Transformer-Based Learned Optimization
Supplementary Material

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

In this supplementary material, we include additional results and visualizations (sec. 1), describe details of our implementation of Optimus (sec. 2), list the input features (sec. 3), summarize the hyperparameters of our method (sec. 4) and include the details of the datasets used in the paper (sec. 5).

1. Additional visualizations

The following section provides additional visualizations of the experiments in the main paper.

Results on standard evaluation functions. We evaluate Optimus on a benchmark of 15 classical optimization test functions. Fig. 5 and tab. 4 provides detailed per-function results. These results were then aggregated to calculate the performance profile (fig. 3 in the main paper).

Comparison of optimizer efficiency. Fig. 4 shows the efficiency of Optimus on N-dimensional Rosenbrock functions and compares it to other optimization algorithms. Note that Optimus achieves better loss values while using significantly less compute than BFGS and Adafactor MLP. For $N = 1000$, Optimus converges after $\sim$60 function evaluations achieving a loss value $\sim$5 while BFGS achieves a loss of $\sim 10^6$ using $\sim$240 function evaluations. The plot was generated using the same Rosenbrock optimization trajectories as presented in the main paper.

Runtime Comparison. In fig. 1 we plot the runtime of the optimization step for Optimus and other optimization methods on the task of finding minima of N-dimensional Rosenbrock functions while varying N between 2 and 1000. In this comparison we use a widely adopted implementation of BFGS from the SciPy [15] package. Both Optimus and BFGS [4] have $\mathcal{O}(N^2)$ time complexity. Interestingly for large N Optimus optimization step is faster than BFGS, which is likely to be due to its more efficient Jax [3] implementation.

*Work done during an internship at Google.

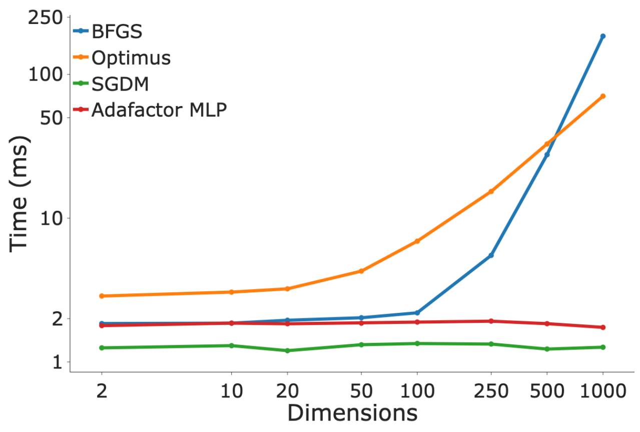


Figure 1. Time required to compute the optimizer update step as a function of dimensionality of the objective function.

Additional Optimization Trajectories. Fig. 6 presents additional visualizations of optimization trajectories on N-dimensional Rosenbrock functions. Note how Optimus tends to vary its step size throughout the trajectories while Adafactor MLP tends to monotonically decrease its step size. We have observed that Adafactor MLP generally tends to learn a learning rate schedule based on the current iteration (a feature to the networks) and decrease its step size monotonically.

2. Implementation details

The Optimus algorithm is shown in alg. 1 and the corresponding neural network architecture is shown in fig. 2.

Training. We implement Optimus in Jax [3] using Haiku [7] and the *learned_optimization*[1] framework. When training Optimus on human motion reconstruction task we using a batch size of 20. We generate the batches by sampling

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

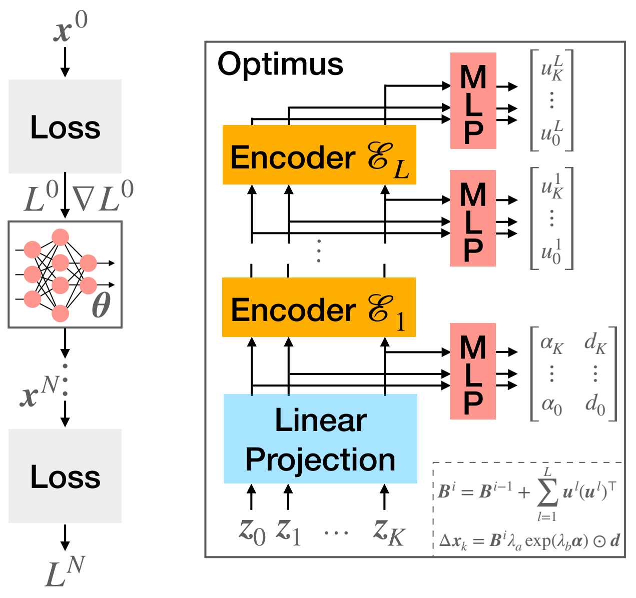


Figure 2. Left: schematic overview of applying our Transformer-based learned optimizer, *Optimus*, to iteratively minimize the loss L. Right: architecture of Optimus consisting of L stacked Transformer encoders that predict the parameter update $\Delta \mathbf{x}_k$ given the feature vector $\mathbf{z}_k$ consisting of the associated gradient $\frac{dL}{d\mathbf{x}_k}$ together with the features from [10].

