Transformer-Based Learned Optimization

Erik Gärtner[1,2]* Luke Metz[1] Mykhaylo Andriluka[1]
C. Daniel Freeman[1] Cristian Sminchisescu[1]

[1]Google Research [2]Lund University

erik.gartner@math.lth.se
{lmetz,mykhayloa,cdfreeman,sminchisescu}@google.com

Abstract

We propose a new approach to learned optimization where we represent the computation of an optimizer's update step using a neural network. The parameters of the optimizer are then learned by training on a set of optimization tasks with the objective to perform minimization efficiently. Our innovation is a new neural network architecture, **Optimus***, for the learned optimizer inspired by the classic BFGS algorithm. As in BFGS, we estimate a preconditioning matrix as a sum of rank-one updates but use a Transformer-based neural network to predict these updates jointly with the step length and direction. In contrast to several recent learned optimization-based approaches [24, 27], our formulation allows for conditioning across the dimensions of the parameter space of the target problem while remaining applicable to optimization tasks of variable dimensionality without retraining. We demonstrate the advantages of our approach on a benchmark composed of objective functions traditionally used for the evaluation of optimization algorithms, as well as on the real world-task of physics-based visual reconstruction of articulated 3d human motion.*

1. Introduction

This work focuses on a new learning-based optimization methodology. Our approach belongs to the category of learned optimization methods, which represent the update step of an optimizer by means of an expressive function such as a multi-layer perceptron. We then learn the parameters of this function on a set of training optimization tasks. Since the update function of the learned optimizers is estimated from data, it can in principle learn various desirable behaviors such as learning-rate schedules [22] or strategies for the exploration of multiple local minima [23]. This is in contrast to traditional optimizers such as Adam [15], or BFGS [11] in which updates are derived in terms of first-principles. However, as these are general and hard-coded, they may not be able to take advantage of the regularities in the loss functions for specific classes of problems.

Learned optimizers are particularly appealing for applications that require repeatedly solving related optimization tasks. For example, 3d human pose estimation is often formulated as the minimization of a particular loss function [12, 19, 30, 46]. Such approaches estimate the 3d state (*e.g.* pose and shape) given image observations by repeatedly optimizing the same objective function for many closely related problems, including losses and state contexts. Traditional optimization treats each problem as independent, which is potentially suboptimal as it does not aggregate experience across multiple related optimization runs.

The main contribution of this paper is a novel neural network architecture for learned optimization. Our architecture is inspired by classical BFGS approaches that iteratively estimate the Hessian matrix to precondition the gradient. Similarly to BFGS, our approach iteratively updates the preconditioner using rank-one updates. In contrast to BFGS, we use a transformer-based [40] neural network to generate such updates from features encoding an optimization trajectory. We train the architecture using Persistent Evolution Strategies (PES) introduced in [41]. In contrast to prior work [4, 24, 27], which rely on updates over each target parameter independently (or coupled only via normalization), our approach allows for more complex inter-dimensional dependencies via self-attention while still showing good generalization to different target problem sizes than those used in training. We refer to our learned optimization approach as **Optimus** in the sequel.

We evaluate Optimus on classical optimization objectives used to benchmark optimization methods in the literature [17, 31, 37] (cf. fig. 1) as well as on a real-world task of physics-based human pose reconstruction. In our experiments, we typically observe that Optimus is able to reach a lower objective value compared to popular off-the-shelf

*Work done during an internship at Google.

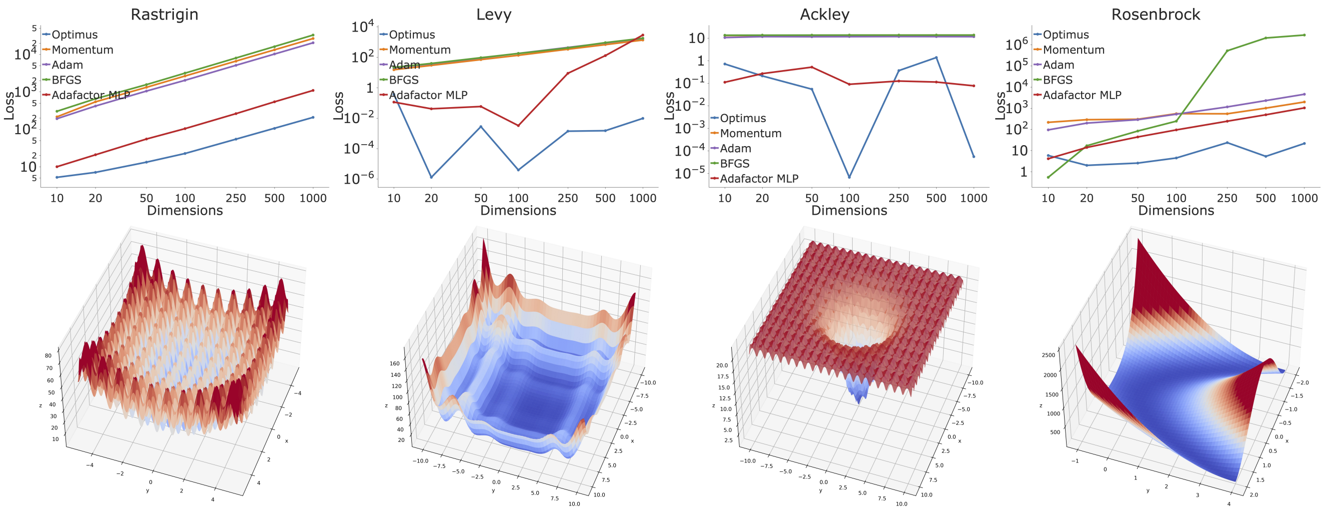

Figure 1. Top row: Evaluation results showing average objective value reached by the optimizer for the corresponding objective function in the top row (y-axis) vs. dimensionality of the objective function (x-axis). Bottom row: examples of objective functions used for evaluation of our approach. From left to right: Rastrigin [28], Levy [17], Ackley [1] and Rosenbrock [31] functions. For each function, we visualize the surface of the 2d version.

optimizers while taking fewer iterations to converge. For example, we observe at least a 10x reduction in the number of update steps for half of the classical optimization problems (see fig. 4). To evaluate Optimus in the context of physics-based human motion reconstruction, we apply it in conjunction with DiffPhy, which is a differentiable physics-based human model introduced in [12]. We experimentally demonstrate that Optimus generalizes well across diverse human motions (*e.g*. from training on walking to testing on dancing), is notably (5x) faster to meta-train compared to prior work [24], leads to reconstructions of better quality compared to BFGS, and is faster in minimizing the loss.

