given each subset size (mean and standard deviation are taken over the results of all hyperparameter sets at a fixed subset size). For the smallest and largest subset sizes (s=128 and s=50000), the GP noise has mostly captured the sub-sampling noise, since the validation error results are more consistent (*i.e.*, given a very small data subset, the error is likely to be high).

However, for intermediate subset sizes, the GP noise has significantly underestimated the sub-sampling noise. This shows that selected training examples can significantly affect the GP model and hence negatively impact BO performance. Hence, although relying on one randomly chosen subset of data decreases the training cost, it may also significantly hinder optimization. On the other hand, spending additional effort to select more informative training examples, as done by IBO, can benefit the overall multiround optimization process.

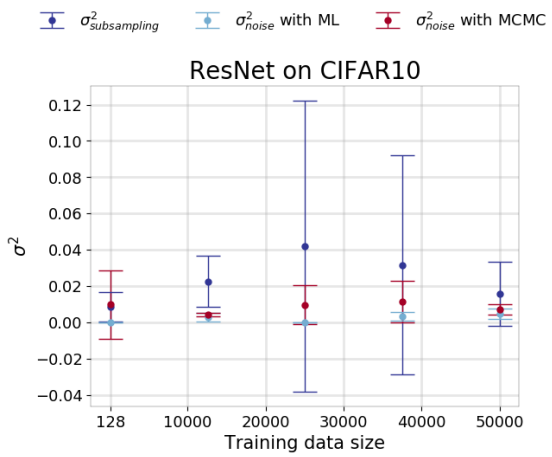

Figure 4: Estimated sub-sampling noise and GP noise on the collected validation errors using US. The sub-sampling noise is larger than the estimated GP noise particularly for intermediate sizes.

B Multi-task Kernel Design

We define the multi-task kernel for the GP that models $f_n(x_i \mid B_i)$ as

$$k^{(f)}\big((x, B), (x', B')\big) = k_X^{(f)}(x, x') \cdot k_B^{(f)}(B, B'),$$

with the sub-task kernels defined as

$$\begin{aligned} k_X^{(f)}(x, x') &= \text{Matérn}_{5/2}(x, x') \\ k_B^{(f)}(B, B') &= (1 - B)^{\nu}(1 - B')^{\nu} + 1. \end{aligned} \tag{7}$$

We choose the covariance function

$$k^{(c)}\big((x, B), (x', B')\big) = k_X^{(c)}(x, x') \cdot k_B^{(c)}(B, B'),$$

where this time we modify the kernel on B to reflect that larger B increases training time:

$$\begin{aligned} k_X^{(c)}(x, x') &= \text{Matérn}_{5/2}(x, x'), \\ k_B^{(c)}(B, B') &= B^{\lambda} B'^{\lambda} + 1. \end{aligned} \tag{8}$$

C Implementation Details

For IBO, we use task kernels $k_B^{(f)}$ (Eq. 7) and $k_B^{(c)}$ (Eq. 8), with kernel hyperparameters $\lambda = 1$ and $\nu = 2$. Following [Snoek et al., 2012], we marginalize over the GPs' hyperparameters using MCMC for all methods. To set the time budget, we fix a total number of BO iterations for each method; the time at which the fastest method completes its final iteration acts as the maximum amount of time available to any other method.

All initial design evaluations also count towards the runtime; this slightly advantages non-IS methods, which have cheaper initializations.

We report the performance of each method as a function of wall-clock time, since the methods differ in per-iteration complexity (App. 5 reports results vs. iteration number). We measure the performance of each method by taking the predicted best hyperparameter values x^* after each BO iteration, then training a model with hyperparameters x^*, using the *entire* training set and vanilla SGD. Recall that for Fabolas, Fabolas-IS, and IBO, the incumbent x^* is the set of hyperparameters with the best predicted objective on the target task (*e.g.*, using the full training data for Fabolas).

We run each method five times unless otherwise stated, and report the median performance and 25^{th} and 75^{th} percentiles (mean and standard deviation results are included in Appendix F.1 for completeness). We have included the following baselines:

- **ES**: Bayesian optimization with the entropy search acquisition function [Hennig and Schuler, 2012],
- Hyperband: the bandits-based hyperparameter tuning method introduced in [Li et al., 2016],
- **ES-IS**: BO with entropy search; inner optimization is performed using IS. For each black-box query, we draw the presample size B uniformly at random from $\{2, \ldots, 6\}\times$ batch size as prescribed in [Katharopoulos and Fleuret, 2018]; B is constant during the n rounds of SGD.
- **Fabolas-IS**: Fabolas, training with SGD-IS. For this method, a fraction s of the training data is uniformly chosen as in Fabolas, but training is performed with SGD-IS. The pre-sample batch size B is the randomly uniformly sampled in $\{2, \ldots, 6\}\times$ batch size.

