AutoSimulate: (Quickly) Learning Synthetic Data Generation

Harkirat Singh Behl[1], Atilim Güneş Baydin[1], Ran Gal[2], Philip H.S. Torr[1], and Vibhav Vineet[2]

[1] Univeristy of Oxford, Oxford, UK
{harkirat,gunes,phst}@robots.ox.ac.uk
[2] Microsoft Research, Redmond, USA
{rgal,vibhav.vineet}@microsoft.com
https://harkiratbehl.github.io/autosimulate

Abstract. Simulation is increasingly being used for generating large labelled datasets in many machine learning problems. Recent methods have focused on adjusting simulator parameters with the goal of maximising accuracy on a validation task, usually relying on REINFORCE-like gradient estimators. However these approaches are very expensive as they treat the entire data generation, model training, and validation pipeline as a black-box and require multiple costly objective evaluations at each iteration. We propose an efficient alternative for optimal synthetic data generation, based on a novel differentiable approximation of the objective. This allows us to optimize the simulator, which may be non-differentiable, requiring only one objective evaluation at each iteration with a little overhead. We demonstrate on a state-of-the-art photorealistic renderer that the proposed method finds the optimal data distribution faster (up to 50×), with significantly reduced training data generation and better accuracy on real-world test datasets than previous methods.

Keywords: synthetic data, training data distribution, simulator, optimization, rendering

1 Introduction

Massive amounts of data needs to be collected and labelled for training neural networks for tasks such as object detection [37, 15], segmentation [46] and machine translation [25]. A tantalizing alternative to real data for training neural networks has been the use of synthetic data, which provides accurate labels for many computer vision and machine learning tasks such as (dense) optical flow estimation [6, 40], pose estimation [52, 5, 54, 50, 22], among others [49, 11, 17, 35, 8, 42]. Current paradigm for synthetic data generation involves human experts manually handcrafting the distributions over simulator parameters [23, 44], or randomizing the parameters to synthesize large amounts of data using game engines or photorealistic renderers [6, 40, 42]. However, photorealistic data generation with these approaches is expensive, needs significant human effort and

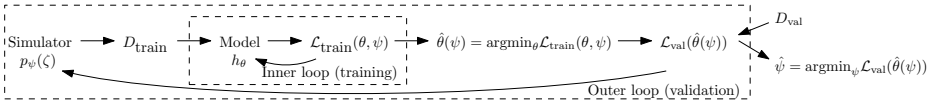


Fig. 1. Overview of the bilevel optimization setup. A simulator $p_{\boldsymbol{\psi}}(\zeta)$ is used to generate a synthetic dataset D_{train}; **inner loop**: a model $h_{\boldsymbol{\theta}}$ is then trained on this dataset with training loss $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi})$ to obtain optimal model paramaters $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$; a real-data validation set D_{val} is used to evaluate the performance of this trained model with validation loss $\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$, providing a measure of goodness of simulator parameter $\boldsymbol{\psi}$; **outer loop**: $\boldsymbol{\psi}$ is updated until we find optimal simulator parameters $\hat{\boldsymbol{\psi}}$.

expertise, and can be sub-optimal. This raises the question, has the full potential of synthetic data really been utilized?

Recent approaches [24, 12, 43, 21] have formulated the setting of simulator parameters as a learning problem. A few of these methods [24, 12] learn simulator parameters to minimize the distance between distributions of simulated data and real data. Ruiz et al.[43] proposed to learn the optimal simulator parameters to directly maximise the accuracy of a model on a defined task. However these approaches [21, 43] are very expensive, as they treat the entire data generation and model training pipeline (Figure 1, outer loop) as a black-box, and use policy gradients [53], which require multiple expensive objective evaluations at each iteration. As a result, learning synthetic data generation with photorealistic renderers has remained a challenge.

In this work, we propose a fast optimization algorithm for learning synthetic data generation, which can quickly optimize state-of-the-art photorealistic renderers. We look at the problem of finding optimal simulator parameters as a bilevel optimization problem (Figure 1) of training (inner) and validation (outer) iterations, and derive approximations for their corresponding objectives. Our key contribution lies in proposing a novel differentiable approximation of the objective, which allows us to optimize the simulator requiring only one objective evaluation at each iteration, with improved speed and accuracy. We also propose effective numerical techniques to optimize the approximation, which can be used to derive terms depending on desired speed-accuracy tradeoff. The proposed method can be used with non-differentiable simulators and handle very deep neural networks. We demonstrate our method on two renderers, the Clevr data generator [20] and the state-of-the-art photorealistic renderer Arnold [13].

2 Related Work

Expert Involvement and Random Generation One of the initial successful work on training deep neural networks on synthetic data for a computer vision problem was done on optical flow estimation, where Dosovitskiy et al. [6] created a large dataset by randomly generating images of chairs using an OpenGL pipeline by pasting objects onto randomly selected real-world images. This strategy has been applied in other problems like object detection, instance

segmentation and pose estimation [49, 17, 35, 50, 8]. Though this approach is simple to implement, the foreground objects are always pasted onto out-of-context background images, thereby requiring careful selection of the background images to achieve good accuracy as shown by Dvornik et al. [7]. Other issues with this technique include these images not being realistic, objects not having accurate shading, and shadows being inconsistent with the background.

Another line of work explores generation of photorealistic images with objects rendered within complete 3D scenes [39, 40, 19, 42, 11, 51, 14, 56]. Though this is a well accepted approach for synthetic data generation, it suffers from several issues. First, given the data generation process is independent of the neural network training, these approaches synthesize a large set of redundant training images, as shown in experiments section. This might add a massive redundant burden on the rendering infrastructure. Second, this requires human expert involvement, e.g., to set the right scene properties, material and texture of objects, quality of rendering, among several other simulator parameters [19]. This hinders widespread adaptation of synthetic data to different tasks. Finally, some sub-optimal synthesized data can corrupt the neural network training.