2. Related Work

Learned optimization is an active area of research, and we refer the reader to an excellent tutorial [3] and survey [9] for a comprehensive review of the literature. Our approach is generally inspired by [4, 18, 27, 44] and is most closely related to Adafactor MLP [24]. One of the distinguishing properties of Optimus compared to Adafactor MLP is the ability to couple optimization updates along different dimensions. Arguably coupling of dimensions can be added to Adafactor MLP through additional features such as radial features from [23] that capture pairwise interactions between dimensions. In contrast, the advantage of Optimus over such extensions is that dimension coupling is learned from data and is not limited to be pairwise. Other work incorporates conditioning in some aggregated space. For example, both [44] and [26] introduce hierarchical conditioning mechanisms that operate on individual layers and in a global setting. The Optimus approach can be seen as constructing a learned preconditioner to account for the underlying cost function curvature. The use of meta-learned curvature has been explored in [29], though only in the context of few-shot learning strategies.

There exist prior work on using learned optimization in the human pose estimation literature [35] as well as approaches that iteratively refine the solution in an optimization-like fashion based on recurrent neural networks *e.g*. [8,50]. The work of [35] is perhaps the most similar to ours in that it genuinely employs a learned optimization in the framing described in sec. 3. However, [35] applies it to the simpler problem of monocular 3d pose estimation. On that task their method converges in as few as four iterations, thus making meta-training based on stochastic gradient descent feasible. By design, the approach of [35] does not generalize to optimization instances with variable (thus different) dimensionality with respect to training, as they employ a single MLP that predicts the entire update vector. Moreover, their work has yet to be evaluated for more complex tasks such as those considered in this paper.

3. Overview and Background

In this paper, we leverage a general approach to learned optimization as introduced in [4, 6, 7, 27], which we review in sec. 3.1. Equipped with this background, we then introduce the details of our new Optimus architecture for learned optimization in sec. 4. Then in sec. 5 we present experimental results comparing Optimus to prior work in learned optimization [27] and against standard off-the-shelf optimizers.

3.1. Learning an Optimizer

Learned optimizers are a particular type of meta-learned system which commonly uses a neural network to param-

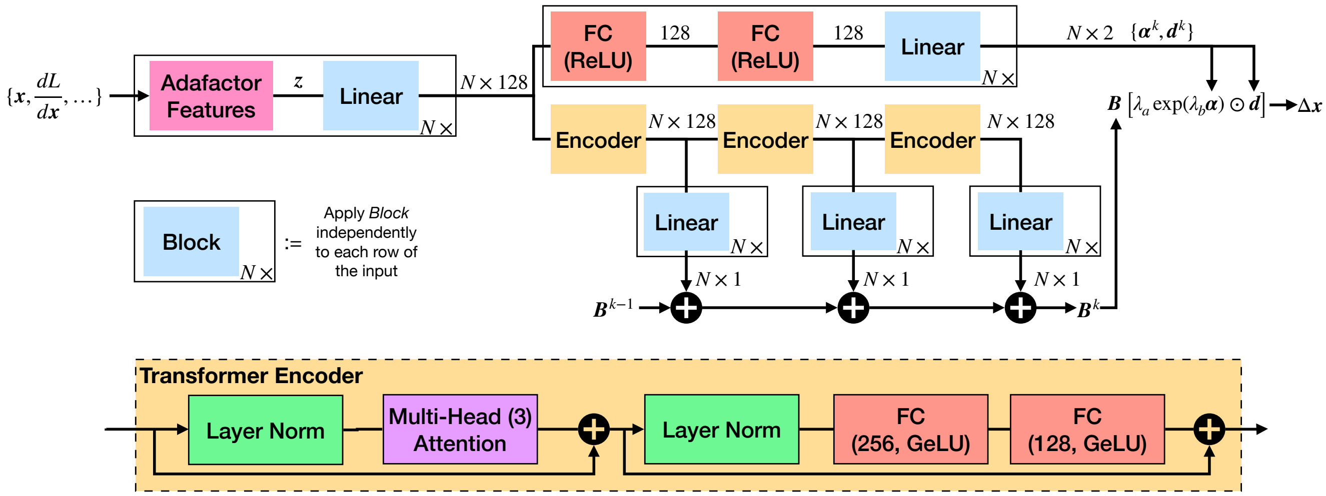

Figure 2. Schematic overview of applying our Transformer-based learned optimizer, *Optimus*. The architecture of Optimus consisting of $L = 3$ stacked Transformer encoders that predict rank one updates to a learned pre-conditioning matrix $\mathbf{B}$ and a separate branch that predicts step size $\mathbf{a}$ and step direction $\mathbf{d}$. The network uses the *Adafactor MLP* features introduced in [24] as input. Note: to allow our architecture to generalize to function of different dimensions all *linear* and *FC* layers are applied per-parameter.

