CSC 2541: Neural Net Training Dynamics

Lecture 12 - Bilevel Optimization and Generalization

Roger Grosse

University of Toronto, Winter 2022

Empirical Deep Learning

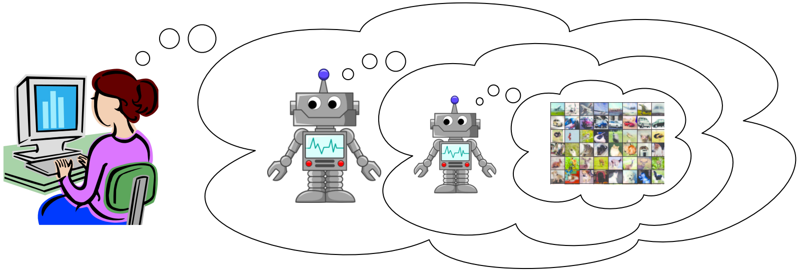
- Studying neural nets presents an unusual set of scientific challenges
- AI used to feel like engineering
  - Start with a goal (optimization, prediction, etc.), ask how a rational agent would solve it, and figure out how to implement that solution
- Now we're doing a lot more reverse engineering
  - The neural net somehow (apparently) solves a problem, and we have to figure out how
- This course didn't give the answers, but it did cover some conceptual tools we need to look for the answers
  - Linearization, metrics, implicit regularization, stochasticity, infinite limits, dynamical systems, etc.

Bilevel Optimization

- Much of the progress of AI has been about automating aspects of AI engineering
  - Hand-coded knowledge $\Rightarrow$ statistical learning
  - Hand-coded reasoning $\Rightarrow$ SAT solvers, probabilistic inference
  - Feature engineering $\Rightarrow$ deep learning
- What's next?
  - Hyperparameters, optimizers, architectures, regularizers, curricula, data augmentation strategies, self-supervised learning objectives, search algorithms, debiasing
- In principle, much of this can be formulated as bilevel optimization
- Our understanding of bilevel optimization is comparable to deep learning circa 2008. Things sometimes work if we get lucky

Bilevel Optimization and NNTD

- Understanding bilevel optimization, meta-learning, etc. requires thinking about NNTD in both the inner and outer levels



This Lecture

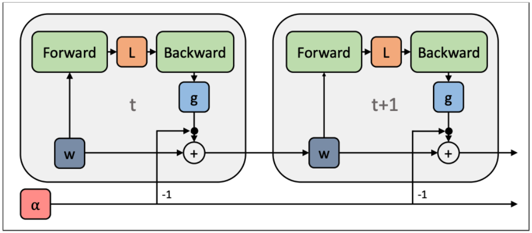
- Last week covered bilevel optimization from an optimization perspective: how to find a Stackelberg equilibrium
- This week: does your solution generalize...
  - ...to unrolling the inner optimization for much longer?
  - ...to restarting the inner optimization from scratch?
  - ...beyond the training data?
- Understanding these issues requires essentially everything we've covered in this course.

Short-Horizon Bias

(Based on Wu et al., “Understanding short-horizon bias in stochastic meta-optimization”)

Short-Horizon Bias

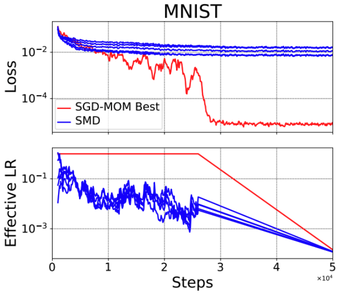
- We saw that it's possible (in principle) to learn an optimizer using meta-descent
- Can we solve the much easier problem of adapting the learning rate? If we can't even do this, then meta-optimization is hopeless!
- We'll even ignore the computational cost of the meta-optimization itself and just ask if it gives a reasonable solution.

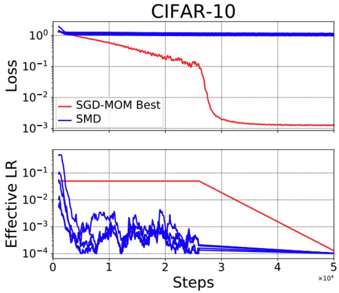


What actually happens?

Short-Horizon Bias

- In online stochastic meta-descent (SMD), over the course of a training run, we periodically update the learning rate with unrolled differentiation with a short horizon (in this case, 5 steps).
- In order to optimize the meta-objective well, we perform 10 steps of meta-descent (using Adam) for each SGD step.
  - This is too expensive for practical use, but let's ignore that.
  - Papers sometimes erroneously report successful results with SMD because they failed to optimize the meta-objective well enough.



