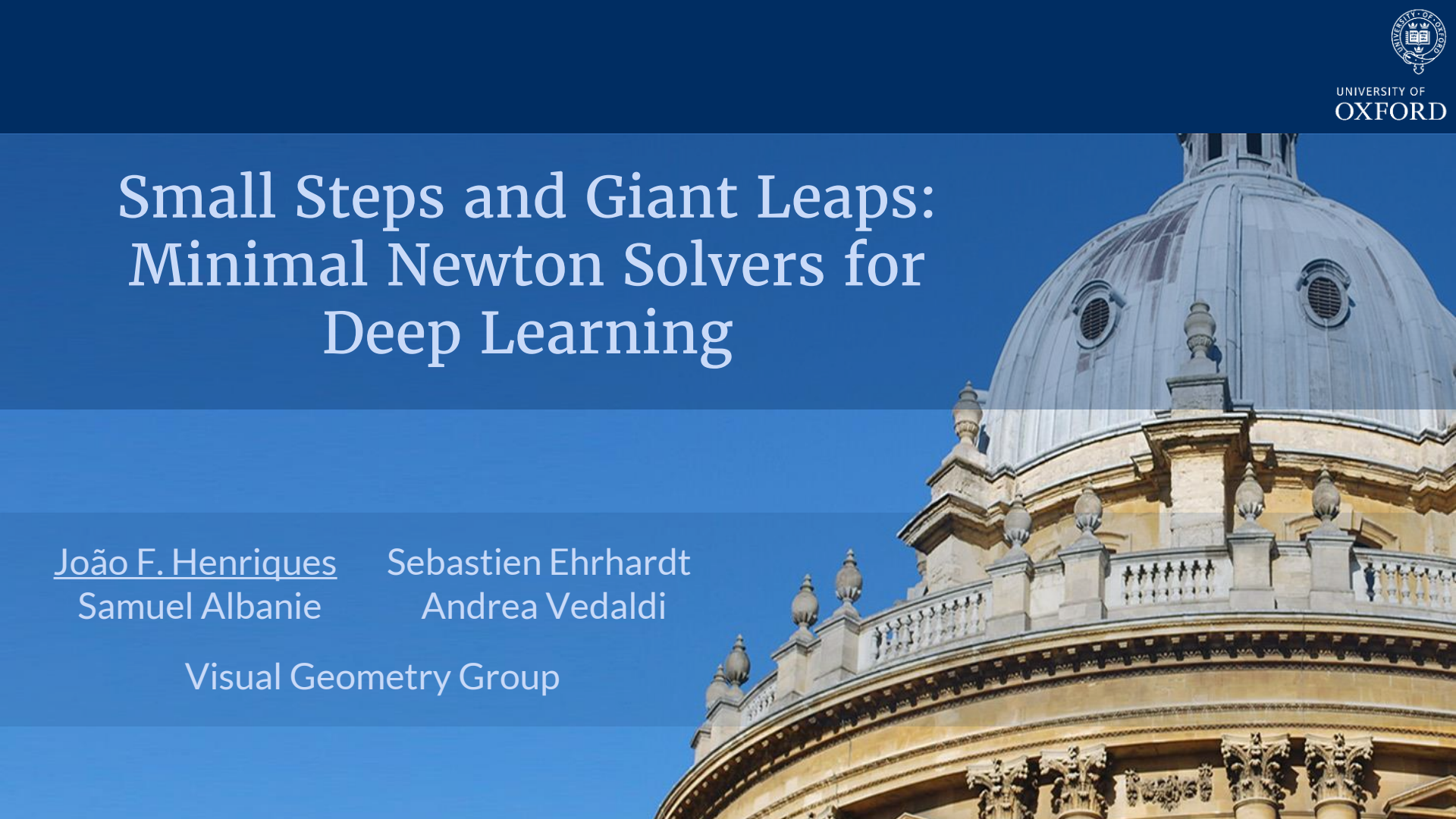


UNIVERSITY OF OXFORD

Small Steps and Giant Leaps: Minimal Newton Solvers for Deep Learning

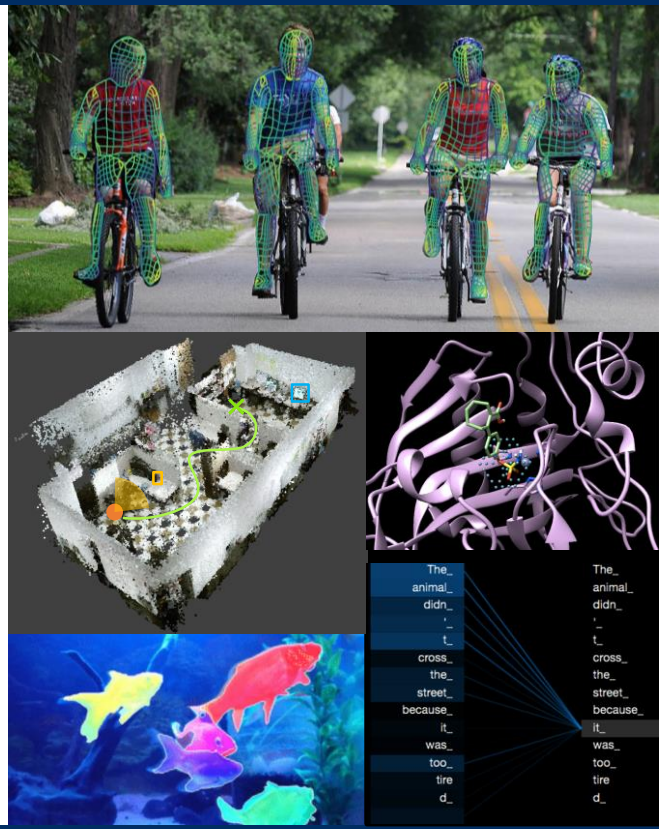
João F. Henriques
Sebastien Ehrhardt
Samuel Albanie
Andrea Vedaldi

Visual Geometry Group

The state of deep learning

UNIVERSITY OF OXFORD

- Deep learning is **everywhere**.