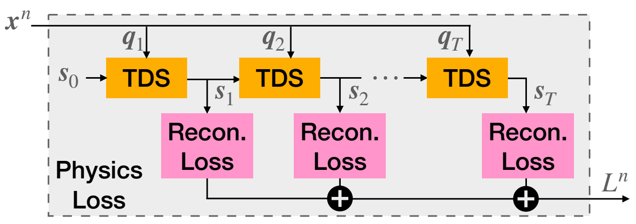


Figure 3. Overview of the physics loss introduced in [5] where the optimization variable $\mathbf{x}$ corresponds to joint torques of a physically simulated human with the goal of minimizing a pose reconstruction loss between the physical character and observed visual evidence.

random windows from the Human3.6M [8] training set, then rolling out the optimization for 50 steps, and computing the PES [14] gradients every 5 steps using 2 antithetic samples. We learn the weights using Adam [9] with the learning rate of 5×10^{-4}. To stabilize the training we perform gradient clipping to gradients with a norm greater than 3.

2.1. Distributed Physics Loss

We use the differentiable physics loss introduced in [5] using the Tiny Differentiable Simulator [6] (TDS) that was implemented in C++. A schematic overview of the loss function is available in fig. 3, however, see [5] for an in-depth explanation. The gradients of the loss are computed using the automatic differentiation framework CppAD [2]. We wrap the simulation step function as a custom TensorFlow [1] function to enable easier integration with Jax when training Optimus. We sample rollouts from 400 distributed servers exposing the loss function using the Courier[2] framework. We do this to overcome the issue of the slow evaluation time of the physics loss.

3. Input features

In the following we list the features $\mathbf{z}$ used as input by Optimus. The features are identical to the features used in [10] and we list them here to make the paper more self-contained.

- the parameter values
- the 3 momentum values (m)
- the second moment value (v)
- 3 values consisting of momenta normalized by rms gradient norm $-m/\sqrt{v}$
- the $(\sqrt{v+\epsilon})^{-1}$
- 3 AdaFactor normalized gradient values
- the tiled, AdaFactor row features (3 features)
- the tiled, AdaFactor column features (3 features)
- 1.0 divided by the square root of these previous 6 AdaFactor features
- 3 features consisting AdaFactor normalized momentum values
- 11 features formed by taking the current timestep, t, and computing $\tanh(t/x)$ where $x \in \{1, 3, 10, 30, 100, 300, 1000, 3000, 10k, 30k, 100k\}$.

4. Hyperparameters and Computional Resources

We train Optimus using a distributed setup on the Google Compute Engine[3]. When training on the DiffPhy [5] reconstruction task, we distributed the loss function on 400 vCPU instances. This significant computational cost is what motivated us to find a model which converges faster than prior work (e.g. Adafactor MLP [10]). We trained approximately 30 such models throughout experimentation.

Training models on N-dimensional Rosenbrock functions was much faster as its loss function evaluates in ∼5ms rather than ∼4s. We trained those models for 48 hours using 40

[2]https://github.com/deepmind/launchpad/tree/master/courier
[3]https://cloud.google.com/compute

vCPU instances. In total, we trained approximately 100 such models.

The primary hyperparameters and the ranges of them that we tested are presented in tab. 3. Due to the high computational expense, we could not test all combinations exhaustively. We chose a short truncation length (5) on the physics tasks as it allows us to update the network more often which speed up the training convergence. Similarly, having a smaller batch size allowed us to update the model more frequently as gathering training batches was faster.

5. Datasets Details

We use three sets of data in the paper. Firstly, we evaluate on the established Human3.6M [8] dataset, which is recorded in a laboratory setting with the permission of the actors. We list the sequences used in our validation set in tab. 2. When comparing to prior work we use the protocol established in [11] namely evaluating on sequences *Directions, Discussions, Greeting, Posing, Purchases, Taking Photos, Waiting, Walking, Walking Dog and Walking Together* from subjects S9 and S11. We evaluate the motions using only camera *60457274* and following prior work [11, 16] we downsample the sequences from 50 FPS to 25 FPS.

Next, we use the AIST[4] [13] dataset which features professional dancers performing to copyright-cleared dance music. The sequences we evaluate on are given in tab. 1. Finally, we provide qualitative examples of our method on "in-the-wild" internet videos that were released under creative common licenses.

[4]https://aistdancedb.ongaaccel.jp/

Sequence	Frames
gBR_sBM_c06_d06_mBR4_ch06	1-120
gBR_sBM_c07_d06_mBR4_ch02	1-120
gBR_sBM_c08_d05_mBR1_ch01	1-120
gBR_sFM_c03_d04_mBR0_ch01	1-120
gJB_sBM_c02_d09_mJB3_ch10	1-120
gKR_sBM_c09_d30_mKR5_ch05	1-120
gLH_sBM_c04_d18_mLH5_ch07	1-120
gLH_sBM_c07_d18_mLH4_ch03	1-120
gLH_sBM_c09_d17_mLH1_ch02	1-120
gLH_sFM_c03_d18_mLH0_ch15	1-120
gLO_sBM_c05_d14_mLO4_ch07	1-120
gLO_sBM_c07_d15_mLO4_ch09	1-120
gLO_sFM_c02_d15_mLO4_ch21	1-120
gMH_sBM_c01_d24_mMH3_ch02	1-120
gMH_sBM_c05_d24_mMH4_ch07	1-120

Table 1. Sequences from the dance dataset AIST [13] used for evaluation.

