Second-order optimization made practical for deep learning: a preliminary analysis

https://github.com/wang-zixuan/Second-Order-Optimizer

Dec 4, 2023

Zixuan Wang, Mingzheng Huang

Motivation

Basics of second-order optimization

- Calculate the second derivative of loss function, called **Hessian** matrix
  - Newton's method for finding minimum
    - $x^{t+1} = x^t - (\nabla^2 f(x^t))^{-1} \nabla f(x^t)$, where $\nabla^2 f(x^t)$ is the Hessian matrix of f at x^t .

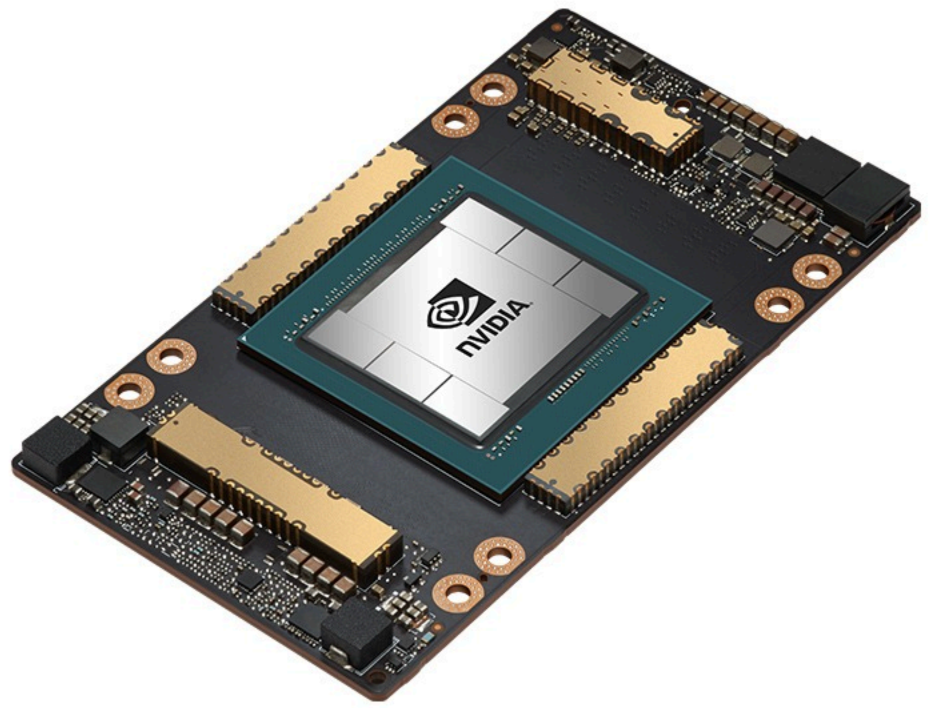
- Very effective in (convex) quadratic problems
  - $f(x) = \frac{1}{2} x^T A x - b^T x$
  - where $\nabla^2 f(x) = A, \nabla f(x) = Ax - b$
  - Newton's method: $x^* \leftarrow x - (\nabla^2 f(x))^{-1} \nabla f(x) = x - A^{-1}(Ax - b) = A^{-1} b$
  - $x^* = A^{-1} b \iff Ax^* - b = 0 \iff \nabla f(x^*) = 0$

Motivation

Second-order optimization

- However… it's widely considered as not suitable for deep learning models
  - Calculating Hessian and its inversion has massive computational demand $O(n^3)$
    - ChatGPT-4 has ~1.7×10^{12} params
    - Calculate inversion of Hessian need ~5×10^{36} floating point parameters
    - ~4×10^{27} GB GPU memory (8 bytes per floating point number)
    - GPU with largest memory: NVIDIA A100 80GB

- Our goal: how to make it practical for DL?



NVIDIA A100 for HGX

Methods

How to make second-order optimization practical for DL?

- Choose small dataset & shallow, simple network as a starting point
    - allows calculation of Hessian and its inversion

- Dataset: MNIST
- Training framework: JAX
- Device: NVIDIA V100 32GB
- Model: input -> linear layer -> ReLU -> linear layer -> prediction

Implementation

Experiment #1: Newton - doesn’t converge; slow.

