He did it first!

Luca Franceschi et al. (2017)

Online hyperparameter tuning

Toward real-time hyperparameter learning

Jiwoong Daniel Im, Cristina Savin, Kyunghyun Cho. NYU.

Optimization in deep learning

1. **Preparation**
   - M hypothesis sets: $\{H^1, \ldots, H^M\}$
   - The data distribution $D(x, y)$
   - A per-example loss function $l(\cdot\,, \cdot)$
2. **Training**: $\hat{f}^m = \arg\min_{f \in H^m} \mathbb{E}_{(x,y)\sim D}\left[l(y, f(x))\right]$
3. **Model selection**: $\hat{f} = \arg\min_{f \in \{\hat{f}^1, \ldots, \hat{f}^M\}} \mathbb{E}_{(x,y)\sim D}\left[l(y, f(x))\right]$
4. **Test error**: $\mathbb{E}_{(x,y)\sim D}\left[l(y, \hat{f}(x))\right]$

Optimization in deep learning

Training: stochastic gradient descent

- $$\min_{\theta \in H^m} \sum_{(x,y) \in D_{\text{train}}} l(y, f(x;\theta)) \Big/ |D_{\text{train}}| + \alpha R(\theta)$$
  - $D_{\text{train}} = \left\{(x^1, y^1), \ldots, (x^N, y^N)\right\}$, where $(x^n, y^n) \sim D$
  - A classifier $f(x;\theta)$ parametrized with θ
  - A differentiable per-example loss $l(y, f(x;\theta))$ w.r.t. θ
  - A differentiable regularizer $R(\theta)$

Optimization in deep learning

Training: stochastic gradient descent

- Stochastic gradient descent $\theta \leftarrow \theta + \Delta(\theta, B; \lambda)$
- $\Delta(\theta, B; \lambda) = -\eta \left[\sum_{(x,y)\in B} \nabla_\theta l(y, f(x;\theta)) \Big/ |B| + \alpha \nabla_\theta R(\theta) \right]$, where
  - $B \subset D_{\text{train}}$ is a minibatch.
  - $\eta \in \lambda$ is a learning rate, and $\alpha \in \lambda$ is a regularization coefficient.

Optimization in deep learning

Training: stochastic gradient descent

- Stochastic gradient descent $\theta \leftarrow \theta + \Delta(\theta, B; \lambda)$
- $\min_{\theta \in H^m} \sum_{(x,y) \in D_{\text{train}}} l(y, f(x;\theta)) \Big/ |D_{\text{train}}| + \alpha R(\theta)$
- The optimization hyperparameters λ determine the hypothesis set H^m
  - Implicit regularization [Neyshabur et al., 2016; Gunasekar et al., 2018]
  - Importance of the early stage of learning [Golatkar et al., 2019; Jastrzebski et al., 2020]