- As a point of comparison, AlexNet (Krizhevsky et al.) arguably brought deep learning to “mainstream” computer vision in **2012**.
- AlexNet was trained with Stochastic Gradient Descent (SGD).
- Almost a decade later, we’re **still using SGD** (and other first-order variants)!

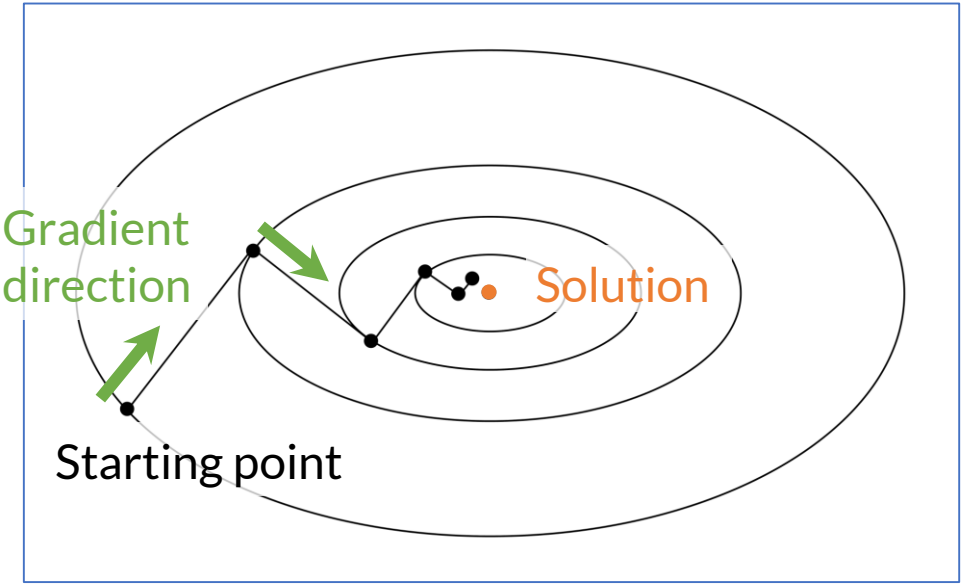


The problem

- First-order solvers (SGD, Adam, etc) are **slow** to converge even on simple problems.