eterize a gradient-based step calculation that can then be used to optimize some objective function [4]. To demonstrate this class of models, let us consider an optimization problem $\text{argmin}_{\mathbf{x}} L(\mathbf{x})$. Gradient-based optimization algorithms such as gradient descent (GD) aim to solve the problem by iteratively modifying the parameters $\mathbf{x}$ using an update function U which takes gradients of L along the optimization trajectory as input $\mathbf{x}_{t+1} = \mathbf{x}_t - U(\nabla_{\leq t} L(\mathbf{x}_{1:t}))$, where $\mathbf{x}_t$ are the parameters at step t and $\nabla_{\leq t} L(\mathbf{x}_{1:t}) = \{\nabla L(\mathbf{x}_1), \ldots, \nabla L(\mathbf{x}_t)\}$. For example, in the case of GD, the update function is simply $U_{\text{gd}}(\nabla_{\leq t} L(\mathbf{x}_{1:t})) = -\alpha \nabla L(\mathbf{x}_t)$, where α is a learning rate hyperparameter. A learned optimizer is a particular type of update function, which itself is parameterized by a set of meta-parameters $\boldsymbol{\theta}$, and with possibly more features (*e.g.*, the current parameter values). Then, as with GD, it can be iteratively applied to improve the loss.

In this paper, we build on learned optimization as proposed in [24, 27, 41]. In that approach, the update function $U = U(\mathbf{z}|\boldsymbol{\theta})$ is parameterized based on a small multilayer perceptron (MLP) with weights $\boldsymbol{\theta}$, which is applied independently to each dimension of a feature vector $\mathbf{z}$. For each parameter, the update function takes a vector of features $\mathbf{z}$ as input, including gradient information, as well as additional features such as exponential averages of squared past gradients as done in Adam [15] or RMSProp [38], momentum at multiple timescales [20], as well as factored features inspired by Adafactor [32]. We refer to this approach as Adafactor MLP in the sequel. Training an Adafactor MLP optimizer amounts to minimizing a meta-loss with respect to parameters $\boldsymbol{\theta}$ on a meta training-set of optimization problem instances. The meta-loss is given by $\sum_t L(\mathbf{x}_t)$ where the sum runs over the parameter states of the optimization trajectory. Minimizing the meta-loss is often implemented via truncated backpropagation through unrolled optimization trajectories [4, 21, 43, 44]. As discussed in [25, 27, 41] typical meta-loss surfaces are noisy and direct gradient-based optimization is difficult due to exploding gradients. To address this, we minimize the smoothed version of a meta-loss as in [27] using Adam [15] and adopt Persistent Evolution Strategies (PES) [41] to compensate for bias due to truncated back-propagation [45].

4. Our Approach

Our transformer-based learned optimizer, *Optimus*, is inspired by the BFGS [11] rank-one approximation approach to estimating the inverse Hessian, which is applied as a preconditioning matrix in order to obtain the descent direction. The parameter update is the product of a descent direction ($\mathbf{s}^k$) produced by a learned optimizer that operates on each parameter independently, and a learned preconditioner ($\mathbf{B}^k$) where $\mathbf{B}^k$ is an $N \times N$ matrix which supports conditioning over the entire parameter space. We update $\mathbf{B}^k$ with L rank-one updates on each iteration. The full update is thus given by

$$\Delta \mathbf{x}^k = \mathbf{B}^k \mathbf{s}^k, \tag{1}$$

where $\Delta \mathbf{x}^k$ is the parameter update at iteration k. See fig. 2 for an overview.

Per-Parameter Learned Descent Direction ($\mathbf{s}^k$). Let us denote a feature vector describing the optimization state of the n-th parameter at iteration k as $\mathbf{z}_n^k$. As in [24] we predict per-parameter updates using a simple MLP that takes the feature vector $\mathbf{z}_n^k$ as input and outputs a log learning rate

α_n^k and update direction d_n^k that are combined into a per-parameter update as

$$s_n^k = \lambda_a \exp(\lambda_b \alpha_n^k) d_n^k, \quad (2)$$

where $\lambda_a = 0.1$, and $\lambda_b = 0.1$ are hyperparameters which are constant throughout meta-training. Note that at that stage we *independently* predict the update direction and magnitude for each dimension of the vector $\mathbf{x}$. In particular, the MLP weights are shared across all the dimensions of $\mathbf{x}$. We use a small 4-layer MLP with 128 units per layer at that stage and did not observe improvement when with larger models (see tab. 4). We use the same features $\mathbf{z}_n^k$ as [24] and similarly to [24] normalize the features to have a second moment of 1. We include the feature list in the supplementary material for completeness.

Learned Preconditioning ($\mathbf{B}^k$). Next, we introduce a mechanism to couple the optimization process of each dimension of $\mathbf{x}$ and enable the optimization algorithm to store information across iterations. Intuitively such coupling should lead to improved optimization trajectories by capturing curvature information, similar to how second-order and quasi-Newton methods improve over first-order methods such as gradient descent. We define these updates as a low-rank update followed by normalization:

$$\tilde{\mathbf{B}}^{k+1} = \mathbf{B}^k + \sum_{l=1}^{L} \mathbf{u}_l^k (\mathbf{u}_l^k)^\top, \quad \mathbf{B}^{k+1} = \tilde{\mathbf{B}}^{k+1} \,/\, \|\tilde{\mathbf{B}}^{k+1}\|,$$

where we initialize with $\mathbf{B}^0 = \mathbf{I}^{N \times N}$. To predict the N-dimensional vectors $\mathbf{u}_l^k$ we apply a stack of L Transformer encoders [40] to a set of per-parameter features linearly mapped to $d = 128$ dimensions. Note that traditionally Transformer architecture has been applied to sequential data, whereas here we use it to aggregate information along the parameter dimensions. We visualize the architecture in fig. 2 for the case $L = 3$. The i-th element of $\mathbf{u}_l^k$ is computed by applying a layer-dependent linear mapping $\mathbf{M}_l$ to the i-th row of the output $\mathbf{E}^{kl}$ of the Transformer encoder at the layer l: $\mathbf{u}_{li}^k = \mathbf{M}_l(\mathbf{E}_{i:}^{kl})$.