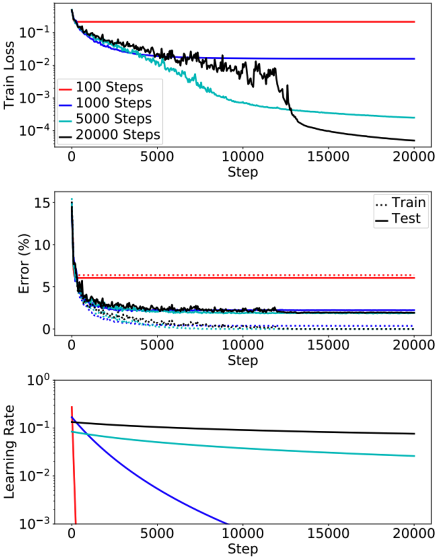


Short-Horizon Bias

- In offline SMD, we unroll many short training runs, and adapt the hyperparameters of a learning rate schedule.

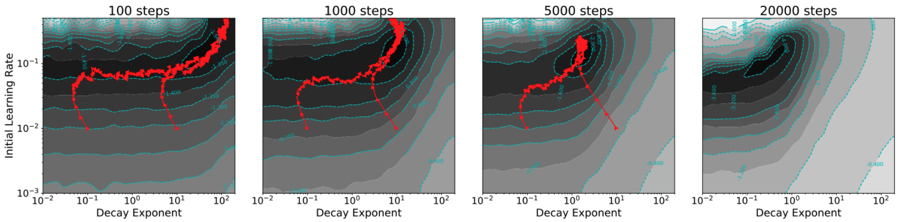
$$\alpha_k = \frac{\alpha_0}{(1 + k/K)^\beta}$$

- Hyperparameters α_0, K, β
- Estimate hypergradient with unrolling
- Evaluate the validation loss after $\{100, 1000, 5000, 20000\}$ steps (the horizon)
- MNIST dataset



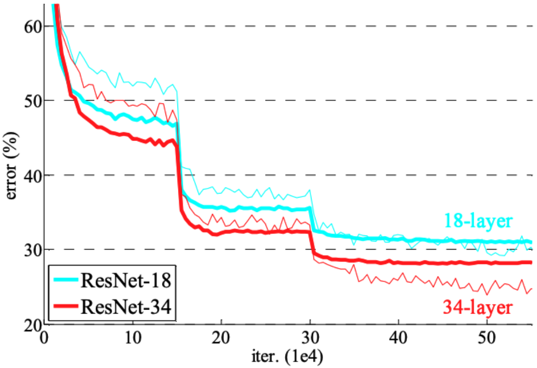
Short-Horizon Bias

Hyperparameter trajectories for different horizons



Why does this happen?

Short-Horizon Bias



(He, 2015, "Deep residual learning for image recognition")

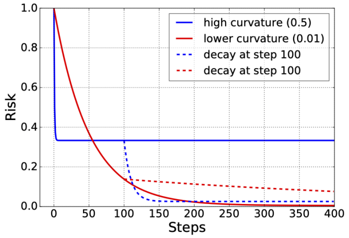
- **The intuition:** you get a big immediate reduction in training error as soon as you decay the learning rate.
- Therefore, an adaptive learning rate method with a short horizon will decay it very quickly.
- Can we model this phenomenon mathematically?

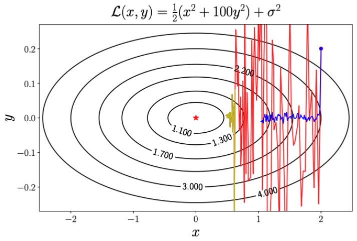
Short-Horizon Bias

- If you are minimizing a deterministic quadratic objective using gradient descent with momentum, then the greedy choice of α and β is optimal!
  - This is because it's equivalent to conjugate gradient (Lecture 9).
- Remember the Noisy Quadratic Model? (Lecture 7)
  - If the curvature and the gradient noise are both spherical, then the greedy choice of α for SGD is optimal!
  - Each coordinate evolves independently, so it reduces to the scalar case.
  - The state of the system can be summarized with a single statistic, $\mathbb{E}[\theta^2]$. We'd always prefer for this to be smaller (in terms of the achievable loss at some later iteration).
  - Therefore, choosing α to minimize $\mathbb{E}[\theta^2]$ one step later is always optimal.
- The short-horizon bias only arises if the objective is both stochastic *and* ill-conditioned!