Learning Simulator Parameters In order to resolve these issues, recent research has focused on learning the simulator parameters. The non-differentiability of the simulators has posed a challenge for optimization. Louppe et al. [24] proposed an adversarial variational optimization technique for learning parameters of non-differentiable simulators, by minimizing the Jensen–Shannon divergence between the distribution of the synthetic data and distribution of the real data. Ganin et al. [12] incorporated a non-differentiable simulator within an adversarial training pipeline for generating realistic synthetic images.

Ruiz et al. [43] focused on optimization of simulator parameters with the objective of generating data that directly maximizes accuracy on downstream tasks such as object detection. They treated the entire pipeline of data generation and neural network training as a black-box, and used classical REINFORCE-based [53] gradient estimation. However, this approach suffers from scalability issues. A single objective evaluation involves generating a synthetic dataset, training a neural network for multiple epochs, and calculating the validation loss. And this method requires mutiple such expensive objective evaluations for taking a single step. Thus it has a very slow convergence and is difficult to scale to photorealistic simulators which have hundreds of parameters. In contrast, our method AutoSimulate, requires only a single objective evaluation at each iteration and works well with state-of-the-art photorealistic renderers. Making an assumption that a probabilistic grammar is available, Kar et al. [21] proposed to learn to transform the scene graphs within this probabilistic grammar, with the objective of simulataneously optimizing performance on downstream task and matching the distribution of synthetic images to real images. They also use REINFORCE-based [53] gradient estimation for the first objective like [43], whose limitations were discussed above.

In bi-level optimization, differentiating through neural network training is a challenge. [26] proposed to learn an approximation of inner loop using another network. Concurrent work [55] makes an assumption that the neural network is trained only for one or few iterations (not epochs), so they can store the computation graph in memory and back-propagate the derivatives, in a similar spirit as MAML [9, 1]. In constrast, we proposed a novel differentiable approximation of the inner loop using a Newton step which can handle many epochs without memory constraints. We also proposed efficient approximations which can be used for desired speed-accuracy tradeoff.

3 Problem Formulation

In supervised learning, a training set $D_{\text{train}} = \{z_1, ..., z_m\}$ of input–output pairs $z_i = (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{X} \times \mathcal{Y}$ is used to learn the parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ of a model $h_{\boldsymbol{\theta}}$ that maps the input domain $\mathcal{X}$ to the output codomain $\mathcal{Y}$. This is accomplished by minimizing the empirical risk $\frac{1}{m}\sum_{i=1}^{m} l(z_i, \boldsymbol{\theta})$, where $l(z, \boldsymbol{\theta}) \in \mathbb{R}$ denotes the loss of model $h_{\boldsymbol{\theta}}$ on a data point z.

Our goal is to generate synthetic training data using a simulator such that the model trained on this data minimizes the empirical risk on some real-data validation set D_{val}. The simulator defines a data generating distribution $p_{\boldsymbol{\psi}}(\zeta)$ given simulator parameters $\boldsymbol{\psi} \in \mathbb{R}^m$, from which we can sample training data instances $\zeta \sim p_{\boldsymbol{\psi}}(\zeta)$, where we use ζ to denote simulated data as opposed to real data z. The objective of finding optimal simulator parameters $\hat{\boldsymbol{\psi}}$ can then be formulated as the optimization problem

$$\min_{\boldsymbol{\psi}} \quad \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big) \tag{1a}$$

$$s.t. \quad \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}) \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}) \ , \tag{1b}$$

where $\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big) = \sum_{z_i \in D_{\text{val}}} l\big(z_i, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)$ is the validation loss, $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}) = \mathbb{E}_{\zeta \sim p_{\boldsymbol{\psi}}}\big[l(\zeta, \boldsymbol{\theta})\big]$ is the training loss, $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$ denote the optimum of model parameters after training on data generated from the simulator parameterised by $\boldsymbol{\psi}$, and $\hat{\boldsymbol{\psi}}$ denote the optimum simulator parameters that minimize $\mathcal{L}_{\text{val}}$. In this paper we will refer to Equations 1a and 1b as the outer and inner optimization problems respectively. This formulation is illustrated in Figure 1.

Equations 1a and 1b represent a bi-level optimization problem [4, 10, 2], which is a special kind of optimization where one problem is nested within another. To compute the gradient of the objective $\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)$ with respect to $\boldsymbol{\psi}$, one needs to propagate derivatives through the training of a model and data generation from a simulator, which is often impossible due to the simulator being non-differentiable [24]. Even in the case of a differentiable simulator, backpropagating through entire training sessions is impracticable because it requires keeping a large number of intermediate variables in memory [27]. One technique to address this challenge is to treat the entire system as a black-box and use off-the-shelf hyper-parameter optimization algorithms such as REINFORCE [53], evolutionary algorithms [30]

or Bayesian optimization [48], which require multiple costly evaluations of the objective in each iteration. An important distinction from neural network hyperparameter optimization is that evaluating the objective $\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)$ at a given $\boldsymbol{\psi}$ is much more expensive in our setting because it involves the expensive step of running the simulation for synthetic dataset generation along with neural network training.

In this paper we propose an efficient technique based on locally approximating the objective function $\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)$ at a point $\boldsymbol{\psi}$, together with an effective numerical procedure to optimize this local model, enabling the efficient tuning of simulator parameters in state-of-the-art computer vision workflows.

4 AutoSimulate

We will derive differentiable approximations of the outer and inner optimization problems (Figure 1) using Taylor expansions of the objectives $\mathcal{L}_{\text{val}}$ and $\mathcal{L}_{\text{train}}$.