⇒ Main cause: **poor scaling** of objective function.

Gradient descent with optimal learning rate on a 2D quadratic loss surface



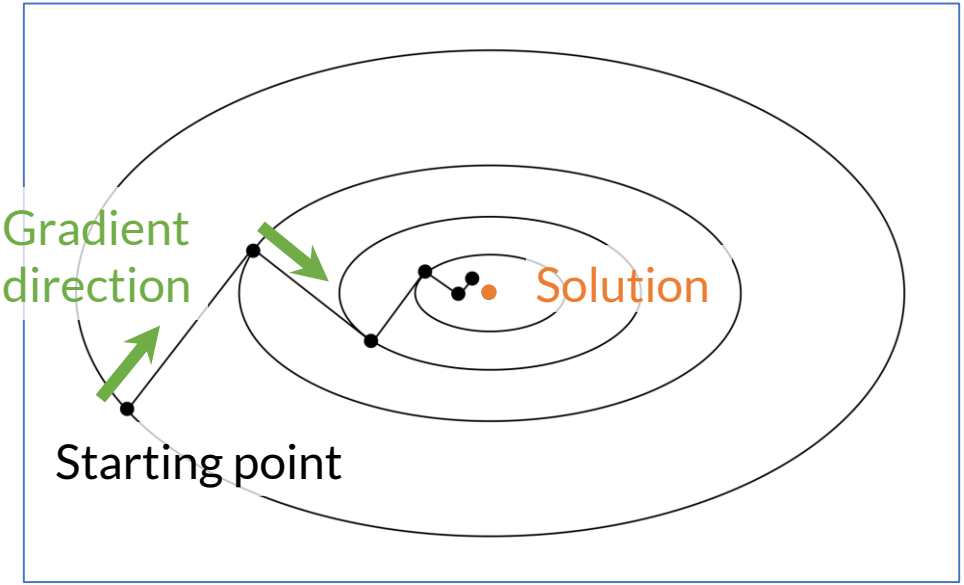
The problem

UNIVERSITY OF OXFORD

More problems:

- Still happens if each parameter is scaled independently (e.g. AdaGrad/Adam/etc, batch/layer normalization).
- Due to nonlinearity of deep nets, the optimal scaling will change as the parameters change.

Gradient descent with optimal learning rate on a 2D quadratic loss surface



The classical solution

- Use a 2nd-order solver (Newton method):

The step → $z = -\widehat{H}^{-1}J$ ← The gradient

↑

The scaling (Hessian/2nd order gradient)

Incompatible with deep learning:

- Hessian matrix size **quadratic** in #parameters (e.g. terabytes).
- Costly to invert even if small.

The modern approach

- **Hessian-free** methods use **automatic differentiation** (e.g. PyTorch) to multiply vectors with the Hessian without storing it.
- These Hessian-vector products can cost only ≃ 2 back-propagations.