Short-Horizon Bias

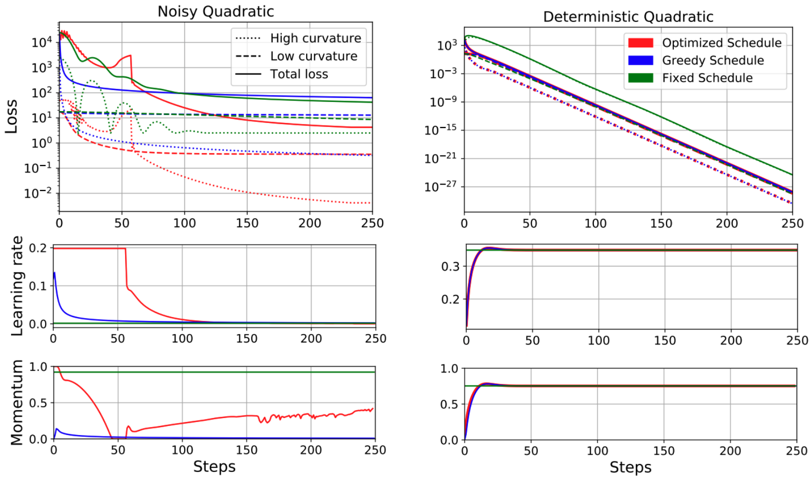
- The NQM captures the phenomenon
  - Over a short horizon, you want to use a small learning rate to reduce the effects of gradient noise
  - Over a long horizon, you want to keep a high learning rate (to make more progress in low-curvature directions) and then decay at the end (to eliminate noise)





Short-Horizon Bias

- Using dynamic programming, we can determine the expected loss under any learning rate and momentum schdule (Lecture 7)
- We can optimize the schedule using dynamic programming



Short-Horizon Bias

- The NQM gives a clear model for why short horizons bias us towards small learning rates
  - This is a tough problem to get around, since large learning rates help by making progress in low-curvature directions, which is invisible if you only measure the loss over the short term
  - Maybe measuring more information would make meta-descent work? But what information?
- I believe this is a fundamental problem not just for meta-descent on learning rate (schedules), but also for any meta-optimizer that can express a learning rate (schedule), e.g.
  - Rescaling a preconditioner is equivalent to changing the learning rate
  - ϵ in RMSprop/Adam, damping parameter in K-FAC
  - Batch norm implicit decay effect

Implicit Bias in Bilevel Optimization

(Based on Vicol et al., “On implicit bias in overparameterized bilevel optimization”)

Non-Uniqueness

- So far, we've made a simplifying assumption that the inner objective has a unique optimum:

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) \qquad \text{s.t.} \qquad \mathbf{w}^*(\boldsymbol{\lambda}) \triangleq \arg\min_{\mathbf{w}} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \mathbf{w}).$$

- In practice, the inner objective is often overparameterized, e.g. training a neural net.
  - There's an entire manifold of optima!
- It's hard to write down a general formulation, hence the scare quotes:

$$\boldsymbol{\lambda}^* \in \text{``}\arg\min_{\boldsymbol{\lambda}}\text{''} \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \mathbf{w}^*) \qquad \text{s.t.} \qquad \mathbf{w}^* \in \arg\min_{\mathbf{w}} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \mathbf{w}),$$

Non-Uniqueness

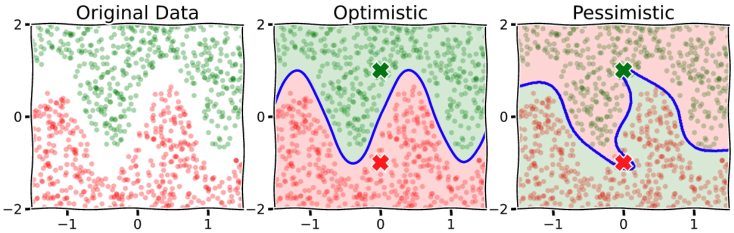
- Let $\mathcal{S}(\boldsymbol{\lambda}) = \arg\min_{\mathbf{w}} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \mathbf{w})$ denote the optimal solution set.
- Traditional ways to disambiguate the solution:
  - Optimistic: disambiguate by minimizing the outer cost

$$\mathbf{w}^*(\boldsymbol{\lambda}) = \arg\min_{\mathbf{w}\in\mathcal{S}(\boldsymbol{\lambda})} \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \mathbf{w})$$

  - Pessimistic: disambiguate by maximizing the outer cost

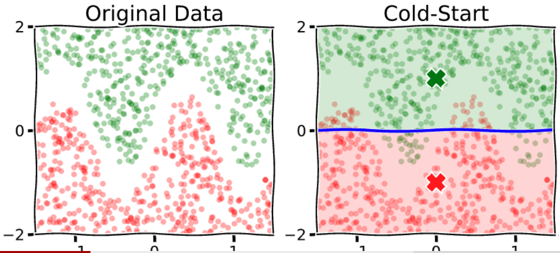
$$\mathbf{w}^*(\boldsymbol{\lambda}) = \arg\max_{\mathbf{w}\in\mathcal{S}(\boldsymbol{\lambda})} \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \mathbf{w})$$