Outer Problem Our goal is to find $\hat{\boldsymbol{\psi}}$, the optimal simulator parameters which minimise $\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$ in the outer (validation) problem, so we construct a Taylor expansion of $\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$ around $\boldsymbol{\psi}_t$ at iteration t as

$$\begin{aligned}\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})\big) &= \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big) + \Delta\boldsymbol{\psi}\frac{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}{d\boldsymbol{\psi}}\frac{d\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)}{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)} + ... \\ &= \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big) + \Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}}\frac{d\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)}{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)} + ... \,, \end{aligned} \tag{2}$$

where $\Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}} = \Delta\boldsymbol{\psi}\frac{d\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}{d\boldsymbol{\psi}} \approx \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + d\boldsymbol{\psi}) - \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)$.

Inner Problem To obtain parameter update $\Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}}$ for the inner (training) problem, which requires retraining on the dataset generated with the new simulator parameter $\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}$, we write the loss function $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})$ as its Taylor series approximation around the current $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$ as

$$\begin{aligned}\mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t) + \Delta\boldsymbol{\theta}, \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big) &= \mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big) \\ &\quad + \Delta\boldsymbol{\theta}^\top \frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big) \\ &\quad + \frac{1}{2}\Delta\boldsymbol{\theta}^\top \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)\Delta\boldsymbol{\theta} + ... \,, \end{aligned} \tag{3}$$

where the Hessian $\boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big) \overset{\text{def}}{=} \frac{\partial^2}{\partial\boldsymbol{\theta}^2}\mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big) \in \mathbb{R}^{n\times n}$.

We are interested in our local model in the limit $\Delta\boldsymbol{\psi} \to 0$, implying that our initial point $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)$ will be in close vicinity to the optimal $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})$. Thus we utilize the local convergence of the Newton method [32] and approximate $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})$ by the quadratic portion. Assuming the $\boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)$ is positive definite, and minimizing the quadratic portion with respect to $\Delta\boldsymbol{\theta}$,

we get

$$\Delta\hat{\boldsymbol{\theta}}_{\boldsymbol{\psi}} \approx \arg\min_{\Delta\boldsymbol{\theta}} \left(\Delta\boldsymbol{\theta}^{\top} \frac{\partial \mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)}{\partial \boldsymbol{\theta}} + \frac{1}{2} \Delta\boldsymbol{\theta}^{\top} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big) \Delta\boldsymbol{\theta} \right)$$
$$= -\boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)^{-1} \frac{\partial \mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)}{\partial \boldsymbol{\theta}} \,. \tag{4}$$

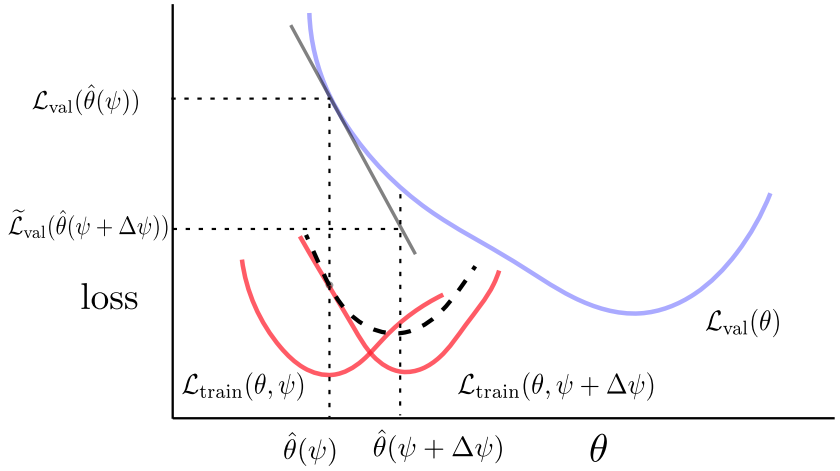


Fig. 2. Visualization of proposed differentiable approximation of objective $\mathcal{L}_{\textbf{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$ (blue). Red curves show the loss surface $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi})$ for the current training data and the loss surface $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \boldsymbol{\psi} + \Delta\boldsymbol{\psi})$ for data after a small update $\Delta\boldsymbol{\psi}$ in the simulator parameters. We assume $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi} + \Delta\boldsymbol{\psi})$ to be close to $\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})$, and use a single step of Newton's method to update $\boldsymbol{\theta}$. This gives us an approximation for updates in optimal $\hat{\boldsymbol{\theta}}$. We then use these to construct an approximation (black) for updates in optimal $\boldsymbol{\psi}$.

Differentiable Approximation Putting this back into Equation 2, and ignoring higher-order terms in $\Delta\boldsymbol{\theta}$, we get the approximation for our objective as

$$\widetilde{\mathcal{L}}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t + \Delta\boldsymbol{\psi})\big) = \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big) -$$
$$\frac{\partial \mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)}{\partial \boldsymbol{\theta}}^{\top} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t + \Delta\boldsymbol{\psi}\big)^{-1} \frac{d\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)}{d\boldsymbol{\theta}} \,, \tag{5}$$

which is equivalent to writing

$$\widetilde{\mathcal{L}}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big) = \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big) - \frac{\partial \mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)}{\partial \boldsymbol{\theta}}^{\top} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)^{-1} \frac{d\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)}{d\boldsymbol{\theta}}, \tag{6}$$

and is our local approximation of the objective. A visualization of this approximation is provided in Figure 2.