$$w^{t+1} = w^t - H^{-1}g$$

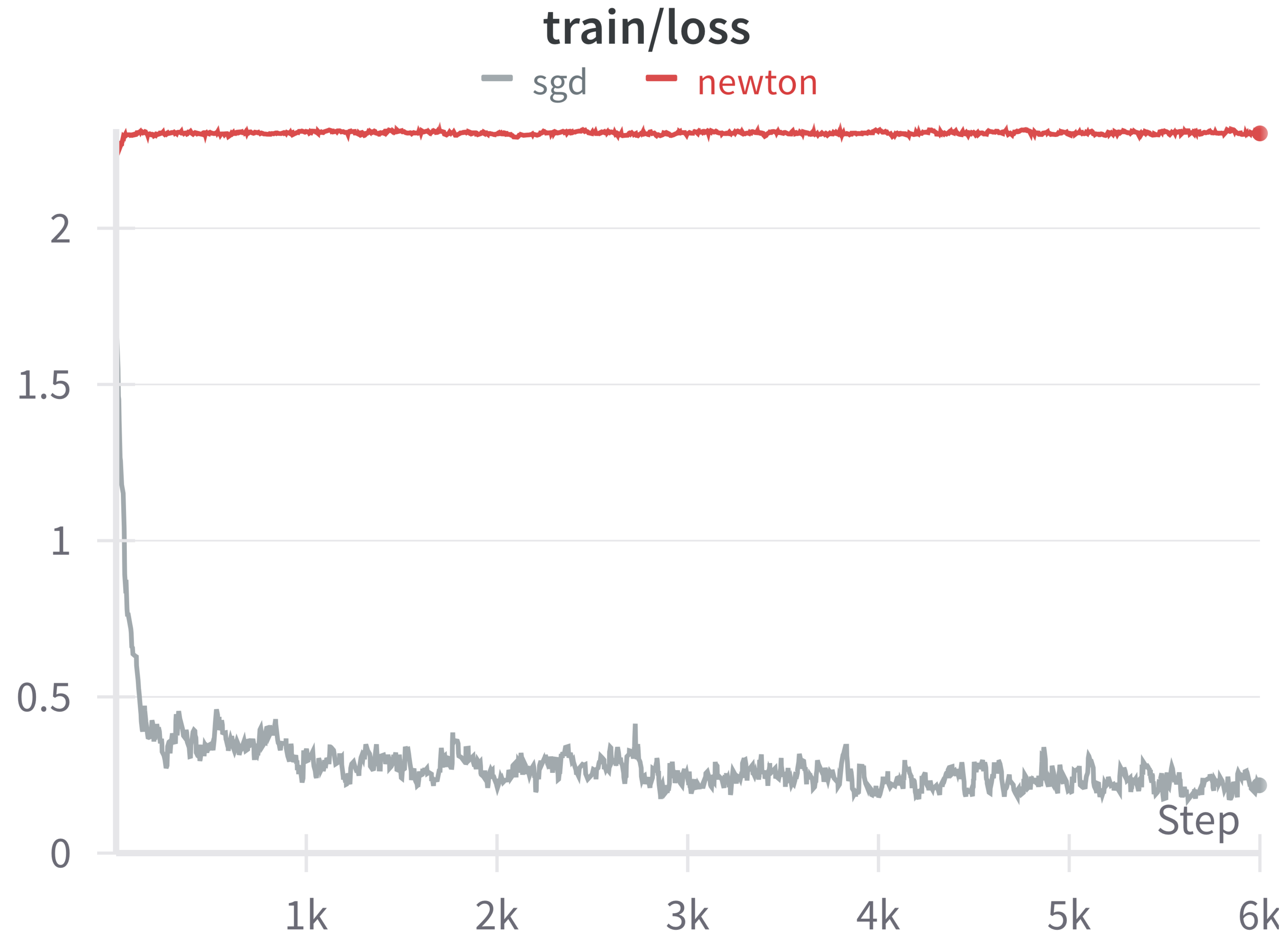


Fig. 1: SGD and Newton training loss comparison based on steps.

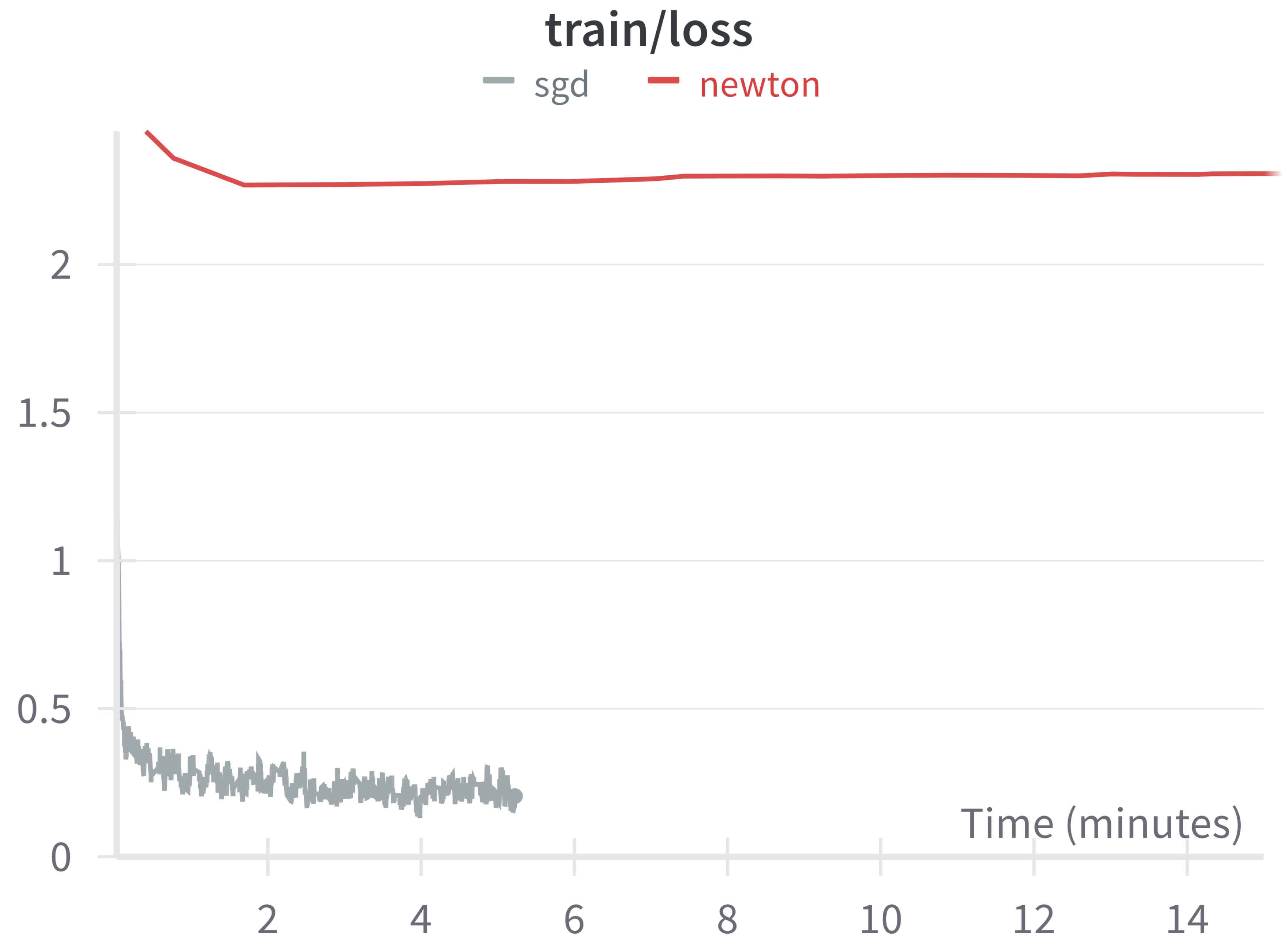


Fig. 2: SGD and Newton training loss comparison based on time.
SGD: ~6mins; Newton: ~6h.