Sequence	Subject	Camera Id	Frames
Phoning	S11	55011271	400-599
Posing_1	S11	58860488	400-599
Purchases	S11	60457274	400-599
SittingDown_1	S11	54138969	400-599
Smoking_1	S11	54138969	400-599
TakingPhoto_1	S11	54138969	400-599
Waiting_1	S11	58860488	400-599
WalkDog	S11	58860488	400-599
WalkTogether	S11	55011271	400-599
Walking_1	S11	55011271	400-599
Greeting_1	S9	54138969	400-599
Phoning_1	S9	54138969	400-599
Purchases	S9	60457274	400-599
SittingDown	S9	55011271	400-599
Smoking	S9	60457274	400-599
TakingPhoto	S9	60457274	400-599
Waiting	S9	60457274	400-599
WalkDog_1	S9	54138969	400-599
WalkTogether_1	S9	55011271	400-599
Walking	S9	58860488	400-599

Table 2. Sequences from the Human3.6M [8] pose dataset used in our ablations and experiments with reinforcement learning agents.

Algorithm 1 The Optimus algorithm. $\mathbf{z}^k$ denotes the Adafactor MLP features from [10] and $\odot$ denotes elementwise multiplication.

```
Initial guess $\mathbf{x}^0$
$\boldsymbol{B}^0 \leftarrow \boldsymbol{I}$
$k \leftarrow 0$
**repeat**
    $\{\boldsymbol{\alpha}, \mathbf{d}, \Delta\boldsymbol{B}^k\} \leftarrow$ Optimus$(\mathbf{z}^k)$
    $\boldsymbol{B}^k \leftarrow \boldsymbol{B}^{k-1} + \Delta\boldsymbol{B}^k$        ▷ Update learned preconditioning matrix.
    $\mathbf{s}^k \leftarrow \mathbf{B}^k[\lambda_a \exp(\lambda_b\boldsymbol{\alpha}) \odot \mathbf{d}]$
    $\mathbf{x}^k \leftarrow \mathbf{x}^{k-1} + \mathbf{s}^k$
    $k \leftarrow k + 1$
**until** $k >$ MAX_ITERS or $f(\mathbf{x}^k) > \frac{1}{N}\sum_{i=1}^{N} \beta f(\mathbf{x}^{k-i}) + \epsilon$
```

Variable	Search grid	Value Used
Learning Rate	$\{0.1, 0.01, 0.001, 5 \times 10^{-4}, 1 \times 10^{-4}\}$	5×10^{-4}
Model Dimension	$\{64, 128, 256, 512\}$	128
# Encoders	$\{1, 2, 3, 4, 5\}$	3
λ_a	$\{0.001, 0.01, 0.1\}$	0.1
λ_b	$\{0.001, 0.01, 0.1\}$	0.1
Batch Size	$\{20, 50, 128\}$	20
PES truncation length	$\{5, 10, 20\}$	5

Table 3. An overview of the different hyperparameters tested during designing of Optimus. *Value Used* refers to the value used on the human pose reconstruction task from video.

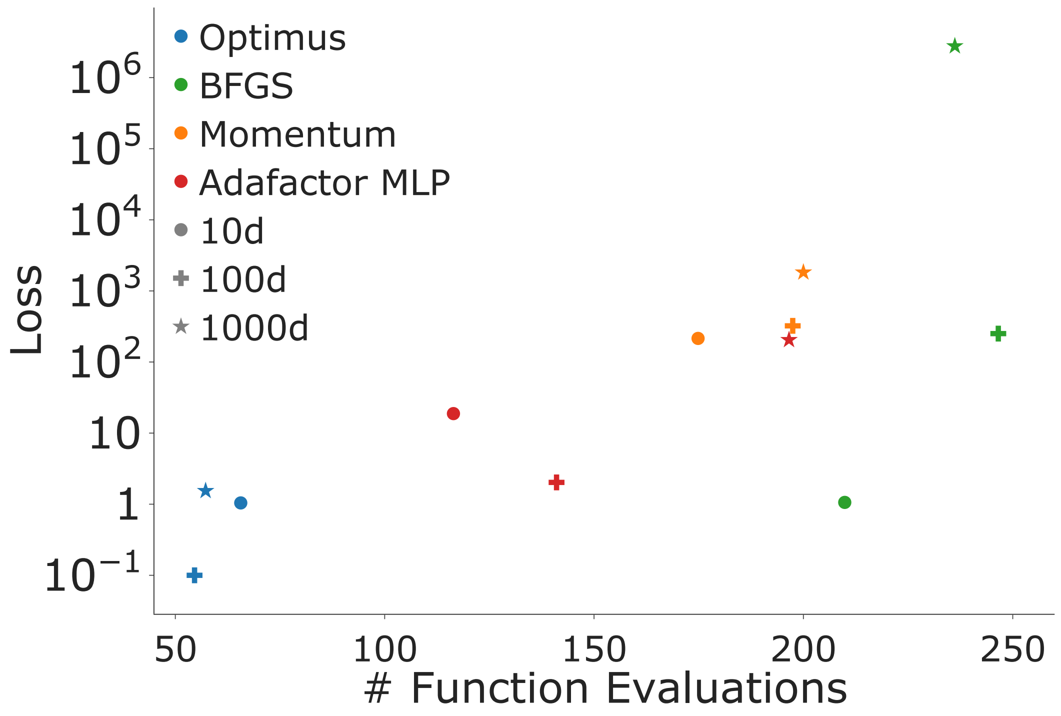
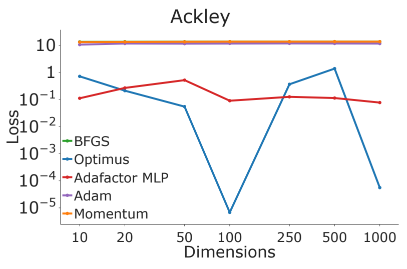
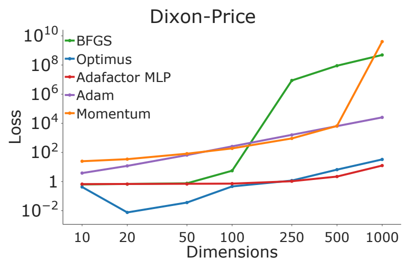