We propose to optimize our local model using gradient descent, and its derivative at point $\boldsymbol{\psi}_t$ (simulator parameter at iteration t) can be written as

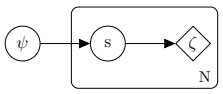
$$\frac{\partial \widetilde{\mathcal{L}}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)}{\partial \boldsymbol{\psi}}\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} = -\frac{\partial}{\partial \boldsymbol{\psi}}\left[\frac{\partial \mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)}{\partial \boldsymbol{\theta}}^{\top} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)^{-1} \frac{d\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)}{d\boldsymbol{\theta}}\right]\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}. \tag{7}$$

Using the definition of $\mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)$ and ignoring the higher order derivative $\frac{\partial}{\partial \boldsymbol{\psi}} H\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)$, we get

$$\frac{\partial \widetilde{\mathcal{L}}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)}{\partial \boldsymbol{\psi}}\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} = -\frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{\zeta\sim p_\psi}\Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]^{\top}\bigg|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t\big)^{-1} \frac{d}{d\boldsymbol{\theta}} \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big). \tag{8}$$

Next we show how to approximate the term $\frac{\partial}{\partial \boldsymbol{\psi}} \mathbb{E}_{\zeta\sim p_\psi}\Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big] \in \mathbb{R}^{m\times n}$ which requires backpropagation through the dataset generation.

4.1 Stochastic Simulator (Data Generating Distribution)



We assume a stochastic simulator that involves a deterministic renderer, which may be non-differentiable, and we make the stochasticity in the process explicit by separating the stochastic part of the simulator from the deterministic rendering. Given the deterministic renderer component $\zeta = \mathrm{r}(s)$, we would like to find the optimal values of simulator parameters $\boldsymbol{\psi}$ that parameterize $s \sim q_{\boldsymbol{\psi}}(s)$ representing the stochastic component, expressing the overall simulator as $\zeta \sim p_{\boldsymbol{\psi}}(\zeta)$. Thus we can write

$$p_{\boldsymbol{\psi}}(\zeta) = \int_{s\in\{s|r(s)=\zeta\}} q_{\boldsymbol{\psi}}(s) ds\ . \tag{9}$$

For example, lets say we want to optimize the location of an object in a scene. Then ψ could be the parameters of a Gaussian distribution $q_\psi(.)$ that is used to sample the location of the object in world coordinates, and s denotes the location of the object sampled as $s \sim q_\psi(s)$. Now this sampled location s is given as input to the renderer to generate an image ζ as $\zeta = \mathrm{r}(s)$. The overall simulator, including the stochastic sampling and the deterministic renderer, thus samples the images ζ as $\zeta \sim p_\psi(\zeta)$, where $p_\psi(\zeta)$ denotes the distribution over the images, parameterized by ψ. Therefore we get

$$\frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{\zeta\sim p_{\boldsymbol{\psi}}}\Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big] = \frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{s\sim q_{\boldsymbol{\psi}}}\Big[\frac{\partial}{\partial \boldsymbol{\theta}} l\big(r(s), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]. \tag{10}$$

Algorithm 1: AutoSimulate

for number of iterations **do**
  Sample dataset of size K: $D_{\text{train}} \sim p_{\boldsymbol{\psi_t}}(\zeta)$
  Fine-tune model for ϵ epochs on D_{train}
  Compute $\boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t\big)^{-1} \frac{d}{d\hat{\boldsymbol{\theta}}} \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)$ using CG
  Compute gradient of expectation as $\sum_{k=1}^{K} \frac{d}{d\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(s_k) . \Big[\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} l\big(r(s_k), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]^{\top}$
  Update simulator by descending the gradient
  $-\frac{\partial}{\partial \boldsymbol{\psi}} \mathbb{E}_{\zeta \sim p_{\psi}} \Big[\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} l\big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]^{\top} \Big|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t\big)^{-1} \frac{d}{d\hat{\boldsymbol{\theta}}} \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)$
end for

The gradient of expectation term for continuous distributions can be computed using REINFORCE [53] as

$$\frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{s \sim q_{\boldsymbol{\psi}}} \Big[\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} l\big(r(s), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big] = \mathop{\mathbb{E}}_{s \sim q_{\boldsymbol{\psi}}} \Big[\frac{d}{d\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(s) . \Big[\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} l\big(r(s), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]^{\top}\Big]$$
$$\approx \sum_{k=1}^{K} \frac{d}{d\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(s_k) . \Big[\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} l\big(r(s_k), \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]^{\top}, \quad (11)$$

where s_k denotes samples drawn from the distribution $q_{\boldsymbol{\psi}}$. This can be derived similarly for discrete distributions [45, 38], and we provide a derivation in the supplementary material. Therefore we can write the update rule as

$$\boldsymbol{\psi}_{t+1} \leftarrow \boldsymbol{\psi}_t + \alpha \frac{\partial}{\partial \boldsymbol{\psi}} \mathop{\mathbb{E}}_{\zeta \sim p_{\psi}} \Big[\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} l\big(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)\Big]^{\top} \Big|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \boldsymbol{H}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t\big)^{-1} \frac{d}{d\hat{\boldsymbol{\theta}}} \mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big). \quad (12)$$

It can be seen that we have transformed our original bi-level objective in Equation 1a, into iteratively creating and minimizing a local model $\widetilde{\mathcal{L}}_{\text{val}}(\hat{\boldsymbol{\theta}}\big(\boldsymbol{\psi}_t + \boldsymbol{\Delta\psi}\big)\big)$. An overview of the method can be found in algorithm 1.

4.2 Efficient Numerical Computation

The benefit of the proposed approximation is that it enables us to use techniques from unconstrained optimization. The update rule in Equation 12 requires an inverse Hessian computation at each iteration, which is common in second-order optimization. We now discuss an efficient strategy for optimizing our model.