Implementation

Experiment #1: Newton - doesn't converge; slow

- Why?
  - Loss is non-convex
  - Hessian might not be positive definite -> local maxima, saddle point…

- In order to make Hessian PD
  - We add "damping" parameter λ
  - $w^{t+1} = w^t - (H + \lambda I)^{-1} g$

Implementation

Experiment #2: Newton with damping - still slow

$$w^{t+1} = w^t - (H + \lambda I)^{-1} g, \lambda = 32$$

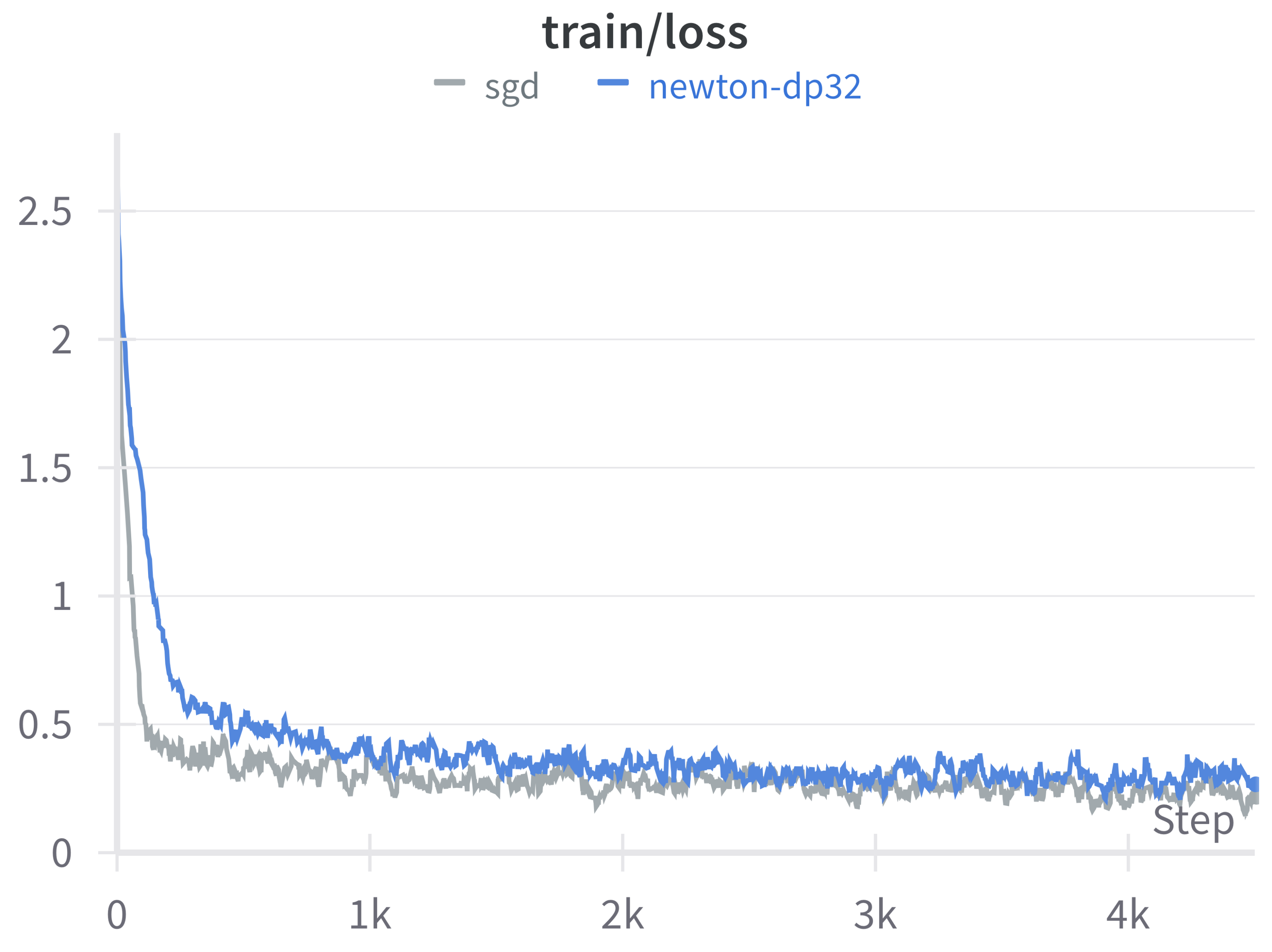


Fig. 3: SGD and Newton training loss comparison based on steps

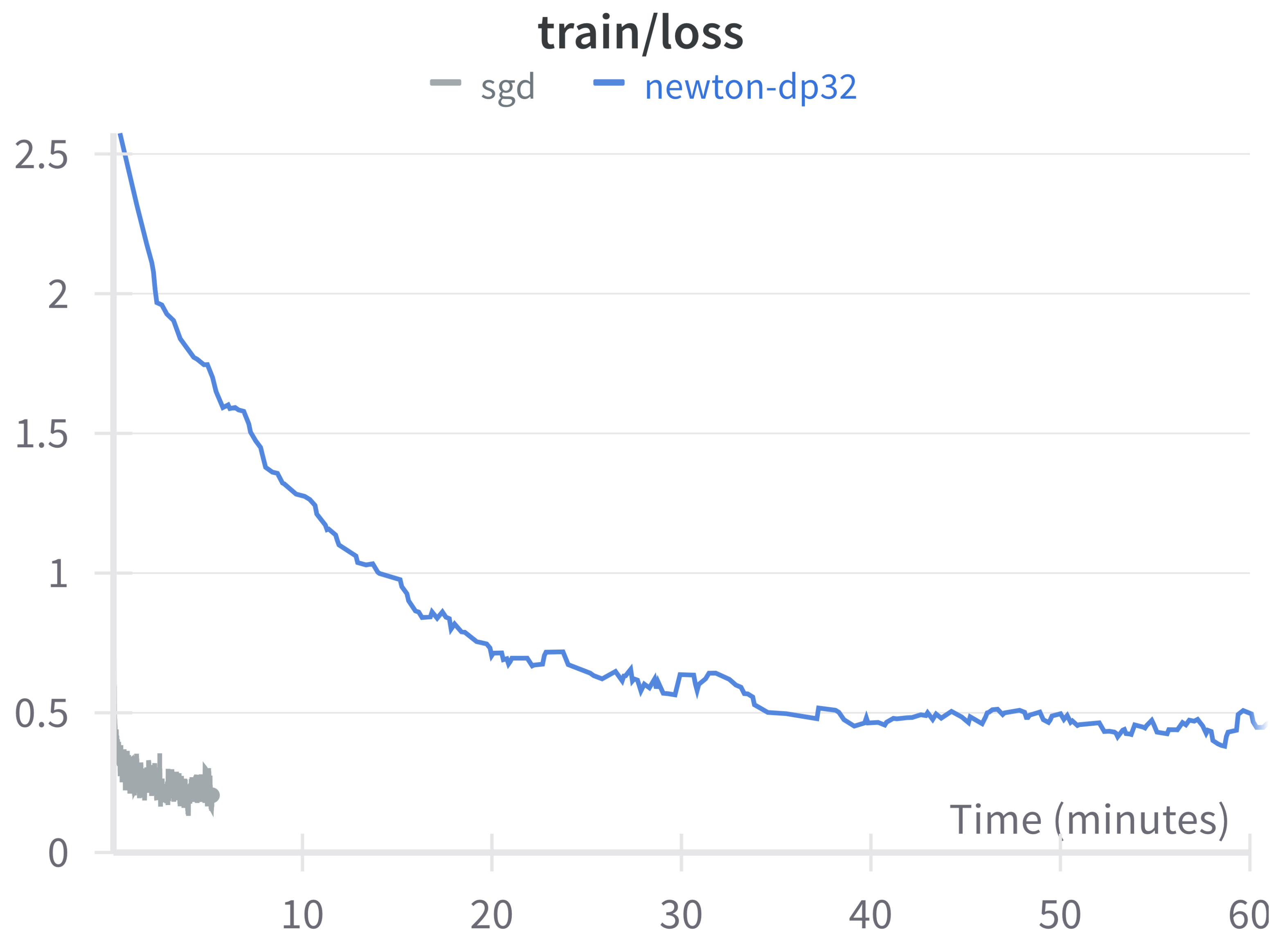