- Consider the example of dataset distillation. Neither solution concept really makes sense here:



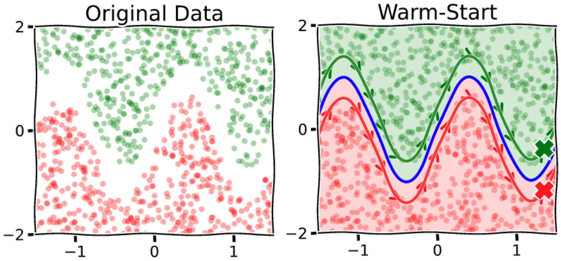
Non-Uniqueness

- More relevant to deep learning is the cold-start solution: define $\mathbf{w}^*(\boldsymbol{\lambda})$ to be the result of running an optimizer to convergence.
- We can compute the hypergradient by unrolling the entire inner optimization procedure (see Lecture 11).
- Recall from Lectures 1 and 6:
  - Gradient descent prefers solutions that minimize the distance to the initialization (exactly for linear regression, approximately for neural nets).
  - For many feature maps and neural net architectures, this creates an inductive bias towards smooth functions.
- Problem: this is very expensive!

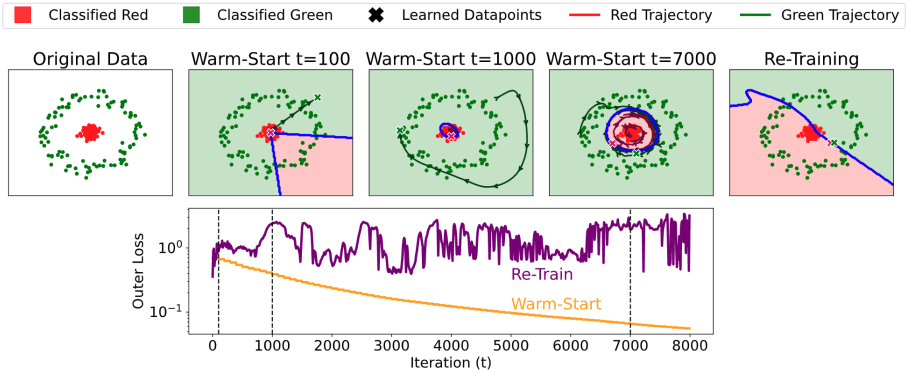


Non-Uniqueness

- In practice, we usually do warm-start training: alternate between
  - Optimize $\mathbf{w}$ to convergence (or just for a handful of steps) starting from its current value
  - Update $\boldsymbol{\lambda}$ using the hypergradient
- Now $\mathbf{w}$ is implicitly regularized to be close to its previous value, not the initialization. This gives the optimization procedure a memory of past solutions.