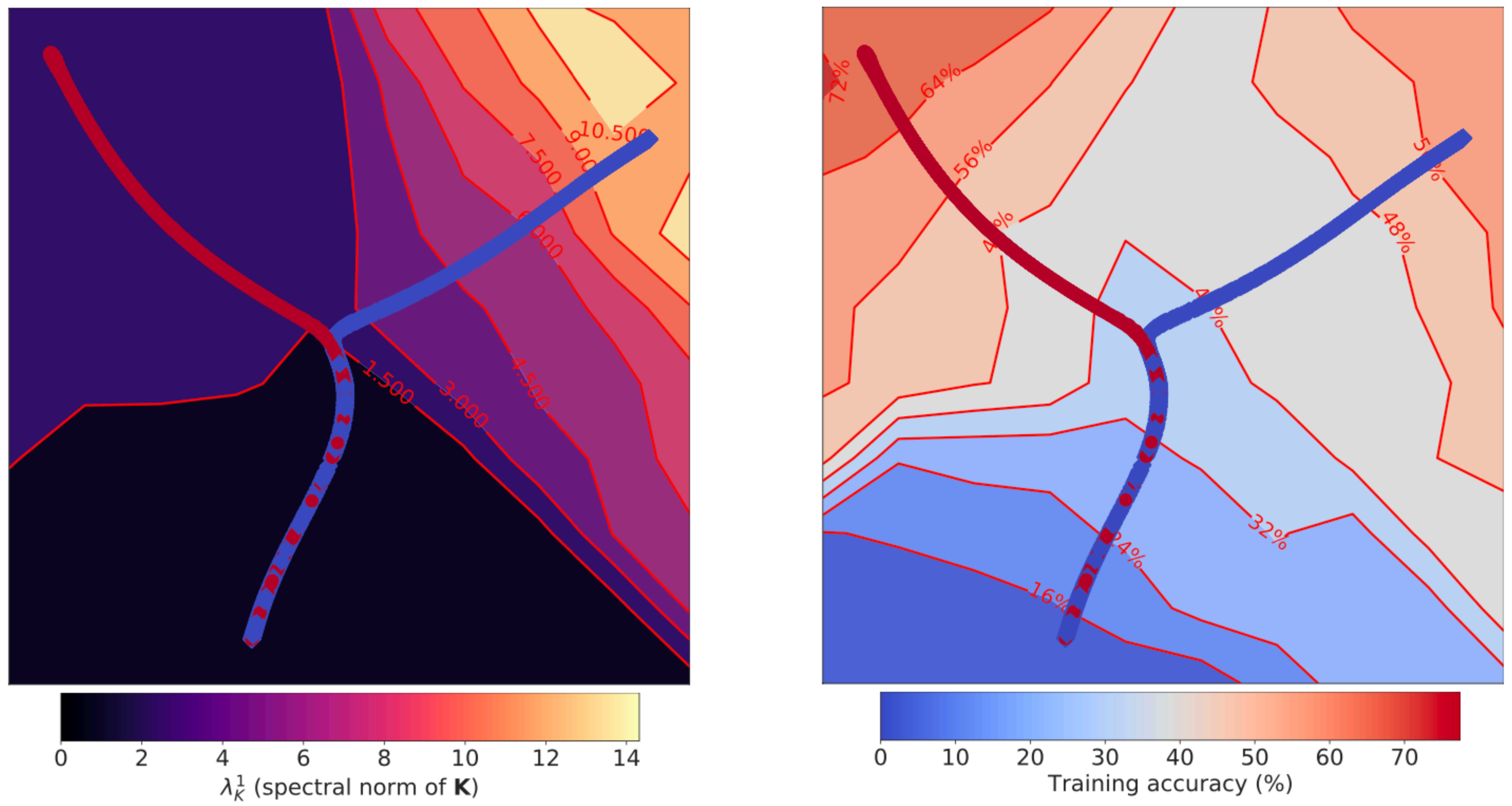


Figure 1: Visualization of the early part of the training trajectories on CIFAR-10 (before reaching 65% training accuracy) of a simple CNN model optimized using SGD with learning rates $\eta = 0.01$ (red) and $\eta = 0.001$ (blue). Each model on the training trajectory, shown as a point, is represented by its test predictions embedded into a two-dimensional space using UMAP. The background color indicates the spectral norm of the covariance of gradients $\mathbf{K}$ (λ_K^1, left) and the training accuracy (right). For lower η, after reaching what we call the break-even point, the trajectory is steered towards a region characterized by larger λ_K^1 (left) for the same training accuracy (right). See Sec. 4.1 for details. We also include an analogous figure for other quantities that we study in App. A.

Optimization in deep learning

Model selection - blackbox optimization

- $\min_{\lambda \in \Lambda} \sum_{(x,y) \in D_{\text{val}}} l(y, f(x; \hat{\theta}(\lambda))) \Big/ |D_{\text{val}}|$
  - $D_{\text{valid}} = \left\{(x^1, y^1), \ldots, (x^{N'}, y^{N'})\right\}$, where $(x^n, y^n) \sim D$
  - $\hat{\theta}(\lambda)$: a fixed point* of SGD updates with the hyperparameters $\lambda \in \Lambda$
- It is often solved as a blackbox optimization problem.

* Not really…

Optimization in deep learning

Model selection - blackbox optimization

- Grid search: $\min_{\lambda \in G(\Lambda)} \sum_{(x,y) \in D_{\text{val}}} l(y, f(x; \hat{\theta}(\lambda))) / |D_{\text{val}}|$
  - $G(\Lambda)$: a set of uniformly-spaced points in Λ.
  - A pretty standard approach: available in **scikit-learn**.
- It does not scale well with the # of hyperparameters.

sklearn.model_selection.GridSearchCV

class sklearn.model_selection.**GridSearchCV**(*estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False*) [source]

Exhaustive search over specified parameter values for an estimator.

Important members are fit, predict.

GridSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

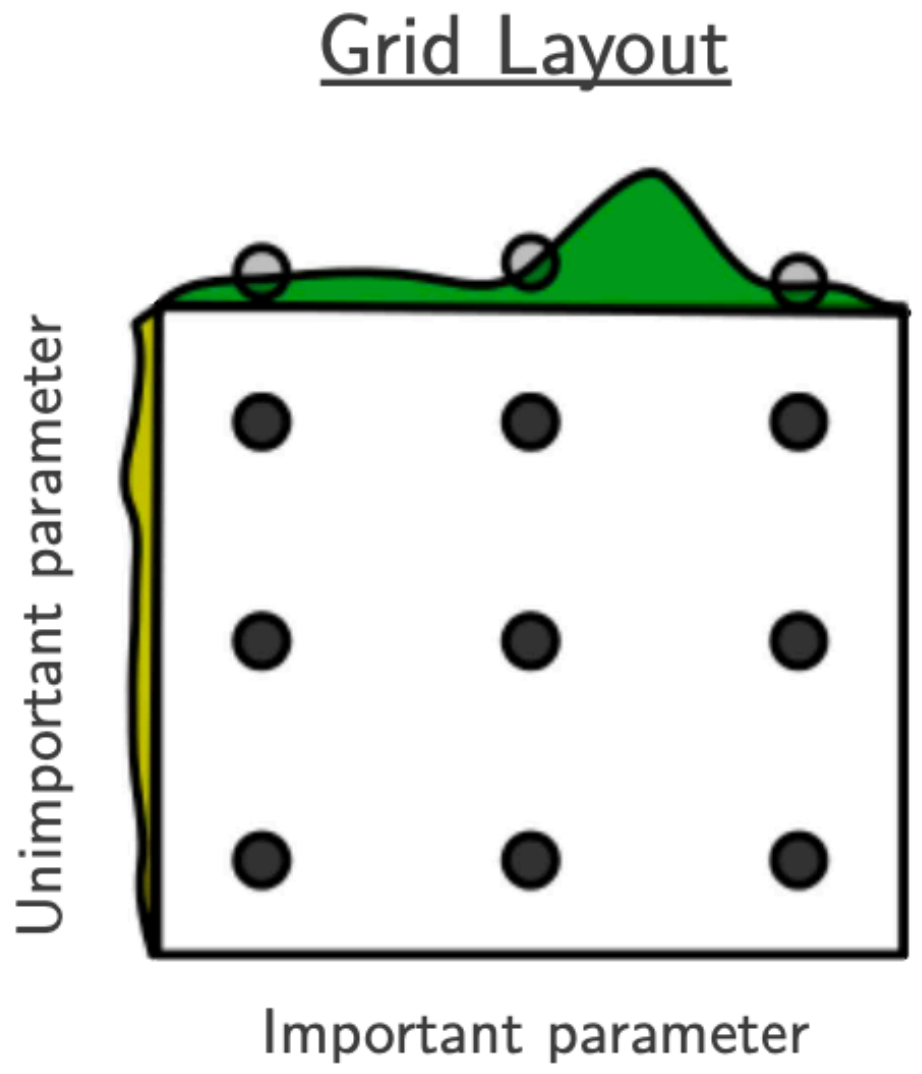
The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