Fig. 4: SGD and Newton training loss comparison based on time

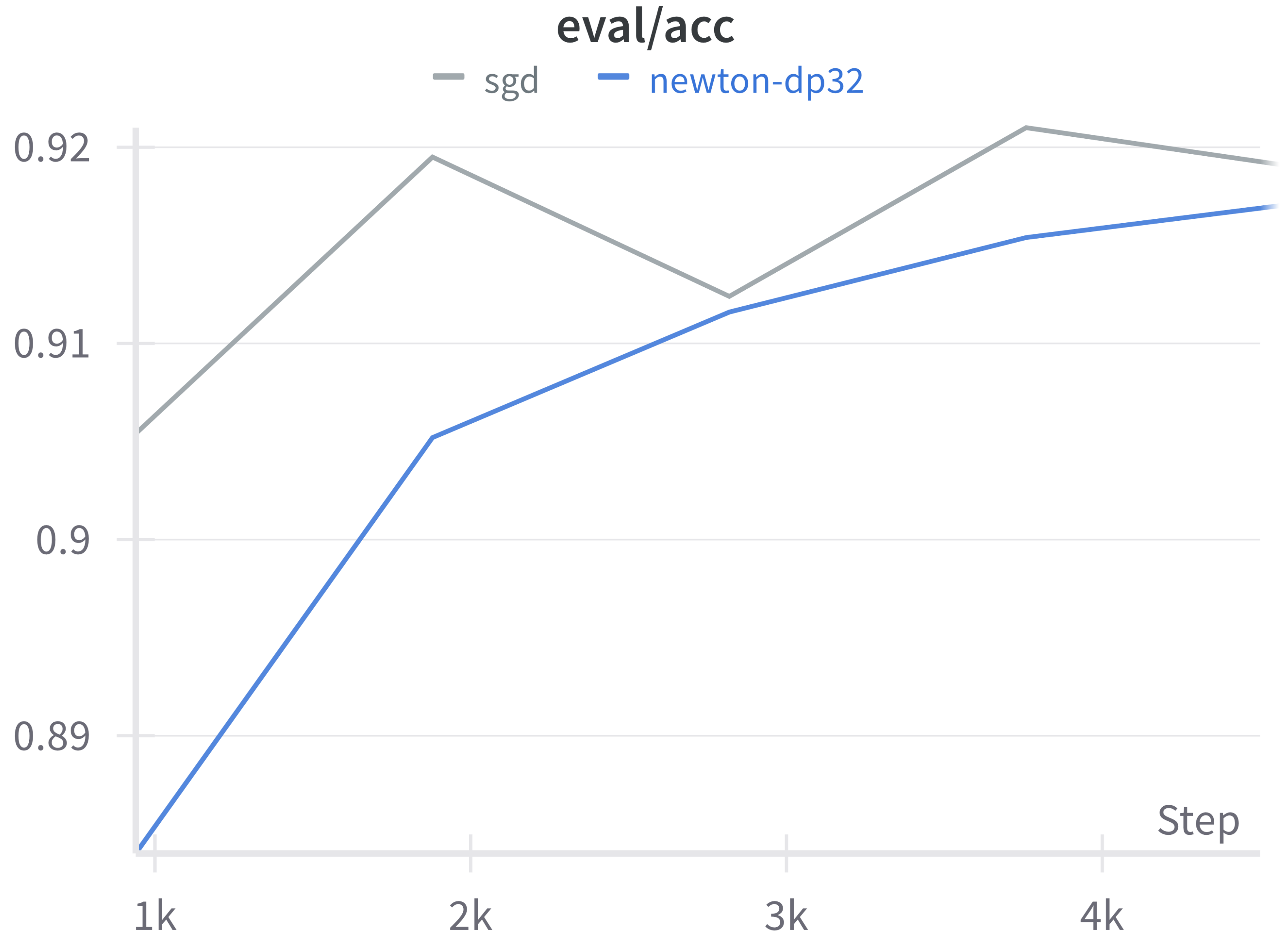


Fig. 5: SGD and Newton test acc comparison

Implementation

Experiment #2: Newton with damping - still slow

- Can we get rid of calculating Hessian and its inversion?
  - Hessian-free method [1]
  - We want to get $H^{-1}g$
  - To approximately solve x in $Hx = g$ (H still should be PD and need damping), we can use matrix-vector product (MVP) and conjugate gradient (CG)
    - MVP is easy to implement in JAX