Regularization for Hessian The first challenge is that the Hessian might have negative eigenvalues. Thus the inverse of the Hessian may not exist. We regularize the Hessian using the Levenberg method [31] and use $\boldsymbol{H} + \lambda \boldsymbol{I}$, where λ is the regularization constant and $\boldsymbol{I}$ denotes the identity matrix. This is common in second-order optimization of Neural Networks [3, 16, 29].

Inverse Hessian–vector product computation Secondly, to compute the update term in Equation 12, we split it as follows: we first compute $\boldsymbol{v}_{\boldsymbol{\psi}_t} \stackrel{\text{def}}{=}$

$\boldsymbol{H}^{-1}_{\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}\boldsymbol{g}_{\text{val}}$ and then compute $\frac{\partial}{\partial \boldsymbol{\psi}}\widetilde{\mathcal{L}}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi})\big)\Big|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} = -\boldsymbol{v}_{\boldsymbol{\psi}_t}.\nabla_{\boldsymbol{\psi}}\,\boldsymbol{g}_{\text{train}}$, where $\boldsymbol{g}_{\text{val}} \stackrel{\text{def}}{=} \frac{d}{d\boldsymbol{\theta}}\mathcal{L}_{\text{val}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)\big)$ and $\boldsymbol{g}_{\text{train}} \stackrel{\text{def}}{=} \frac{d}{d\boldsymbol{\theta}}\mathcal{L}_{\text{train}}\big(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}\big)$.

The inverse Hessian vector product $\boldsymbol{H}^{-1}\boldsymbol{g}$ is computed by using the conjugate gradient method [47] to solve $\min_{\boldsymbol{v}}\{\frac{1}{2}\boldsymbol{v}^\top\boldsymbol{H}\boldsymbol{v} - \boldsymbol{g}^\top\boldsymbol{v}\}$. This is common in second-order optimization. Thus the update term in Equation 12 can be obtained as

$$\boldsymbol{v}_{\boldsymbol{\psi}_t} \equiv \arg\min_{\boldsymbol{v}} Q(\boldsymbol{v}) \stackrel{\text{def}}{=} \{\frac{1}{2}\boldsymbol{v}^\top\boldsymbol{H}_{\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t)}\boldsymbol{v} - \boldsymbol{g}_{\text{val}}^\top\boldsymbol{v}\}\,, \tag{13a}$$

$$\boldsymbol{\psi}_{t+1} \leftarrow \boldsymbol{\psi}_t + \alpha\,\boldsymbol{v}_{\boldsymbol{\psi}_t}.\nabla_{\boldsymbol{\psi}}\boldsymbol{g}_{\text{train}}\,. \tag{13b}$$

The CG approach only requires the evaluation of $\boldsymbol{H}_{\hat{\boldsymbol{\theta}}}\boldsymbol{v}$. Using automatic differentiation, Hessian-vector product requires only one forward and backward pass, same as a gradient. In practise, a good approximation for the inverse hessian vector product can be obtained with few iterations.

Approximations for $\Delta\hat{\theta}_\psi$ We proposed a novel approximation for the solution of the inner problem. To further reduce the compute overhead, we propose approximations for $\Delta\hat{\theta}_\psi$. Table 1 shows some other alternative approximations for the inner problem, which can be obtained by using a linear approximation for the inner problem or using an approximate quadratic approximation. Using automatic differentiation, Hessian-vector product requires only one forward and backward pass, same as a gradient. Another baseline we try is a constant approximation for the inner problem where the method does not use the real validation set at all and just finds data which gives minimum model loss.

Table 1. Proposed approximations for $\Delta\hat{\theta}_\psi$.

	Approximation ($\Delta\hat{\theta}_\psi$)	Derivative Term ($\frac{\partial}{\partial\boldsymbol{\psi}}\widetilde{\mathcal{L}}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}))$)
Quadratic	$-H(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})^{-1}\frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{L}_{\text{tr.}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})$	$-\frac{\partial}{\partial\boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial\boldsymbol{\theta}}l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top\Big\|_{\boldsymbol{H}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t)^{-1}\frac{d}{d\boldsymbol{\theta}}\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))}_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}$
Approx. Quadratic	$H(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})\frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{L}_{\text{tr.}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})$	$\frac{\partial}{\partial\boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial\boldsymbol{\theta}}l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top\Big\|_{\boldsymbol{H}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi}_t)\frac{d}{d\boldsymbol{\theta}}\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))}_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}$
Linear	$-\frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{L}_{\text{tr.}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t), \boldsymbol{\psi})$	$-\frac{\partial}{\partial\boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial\boldsymbol{\theta}}l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top\Big\|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}\frac{d}{d\boldsymbol{\theta}}\mathcal{L}_{\text{val}}(\hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))$
No Val	$\mathbf{1}$	$-\frac{\partial}{\partial\boldsymbol{\psi}}\,\mathbb{E}_{\zeta\sim p_\psi}[\frac{\partial}{\partial\boldsymbol{\theta}}l(\zeta, \hat{\boldsymbol{\theta}}(\boldsymbol{\psi}_t))]^\top\Big\|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t}$

5 Experiments

In this section, we demonstrate the effectiveness of the proposed method in learning simulator parameters in two different scenarios. First we evaluate our method on a simulator with the goal of performing a per-pixel semantic segmentation task. Second, we also conduct experiments with physically based rendering for solving object detection task on real world data. In supplementary material we provide more details about the data generation process from a simulator. In

our experiments, we have used two physically based simulators: Blender-based CLEVR and the Arnold renderer.