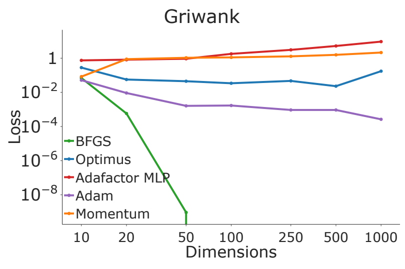
Figure 4. Comparison of loss vs number of function evaluations on the N-dimensional Rosenbrock functions for a maximum budget of 200 steps. Note that BFGS may evaluate the function multiple times per step due to its line search.



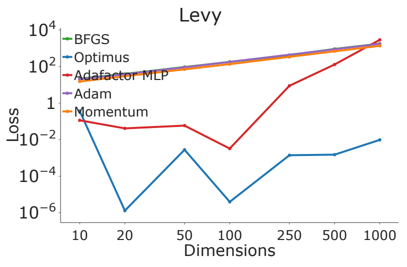
(a) Ackley



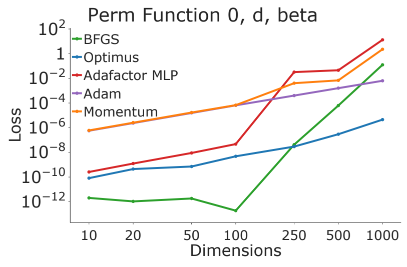
(b) Dixon-Price



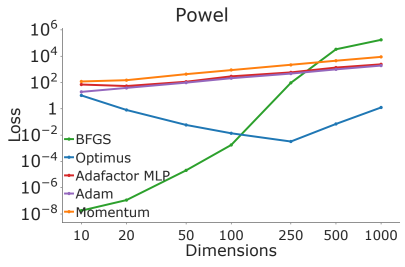
(c) Griwank



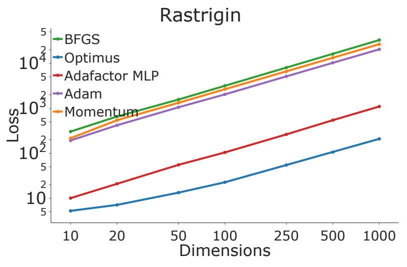
(d) Levy



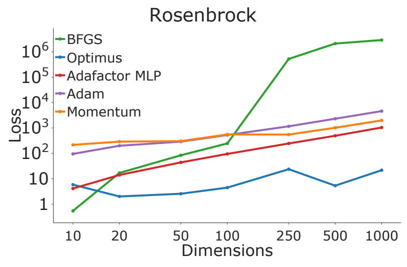
(e) Perm Function 0, d, beta



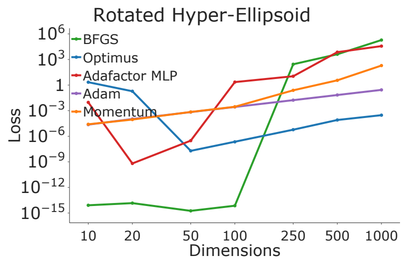
(f) Powel



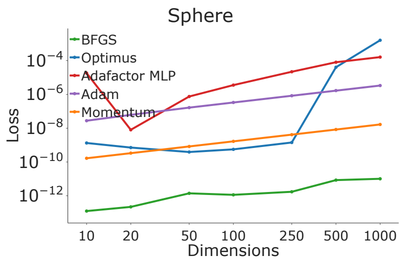
(g) Rastrigin



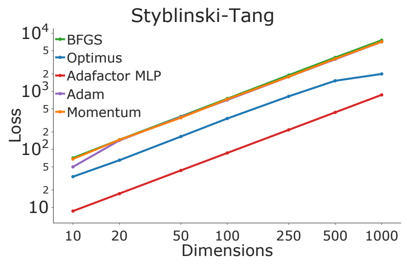
(h) Rosenbrock



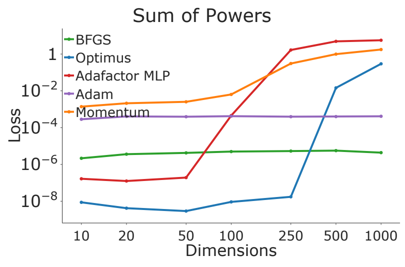
(i) Rotated Hyper-Ellipsoid



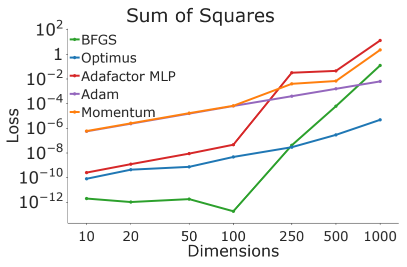
(j) Sphere



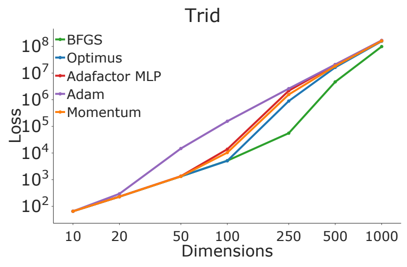
(k) Styblinski-Tang



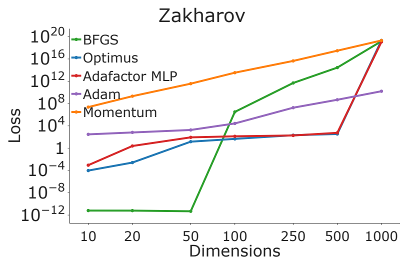
(l) Sum of Powers



(m) Sum of Squares

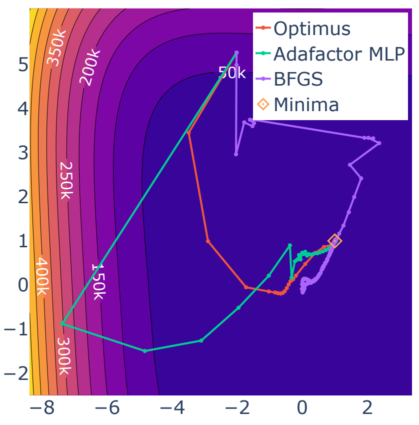


(n) Trid

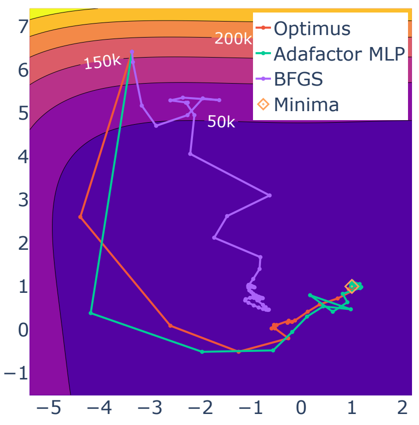


(o) Zakharov

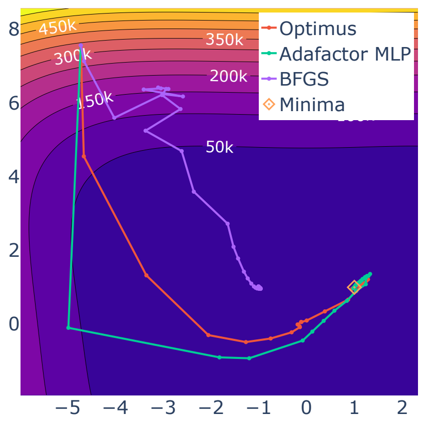
Figure 5. Evaluation results used to generate performance profile plot in Sec. 5.1 of the paper. Here we show results separately for each objective function. We plot objective value averaged over 64 randomly initialized optimization runs on the y-axis and dimensionality of the objective function on the x-axis. Each method has a fixed budget of 200 iterations.