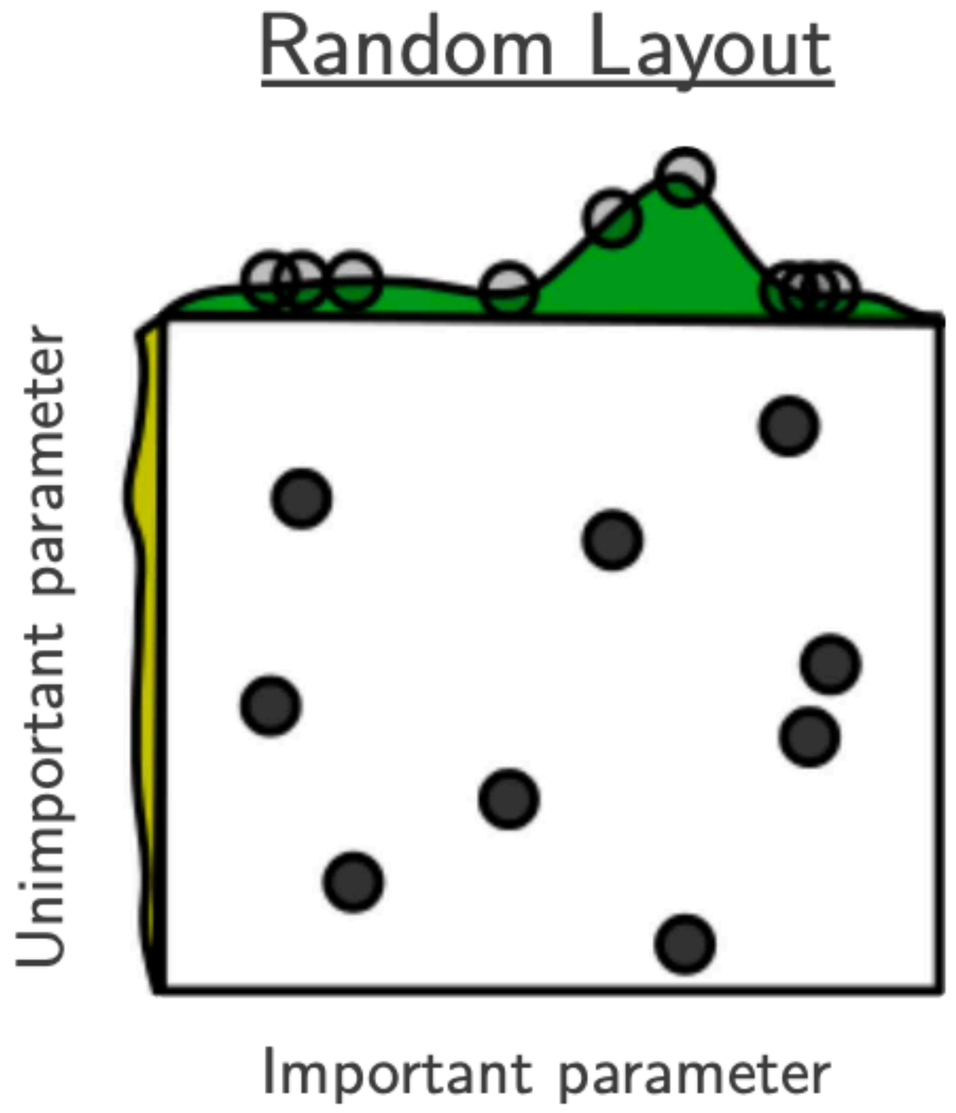
Read more in the User Guide.

Optimization in deep learning

Model selection - blackbox optimization

- Random search: $\min_{\lambda \in \tilde{G}(\Lambda)} \sum_{(x,y)\in D_{\text{val}}} l(y, f(x; \hat{\theta}(\lambda))) \Big/ |D_{\text{val}}|$
  - $\tilde{G}(\Lambda) = \{\lambda \sim \mathcal{U}(\Lambda)\}$: a random set of hyperparameters drawn uniformly.
  - Quite effective and widely used in deep learning
  - Bergstra & Bengio (2012): "*Compared with neural networks configured by a pure grid search, we find that random search over the same domain is able to find models that are as good or better within a small fraction of the computation time.*"