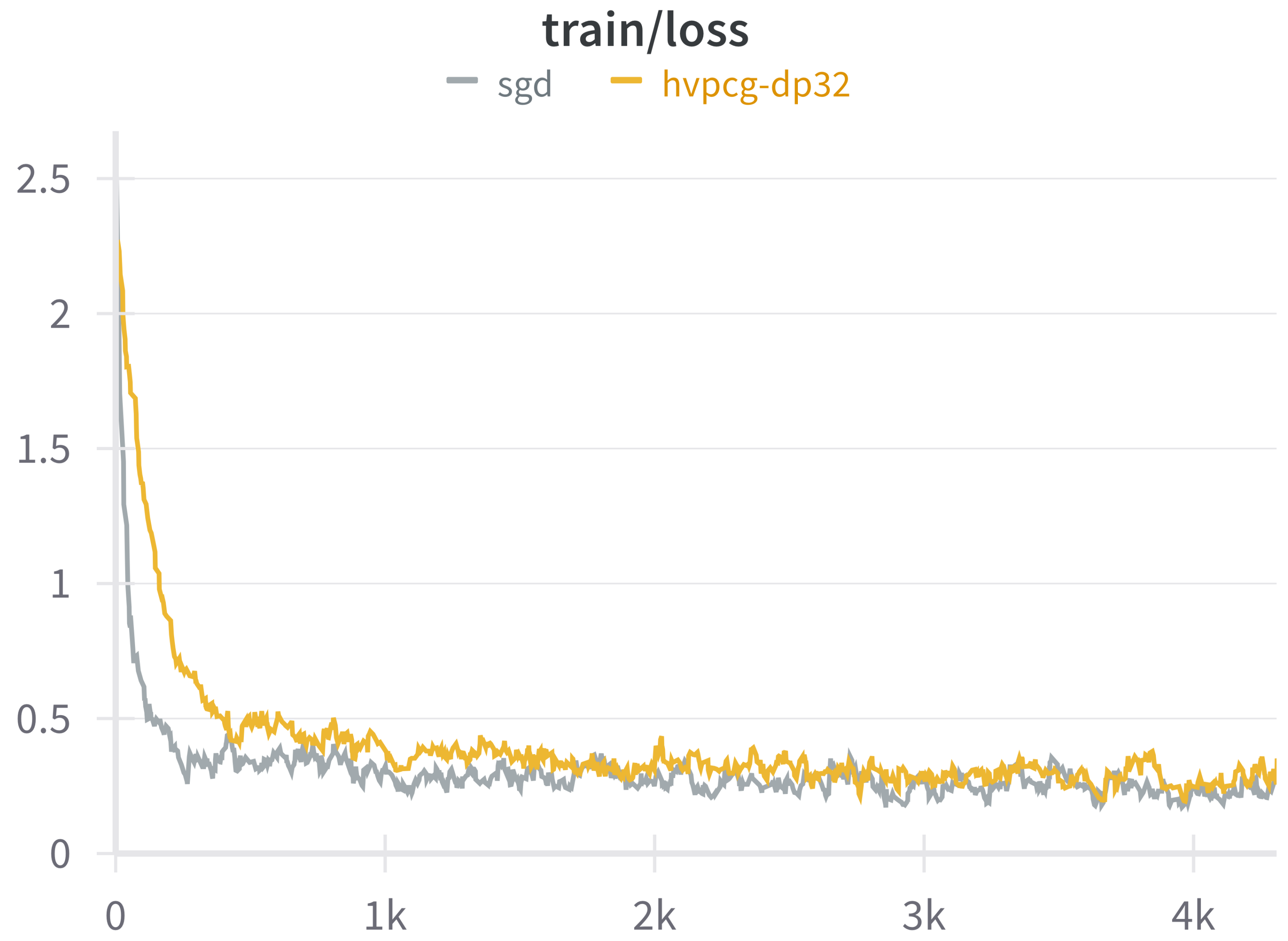
```
def hvp(f, x, v):
    return grad(lambda x: jnp.vdot(grad(f)(x), v))(x)
```

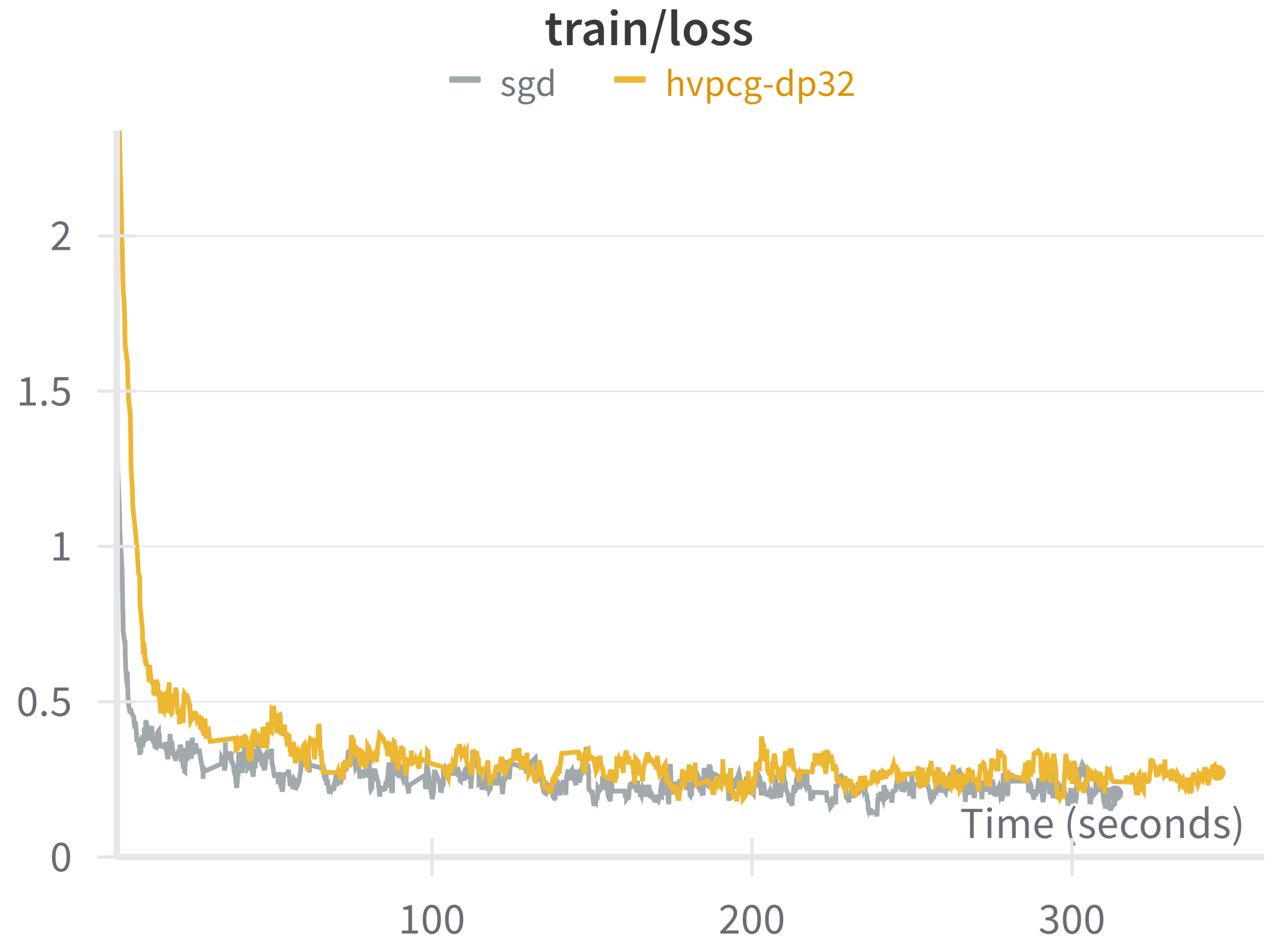
[1] James Martens. Deep learning via Hessian-free optimization.

