Optimizing Millions of Hyperparameters with Implicit Differentiation

Jonathan Lorraine, Paul Vicol, David Duvenaud

VECTOR INSTITUTE

UNIVERSITY OF TORONTO

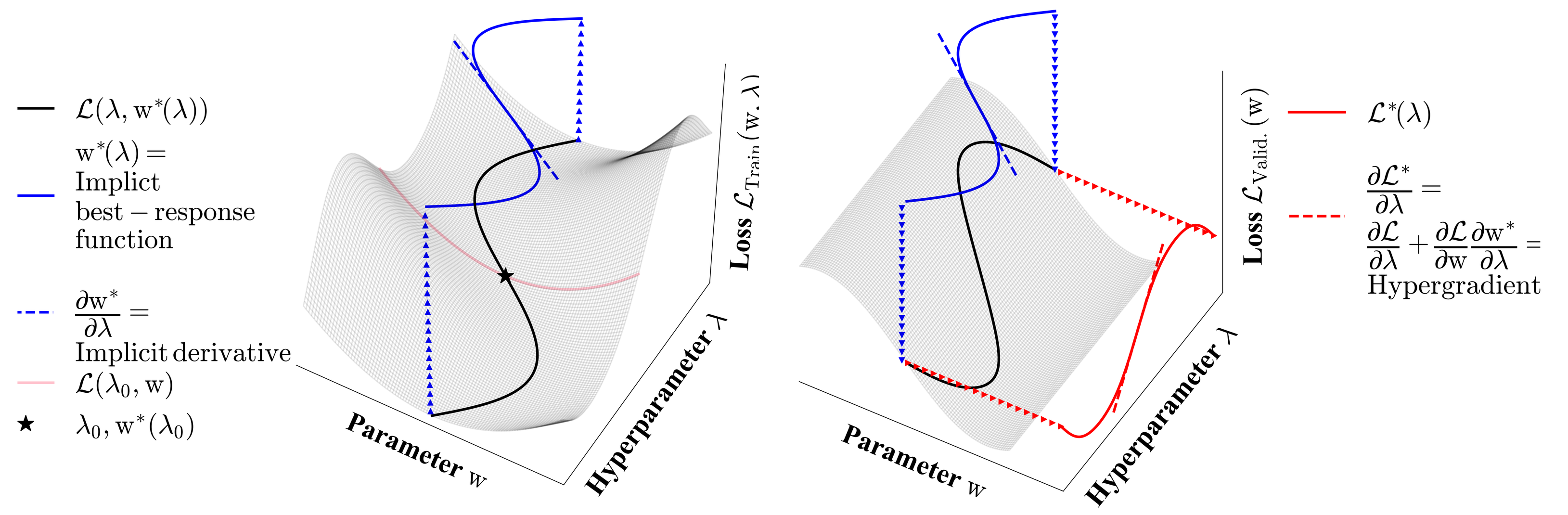
Motivation and Contributions

- Tuning hyperparameters can be critical for generalization, but **difficult when hyperparameters are high-dimensional**.
- If we can tune as many hyperparameters as weights, we have a new paradigm of learned, flexible regularization.
  - Learned data augmentation
  - Data distillation
  - Tuning millions of regularization hyperparameters
- We **scale** gradient-based hyperparameter optimization to high dimensions by combining **implicit differentiation with efficient & stable inverse-Hessian approximations**.

Hyperparameter Optimization is Nested Optimization

$$\boldsymbol{\lambda}^* := \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V^*(\boldsymbol{\lambda})$$

$$\text{where } \mathcal{L}_V^*(\boldsymbol{\lambda}) := \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) \text{ and } \mathbf{w}^*(\boldsymbol{\lambda}) := \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w})$$