All methods are initialized with 5 hyperparameters drawn from a Latin hypercube design. For IBO, we evaluate each configuration on the maximum value of its target task B. For Fabolas, Klein et al. [2016a] suggest initializing by evaluating each hyperparameter on an increasing series of task values. This aims to capture the task variable's effect on the objective. However, we empirically observed that following an initial design strategy similar to IBO's, *i.e.*, evaluating each hyperparameter on the maximum target value s, worked better in practice for both Fabolas and Fabolas-IS. This is the method we use in our experiments; App. F includes results for both initialization schemes.

For IBO, Fabolas-IS and ES-IS, we reparameterize the pre-sample size B as $B = b \times s_B$. As recommended by Katharopoulos and Fleuret [2018], we set $s_B \in [2, 6]$. For Fabolas-IS, if B is larger than the training subset size, we use the entire subset to compute the importance distribution.

In experiment 4.1, following [Dai et al., 2019], we tune six hyperparameters: number of convolutional filters $n_c \in \{128, \ldots, 256\}$, number of units in the fully connected layer $n_u \in \{256, \ldots, 512\}$, batch size $b \in \{32, \ldots, 512\}$, initial learning rate $\eta \in [10^{-7}, 0.1]$, weight decay $\beta \in [10^{-7}, 10^{-3}]$, and regularization weight $\upsilon \in [10^{-7}, 10^{-3}]$

In experiments 4.2 and 4.3, we tune four hyperparameters: initial learning rate $\eta \in [10^{-6},\ 1]$, weight decay $\beta \in [10^{-4},\ 1]$, momentum $\omega \in [0.1, 0.999]$ and L_2 regularization weight $\upsilon \in [10^{-6},\ 1]$. Following Klein et al. [2016a], all but the momentum are optimized over a log-scale search space. ES and Fabolas variations are run using RoBO. For importance sampling, we used the code provided by Katharopoulos and Fleuret [2018].

D Feed-forward Neural Network on MNIST

Our first experiment is based on a common Bayesian optimization benchmark problem [Falkner et al., 2018, Domhan et al., 2015, Hernández-Lobato et al., 2016]. We tune a fully connected neural network using RMSProp on MNIST [LeCun, 1998]. The number of training epochs n and the number of BO rounds are set to 50. We tune six hyperparameters: number of hidden layers, number of units per layer, batch size, learning rate, weight decay, and dropout rate (see App. D).

Given the well-known straightforwardness of the MNIST dataset, we do not expect to see significant gains when using importance sampling during training. Indeed, we see that all methods perform similarly after exhausting their BO iteration budget, although Fabolas does reach a low validation error slightly earlier on, since training on few data points is sufficient.

Per BO iteration (Fig. 5, last row), IBO is amongst the best performing methods gaining higher utility compared to the others. However, since performing importance sampling is expensive, IBO's performance degrades over wall-clock time. After spending roughly 30% of the time budget (around two hours), Fabolas outperforms the other methods. This is expected since Fabolas utilizes cheap approximations by using training subsets. Although such approximations are noisy, we speculate that it does not significantly harm the performance, specially for simpler datasets and models such as a feed-forward network on MNIST.

We tune six hyperparameters: number of hidden layers $n_\ell \in \{1, \ldots, 5\}$, number of units per layer $n_u \in \{16, \ldots, 256\}$, batch size $b \in \{8, \ldots, 256\}$, initial learning rate $\eta \in [10^{-7}, \ldots,\ 10^{-1}]$, weight decay

$\beta \in [10^{-7},\ 10^{-3}]$ and dropout rate $\rho \in [0,\ 0.5]$. Following [Falkner et al., 2018], the batch size, number of units, and learning rate are optimized over a log-scale.

All methods are run for 50 BO iterations. The performance is averaged over five random runs and shown in the last row of Figure 5 (median with 25 and 75 percentiles over time and iteration budget) and Figure 6 (mean with standard deviation over time).