Baselines In all our experiments, we compare our proposed method for learning simulator parameters against three state-of-the-art baseline algorithms. The main baseline is "learning to simulate" (LTS) [43] which uses the REINFORCE gradient estimator. As the code for this is not public, we implemented it. Please note that Meta-sim [21] also uses REINFORCE. In addition, we also compare against the two most established hyper-parameter optimization algorithms in machine learning; for Bayesian optimization we use the opensource Python package `bayesian-optimization` [33] and for random search we used the Scikit-learn Python library [34]. Please note that prior work [43, 21, 55] did not compare against these two approaches and in our results we found that out-of-the-box BO and Random search outperform REINFORCE [43, 21].

5.1 CLEVR Blender

In this experiment, we use the CLEVR simulator [20] which generates physically based images. The main task is semantic segmentation of the three classes present in the CLEVR benchmark, namely, Sphere, Cube and Cylinder. The images are generated using the CLEVR dataset generator. We optimize multiple CLEVR rendering parameters including the intensity of ambient light, back light, number of samples, number of bounces of light, image size, location of the objects in the scene, and materials of objects. The validation set is composed of synthetic images generated with a particular simulator configuration shown in Figure 5.

Task Network For the task network, we use a UNet [41] with eight convolutional layers. UNet is very common for segmentation and we have used an openly available Pytorch implementation[3] of UNet. The performance is measured in terms of mean IoU (intersection over union).

Results Quantitative results are shown in Table 2. BO and LTS methods to learn simulator parameters achieve similar test accuracy to ours. However, both these methods generate significantly more images to reach a similar accuracy. Essentially, to reach to the same level of test accuracy, the proposed approach requires 2.5× and 5× less data than BO and LTS methods respectively. This translates into saving time and resources required for the data generation and CNN training steps. Figure 5 shows some qualitative examples for this task.

Table 2. Segmentation on Clevr. Comparison of time, number of images, and test accuracy achieved by different methods.

Method	Time	Images	Test mIoU
REINFORCE (LTS, MetaSim)	3h2m	3,750	61.0
Random search	2h38m	2,090	60.8
BO	**43m**	960	62.9
AutoSimulate	53m	**390**	**62.9**

[3] https://github.com/jvanvugt/pytorch-unet

5.2 Photorealistic Renderer Arnold

We next evaluate the performance of our proposed method on real-world data. For this, we use LineMod-Occluded (LM-O) dataset [18] for object detection task that consists of 3D models of objects. The dataset consists of eight object classes that includes metallic, non-Lambertian objects, e.g., metallic cans. The data has recently been used for benchmarking object detection problem [19]. We use the same test split for evaluating the performance of our method. Further, we use the same simulator as Hodan et al. [19], based on Arnold [13], to generate photo-realistic synthetic data for training an object detector model. Note that Hodan et al. [19] heavily relied on human expert knowledge to correctly decide the distributions for different simulator parameters. In comparison, we show how our approach can be used to instead learn the optimal distribution over the simulator parameters without sacrificing accuracy.

In this experiment, we are given three scenes and nine locations within each scene. These locations signify the locations within the scene where objects can be placed. They can be arbitrarily chosen or can be selected by a human. Further, there are two rendering quality settings (high and low). The task is to optimize the categorical distribution for finding the fraction of data to be generated from each of these locations from different scenes under the two quality settings. Thus, the problem requires optimising 54 simulator parameters. Some of the images generated during simulator training have been shown in Figure 3.

Fig. 3. Synthetic images generated with Arnold renderer used for training.

Task Network For this task we use Yolo [36], which is an established method for object detection and the code is freely available and easy to use[4]. We use Yolo-spp which has 112 layers with default parameter setting. The object detection performance is measured in terms of mean average precision (mAP@0.50).

Results Quantitative results are provided in Table 3 where we compare the presented method against the baselines. We evaluate these methods on three

[4] https://github.com/ultralytics/yolov3

Table 3. Object Detection. Comparison of methods. We run each method for a 1,000 epochs and report: *Val. mAP*: maximum validation mAP, *Images* and *Time*: number of images generated and time spent to reach maximum validation mAP, *Test mAP*: test mAP of the result.

Method	Val. mAP	Images	Time(s)	Test mAP
REINFORCE (LTS, MetaSim)	40.2	86,150	114,360	37.2
Bayesian Optimization	39.3	9,200	83,225	37.5
Random Search	40.3	34,300	134,318	37.0
AutoSimulate	37.1	8,950	23,193	36.1
AutoSimulate(Approx Quad)	40.1	**2,950**	**2,321**	37.4
AutoSimulate (Linear)	**41.4**	17,850	30,477	**45.9**

different criteria: mAP accuracy achieved on the object detection test set, total images generated during training of the simulator, and total time taken to complete simulator training. AutoSimulate provides significant benefit over the baseline methods on all the criteria. Our method, AutoSimulate (Linear), achieves a remarkable improvement of almost 8 percent in mAP on test set. Further, it requires much lesser data (Figure 6) generation in comparison to baseline methods, and takes almost 2.5–4× less time to train simulator parameters compared to all the baselines including LTS, BO and random search. Our AutoSimulate (Approx Quad) is almost 35–60× faster than the baselines while also achieving the same mAP accuracy. This shows the effectiviness of the proposed method for learning simulator parameters. In Figure 4, we show qualitative examples of object detections in real-world images from the LM-O dataset, using a neural network trained on synthetic images generated by Arnold renderer optimized using AutoSimulate.

Fig. 4. Sample detections on real images from LM-O dataset, using a network trained on synthetic images generated by Arnold renderer, which is optimized with AutoSimulate using real-world images.

In supplementary material, we also show results on training simulator along with Faster-rcnn [37] another popular object detection model.