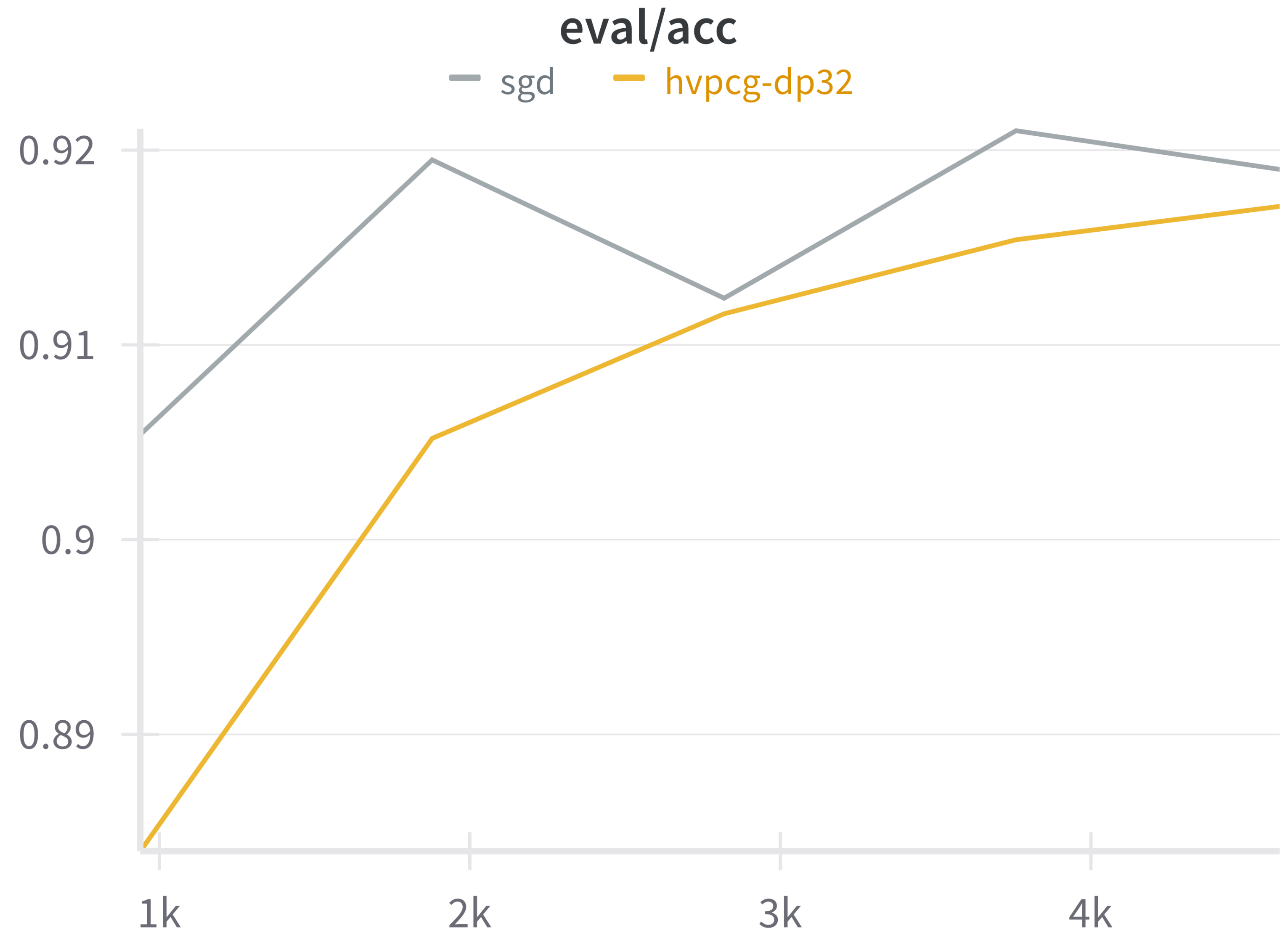
Implementation

Experiment #3: Hessian-free Newton (MVP + CG)

$$w^{t+1} = w^t - (H + \lambda I)^{-1} g, \lambda = 32$$







Speed is on par with that of SGD!

Fig. 6: SGD and MVP-CG training loss comparison based on steps

Fig. 7: SGD and MVP-CG training loss comparison based on time

Fig. 8: SGD and MVP-CG test acc comparison

Implementation

Experiment #3: Hessian-free Newton (MVP + CG)

- This method still depends on damping parameter
- $J_{x,t}(w) = L(f(w,x),t)$, decompose its Hessian, we can get
- $$\nabla^2 J_{x,t}(w) = J_{zw}^T H_z J_{zw} + \sum_a \frac{\partial L}{\partial y_a} \nabla_w^2 [f(x,w)]_a$$
- where $H_z = \nabla_z^2 L(z,t)$ is the output Hessian
- If we drop the second term, we can get Gauss Newton Hessian
- $G = J_{zw}^T H_z J_{zw}$ involves first derivatives of the network and the second derivatives of the loss function. Guaranteed to be positive semidefinite.