Algorithm 2. Simplified Hessian-free method.

for $t = 0, ..., T-1$ **do**
  $z_0 = -J(w_t)$
  for $r = 0, ..., R-1$ (or convergence) **do**
    $z_{r+1} = \mathrm{CG}(z_r, \hat{H}(w_t)z_r, J(w_t))$
  end for
  $w_{t+1} = w_t + z_R$
end for

← Iteratively solve $z = -\widehat{H}^{-1}J$ by Conjugate Gradient (CG) (compute Newton step)

← Apply step to parameters

Problems with the modern approach

UNIVERSITY OF OXFORD

Algorithm 2. Simplified Hessian-free method.

for $t = 0, ..., T-1$ **do**
  $z_0 = -J(w_t)$
  for $r = 0, ..., R-1$ (or convergence) **do**
    $z_{r+1} = \mathrm{CG}(z_r, \hat{H}(w_t) z_r, J(w_t))$
  end for
  $w_{t+1} = w_t + z_R$
end for

Hessian-free methods:

- Still **dozens of times more costly** than gradient methods (due to inner loop).
- Must fix and run CG over a single batch because it is **unstable under noise**.

The *even more* modern approach (ours)

UNIVERSITY OF OXFORD

- We need an alternative to Conjugate-Gradient (CG) for matrix inversion.

$$z = -\hat{H}^{-1} J \quad \leftarrow \text{(Newton step, costly to compute explicitly)}$$

- Notice this inversion can be written as a **minimization**:

$$z = \arg\min_{z'} \tfrac{1}{2} z'^{T} \hat{H} z' + z'^{T} J$$

- So we can **replace CG** with **gradient descent**, using the gradient (over z):

$$\triangle_z = \hat{H} z + J$$

The *even more* modern approach (ours)

- **Proposed changes** to Hessian-free method:

Algorithm 2. Simplified Hessian-free method.

1: **for** $t = 0, ..., T-1$ **do**
2: ~~$z_0 = J(w_t)$~~
3: ~~**for** $r = 0, ..., R-1$ (or convergence) **do**~~
4: $z_{r+1} = \mathrm{CG}(z_r, \hat{H}(w_t)z_r, J(w_t))$
5: ~~**end for**~~
6: $w_{t+1} = w_t + z_R$
7: **end for**

← Warm-start from prev. iteration
← Only do 1 iteration of inner loop
← Replace CG with gradient descent (robust to warm-starts and noise)

The result – CurveBall

UNIVERSITY OF OXFORD

Algorithm 1. CURVEBALL (proposed).

$z_0 = \mathbf{0}$

for $t = 0, ..., T-1$ **do**

$\Delta_z = \hat{H}(w_t)z_t + J(w_t)$ ← Gradient for $z = -\hat{H}^{-1}J$

$z_{t+1} = \rho z_t - \beta \Delta_z$ ← Gradient descent over z

$w_{t+1} = w_t + z_{t+1}$

end for

The result – CurveBall

UNIVERSITY OF OXFORD

Algorithm 1. CURVEBALL (proposed).

1: $z_0 = \mathbf{0}$
2: **for** $t = 0, ..., T-1$ **do**
3: $\quad \Delta_z = \hat{H}(w_t) z_t + J(w_t)$ ← Gradient for $z = -\widehat{H}^{-1} J$
4: $\quad z_{t+1} = \rho z_t - \beta \Delta_z$ ← Gradient descent over z
5: $\quad w_{t+1} = w_t + z_{t+1}$
6: **end for**

Main characteristics:

- Cost of inverting the Hessian is amortized over time.
- The buffer z adapts over time to approximate $z \simeq -\widehat{H}^{-1} J$.

⇒ Approximates the Newton step!

The result – CurveBall

UNIVERSITY OF OXFORD

Algorithm 1. CURVEBALL (proposed).

$z_0 = \mathbf{0}$
for $t = 0, ..., T-1$ **do**
  $\Delta_z = \hat{H}(w_t)z_t + J(w_t)$ ← Gradient for $z = -\widehat{H}^{-1}J$
  $z_{t+1} = \rho z_t - \beta\Delta_z$ ← Gradient descent over z
  $w_{t+1} = w_t + z_{t+1}$
end for

More :

- Size of z is $\mathcal{O}(p)$ (a momentum buffer) instead of $\mathcal{O}(p^2)$ (approximate Hessian).
- The implicit Hessian is averaged over **many batches of data**, as opposed to computing a Hessian for a single batch (which would be noisy).