Note that our formulation of the update equations for $\mathbf{B}^k$ supports several desirable properties. First, it enables coupling between updates of individual parameters through the self-attention of the encoders. Secondly, our formulation does so without making the network specific to the objective function dimensions used during training. This allows us to readily generalize to problems of different dimensionality (see sec. 5.1). Finally, it allows the optimizer to accumulate information across iterations, similarly to how BFGS [11] incrementally approximates the inverse Hessian matrix as optimization goes on. Effectively our methodology works by learning a *preconditioning* for the first-order updates estimated in other learned optimizers, such as the Adafactor MLP [24]. While this preconditioner considerably increases the step quality, its computational cost grows quadratically in the number of parameters.

Stopping Criterion. During meta-training, we unroll the optimizer for 50 steps, but at test time, we run Optimus using a stopping criterion based on relative function value decrease, as in classical optimization. We terminate the search if $f(\mathbf{x}^k) > \frac{1}{N}\sum_{i=1}^{N} \beta f(\mathbf{x}^{k-i}) + \epsilon$, *i.e.* if the function value at step k is greater than the average function value in the previous N steps. We do not apply this criterion for the first N steps. We set $N = 5$ in our experiments.

5. Experiments

We evaluate our learned optimization methodology on two tasks. We first present results of a benchmark composed of objective functions typically used for evaluation of optimization methods, and then present results for articulated 3d human motion reconstruction in sec. 5.2.

Baselines. We compare the performance of our Optimus optimizer to standard optimization algorithms BFGS [11], Adam [15], and gradient descent with momentum (GD-M). We independently tune the learning rate of Adam and GD-M for each optimization task given by objective function and input dimensionality using grid-search. To that end we test 100 candidate learning rates between 10^{-6} and 1 and choose the learning rate that results in lowest average objective value after running optimization for 64 random initializations. Finally, we also compare our approach to the state-of-the-art learned optimizer Adafactor MLP [24, 27] using the publicly available implementation[1].

5.1. Standard evaluation functions

Evaluation benchmark. We define a benchmark composed of 15 objective functions frequently used for evaluation of optimization algorithms. We show a few examples of such functions in fig. 1 and provide a full list in the supplementary material. To define the benchmark we use the catalog of objective functions available at [37], focusing on the functions that can be instantiated for any dimensionality of the input. We include both seemingly easy to optimize functions (*e.g.* "Sphere" function[2]) as well as more challenging functions with multiple local minima (*e.g.* Ackley function [1]) or difficult to find global minima located at the bottom of an elongated valley (*e.g.* Rosenbrock function [31]). We use versions of these functions with input dimensionality of $2-100$ for training, and then evaluate on the dimensions 250, 500 and 1000 to show that optimizer can generalize to different dimensionality of the input. This gives us a test set of 45 objective functions to evaluate on. To prevent the learned optimizer from memorizing the po-

[1] https://github.com/google/learned_optimization

[2] $f_{\text{sphere}}(x) = \sum_{i=1}^{d} x_i^2$, where $x \in \mathbf{R}^d$

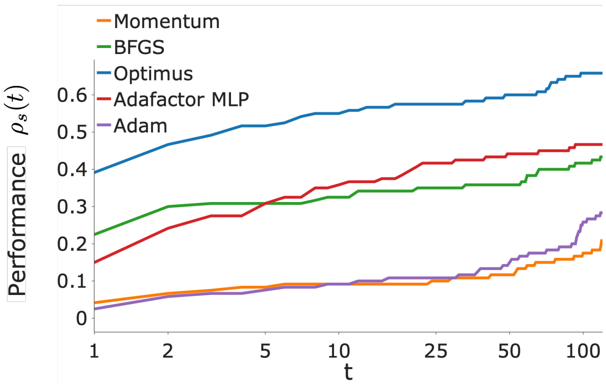

Figure 3. Comparison of optimizers using the performance profile metric from [5] that incorporates relative distance from the global minimum and relative performance of each algorithm with respect to the best algorithm (higher values on the y-axis mean better performance).

sition specific information we add random offsets to the objective functions used for training.

Evaluation metrics. We use two types of aggregate metrics in our evaluation to assess both the quality of the minima that were found, as well as the number of iterations an optimizer needed in order to reach the minimum.

We rely on *performance profiles* [2, 5, 10, 36] to compare Optimus to the baseline methods. Let $\mathcal{P}$ be the set of test problems and $\mathcal{S}$ is the set of optimizers tested. To define a performance profile one first introduces a performance measure $m_{p,s} = \frac{\hat{f}_{p,s} - f^*_p}{f^w_p - f^*_p}$, where $\hat{f}_{p,s}$ is best solution of method s for problem p, f^w_p is the worst solution out of all methods, and f^*_p is the global minimum. The measure $m_{p,s}$ is useful because if allows to compare performance of the optimizer across a set of different optimization tasks. The following ratio then compares each optimizer with the best performing one on the problem p: $r_{p,s} = \frac{m_{p,s}}{\min\{m_{p,s}: s \in \mathcal{S}\}}$, where the best solver for each problem has ratio $r_{p,s} = 1$. Given a threshold t for each optimizer s we can now compute the percentage of problems $\rho_s(t)$ for which the ratio $r_{p,s} \leq t$:

$$\rho_s(t) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : r_{p,s} \leq t\}. \tag{3}$$

Thus, the performance profile $\rho_s(t)$ is the proportion of problems a method's performance ratio $r_{p,s}$ is within a factor of t of the best performance ratio. Hence, $\rho_s(1)$ represents the percentage of tasks for which optimizer s has best performance (lowest function value) out of the tested methods. We compare the performance profiles of Optimus with our baselines in fig. 3.

Finally, we also measure the relative number of iterations Optimus needed to reach the value of the minima found by another baseline optimizer. We use Adam, BFGS and Adafactor MLP as baselines for such comparison.