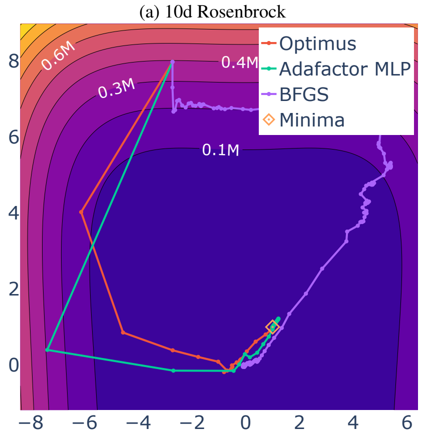
(a) 10d Rosenbrock



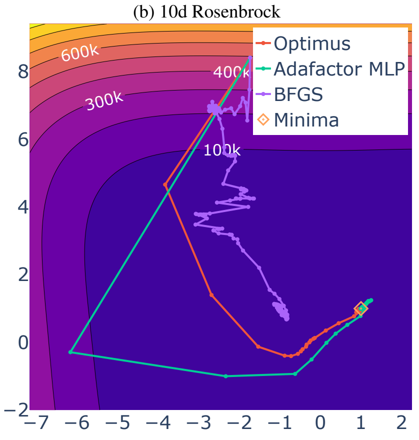
(b) 10d Rosenbrock



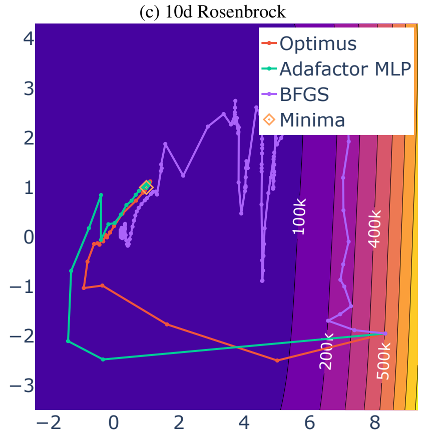
(c) 10d Rosenbrock



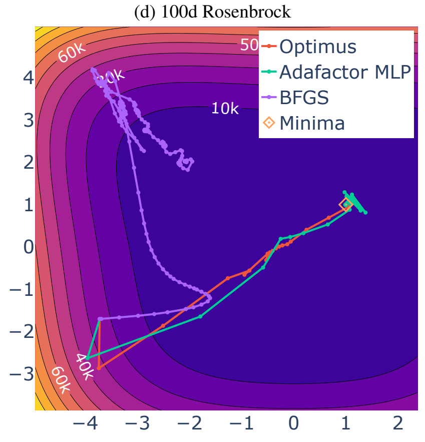
(d) 100d Rosenbrock



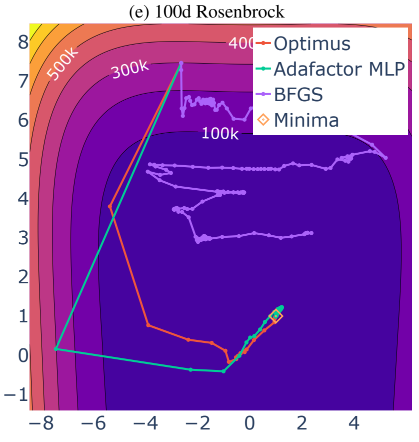
(e) 100d Rosenbrock



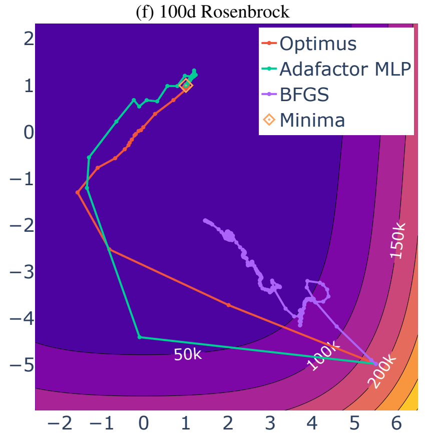
(f) 100d Rosenbrock



(g) 1000d Rosenbrock



(h) 1000d Rosenbrock



(i) 1000d Rosenbrock

Figure 6. Example trajectories on N-dimensional Rosenbrock functions. We visualize the trajectory for the first two dimensions.

Table 4. Full results for experiments on classical optimization functions. Each value is averaged over 64 random initializations. We implement the functions according to the formulas presented in [12].

Function	BFGS	Optimus	Adafactor MLP	Adam	Momentum
Ackley 2d	1.11e+01	6.65e+00	**1.69e-01**	8.51e+00	8.99e+00
Ackley 10d	1.34e+01	7.16e-01	**1.11e-01**	1.06e+01	1.31e+01
Ackley 20d	1.36e+01	**2.12e-01**	2.68e-01	1.16e+01	1.34e+01
Ackley 50d	1.36e+01	**5.46e-02**	5.19e-01	1.15e+01	1.34e+01
Ackley 100d	1.37e+01	**6.80e-06**	9.02e-02	1.18e+01	1.36e+01
Ackley 250d	1.36e+01	3.65e-01	**1.26e-01**	1.18e+01	1.36e+01
Ackley 500d	1.37e+01	1.38e+00	**1.13e-01**	1.17e+01	1.37e+01
Ackley 1000d	1.37e+01	**5.60e-05**	7.71e-02	1.16e+01	1.37e+01
Dixon-Price 2d	**4.64e-13**	4.67e+00	8.86e-06	2.83e-01	5.42e+00
Dixon-Price 10d	6.35e-01	**4.24e-01**	6.67e-01	3.81e+00	2.49e+01
Dixon-Price 20d	6.67e-01	**7.60e-03**	6.68e-01	1.19e+01	3.44e+01
Dixon-Price 50d	7.58e-01	**3.62e-02**	6.77e-01	6.56e+01	8.19e+01
Dixon-Price 100d	5.54e+00	**4.63e-01**	7.20e-01	2.56e+02	1.89e+02
Dixon-Price 250d	8.67e+06	1.17e+00	**1.05e+00**	1.58e+03	9.12e+02
Dixon-Price 500d	8.93e+07	6.57e+00	**2.20e+00**	6.39e+03	6.60e+03
Dixon-Price 1000d	4.90e+08	3.30e+01	**1.25e+01**	2.53e+04	4.04e+09
Griwank 2d	2.00e-02	9.04e-01	1.43e-01	1.96e-02	**1.91e-02**
Griwank 10d	6.92e-02	2.81e-01	7.45e-01	**5.16e-02**	8.33e-02