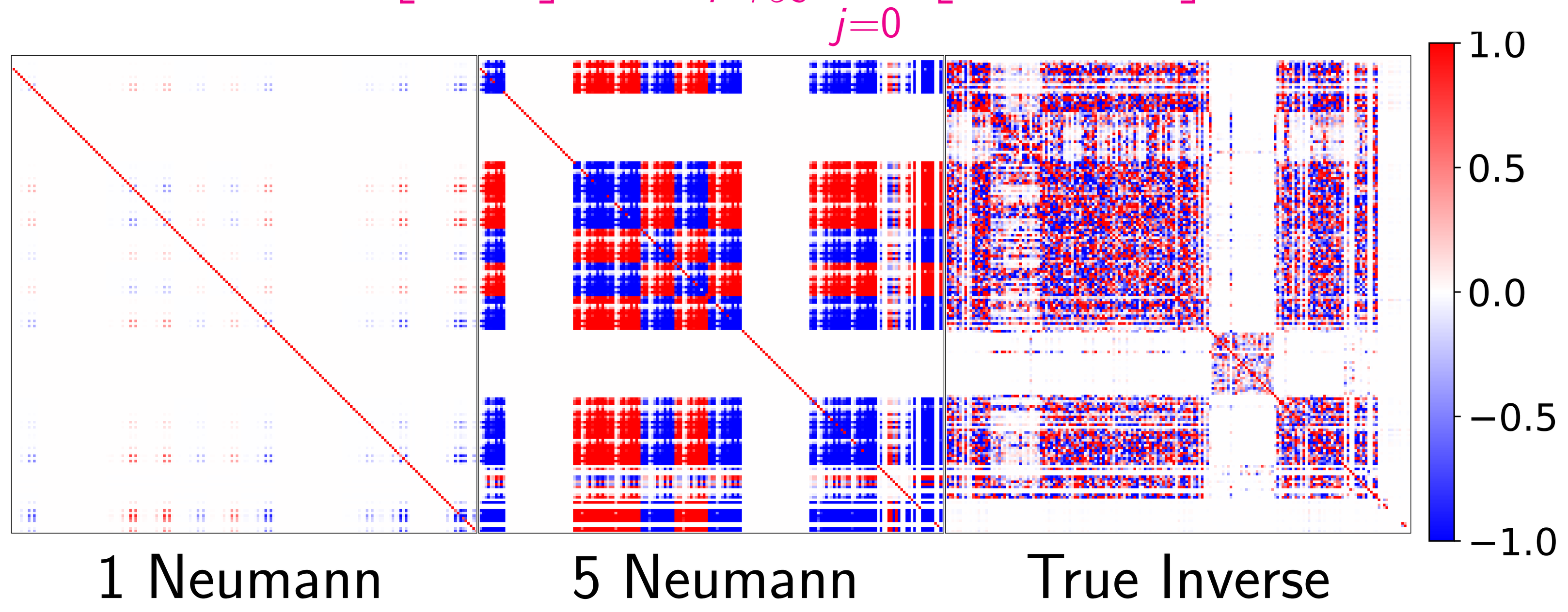
Hypergradients with Implicit Differentiation

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \underbrace{\cancel{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}^{\,0}}_{\text{hyperparam direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \times \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparam indirect grad.}}$$

$$\left.\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}'} = -\underbrace{\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1}}_{\text{training Hessian}} \times \underbrace{\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}}_{\text{training mixed partials}}\Bigg|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')} \quad \text{(implicit differentiation)}$$