Results. We observe that Optimus typically converges to a lower objective value compared to other optimizers (see

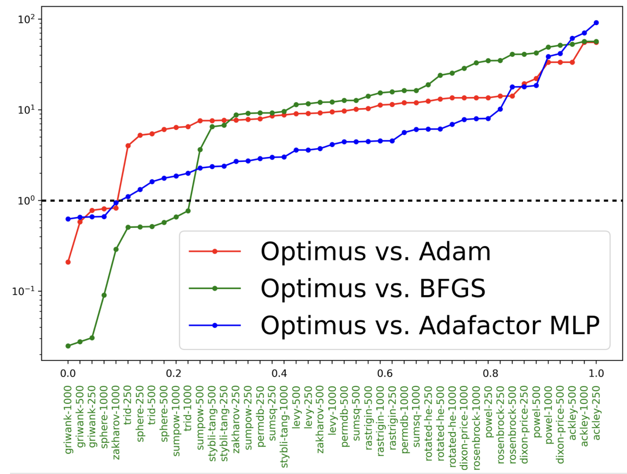

Figure 4. Comparison of Optimus with Adam, BFGS and Adafactor MLP in terms of relative number of iterations required to reach a pre-defined minimum of the objective function. Black dotted line corresponds to break-even point, all the dots above this line correspond to optimization problems where Optimus required fewer iterations than corresponding baseline. The units of the x-axis are percentiles of the total number of tasks. For illustration, we mark the points on the x-axis with the name and dimensionality of the objective function using the ordering of the Optimus vs. BFGS comparison (as indicated by the green color).

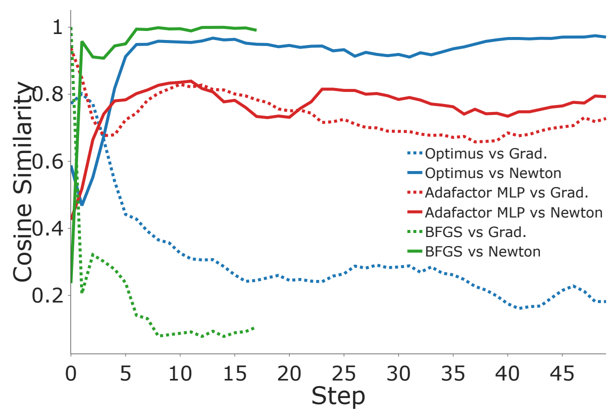

Figure 5. Left: results on evaluating Optimus trained on 2-100 dimensional Rosenbrock functions and then evaluating on 10d to 1000d Rosenbrocks. Mid: comparing the generalizing capability of Optimus and Adafactor MLP for training on different dimensions of the Rosenbrock function. Right: mean similarity between Optimus update step and gradient and Newton direction on the 2d Rosenbrock function, averaged over 64 trajectories.

fig. 1 and supplementary material). We use the performance profile metric [5] to aggregate these results across functions. Note that it is not meaningful to directly average the per-function minima since each function is scaled differently and so values of minima are not directly comparable. The performance profile metric tackles this issue by relating minima for each function to its global minima, which is known for all objective functions in our benchmark. The results are shown fig. 3. We observe that performance of Optimus is higher than other optimizers across all values of performance threshold indicating that on average Optimus gets closer to global minimum of each function compared to other optimizers.

In fig. 4 we show relative number of iterations Optimus needs to reach the same objective value as Adam, BFGS and Adafactor MLP optimizers. The target objective value in this evaluation is defined as average objective value achieved by the corresponding baseline after 100 iterations. The y-axis in fig. 4 corresponds to ratio between number of iterations required by a baseline and number of iterations required by Optimus. Values on y-axis larger than 1 indicate that Optimus required fewer iterations. The x-axis in fig. 4 is a percentile of the tasks in the benchmark. A particular point on the plot then tells us what percentage of the benchmark has a ratio between number of iterations greater or equal than a value on y axis at that point.

For example, we observe that for about 50% of the tasks Optimus requires about $10\times$ fewer iterations than Adam and BFGS and about $5\times$ fewer iterations than Adafactor MLP. In fig. 4 we label the x-axis with names of the objective functions corresponding to each point on the curve comparing Optimus to BFGS. We observe that BFGS excels on simple convex objective functions such as "Sphere" or their noisy versions such as "Griewank", where it can quickly converge to global minimum. However BFGS fails on more complex functions with multiple local minima such as "Ackley" or functions where minima is located in a flat elongated valley such as "Rosenbrock" or "Dixon-Price"[3].

Analysis of update step direction. To further highlight differences between Optimus and Adafactor MLP we plot the absolute cosine similarity of their step along the steepest descent direction given by $-\nabla \mathbf{x}$, and the Newton direction given by $-\mathbf{H}_{\mathbf{x}}^{-1}\nabla \mathbf{x}$, where $\mathbf{H}_{\mathbf{x}}^{-1}$ is the inverse Hessian at point $\mathbf{x}$. The results are shown in fig. 5 for optimization of the 2d Rosenbrock function. For clarity we also include the same similarity plot for BFGS. As expected, the direction of BFGS step becomes similar to Newton after a few iterations since the preconditioner in BFGS approximates the inverse Hessian. Note that overall the direction of the Optimus step is much closer to Newton compared to Adafactor MLP and much less similar to steepest descent. This supports the intuition that Optimus's design extends the learned optimizer with a preconditioner similar to BFGS.