E IBO scales with dataset and network complexity

IBO improves upon existing BO methods, moreso when tuning on large complex datasets and architectures. To illustrate, Figure 5 includes the results of all experiments over iteration budget (left column) and wall-clock time budget (right column). Moreover, the plots are sorted such that complexity of dataset and model architecture decreases along the rows; i.e., the most straightforward problem, FCN on MNIST, lies in the bottom row and the most challenging experiment, ResNet on CIFAR100 is in the top row. In the iteration plots (left column), IBO is consistently amongst the best methods (lower curve denotes better performance), achieving high utility per BO iteration. However, since doing importance sampling is inherently expensive, the advantage of IBO over wall-clock time gradually manifests once the tuning becomes more challenging. Specifically, moving from the bottom to the top, as the complexity level of tuning increases, IBO starts to outperform the rest from and earlier stage and with an increasing margin over wall-clock time (right column).

F Initializing Fabolas

Conventionally, Bayesian optimization starts with evaluating the objective at an initial set of hyperparameters chosen at random. To leverage speedup in Fabolas, [Klein et al., 2016a] suggests to evaluate the initial hyperparameters at different, usually small, subsets of the training data. In our experiments, we randomly selected 5 hyperparameters and evaluated each on randomly selected training subsets with sizes $\{\frac{1}{128}, \frac{1}{64}, \frac{1}{32}, \frac{1}{16}\}$ of the entire training data. However, our experimental results show that Fabolas achieves better results faster if during the initial design phase, the objective evaluation use the entire training data. Figure 7 illustrates this point for CNN and ResNet on CIFAR-10. Fabolas with the original initialization scheme performs 20 evaluations (5 hyperparameters each evaluated at 4 budgets) where with the new scheme, Fabolas initializes with 5 evaluations (5 hyperparameters each evaluated at 1 budget). The plots show the mean results (with standard deviation) averaged over five and three runs for CNN and ResNet. Overall, the Fabolas with new initialization achieves better average performance.

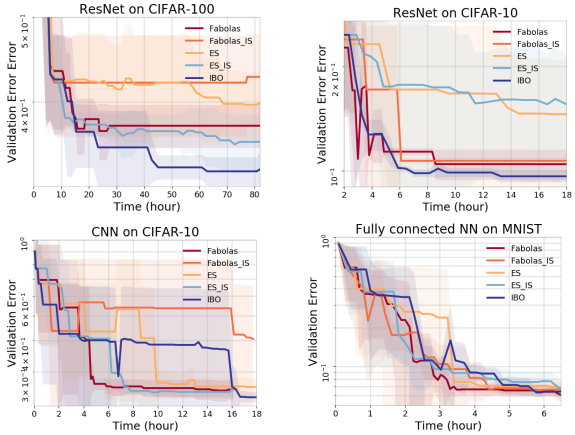

Figure 6: These plots show the mean performance (with standard deviation) of all methods for all the experiments. IBO consistently achieves amongst the lowest validation errors at the maximum budget. For CNN on CIFAR10, IBO suffers 1 relatively weak run (out of 5 total runs) which affects the mean and standard deviation . For a different perspective, see Fig. 5 reporting median and 25/75 %.

F.1 Mean and Standard Deviation Results

For completeness, we include mean and standard deviation of the results in Fig. 6.

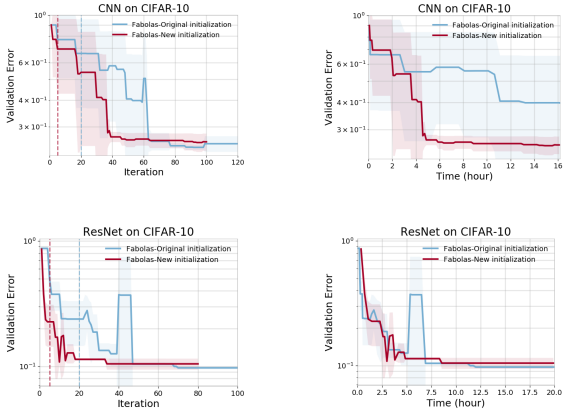

Figure 7: Comparison between two initialization schemes of Fabolas for CNN and ResNet on CIFAR-10. The dashed lines (left column) show the number of initial design evaluations for each method, immediately followed by the start of BO. We observe that with the new initial design scheme, Fabolas can potentially start progressing at a smaller iteration and a lower time, and achieve a reduced variance.