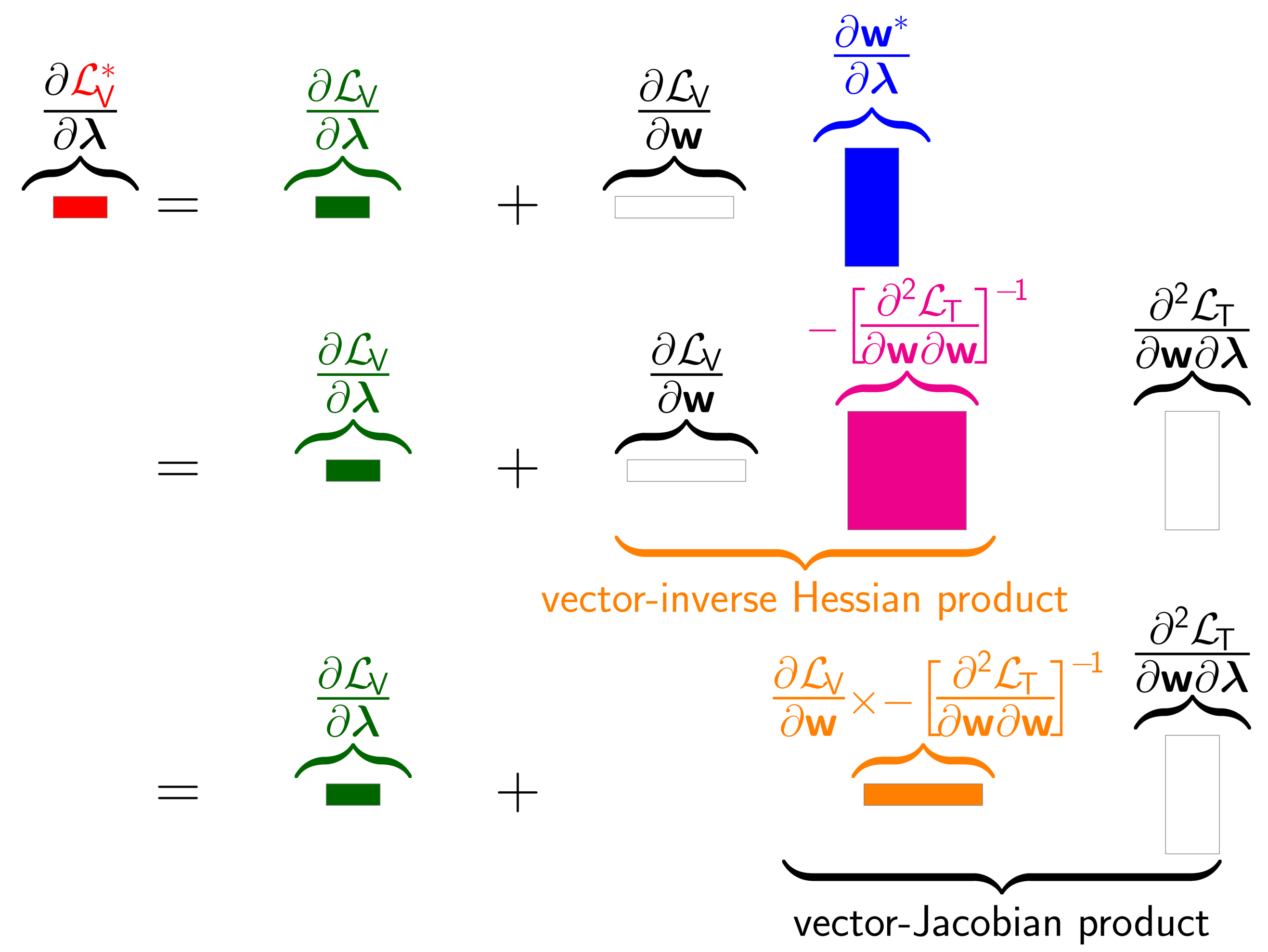
Inverse-Hessian approximations via Neumann series

$$\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} = \lim_{i \to \infty} \sum_{j=0}^{i} \left[I - \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^j$$



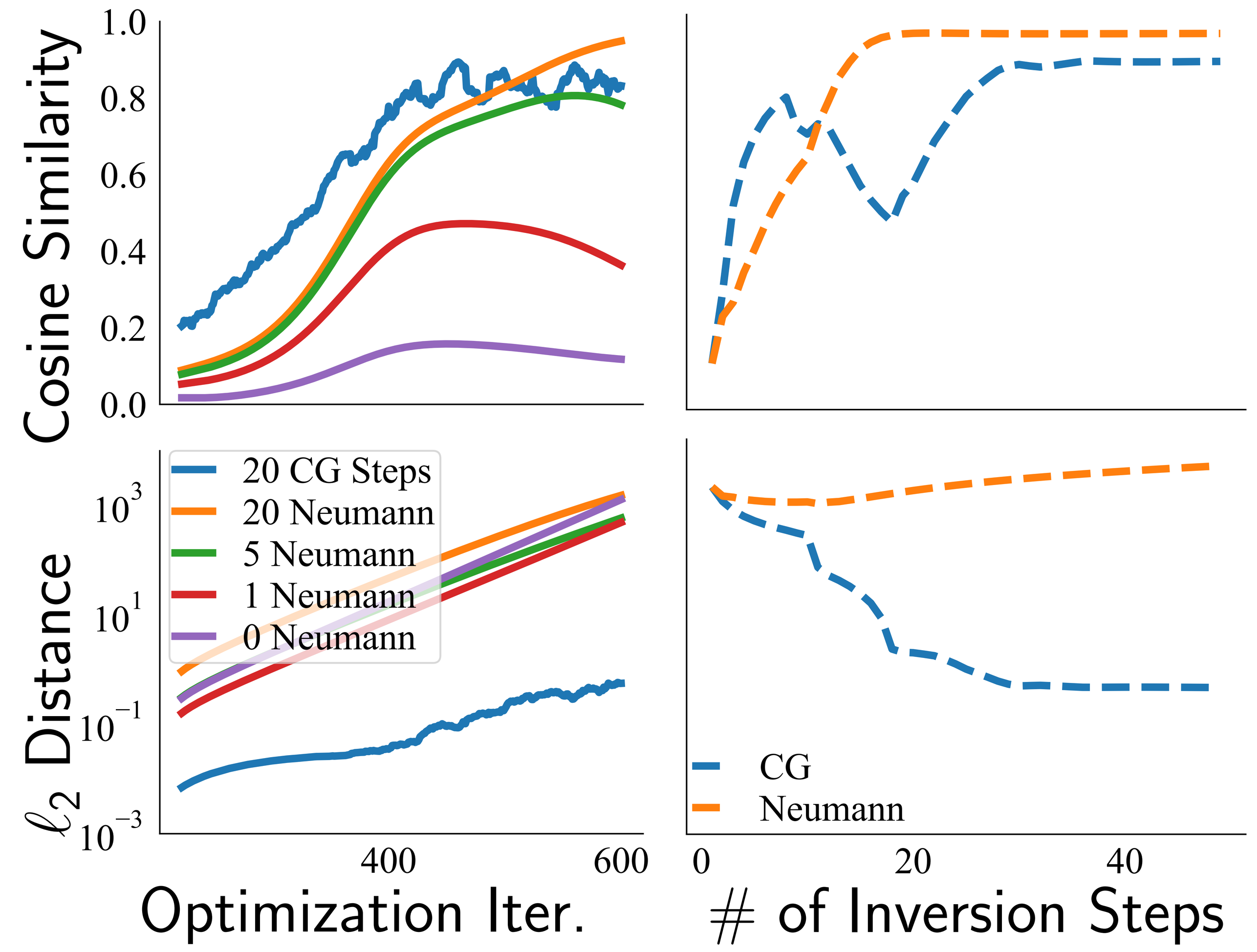
Calculating Hypergradients Efficiently

```
while not converged do
    for k = 1 ... N do
        w' -= ∂L_T/∂w |_{λ',w'}
    λ' -= hypergradient(L_V, L_T, λ', w')
return λ', w'
```