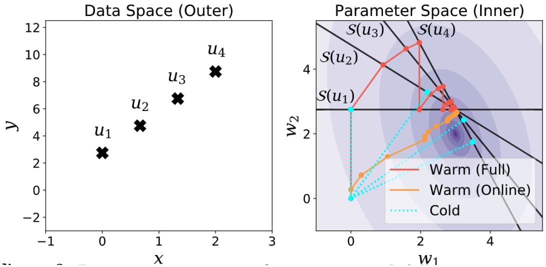


Non-Uniqueness

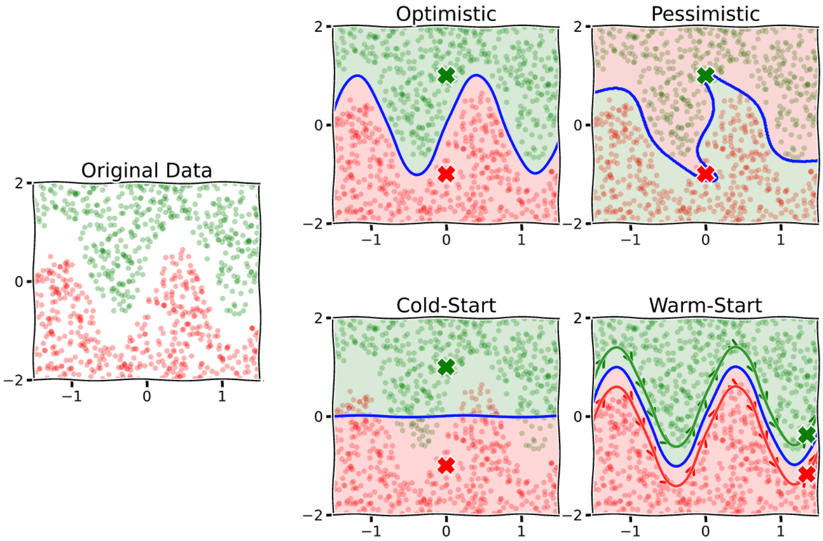


Non-Uniqueness

- Why does warm-start optimization have a memory?
- For simplicity, suppose we're not optimizing $\boldsymbol{\lambda}$, but cycling through a fixed set of values.
- Inner problem: linear regression with one data point.
- The warm-start procedure is equivalent to the Kaczmarz algorithm, the alternating projection method from Lecture 7.
  - Under some conditions, it converges to the intersection of the constraint set (i.e. the weights learn to fit all the training examples.)



Non-Uniqueness



Effect of Hypergradient Approximation

- Another source of implicit bias is that the hypergradient is only computed approximately.
- Recall the IFT hypergradient formula:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\lambda}}\left[\mathcal{J}_{\text{val}}(\boldsymbol{\lambda}, \mathbf{r}(\boldsymbol{\lambda}))\right] = \underbrace{\frac{\partial \mathcal{J}_{\text{val}}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}, \mathbf{r}(\boldsymbol{\lambda}))}_{\text{direct term}} + \underbrace{\left(\frac{\partial \mathbf{r}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda})\right)^{\top} \frac{\partial \mathcal{J}_{\text{val}}}{\partial \mathbf{w}}(\boldsymbol{\lambda}, \mathbf{r}(\boldsymbol{\lambda}))}_{\text{response term}}$$

$$\frac{\partial \mathbf{r}}{\partial \boldsymbol{\lambda}} = \underbrace{\left(\frac{\partial^2 \mathcal{J}_{\text{tr}}}{\partial \mathbf{w}^2}\right)^{-1}}_{=\mathbf{H}^{-1}} \frac{\partial^2 \mathcal{J}_{\text{tr}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}$$

- This requires approximation because of the $\mathbf{H}^{-1}$!

Effect of Hypergradient Approximation

- Neumann iterations are a method for solving high-dimensional linear systems.
- They are based on Neumann series: for any matrix $\mathbf{A}$ such that $\mathbf{I} - \mathbf{A}$ is invertible,

$$(\mathbf{I} - \mathbf{A})^{-1} = \sum_{k=0}^{\infty} \mathbf{A}^k$$

- Plugging in $\mathbf{A} = \mathbf{I} - \alpha \mathbf{H}$ for $\mathbf{H}$ positive definite and $\alpha < 1/\lambda_{\max}(\mathbf{H})$ (where $\lambda_{\max}(\mathbf{H})$ is the largest eigenvalue of $\mathbf{H}$):

$$\mathbf{H}^{-1} = \alpha \sum_{k=0}^{\infty} (\mathbf{I} - \alpha \mathbf{H})^k$$

Effect of Hypergradient Approximation

- Truncating this to the first K terms:

$$\mathbf{H}^{-1} \approx \alpha \sum_{k=0}^{K} (\mathbf{I} - \alpha \mathbf{H})^k$$

- Efficient recurrence for a polynomial $\mathbf{I} + \mathbf{A} + \cdots + \mathbf{A}^{K-1}$:

$$\mathbf{C}_0 = \mathbf{I} \qquad \mathbf{C}_k = \mathbf{A}\mathbf{C}_{k-1} + \mathbf{I}$$

- Using this to approximate $\mathbf{H}^{-1}\mathbf{b}$:

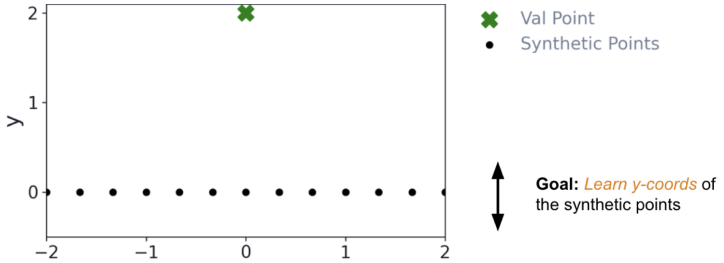
```
$\mathbf{v}_0 \leftarrow \alpha \mathbf{b}$
For $k = 1, \ldots, K-1$,
    $\mathbf{v}_k \leftarrow \alpha \mathbf{b} + (\mathbf{I} - \alpha \mathbf{H})\mathbf{v}_{k-1}$    $(= \mathbf{v}_{k-1} + \alpha \mathbf{b} - \alpha \mathbf{H} \mathbf{v}_{k-1})$
```