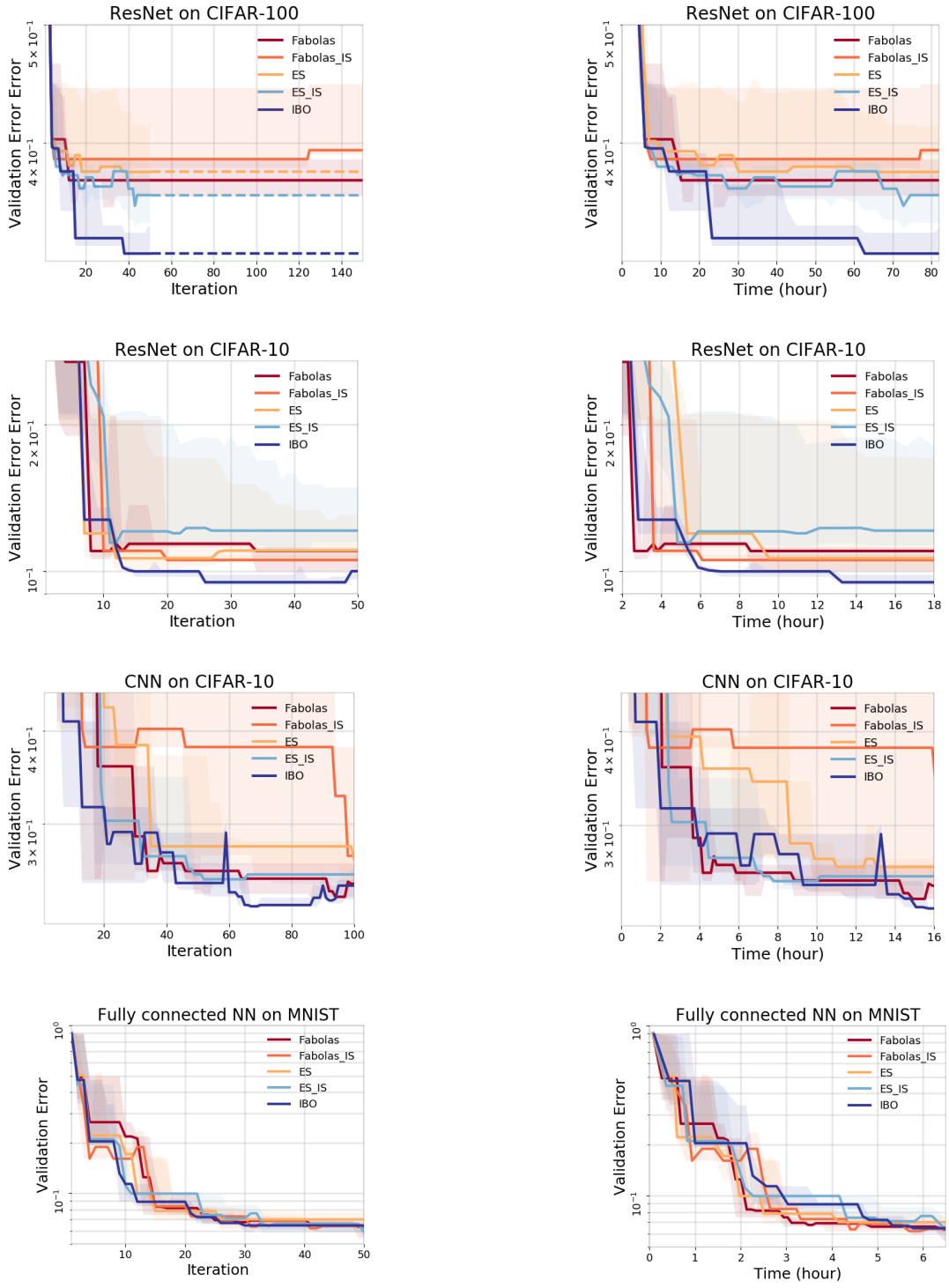

Figure 5: Average performance of all methods for all experiments as a function of both iteration budget (left column) and wall-clock time (right column). Each row represents one experiment such that the difficulty of tuning increases from the bottom row to the top i.e., the most straight-forward problem, FCN on MNIST, lies in the bottom row and the most challenging benchmark, ResNet on CIFAR100 is in the top row. In the iteration plots, IBO is consistently amongst the best methods (lower curve denotes better performance), achieving high utility per BO iteration. However, since doing importance sampling is inherently expensive, the advantage of IBO over wall-clock time gradually manifests once the tuning becomes more challenging. Specifically, IBO starts to outperform the rest earlier with an increasing margin over wall-clock time, the more difficult benchmarks become (from the bottom row to the top).