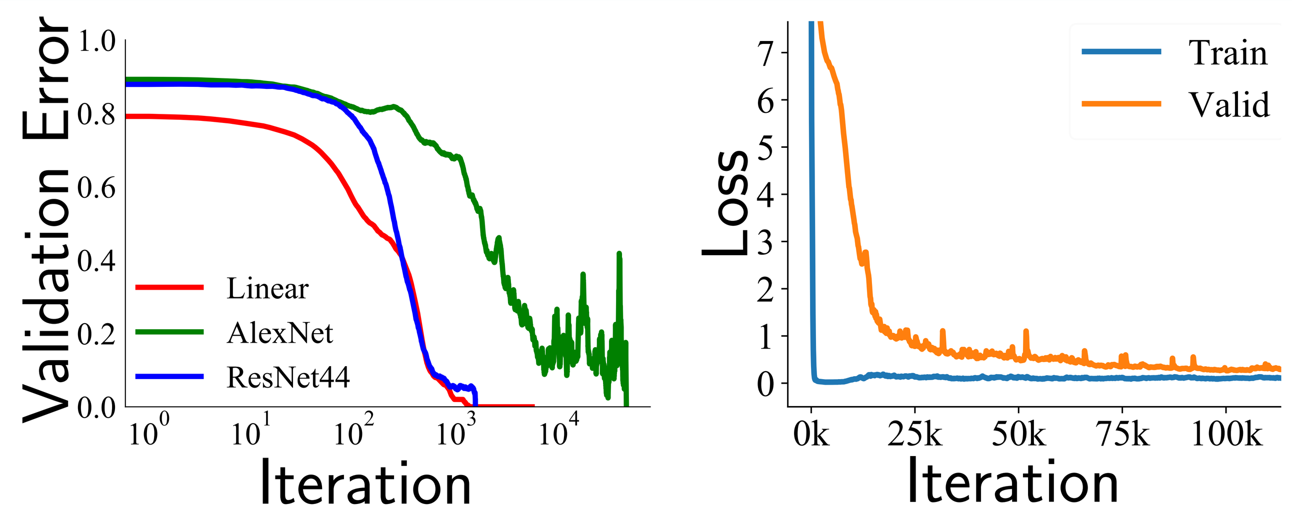


The entire hypergradient computation can be performed efficiently using **vector-Jacobian products**, given an inverse-Hessian-vector approximation.