5.2. Physics-Based Motion Reconstruction

In principle, learned optimization can be applied to any problem that is solvable by means of local descent, and thus to any physics-based reconstruction that formulates human motion reconstruction as loss minimization [12, 19, 46, 49]. In this paper, we build on the DiffPhy approach of [12], who define a differentiable loss function for physics-based reconstruction. DiffPhy relies on the differentiable implementation of rigid body dynamics in [13] and shape-specific body model based on [47] to define a loss function that measures similarity between simulated motion and observations. The observations are either a set of 3d keyframe poses, when the goal is to reproduce articulated 3d motion in physical simulation, or a sequence of image measurements such as 2d image keypoints or estimated 3d poses in each frame. The physical motion in DiffPhy is parameterized via a control trajectory, which is given by a sequence of quaternions defining a target rotation of each body joint over time. The control trajectory implicitly defines the torques applied to each body joint via PD-control. Please refer to [12] for a more extensive explanation of the loss function. Hence, optimization, in this case, aims to infer the control trajectory that re-creates an articulated motion in physical simulation in a way that is close to the given observations and consistent with constraints (*e.g.* lack of footskate and non-intersection with respect to a ground plane).

Datasets. In our human motion experiments, we use the popular Human3.6M articulated pose estimation benchmark [14]. We follow the protocol introduced in [34] to compare to related work for video-based experiments[4]. For experiments with motion capture inputs, we rely the same subset of 20 validation sequences from Human3.6M as used in [12]. We then additionally evaluate on the dancing sequences from the AIST dataset [39] used in [12] to further evaluate the generalization of our approach across qualitatively different motion types.

Evaluation metrics. We report the standard 2d and 3d human pose metrics as well as physics-based metrics. The mean per-joint 3d position error (MPJPE-G), the per-frame translation-aligned error (MPJPE), and the per-frame Procrustes-aligned error (MPJPE-PA) are reported in millimeters. In addition, we measure the 2d keypoint error of the reconstruction (MPJPE-2d) in pixels. Finally, we measure the amount of motion jitter as the total variation in 3d joint acceleration (TV) defined as $\frac{1}{T}\sum_{t\in T}\sum_{k\in K}|\ddot{x}^k_{t+1} - \ddot{x}^k_t|$, where $\ddot{x}^k_t$ is the 3d acceleration of joint k at time t, as well as the percentage of frames exhibiting *foot skate* as defined by prior work [12].

Model training. We train Optimus and Adafactor MLP on the task of minimizing the DiffPhy loss, which measures similarity between the simulated motion and 3d poses estimated based on [50]. Following an initial grid-search to determine the best learning rate for each model, we train these for 10 days using 500 CPUs to generate training batches of optimization rollouts (128 roll-outs of length 50 per batch).

[3] Please see supplementary material for the description of the objective functions.

[4] The protocol excludes motions such as "sitting" and "eating" which require modeling human-object interactions.

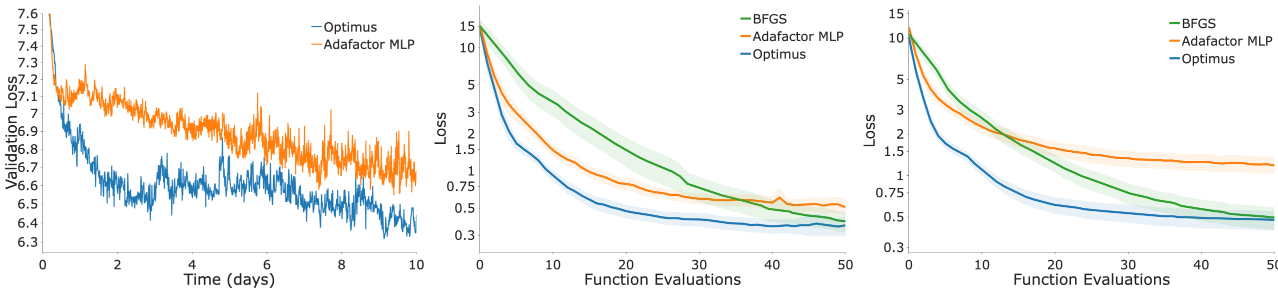

Figure 6. Left: comparison of validation loss during training of Optimus and Adafactor MLP. Note how Optimus converges to lower validation loss much faster than the Adafactor MLP model. Mid: comparison of loss curves during optimizing on "in domain" examples in tab. 1. Right: loss curves during optimizing of "out of domain" examples in tab. 1. Note how Optimus generalizes better than Adafactor MLP on out of domain data and minimizes the loss faster than BFGS. Shaded area denotes 95% confidence interval.

Method	MPJPE-G (in domain)	MPJPE-G (out of domain)	# Func. Evals
DiffPhy + BFGS [12]	38.3	24.7	71
+ AdaFactor MLP [24]	33.0	29.3	50
+ Optimus (Ours)	24.0	25.0	50

Table 1. Comparison of different approaches to trajectory optimization on motion capture data from our Human3.6M validation set.

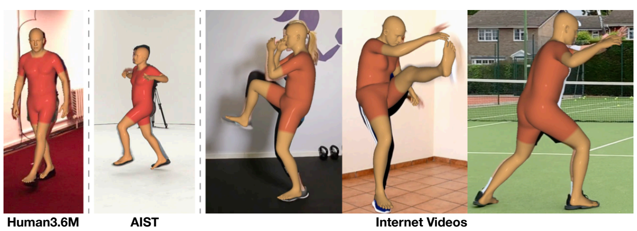

Figure 7. Qualitative examples of video reconstructions by Optimus. We train Optimus on the Human3.6M [14] dataset (left). We then evaluate its performance on the dancing sequence from AIST [39] (see tab. 3) and qualitatively verify that Optimus is applicable to in-the-wild internet videos (rightmost images).

We show loss vs. training time curves on the validation set in fig. 6 (left). Note that Optimus generally converges much faster and to a lower loss value compared to competitors. For example, after 48 hours of training, Optimus has essentially converged to a loss of 6.49 whereas Adafactor MLP requires nearly 240 hours to reach a loss value of 6.55.