- Note that $\mathbf{H}\mathbf{v}_{k-1}$ is just a hessian-vector product.
- This is equivalent to gradient descent on the quadratic:

$$\mathcal{J}(\mathbf{v}) = \frac{1}{2} \mathbf{v}^\top \mathbf{H} \mathbf{v} - \mathbf{b}^\top \mathbf{v}$$

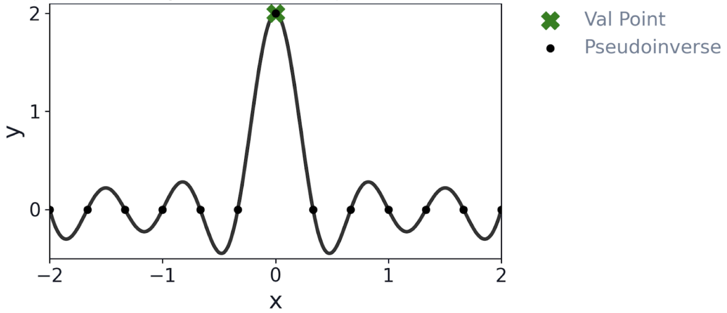
Effect of the Hypergradient Approximation

- Consider another toy problem, anti-distillation. For a 1-D regression problem, we have 1 original training point and 13 distilled points.
- Since the outer objective is overparameterized, there are many global optima.
- The inner objective is a Fourier basis regression with an inductive bias for smoothness.
- What do you think will happen if we optimize the distilled points using the *exact* hypergradient?



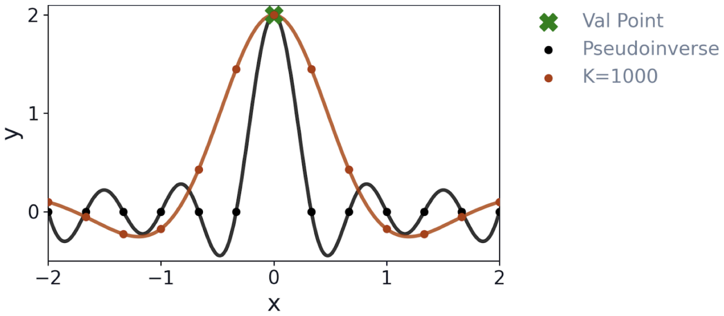
Effect of the Hypergradient Approximation

- Gradient descent on the outer objective tries to find the min-norm solution (i.e. the one closest to the initialization).
- Only the middle point matters, so the min-norm solution only changes this point.
- The inner optimizer has no choice but to fit this dataset, which it can only do with great difficulty.

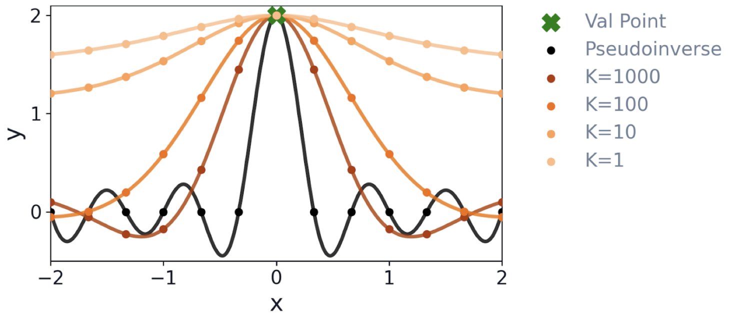


- Now what do you think will happen if we approximate the hypergradient with Neumann iterations?

Effect of the Hypergradient Approximation



Effect of the Hypergradient Approximation



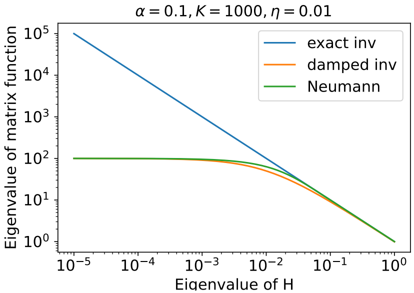
What's going on???

Effect of Hypergradient Approximation

- **Observation**:

$$\alpha \sum_{k=1}^{K} (\mathbf{I} - \alpha \mathbf{H})^k \approx (\mathbf{H} + \eta \mathbf{I})^{-1} \qquad \text{for } \eta = \frac{1}{\alpha K}$$

- The RHS is just the damped inverse (Lecture 4).
- Since the LHS and RHS are both matrix functions of **H**, they share the same eigenvectors as **H**, and their eigenvalues are functions of the eigenvalues of **H**.