The result – CurveBall

UNIVERSITY OF OXFORD

Algorithm 1. CURVEBALL (proposed).

$z_0 = \mathbf{0}$
for $t = 0, ..., T-1$ **do**
  $\Delta_z = \hat{H}(w_t) z_t + J(w_t)$ ← Gradient for $z = -\widehat{H}^{-1} J$
  $z_{t+1} = \rho z_t - \beta \Delta_z$ ← Gradient descent over z
  $w_{t+1} = w_t + z_{t+1}$
end for

Advantages:

- Very **fast** (cost of $\widehat{H}v \simeq 2$ back-props).
- **Easy** to implement.
- Can get hyper-parameters (ρ, β) automatically.

⇓

No hyper-parameter tuning!

The result – CurveBall

UNIVERSITY OF OXFORD

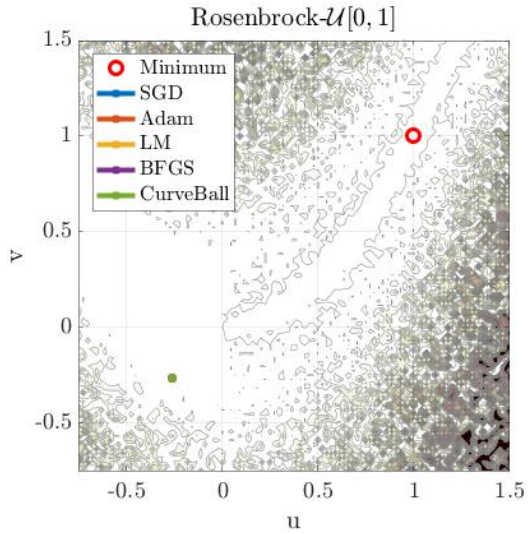
Algorithm 1. CURVEBALL (proposed).

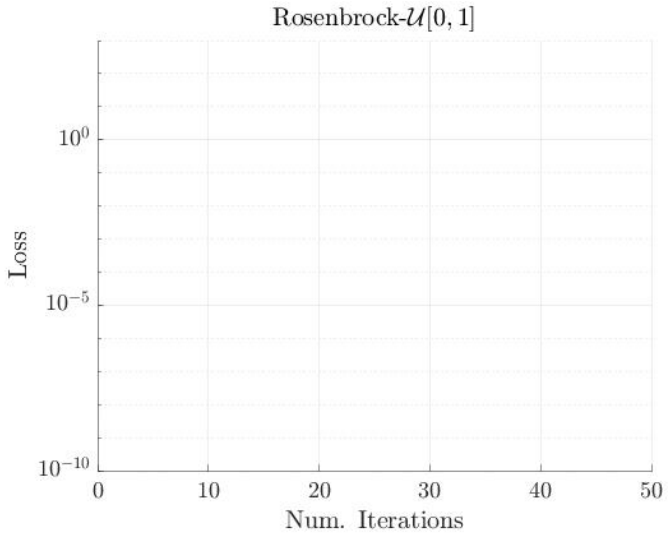
1: $z_0 = \mathbf{0}$
2: **for** $t = 0, ..., T-1$ **do**
3: $\quad \Delta_z = \hat{H}(w_t) z_t + J(w_t)$
4: $\quad z_{t+1} = \rho z_t - \beta \Delta_z$
5: $\quad w_{t+1} = w_t + z_{t+1}$
6: **end for**

Comparison to SGD:

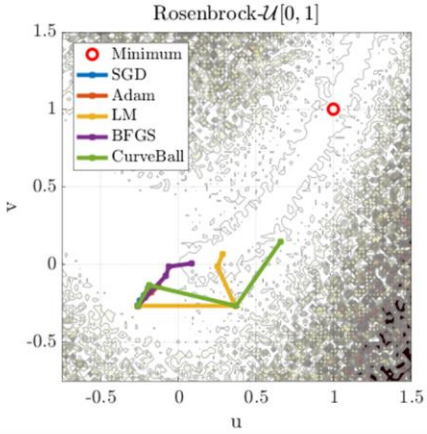
- Reduces *exactly* to Momentum SGD, if we eliminate the Hessian term.
- Momentum SGD is also known as the Heavy-Ball Method.
- Since we add a curvature (Hessian) term to it, we named our method **CurveBall**.