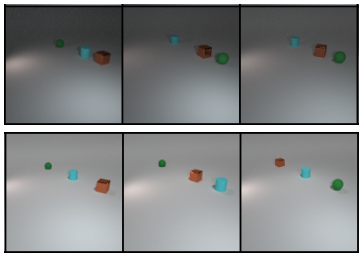


Fig. 5. Images Rendered with the Clevr Simulator. Top: samples from validation set. Bottom: images rendered during the simulator training, showing variation in the quality of images, lighting in the scene, and location of objects in scene.

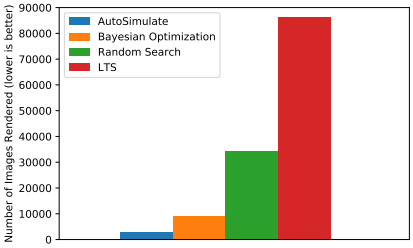


Fig. 6. Comparison of the number of synthetic images required during training using the photorealistic renderer Arnold.

5.3 Additional Studies

Approximations for θ In this ablation, we analyse the different possible approximations for $\Delta\hat{\theta}$. The quantitative results are provided in Table 4. We observe that linear approximation of $\Delta\hat{\theta}$ achieves the best test accuracy (mAP). Further, our Approximate Quadratic takes the least time to converge and requires the least amount of data generation. Thereby giving the user freedom to select the approximation based on their speed–accuracy requirements.

Table 4. Effect of Approximations

Method	Test mAP	Time(s)	Images
Exact Quadratic (Ours)	36.1	2,3193	8,950
Approximate Quadratic (Ours)	37.4	**2,321**	**2,950**
Linear (Ours)	**45.9**	30,477	1,7850
No Validation	29.3	5,539	6,400

Effect of Freezing Layers It is a common practise to train on synthetic data with the initial layers of the network frozen and trained on real data. For this ablation, we use networks pretrained on COCO dataset. The effect of freezing different numbers of layers are shown in Table 5. In particular, we show the effect of freezing 0, 98 and 104 layers out of the total 112 layers. We observe that freezing no layers achieves better accuracy than freezing layers of the CNN model. However, it leads to higher convergence time. The faster convergence of frozen layers can be attributed to fast Hessian approximation computation.

Generelization and Effect of Network Size In this ablation, we study whether a simulator trained on a shallow network generalizes to a deeper network. We first examine the effect of network depth on simulator training. In particular, we show results of using two networks: YOLO-spp with 112 layers

Table 5. Effect of Freezing Layers

	0 frozen layers			98 frozen layers			104 frozen layers		
Method	mAP	Time(s)	Images	mAP	Time(s)	Images	mAP	Time(s)	Images
REINFORCE (LTS)	37.2	114,360	86,150	33.0	114,360	86,150	31.9	145,193	104,600
Bayesian Optimization	37.5	83,225	**9,225**	31.7	13,940	3,550	31.7	30,538	6,050
Random Search	36.8	134,137	34,300	30.2	8,913	3,500	28.9	73,411	21,650
Ours	**45.9**	**30,477**	17,850	**37.1**	**2,321**	**2,950**	**35.8**	**958**	**1,000**

and YOLO-tiny with 22 layers in Table 6. Our approach on shallow network takes almost 7×, 15×, 135× less time to converge than LTS, random search and BO methods respectively. On the other hand, our method on deep network takes 4×, 2.5× and 4× less time than the three baseline methods. This highlights that the relative improvement of our method with the shallow network is much better than the deeper network. Further, in the supplementary material we also show the generalization of simulator parameters trained using shallow network on generating data for training deeper network. It gives users freedom to select size of network according to resources available for training the simulator.

Table 6. Effect of Network Size

	Yolo-spp			Yolo-Tiny		
Method	mAP	Time(s)	Images	mAP	Time(s)	Images
REINFORCE (LTS)	37.2	114,360	86,150	**24.7**	3,475	11,550
Bayesian Optimization	37.5	83,225	**9,225**	19.5	65,760	35,700
Random Search	36.8	134,137	34,300	20.6	7,319	11,620
Ours	**45.9**	**30,477**	17,850	21.2	**484**	**280**

6 Conclusion

Recent methods optimize simulator parameters with the objective of maximising accuracy on a downstream task. However these methods are computationally very expensive which has hindered the widespread use of simulator optimization for generating optimal training data. In this work, we propose an efficient algorithm for optimally generating synthetic data, based on a novel differentiable approximation of the objective. We demonstrate the effectiveness of our approach by optimising state-of-the-art photorealistic renderers using a real-world validation dataset, where our method significantly outperforms previous methods.

Acknowledgements Harkirat is wholly funded by a Tencent grant. This work was supported by ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1 and EPSRC/MURI grant EP/N019474/1. We would like to acknowledge the Royal Academy of Engineering, and also thank Ondrej Miksik, Tomas Hodan and Pawan Mudigonda for helpful discussions.