Griwank 20d	**5.78e-04**	5.63e-02	8.05e-01	9.16e-03	8.78e-01
Griwank 50d	**9.31e-10**	4.48e-02	9.25e-01	1.62e-03	1.06e+00
Griwank 100d	**0.00e+00**	3.41e-02	1.79e+00	1.69e-03	1.11e+00
Griwank 250d	**0.00e+00**	4.68e-02	3.07e+00	9.31e-04	1.28e+00
Griwank 500d	**0.00e+00**	2.30e-02	5.20e+00	9.29e-04	1.57e+00
Griwank 1000d	**0.00e+00**	1.74e-01	9.34e+00	2.64e-04	2.14e+00
Levy 2d	8.36e+00	5.65e+00	**3.74e+00**	8.40e+00	3.80e+00
Levy 10d	2.20e+01	3.72e-01	**1.15e-01**	2.06e+01	1.53e+01
Levy 20d	4.00e+01	**1.31e-06**	4.16e-02	3.81e+01	3.03e+01
Levy 50d	9.51e+01	**2.81e-03**	5.89e-02	9.17e+01	7.12e+01
Levy 100d	1.83e+02	**3.96e-06**	3.27e-03	1.78e+02	1.37e+02
Levy 250d	4.20e+02	**1.41e-03**	8.96e+00	4.34e+02	3.39e+02
Levy 500d	9.28e+02	**1.52e-03**	1.31e+02	8.89e+02	6.95e+02
Levy 1000d	1.81e+03	**9.78e-03**	2.93e+03	1.76e+03	1.38e+03
Perm Function 0, d, beta 2d	**1.76e-12**	1.16e-01	1.07e-01	9.96e-09	5.65e-09
Perm Function 0, d, beta 10d	**2.03e-12**	8.17e-11	2.58e-10	5.69e-07	6.02e-07
Perm Function 0, d, beta 20d	**1.06e-12**	4.45e-10	1.24e-09	2.37e-06	2.60e-06
Perm Function 0, d, beta 50d	**1.83e-12**	7.14e-10	8.96e-09	1.61e-05	1.70e-05
Perm Function 0, d, beta 100d	**1.89e-13**	4.80e-09	4.70e-08	6.49e-05	6.73e-05
Perm Function 0, d, beta 250d	4.21e-08	**2.95e-08**	3.21e-02	4.02e-04	4.05e-03
Perm Function 0, d, beta 500d	6.24e-05	**2.96e-07**	4.53e-02	1.61e-03	6.86e-03
Perm Function 0, d, beta 1000d	1.25e-01	**4.52e-06**	1.33e+01	6.36e-03	2.27e+00
Powel 2d	**1.08e-08**	4.29e-01	5.16e+00	7.83e+00	3.43e+01
Powel 10d	**1.81e-08**	1.04e+01	7.02e+01	1.94e+01	1.20e+02
Powel 20d	**1.15e-07**	8.17e-01	5.39e+01	3.82e+01	1.50e+02
Powel 50d	**2.09e-05**	5.92e-02	1.14e+02	9.74e+01	4.31e+02
Powel 100d	**1.78e-03**	1.38e-02	2.94e+02	2.12e+02	8.98e+02
Powel 250d	9.41e+01	**3.26e-03**	5.87e+02	4.77e+02	2.18e+03
Powel 500d	3.35e+04	**7.06e-02**	1.37e+03	9.98e+02	4.54e+03
Powel 1000d	1.76e+05	**1.26e+00**	2.41e+03	1.93e+03	8.88e+03