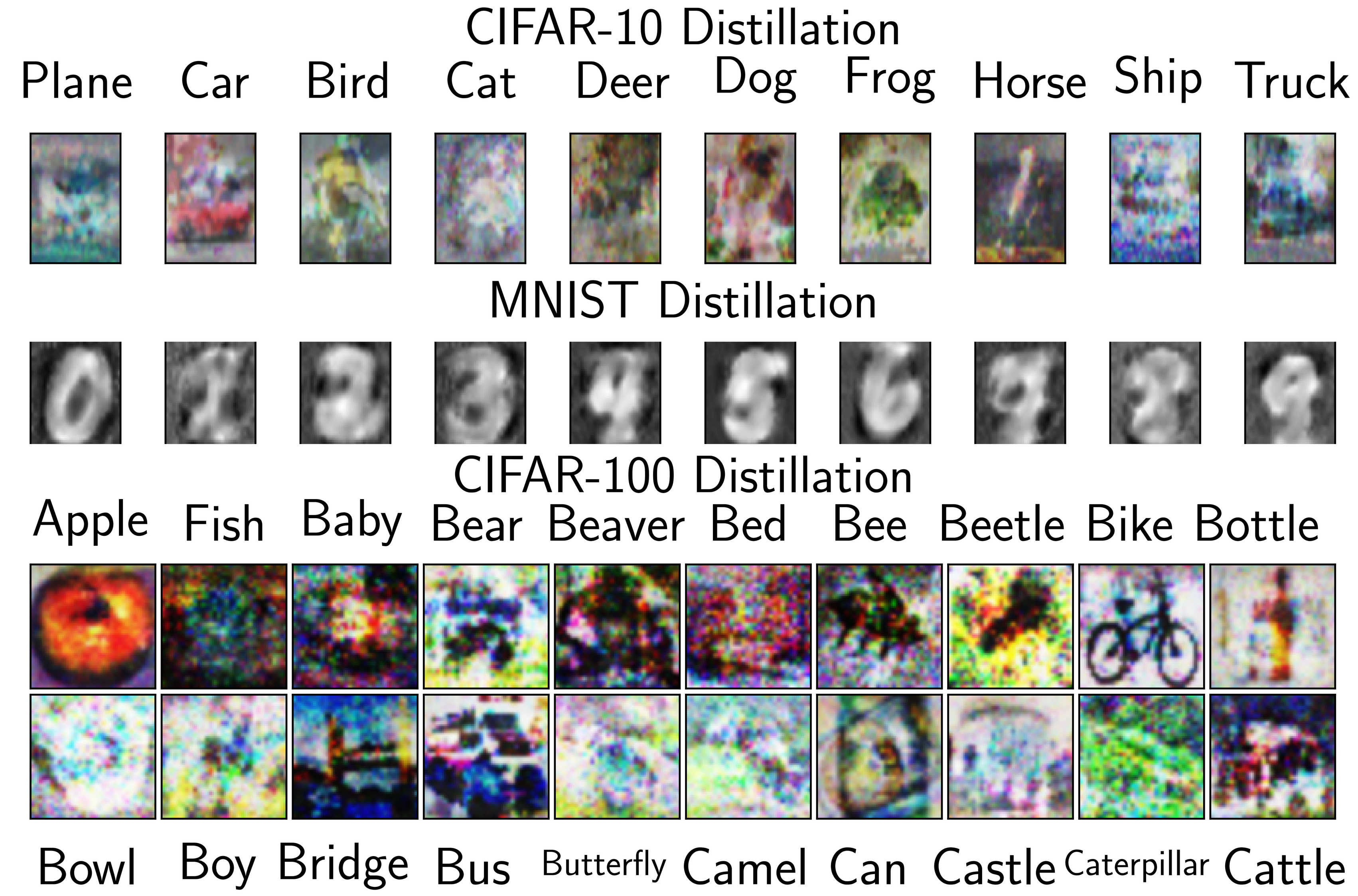
Comparing Inverse-Hessian Approximations



Overfitting the Validation Set



Data Distillation



We learn **one distilled image per class**, so after training a logistic regression classifier on the distillation, it generalizes to the rest of the data.

Learned Data Augmentation

We train a U-net for held-out data with weights as hyperparameters, which **takes data & noise, and outputs augmented data**.

Tuning Regularization Parameters

Method	Validation	Test	Time(s)
Grid Search	97.32	94.58	100k
Random Search	84.81	81.46	100k
Bayesian Opt.	72.13	69.29	100k
STN	70.30	67.68	25k
Ours	69.22	66.40	**18.5k**
Ours, Many	**68.18**	**66.14**	**18.5k**

For LSTMs on PTB, we tune the same 7 (and millions of) hyperparameters **faster, with comparable memory, and to a lower perplexity**.