How to break your optimiser

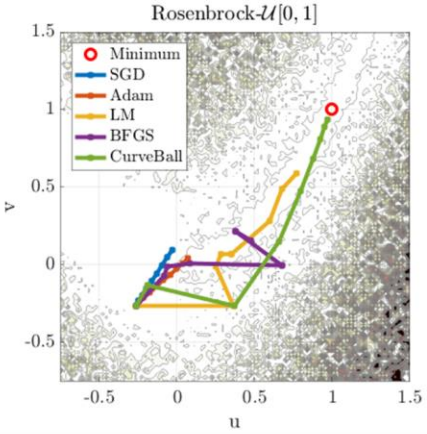




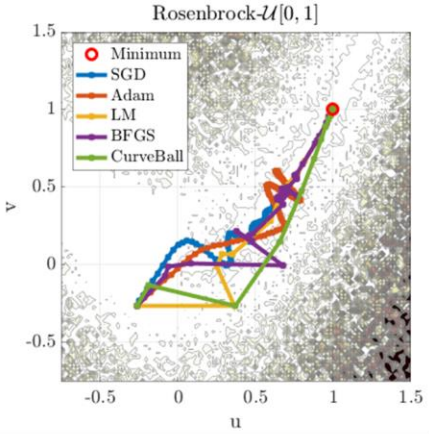
How to break your optimiser



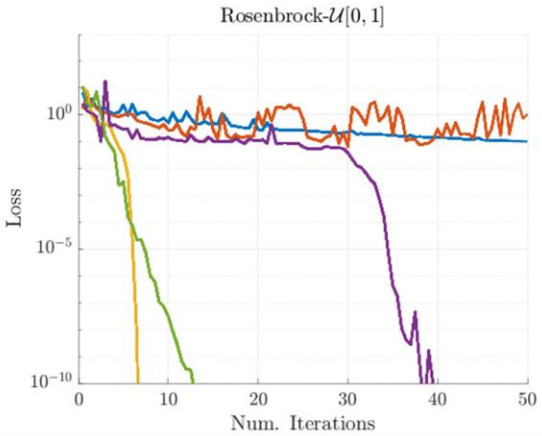
Iteration 3



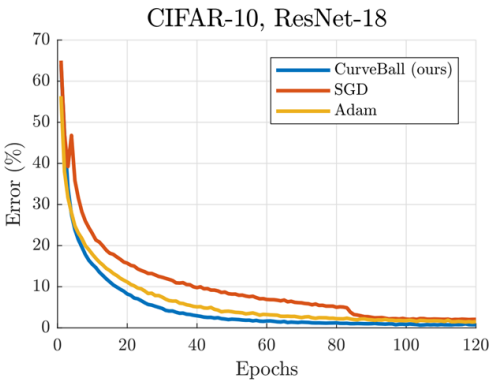
Iteration 8

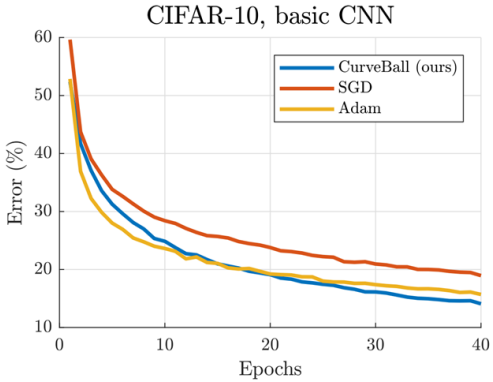