Continued on next page

Table 4 – continued from previous page

Function	BFGS	Optimus	Adafactor MLP	Adam	Momentum
Rastrigin 2d	6.17e+01	3.30e+00	**2.85e+00**	3.99e+01	5.59e+01
Rastrigin 10d	3.04e+02	**5.24e+00**	1.01e+01	1.93e+02	2.16e+02
Rastrigin 20d	6.57e+02	**7.12e+00**	2.10e+01	4.20e+02	5.44e+02
Rastrigin 50d	1.57e+03	**1.33e+01**	5.53e+01	1.05e+03	1.33e+03
Rastrigin 100d	3.18e+03	**2.27e+01**	1.05e+02	2.04e+03	2.66e+03
Rastrigin 250d	8.03e+03	**5.48e+01**	2.63e+02	5.14e+03	6.61e+03
Rastrigin 500d	1.61e+04	**1.06e+02**	5.45e+02	1.03e+04	1.33e+04
Rastrigin 1000d	3.30e+04	**2.10e+02**	1.10e+03	2.05e+04	2.67e+04
Rosenbrock 2d	**5.07e-13**	3.48e+01	5.06e-01	3.58e+00	5.17e+00
Rosenbrock 10d	**5.43e-01**	5.84e+00	4.12e+00	9.52e+01	2.16e+02
Rosenbrock 20d	1.69e+01	**2.00e+00**	1.41e+01	1.99e+02	2.89e+02
Rosenbrock 50d	8.40e+01	**2.56e+00**	4.40e+01	2.89e+02	3.04e+02
Rosenbrock 100d	2.46e+02	**4.47e+00**	9.50e+01	5.26e+02	5.56e+02
Rosenbrock 250d	5.24e+05	**2.37e+01**	2.45e+02	1.16e+03	5.50e+02
Rosenbrock 500d	2.11e+06	**5.33e+00**	4.93e+02	2.31e+03	1.02e+03
Rosenbrock 1000d	2.91e+06	**2.17e+01**	1.04e+03	4.61e+03	1.99e+03
Rotated Hyper-Ellipsoid 2d	**2.36e-14**	2.59e+01	1.74e+00	3.81e-07	5.32e-08
Rotated Hyper-Ellipsoid 10d	**8.08e-15**	2.18e+00	9.38e-03	2.31e-05	2.53e-05
Rotated Hyper-Ellipsoid 20d	**1.45e-14**	1.89e-01	6.46e-10	1.05e-04	9.07e-05
Rotated Hyper-Ellipsoid 50d	**1.80e-15**	1.95e-08	3.06e-07	6.81e-04	7.03e-04
Rotated Hyper-Ellipsoid 100d	**7.18e-15**	2.22e-07	2.24e+00	2.71e-03	2.86e-03
Rotated Hyper-Ellipsoid 250d	2.76e+02	**5.88e-06**	1.08e+01	1.71e-02	2.43e-01
Rotated Hyper-Ellipsoid 500d	4.09e+03	**8.07e-05**	7.24e+03	6.87e-02	3.61e+00
Rotated Hyper-Ellipsoid 1000d	1.91e+05	**2.97e-04**	3.70e+04	2.73e-01	1.94e+02
Sphere 2d	**7.07e-14**	4.24e-04	3.22e-03	1.84e-09	3.68e-11
Sphere 10d	**1.25e-13**	1.33e-09	1.81e-05	2.77e-08	1.66e-10
Sphere 20d	**2.21e-13**	7.14e-10	8.02e-09	6.15e-08	3.35e-10
Sphere 50d	**1.40e-12**	3.89e-10	7.48e-07	1.65e-07	8.28e-10
Sphere 100d	**1.14e-12**	5.59e-10	3.54e-06	3.32e-07	1.66e-09
Sphere 250d	**1.73e-12**	1.42e-09	2.19e-05	8.34e-07	4.13e-09
Sphere 500d	**8.48e-12**	4.04e-05	7.99e-05	1.68e-06	8.37e-09
Sphere 1000d	**1.01e-11**	1.60e-03	1.61e-04	3.33e-06	1.66e-08
Styblinski-Tang 2d	1.40e+01	4.59e+00	**3.40e+00**	1.10e+01	1.04e+01
Styblinski-Tang 10d	7.13e+01	3.39e+01	**8.66e+00**	5.00e+01	6.83e+01
Styblinski-Tang 20d	1.47e+02	6.51e+01	**1.73e+01**	1.43e+02	1.46e+02
Styblinski-Tang 50d	3.69e+02	1.66e+02	**4.34e+01**	3.61e+02	3.50e+02
Styblinski-Tang 100d	7.52e+02	3.40e+02	**8.69e+01**	7.10e+02	7.30e+02
Styblinski-Tang 250d	1.90e+03	8.20e+02	**2.17e+02**	1.78e+03	1.79e+03
Styblinski-Tang 500d	3.83e+03	1.52e+03	**4.34e+02**	3.57e+03	3.64e+03
Styblinski-Tang 1000d	7.59e+03	1.99e+03	**8.70e+02**	7.13e+03	7.17e+03
Sum of Powers 2d	3.86e-08	2.16e-08	**3.42e-09**	2.14e-06	3.96e-05
Sum of Powers 10d	2.16e-06	**8.71e-09**	1.66e-07	2.85e-04	1.38e-03
Sum of Powers 20d	3.60e-06	**4.18e-09**	1.25e-07	4.06e-04	2.11e-03
Sum of Powers 50d	4.24e-06	**2.92e-09**	1.92e-07	3.92e-04	2.53e-03
Sum of Powers 100d	4.99e-06	**9.19e-09**	4.53e-04	4.20e-04	6.37e-03
Sum of Powers 250d	5.32e-06	**1.74e-08**	1.66e+00	3.94e-04	3.05e-01
Sum of Powers 500d	**5.63e-06**	1.45e-02	4.90e+00	3.98e-04	9.88e-01
Sum of Powers 1000d	**4.38e-06**	2.95e-01	5.62e+00	4.14e-04	1.76e+00
Sum of Squares 2d	**1.76e-12**	1.16e-01	1.07e-01	9.96e-09	5.65e-09
Sum of Squares 10d	**2.03e-12**	8.17e-11	2.58e-10	5.69e-07	6.02e-07