Effect of Hypergradient Approximation

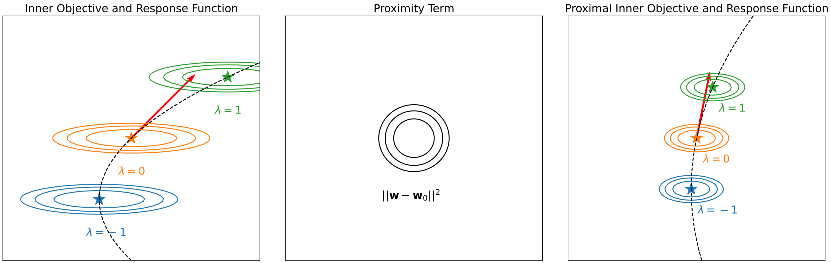
- What is the effect of approximation $\mathbf{H}^{-1}$ with $(\mathbf{H} + \eta \mathbf{I})^{-1}$ in the IFT formula?
- This is the exact hypergradient for an approximate bilevel program with a proximity term added to the inner objective:

$$\begin{aligned} \boldsymbol{\lambda}^* &= \arg\min_{\boldsymbol{\lambda}} \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) \\ \text{s.t.} \quad \mathbf{w}^*(\boldsymbol{\lambda}) &\triangleq \arg\min_{\mathbf{w}} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}, \mathbf{w}) + \frac{\eta}{2}\|\mathbf{w} - \mathbf{w}_0\|^2, \end{aligned}$$

  where $\mathbf{w}_0$ are the current inner parameters, which are an optimal solution to the *original* inner objective $\mathcal{J}_{\text{in}}$.
- Note that the proximity term is minimized at $\mathbf{w}_0$, so $\mathbf{w}_0$ is the unique optimum of the proximal inner objective.

Effect of Hypergradient Approximation

- Why doesn't the proximal hypergradient match the exact hypergradient?



Inner Objective and Response Function
$\lambda = 1$
$\lambda = 0$
$\lambda = -1$
Proximity Term
$||\mathbf{w} - \mathbf{w}_0||^2$
Proximal Inner Objective and Response Function
$\lambda = 1$
$\lambda = 0$
$\lambda = -1$

- The proximal best-response function (and hence the proximal response Jacobian) is insensitive to low-curvature directions of the inner objective.

Effect of Hypergradient Approximation

Farnia and Ozdaglar, ICML 2020, "Do GANs always have Nash equilibria?"

- This paper introduced the notion of proximal equilibrium, a solution concept that interpolates between Nash and Stackelberg equilibria.
- A pair $(\boldsymbol{\lambda}^*, \mathbf{w}^*)$ is a proximal equilibrium with parameter η if it is a solution (Stackerlberg equilibrium) to the bilevel program with the proximity term:

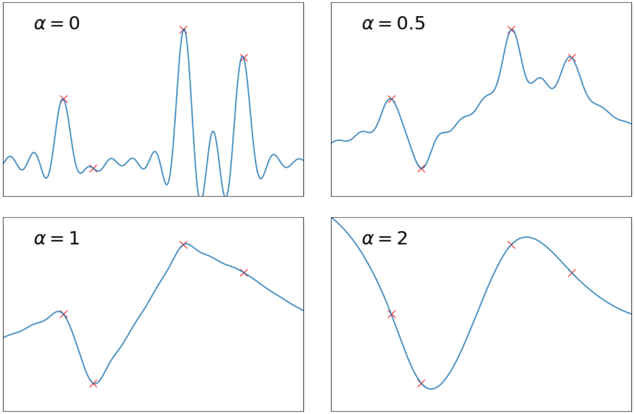
$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}} \mathcal{J}_{\text{out}}(\boldsymbol{\lambda}, r(\boldsymbol{\lambda}))$$

$$\text{s.t.} \quad \mathbf{w}^* = r(\boldsymbol{\lambda}) \triangleq \arg\min_{\mathbf{w}} \mathcal{J}_{\text{in}}(\boldsymbol{\lambda}^*, \mathbf{w}) + \frac{\eta}{2}\|\mathbf{w} - \mathbf{w}^*\|^2,$$

- For $\eta = 0$, this is just the Stackelberg equilibrium.
- For $\eta > 0$, the follower can't react as strongly to the leader.
- As $\eta \to \infty$, the follower can't react at all, so this reduces to the Nash equilibrium.