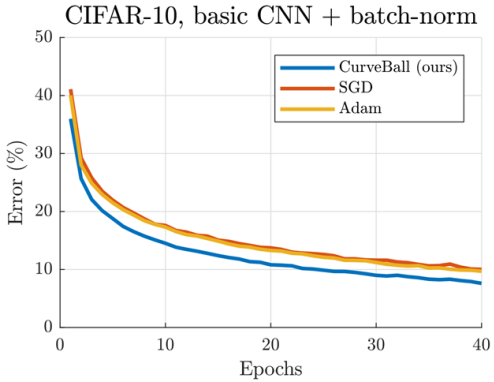
Iteration 100

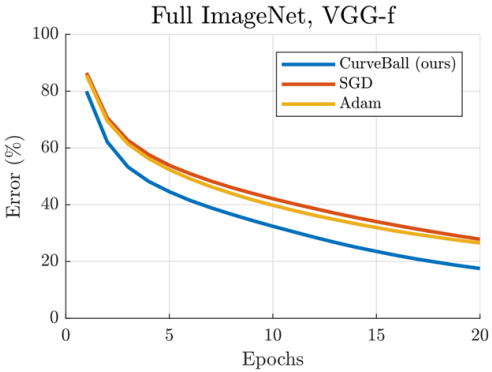


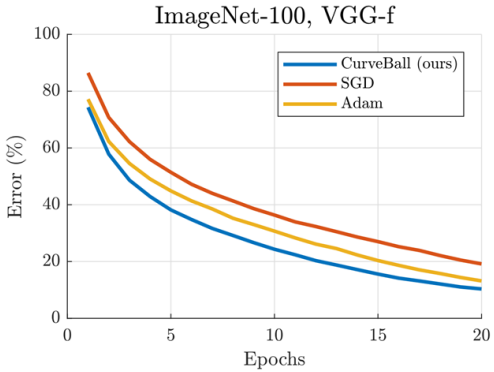
Experiments with standard CNNs

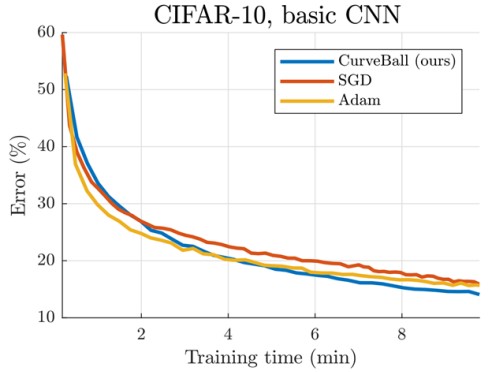












Better convergence with the **same** hyper-parameters across **all datasets**!