Implementation

Experiment #4: Hessian-free Gauss-Newton (MVP + CG)

- Still add damping to get better performance $w^{t+1} = w^t - (G + \lambda I)^{-1} g, \lambda = 4$
-

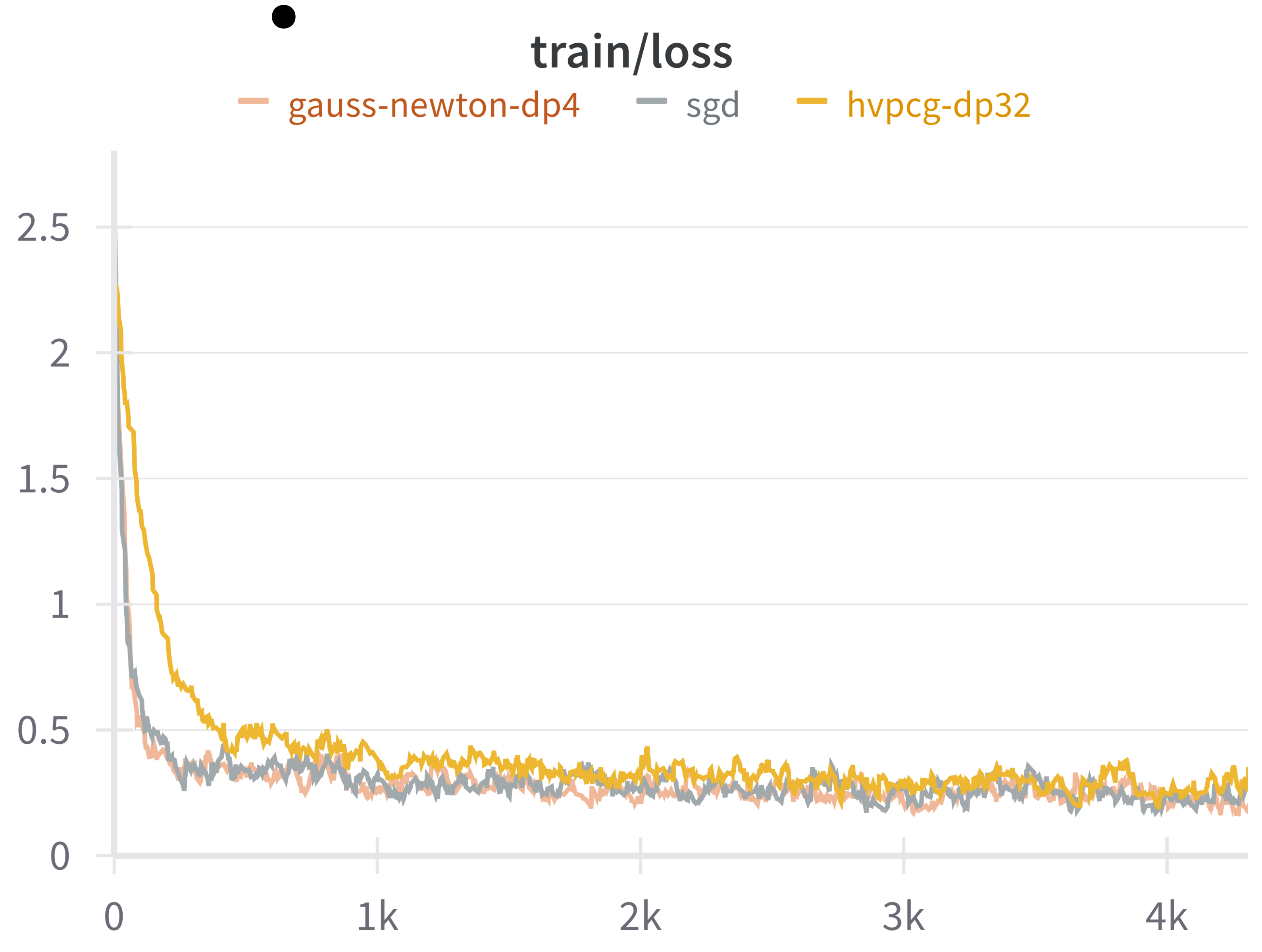


Fig. 9: Training loss comparison based on steps

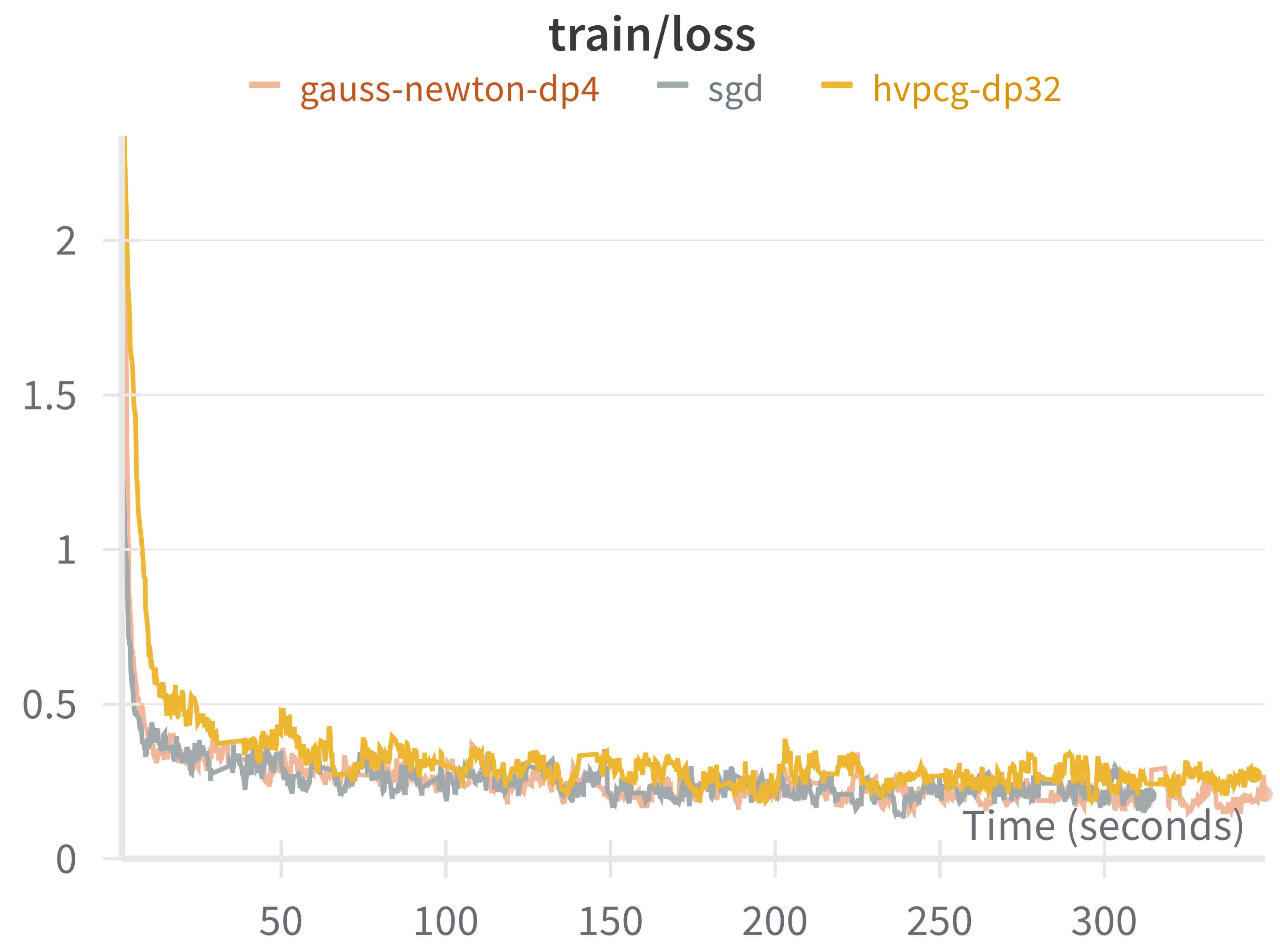