Results. As our first experiment in tab. 1 we compare the performance of Optimus to Adafactor MLP and BFGS using motion capture (mocap) data as input. In fig. 6 (middle and right) we additionally visualize how quickly the loss is reduced by the optimizer at each iteration. Note that iterations of BFGS and of the learned optimizers are not comparable in terms of computational resources because BFGS might evaluate the objective function more than once during line search. To compensate for different iteration costs, in fig. 6 we plot the number of objective function evaluations instead of the optimization iteration along the x-axis. In this experiment we train all models on walking sequences from Human3.6M dataset [5]. We refer to walking sequences as "in domain" in fig. 6 and tab. 1. We then assess performance on a validation set of "in domain" and "out of domain" sequences corresponding to motions other than walking. Note that Optimus improves over other approaches in the "in domain" setting and reaches the loss comparable to BFGS at roughly half of loss function evaluations (*cf*. fig. 6 (middle)). In the "out of domain" setting Optimus reaches nearly the same accuracy compared to BFGS (24.7 vs 25mm.) but again converges much faster. In contrast Adafactor MLP does not improve over BFGS on the "out of domain" motions and converges to a higher loss (*cf*. fig. 6 (right)).

In the second experiment, we evaluate performance of our best model DiffPhy+Optimus for video-based human motion reconstruction. Note that Optimus performs well on the dancing sequences from AIST even though it has been trained only on walking data from Human3.6M (150.2 for DiffPhy vs. 149.8 mm MPJPE-G for Optimus) and that Optimus is able to handle mocap and video inputs without retraining. We observe similar results on Human3.6M, where Optimus performs on par with BFGS (138.6 vs. 139.1 MPJPE-G) while requiring roughly half as many function evaluations (88 for DiffPhy vs. 40 for Optimus). We show a few qualitative examples for Optimus results on Human3.6M, AIST and internet videos in fig. 7.

Ablation experiments. We now evaluate how different design choices affect the model performance. Due to high computational load, we train each model for up to 48 hours. The results are shown in tab. 4. We refer to a version of Optimus that does not incrementally update the matrix $\mathbf{B}^k$ in sec. 4 and instead predicts it from scratch at each iteration

[5] We use "Walking" and "WalkTogether" activities.

Model	MPJPE-G	MPJPE	MPJPE-PA	MPJPE-2d	TV	Foot skate
VIBE [16]	207.7	68.6	43.6	16.4	0.32	27.4
PhysCap [34]	-	97.4	65.1	-	-	-
SimPoE [48]	-	**56.7**	**41.6**	-	-	-
Shimada et al. [33]	-	76.5	58.2	-	-	-
Xie et al. [46]	-	68.1	-	-	-	-
DiffPhy [12]	139.1	82.1	55.9	13.2	0.21	7.2
Optimus (Ours)	**138.6**	82.8	57.0	13.2	**0.20**	**6.5**

Table 2. Quantitative results on Human3.6M [14] comparing our model to prior methods.

Model	MPJPE-G	MPJPE	MPJPE-PA	MPJPE-2d	TV	Foot skate
DiffPhy [12]	150.2	105.5	66.0	12.1	0.44	19.6
Optimus	149.8	104.4	66.4	12.1	0.45	21.5

Table 3. Quantitative results on generalizing to a new dataset. Optimus was trained Human3.6M [14] and is here evaluated on a subset of the dance motion dataset AIST [39].

Variant	MPJPE-G	Loss	# Params	Runtime (ms)
Optimus	33.1	0.534	832,526	78.9
Optimus, no state	36.5	0.675	832,526	115.1
Optimus, no structure	37.3	0.721	832,139	10.7
Adafactor MLP [24] 512x4	40.0	0.723	811,531	3.7
Adafactor MLP [24] 128x4	40.6	0.811	22,411	2.6
Optimus	33.1	0.534	832,526	78.9
Optimus, 50 iterations	28.5	0.383	832,526	78.9
Optimus + BasinHopping [42]	25.6	0.352	832,526	78.9

Table 4. Top: ablation of model components on motion capture from our Human3.6M [14] validation set. Bottom: comparison of different test time operating modes.

as "Optimus, no state", and a version that uses a transformer to directly predict $\Delta \mathbf{x}^k$ from $\mathbf{s}^k$ in Eq. (1) as "Optimus, no structure". Our full Optimus improves over Adafactor MLP by 17.2% and over simpler "Optimus, no structure" by 11.3% while having nearly the same number of parameters. Note that simply adding more parameters to Adafactor MLP barely improves results (40.6 vs. 40.0mm MPJPE-G). Optimus makes 30.3 function evaluations before optimization is terminated by the stopping criterion. In tab. 4 (bottom three lines) we evaluate the effect of running Optimus for a fixed number of 50 function evaluation without stopping criterion and the effect of using even more costly BasinHopping [42] optimization that adds random perturbations after a fixed number of iterations (to improve global exploration) requiring 82.3 function evaluations on average. We observe that running Optimus for longer leads to considerably improved results, at higher computational load (33.1 for Optimus vs. 25.6 mm MPJPE-G for Optimus+BasinHopping).

6. Conclusion

We have introduced a learned optimizer, **Optimus**, based on an expressive architecture that can capture complex dependency updates in parameter space. Furthermore, we have demonstrated the effectiveness of Optimus for the real-world task of physics-based articulated 3D motion reconstruction as well as on a benchmark of classical optimization problems. While Optimus's expressive architecture outperforms simpler methods such as Adafactor MLP, the expressiveness comes at an increased computational cost. As a result, Optimus is best suited for tasks where the loss function dominates the computational complexity of optimization (*e.g.*, physics-based reconstruction) but might be less suited for applications where the computation of the loss function is fast (*e.g.* training neural networks). In future work, we hope to address this limitation by learning factorizations of the estimated prediction matrix.

Ethical Considerations. We aim to improve the realism and quality of human pose reconstruction by including physical constraints. By amortizing the computation through learning from past instances, we hope to reduce the long-term computational demand of these methods. We believe that our physical model's level of detail (e.g. lack of photorealistic appearance) limits its applications in adverse tasks such as person identification or deepfakes. Furthermore, the model is inclusive in supporting a variety of body shapes and sizes, and their underlying physics.