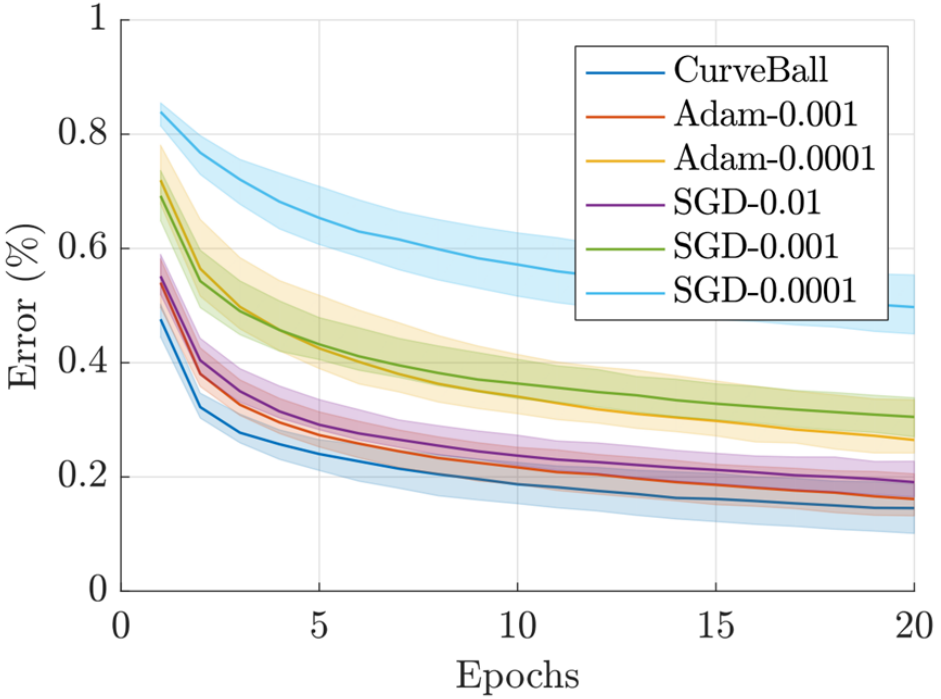
Experiments on 50 random architectures

UNIVERSITY OF OXFORD

- Architectures that didn't work with SGD were discarded early.

  ⇒ So standard deep networks are *biased* to favour 1st-order methods.

- True test of generalization across networks: **random architectures**.



Training vs. validation performance

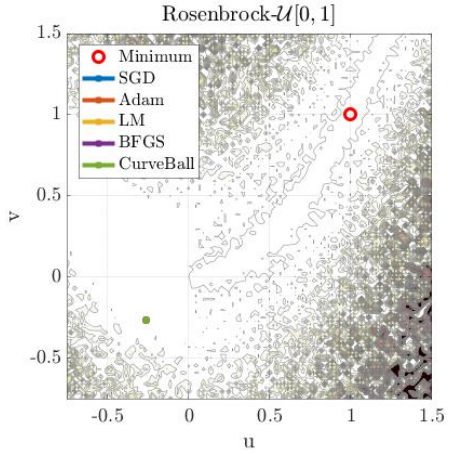
UNIVERSITY OF OXFORD

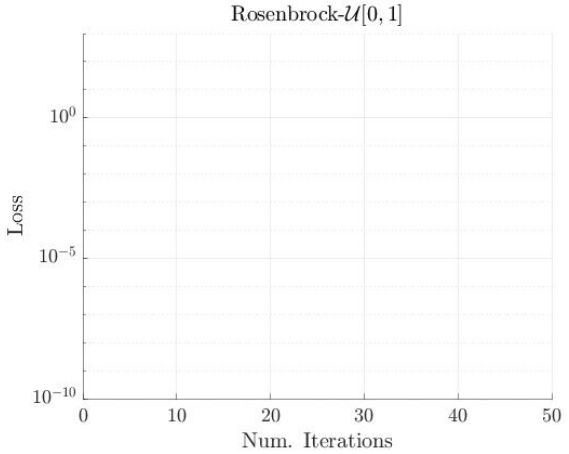
Model	Basic	Basic + BN	ResNet-18	VGG-f
CURVEBALL λ	**14.1** / 19.9	**7.6** / 16.3	**0.7** / 15.3 (13.5)	**10.3** / **33.5**
CURVEBALL	15.3 / **19.3**	9.4 / **15.8**	1.3 / 16.1	12.7 / 33.8
SGD	18.9 / 21.1	10.0 / 16.1	2.1 / **12.8**	19.2 / 39.8
Adam	15.7 / 19.7	9.6 / 16.1	1.4 / 14.0	13.2 / 35.9

Train./val. error

Conclusions

- We propose a practical 2nd-order solver, **CurveBall**, specifically tailored for deep learning.
- Converges to Newton solver in the limit, which is optimal but expensive.
- Applicable to **large-scale settings** (e.g. ImageNet, ResNets).
- **Automatic** hyper-parameter tuning with closed-form solutions.





Project page with code:
www.robots.ox.ac.uk/~joao/curveball