Fig. 10: Training loss comparison based on time

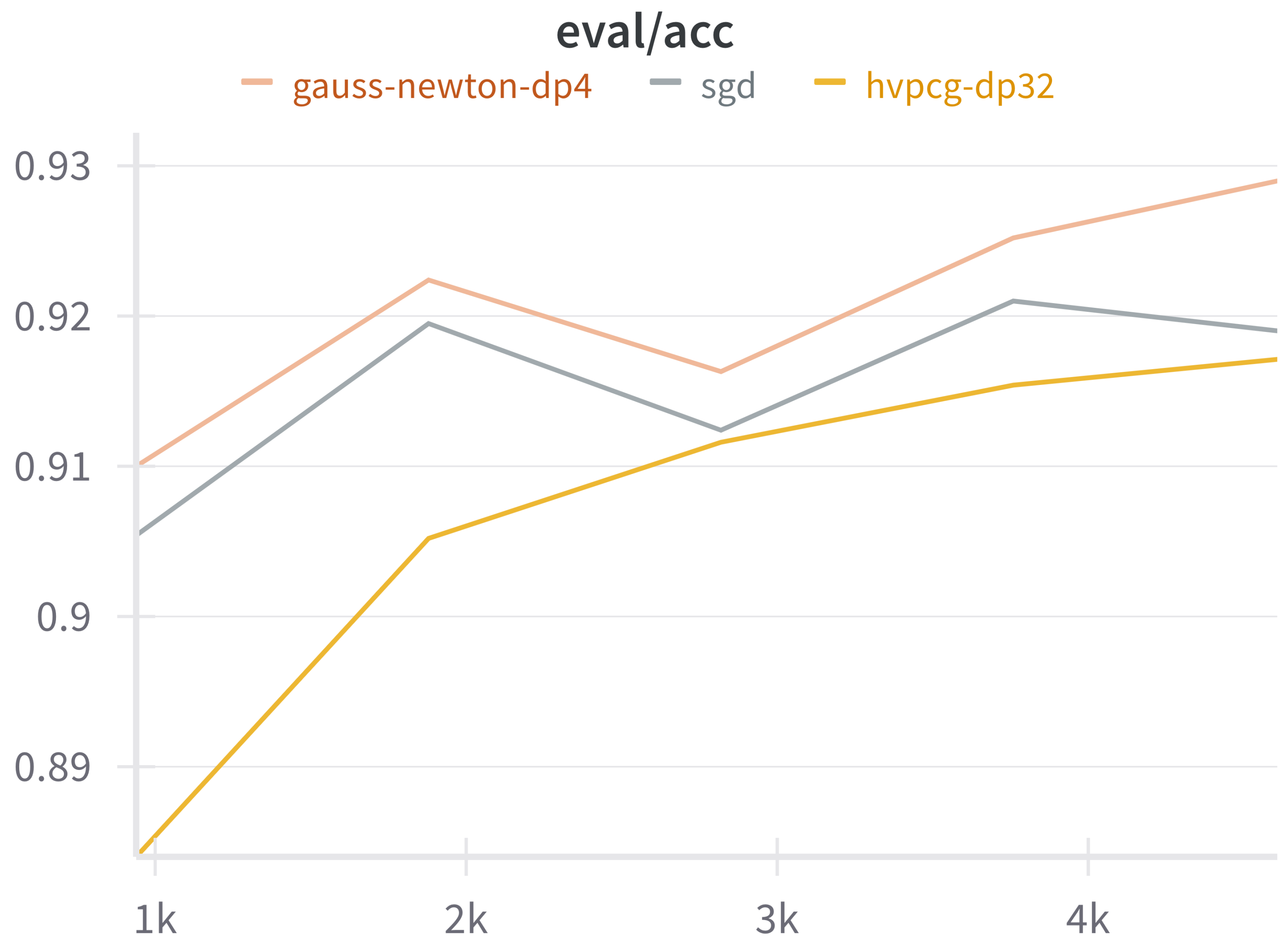


Fig. 11: Test acc comparison

Gauss-Newton receives best results

Future work

- We proved that second-order optimizers are practical in DL models
- Levenberg-Marquardt method for adjusting damping parameter
- Newton-Lanczos method for calculating trust region