References

[1] D.H Ackley. *A Connectionist Machine for Genetic Hillclimbing*, volume SECS28. Kluwer Academic Publishers, Boston, 1987. 2, 4

[2] M Montaz Ali, Charoenchai Khompatraporn, and Zelda B Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of global optimization*, 31(4):635–672, 2005. 5

[3] Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *arXiv preprint arXiv:2202.00665*, 2022. 2

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016. 1, 2, 3

[5] Vahid Beiranvand, Warren Hare, and Yves Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848, 2017. 5

[6] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992. 2

[7] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1990. 2

[8] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4733–4742, 2016. 2

[9] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021. 2

[10] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002. 5

[11] Roger Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, NY, USA, 1987. 1, 3, 4

[12] Erik Gärtner, Mykhaylo Andriluka, Erwin Coumans, and Cristian Sminchisescu. Differentiable dynamics for articulated 3d human motion reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 1, 2, 6, 7, 8

[13] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. NeuralSim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. 6

[14] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014. 6, 7, 8

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1, 3, 4

[16] Muhammed Kocabas, Nikos Athanasiou, and Michael J. Black. Vibe: Video inference for human body pose and shape estimation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2020. 8

[17] Manuel Laguna and Rafael Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33:235–255, 2005. 1, 2

[18] Ke Li and Jitendra Malik. Learning to optimize. In *ICLR*, 2017. 2

[19] Zongmian Li, Jiri Sedlar, Justin Carpentier, Ivan Laptev, Nicolas Mansard, and Josef Sivic. Estimating 3d motion and forces of person-object interactions from monocular video. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 6

[20] James Lucas, Shengyang Sun, Richard Zemel, and Roger Grosse. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018. 3

[21] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015. 3

[22] Niru Maheswaranathan, David Sussillo, Luke Metz, Ruoxi Sun, and Jascha Sohl-Dickstein. Reverse engineering learned optimizers reveals known and novel mechanisms. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 19910–19922. Curran Associates, Inc., 2021. 1

[23] Amil Merchant, Luke Metz, Samuel Schoenholz, and Ekin Dogus Cubuk. Learn2hop: Learned optimization on rough landscapes. In *International Conference on Machine Learning*, pages 8661–8671. PMLR, 2021. 1, 2

[24] Luke Metz, C Daniel Freeman, James Harrison, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Practical tradeoffs between memory, compute, and performance in learned optimizers. *arXiv preprint arXiv:2203.11860*, 2022. 1, 2, 3, 4, 7, 8

[25] Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. Gradients are not all you need, 2021. 3

[26] Luke Metz, Niru Maheswaranathan, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020. 2

[27] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565, 2019. 1, 2, 3, 4

[28] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. 17(6–7):619–632, Sept. 1991. 2

[29] Eunbyung Park and Junier B Oliva. Meta-curvature. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[30] Davis Rempe, Leonidas J. Guibas, Aaron Hertzmann, Bryan Russell, Ruben Villegas, and Jimei Yang. Contact and human dynamics from monocular video. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 1

[31] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960. 1, 2, 4

[32] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018. 3

[33] Soshi Shimada, Vladislav Golyanik, Weipeng Xu, Patrick P'erez, and Christian Theobalt. Neural monocular 3d human motion capture with physical awareness. *ACM Transactions on Graphics (TOG)*, 40:1 – 15, 2021. 8

[34] Soshi Shimada, Vladislav Golyanik, Weipeng Xu, and Christian Theobalt. Physcap: Physically plausible monocular 3d motion capture in real time. *ACM Transactions on Graphics*, 39(6), dec 2020. 6, 8

[35] Jie Song, Xu Chen, and Otmar Hilliges. Human body model fitting by learned gradient descent. In *European Conference on Computer Vision*, pages 744–760. Springer, 2020. 2

[36] Roman Strongin and Yaroslav Sergeyev. *Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms*. Springer US, New York, NY, USA, 2000. 5

[37] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved November 9, 2022, from http://www.sfu.ca/~ssurjano. 1, 4

[38] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. 3

[39] Shuhei Tsuchida, Satoru Fukayama, Masahiro Hamasaki, and Masataka Goto. Aist dance video database: Multi-genre, multi-dancer, and multi-camera database for dance information processing. In *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019*, pages 501–510, Delft, Netherlands, Nov. 2019. 6, 7, 8

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 4

[41] Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10553–10563. PMLR, 18–24 Jul 2021. 1, 3

[42] David J. Wales and Jonathan P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997. 8

[43] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 3

[44] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *International Conference on Machine Learning*, 2017. 2, 3

[45] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018. 3

[46] Kevin Xie, Tingwu Wang, Umar Iqbal, Yunrong Guo, Sanja Fidler, and Florian Shkurti. Physics-based human motion estimation and synthesis from videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11532–11541, October 2021. 1, 6, 8

[47] Hongyi Xu, Eduard Gabriel Bazavan, Andrei Zanfir, William T Freeman, Rahul Sukthankar, and Cristian Sminchisescu. GHUM & GHUML: Generative 3d human shape and articulated pose models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6184–6193, 2020. 6

[48] Ye Yuan, Shih-En Wei, Tomas Simon, Kris Kitani, and Jason Saragih. Simpoe: Simulated character control for 3d human pose estimation. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 8

[49] Andrei Zanfir, Eduard Gabriel Bazavan, Hongyi Xu, Bill Freeman, Rahul Sukthankar, and Cristian Sminchisescu. Weakly supervised 3d human pose and shape reconstruction with normalizing flows. *arXiv preprint arXiv:2003.10350*, 2020. 6

[50] Andrei Zanfir, Eduard Gabriel Bazavan, Mihai Zanfir, William T Freeman, Rahul Sukthankar, and Cristian Sminchisescu. Neural descent for visual 3d human pose and shape. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 6