References

1. Behl, H.S., Baydin, A.G., Torr, P.H.: Alpha maml: Adaptive model-agnostic meta-learning. In: ICML, AutoML Workshop (2019)
2. Bennett, K.P., Kunapuli, G., Hu, J., Pang, J.S.: Bilevel optimization and machine learning. In: WCCI (2008)
3. Botev, A., Ritter, H., Barber, D.: Practical Gauss-Newton optimisation for deep learning. In: ICML (2019)
4. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. Annals of operations research (2007)
5. Doersch, C., Zisserman, A.: Sim2real transfer learning for 3d human pose estimation: motion to the rescue. In: NeurIPS (2019)
6. Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., Brox, T.: Flownet: Learning optical flow with convolutional networks. In: ICCV (2015)
7. Dvornik, N., Mairal, J., Schmid, C.: Modeling visual context is key to augmenting object detection datasets. In: ECCV (2018)
8. Dwibedi, D., Misra, I., Hebert, M.: Cut, paste and learn: Surprisingly easy synthesis for instance detection. In: ICCV (2017)
9. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: ICML (2017)
10. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel programming for hyperparameter optimization and meta-learning. In: ICML (2018)
11. Gaidon, A., Wang, Q., Cabon, Y., Vig, E.: Virtual worlds as proxy for multi-object tracking analysis. In: CVPR (2016)
12. Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S.M.A., Vinyals, O.: Synthesizing programs for images using reinforced adversarial learning. In: ICML (2018)
13. Georgiev, I., Ize, T., Farnsworth, M., Montoya-Vozmediano, R., King, A., Lommel, B.V., Jimenez, A., Anson, O., Ogaki, S., Johnston, E., et al.: Arnold: A brute-force production path tracer. TOG (2018)
14. Handa, A., Patraucean, V., Badrinarayanan, V., Stent, S., Cipolla, R.: Understanding real world indoor scenes with synthetic data. In: CVPR (2016)
15. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: ICCV (2017)
16. Henriques, J.F., Ehrhardt, S., Albanie, S., Vedaldi, A.: Small steps and giant leaps: Minimal newton solvers for deep learning. In: ICCV (2019)
17. Hinterstoisser, S., Lepetit, V., Wohlhart, P., Konolige, K.: On pre-trained image features and synthetic images for deep learning. In: ECCVW (2018)
18. Hodan, T., Michel, F., Brachmann, E., Kehl, W., Buch, A.G., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., et al.: BOP: Benchmark for 6D object pose estimation. In: ECCV (2018)
19. Hodaň, T., Vineet, V., Gal, R., Shalev, E., Hanzelka, J., Connell, T., Urbina, P., Sinha, S., Guenter, B.: Photorealistic image synthesis for object instance detection. ICIP (2019)
20. Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., Girshick, R.: Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In: CVPR (2017)
21. Kar, A., Prakash, A., Liu, M.Y., Cameracci, E., Yuan, J., Rusiniak, M., Acuna, D., Torralba, A., Fidler, S.: Meta-sim: Learning to generate synthetic datasets. In: ICCV (2019)

22. Kehl, W., Manhardt, F., Tombari, F., Ilic, S., Navab, N.: SSD-6D: making rgb-based 3d detection and 6d pose estimation great again. In: ICCV (2017)
23. Le, T.A., Baydin, A.G., Zinkov, R., Wood, F.: Using synthetic data to train neural networks is model-based reasoning. In: IJCNN (2017)
24. Louppe, G., Cranmer, K.: Adversarial variational optimization of non-differentiable simulators. In: AISTATS (2019)
25. Luong, M., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. In: EMNLP (2015)
26. MacKay, M., Vicol, P., Lorraine, J., Duvenaud, D., Grosse, R.: Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In: ICLR (2019)
27. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based hyperparameter optimization through reversible learning. In: ICML (2015)
28. Martens, J.: Deep learning via hessian-free optimization. In: ICML (2010)
29. Martens, J., Grosse, R.: Optimizing neural networks with kronecker-factored approximate curvature. In: ICML (2015)
30. Mitchell, M.: An introduction to genetic algorithms (1998)
31. Moré, J.J.: The levenberg-marquardt algorithm: implementation and theory. In: Numerical analysis (1978)
32. Nocedal, J., Wright, S.: Numerical optimization (2006)
33. Nogueira, F.: Bayesian Optimization: Open source constrained global optimization tool for Python (2014–), https://github.com/fmfn/BayesianOptimization
34. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. JMLR (2011)
35. Rad, M., Lepetit, V.: BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In: ICCV (2017)
36. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. In: CVPR (2017)
37. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: towards real-time object detection with region proposal networks. PAMI (2017)
38. Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: ICML (2014)
39. Richter, S.R., Hayder, Z., Koltun, V.: Playing for benchmarks. In: ICCV (2017)
40. Richter, S.R., Vineet, V., Roth, S., Koltun, V.: Playing for data: Ground truth from computer games. In: ECCV (2016)
41. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: MICCAI (2015)
42. Ros, G., Sellart, L., Materzynska, J., Vázquez, D., López, A.M.: The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In: CVPR (2016)
43. Ruiz, N., Schulter, S., Chandraker, M.: Learning to simulate. In: ICLR (2019)
44. Sakaridis, C., Dai, D., Van Gool, L.: Semantic foggy scene understanding with synthetic data. IJCV (2018)
45. Schulman, J., Heess, N., Weber, T., Abbeel, P.: Gradient estimation using stochastic computation graphs. In: NIPS (2015)
46. Shelhamer, E., Long, J., Darrell, T.: Fully convolutional networks for semantic segmentation. PAMI (2017)
47. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., USA (1994)

48. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: NIPS (2012)
49. Su, H., Qi, C.R., Li, Y., Guibas, L.J.: Render for CNN: viewpoint estimation in images using cnns trained with rendered 3d model views. In: ICCV (2015)
50. Tekin, B., Sinha, S.N., Fua, P.: Real-time seamless single shot 6d object pose prediction. In: CVPR (2018)
51. Tremblay, J., To, T., Birchfield, S.: Falling things: A synthetic dataset for 3d object detection and pose estimation. In: CVPR (2018)
52. Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M.J., Laptev, I., Schmid, C.: Learning from synthetic humans. In: CVPR (2017)
53. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning (1992)
54. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In: RSS 2018
55. Yang, D., Deng, J.: Learning to generate synthetic 3d training data through hybrid gradient. In: CVPR (2020)
56. Zhang, Y., Song, S., Yumer, E., Savva, M., Lee, J., Jin, H., Funkhouser, T.A.: Physically-based rendering for indoor scene understanding using convolutional neural networks. In: CVPR (2017)