Continued on next page

Table 4 – continued from previous page

Function	BFGS	Optimus	Adafactor MLP	Adam	Momentum
Sum of Squares 20d	**1.06e-12**	4.45e-10	1.24e-09	2.37e-06	2.60e-06
Sum of Squares 50d	**1.83e-12**	7.56e-10	8.96e-09	1.61e-05	1.70e-05
Sum of Squares 100d	**1.89e-13**	4.80e-09	4.70e-08	6.49e-05	6.73e-05
Sum of Squares 250d	4.21e-08	**2.95e-08**	3.21e-02	4.02e-04	4.05e-03
Sum of Squares 500d	6.24e-05	**2.96e-07**	4.53e-02	1.61e-03	6.86e-03
Sum of Squares 1000d	1.25e-01	**4.90e-06**	1.33e+01	6.36e-03	2.27e+00
Trid 2d	**5.00e+00**	**5.00e+00**	**5.00e+00**	**5.00e+00**	**5.00e+00**
Trid 10d	**6.50e+01**	**6.50e+01**	6.51e+01	**6.50e+01**	**6.50e+01**
Trid 20d	**2.30e+02**	**2.30e+02**	2.31e+02	2.95e+02	**2.30e+02**
Trid 50d	1.33e+03	**1.32e+03**	1.35e+03	1.47e+04	**1.32e+03**
Trid 100d	5.16e+03	**5.11e+03**	1.39e+04	1.56e+05	1.04e+04
Trid 250d	**5.55e+04**	8.92e+05	2.18e+06	2.61e+06	1.57e+06
Trid 500d	**4.67e+06**	1.64e+07	2.09e+07	2.10e+07	1.83e+07
Trid 1000d	**9.91e+07**	1.57e+08	1.68e+08	1.67e+08	1.62e+08
Zakharov 2d	**3.61e-12**	2.21e-01	8.11e-05	1.58e-01	2.11e-02
Zakharov 10d	**6.40e-12**	9.58e-05	8.85e-04	3.00e+02	2.31e+07
Zakharov 20d	**6.28e-12**	2.61e-03	2.45e+00	6.68e+02	2.09e+09
Zakharov 50d	**4.78e-12**	1.46e+01	8.71e+01	1.83e+03	3.71e+11
Zakharov 100d	3.20e+06	**4.54e+01**	1.34e+02	2.71e+04	3.45e+13
Zakharov 250d	4.94e+11	2.12e+02	**1.89e+02**	1.81e+07	4.45e+15
Zakharov 500d	2.82e+14	**3.46e+02**	5.61e+02	4.80e+08	2.96e+17
Zakharov 1000d	1.53e+19	2.08e+19	1.15e+19	**1.63e+10**	2.08e+19

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2

[2] B. Bell. Cppad: a package for c++ algorithmic differentiation, 2021. 2

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 1

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

[5] 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. 2

[6] 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. 2

[7] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. 1

[8] 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. 2, 3

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[10] 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. 2, 4

[11] 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. 3

[12] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved November 9, 2022, from http://www.sfu.ca/~ssurjano. 7

[13] 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. 3

[14] 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. 2

[15] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 1

[16] 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. 3