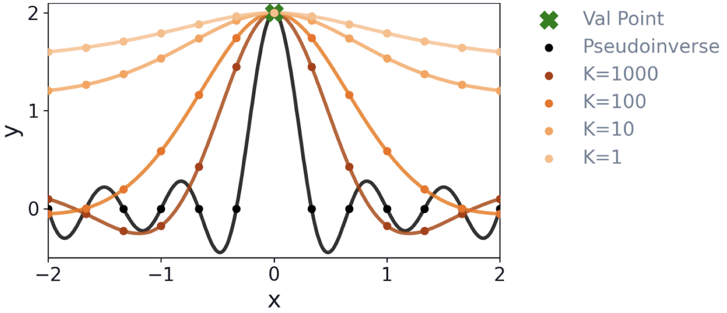
Effect of Hypergradient Approximation

- Recall the effect of min-norm bias in Fourier regression (Lecture 6):



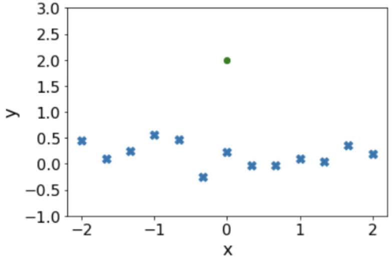
Effect of Hypergradient Approximation

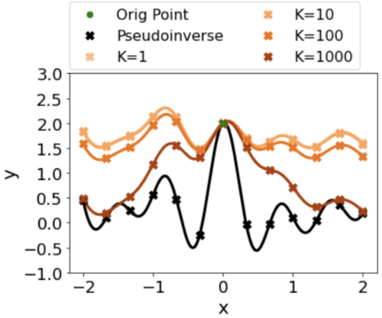
- High curvature directions for inner optimizer = low frequencies
- When approximating the hypergradient with Neumann iterations, the response Jacobian doesn't "know" that the inner optimizer is able to fit high frequencies!
- Hence, it only makes adjustments in the low frequency directions.



Effect of Hypergradient Approximation

- **Note:** this is not the same as an inductive bias for smooth functions. If the initialization is noisy, it remembers the noise.

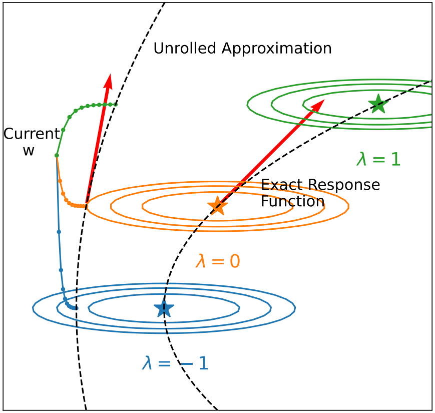




Effect of Hypergradient Approximation

- If we approximate the hypergradient by unrolling gradient descent instead of IFT with Neumann iterations, the implicit bias is similar.

Exact Response vs. Unrolling



Effect of Hypergradient Approximation

- Approximating the hypergradient with Neumann iterations or unrolling (the two most common choices) creates an implicit bias where it only accounts for high-curvature directions of the inner objective.
- The learned objective might not generalize well if you switch to a different inner optimizer at test time, or optimize for more iterations.
- Influence functions (Lecture 11) are typically estimated with a variant of Neumann iterations.
  - Hence, they may be insensitive to lower-curvature directions in weight space.
- While the min-norm bias has a strong effect on single-level optimization, you at least have the same implicit bias regardless of the learning rate, momentum, etc.
  - In bilevel optimization, the implicit bias seems to depend heavily on the hyperparameter!

Summary: Bilevel Optimization and Generalization

- **Summary:** Generalization phenomena in bilevel optimization
  - Meta-descent on the learning rate fails to generalize to long horizons since it myopically lowers α to reduce the gradient noise.
  - Cold-start optimization encourages simple/smooth solutions to the inner objective.
  - Warm-start optimization creates a memory of past iterates, leading to outer solutions that fail to generalize under reinitialization.
  - Approximating the hypergradient with Neumann iterations or unrolling only accounts for high-curvature directions of the inner objective.

Closing Thoughts

Closing Thoughts

Some (mostly) open questions about bilevel optimization dynamics:

- When is the outer objective smooth or chaotic?
- When should we use a simultaneous vs. a Stackelberg game?
- What are the effects of various ways of approximating $\mathbf{H}^{-1}$?
- How is bilevel optimization affected by stochastic gradients?
  - Are we in the noise-dominated or curvature-dominated regime, and are these even the right concepts to consider?
- Can we understand and improve the game dynamics for STNs and other approaches?
  - What do $\mathbf{H}$, $\mathbf{G}$, etc. for the inner and outer objectives tell us about the game dynamics? (E.g., how to understand the centering effect in Δ-STNs?)