Optimization in deep learning

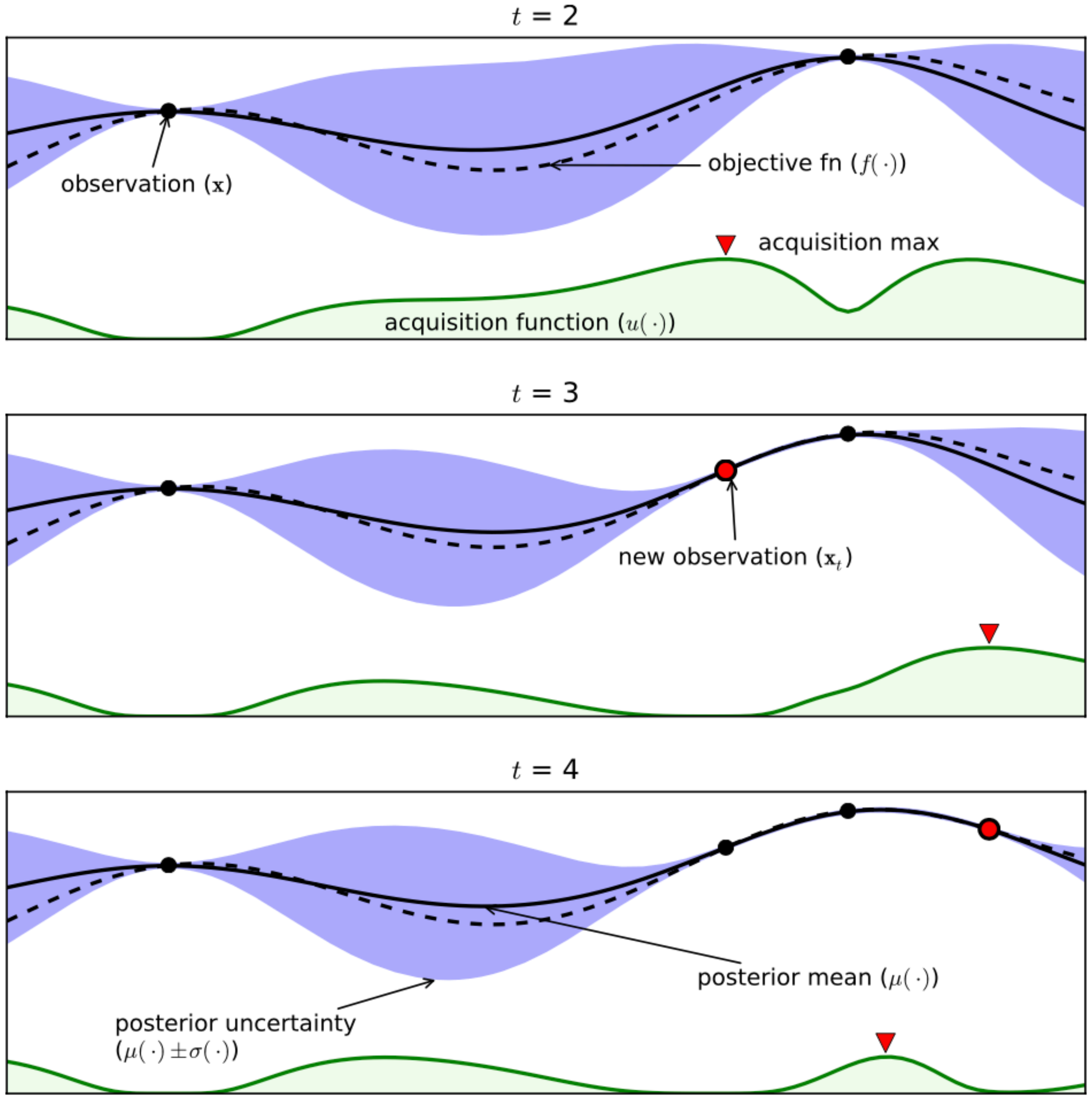
Model selection - sequential model-based optimization

- SMBO [Jones et al., 1998; JGO] for hyperparameter optimization

1. Treat the hyperparameter-to-loss function $\Lambda \to \mathbb{R}_+$ as a blackbox.
2. **Evaluate** an initial set of **hyperparameters** $\tilde{G}(\Lambda)$.
3. **Fit a model** to the underlying blackbox function.
4. **Draw** a set of **hyperparameters** according to an acquisition function.
5. **Repeat 3-4**.
6. Find the hyperparameters that minimize the final model.

Optimization in deep learning

Model selection - sequential model-based optimization



See Brochu et al. [2010] for a great tutorial on this topic.

Optimization in deep learning

Model selection - sequential model-based optimization

- Bayesian optimization with Gaussian Process [Snoek et al., 2015]
  - Use **Gaussian process** as a model.
  - Admits the exact computation of various acquisition functions.
- It has become a standard approach to hyperparameter tuning.
  - Implemented in ***scikit-optimize*** and can be used with ***scikit-learn***.

skopt.BayesSearchCV

class `skopt.`**`BayesSearchCV`**(*estimator, search_spaces, optimizer_kwargs=None, n_iter=50, scoring=None, fit_params=None, n_jobs=1, n_points=1, iid=True, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score='raise', return_train_score=False*) [source][source]

Bayesian optimization over hyper parameters.

BayesSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

Gradient-based hyperparameter tuning

Model selection as bilevel optimization

- **Outer** optimization: $\min_{\lambda \in \Lambda} \sum_{(x,y) \in D_{\text{val}}} l(y, f(x; \hat{\theta}(\lambda))) \Big/ |D_{\text{val}}|$
- **Inner** optimization: $\min_{\theta \in H(\lambda)} \sum_{(x,y) \in D_{\text{train}}} l(y, f(x; \theta)) \Big/ |D_{\text{train}}| + \alpha R(\theta)$

Model selection as bilevel optimization

For deep learning, in practice

1. Approximately solve the inner optimization problem with SGD:
    - $\hat{\theta}(\lambda) = \theta^T$, where $\theta^t = \theta^{t-1} + \Delta(\theta^{t-1}, B^t; \lambda)$ and $\theta^0 \sim \mathrm{Init}.$
2. Stochastic gradients are differentiable w.r.t. λ
    - $\dfrac{\partial \Delta^t}{\partial \lambda} = \displaystyle\sum_{s=1}^{t} \frac{\partial \Delta^t}{\partial \lambda^s} \cancel{\frac{\partial \lambda^s}{\partial \lambda}}$, where $\dfrac{\partial \Delta^t}{\partial \lambda^s} = \dfrac{\partial \Delta^t}{\partial \theta^{t-1}} \left(\displaystyle\prod_{t'=s}^{t-2} \frac{\partial \theta^{t'+1}}{\partial \theta^{t'}} \right) \frac{\partial \Delta^s}{\partial \lambda^s}$
    - as long as Δ is differentiable w.r.t. both θ and λ.
3. We can then compute $\dfrac{\partial \hat{\theta}}{\partial \lambda}$ by backprop-through-backprop

Model selection as bilevel optimization

Gradient-based hyperparameter optimization

- Bengio [2020; NC] "*present[s] a methodology to optimize several hyper-parameters, based on the computation of the gradient of a model selection criterion with respect to the hyperparameters.*"
- The outer and inner loops collapse into one optimization problem:
  - $\min_{\lambda \in \Lambda} \sum_{(x,y) \in D_{\text{val}}} l(y, f(x; \theta^0 + \sum_{t=1}^{T} \Delta(\theta^{t-1}, B^t; \lambda))) \big/ |D_{\text{val}}|$
    - where $\theta^t = \theta^{t-1} + \Delta(\theta^{t-1}, B^t; \lambda)$
  - This allows us to use gradient-based optimization.

Model selection as bilevel optimization

Gradient-based hyperparameter optimization

- Expensive due to the necessity of computing Hessian**s**:
    - $\dfrac{\partial \theta^{t'+1}}{\partial \theta^{t'}} = I + \nabla^2_\theta L(\theta^{t'}, B; \lambda)$, where $L(\theta, B; \lambda) = \dfrac{1}{|B|} \sum_{(x,y)\in B} l(y, f(x;\theta)) + \alpha R(\theta)$
- Some kind of approximation is necessary to make an appropriate trade-off:
    - Truncation of the inner optimization trajectory [Luketina et al., 2016 ICML; Finn et al., 2017 ICML]
    - Implicit function theorem: truncated backprop-through-backprop [Bengio, 2000 NC; Pedregosa, 2016 ICML]
    - Reversible iterative optimization [Maclaurin et al., 2015 ICML]

Model selection as bilevel optimization

Somewhat unsatisfactory trade-offs

	Method	Steps	Eval.	Hypergradient Approximation
Impossible	Exact IFT	∞	$\mathbf{w}^*(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\mathbf{w}^*(\boldsymbol{\lambda})}$
Offline only	Unrolled Diff. [4]	i	$\mathbf{w}_0$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \sum_{j \leq i} \left[\prod_{k<j} I - \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\Big\|_{\mathbf{w}_{i-k}}\right] \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\mathbf{w}_{i-j}}$
Inexact Online	L-Step Truncated Unrolled Diff. [7]	i	$\mathbf{w}_L$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \sum_{L \leq j \leq i} \left[\prod_{k<j} I - \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\Big\|_{\mathbf{w}_{i-k}}\right] \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\mathbf{w}_{i-j}}$
Offline only	Larsen et al. [23]	∞	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \left[\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}} \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}^T\right]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Offline only	Bengio [2]	∞	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Inexact Online	$T1 - T2$ [24]	1	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times [I]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Inexact Online	**Ours**	i	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \left(\sum_{j<i} \left[I - \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^j\right) \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Online	Conjugate Gradient (CG) $\approx$	-	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} - \left(\arg\min_{\mathbf{x}} \left\Vert \mathbf{x} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} - \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \right\Vert\right) \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Exact Offline only	Hypernetwork [25, 26]	-	-	$\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \times \frac{\partial \mathbf{w}^*_\phi}{\partial \boldsymbol{\lambda}}$ where $\mathbf{w}^*_\phi(\boldsymbol{\lambda}) = \arg\min_\phi \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}_\phi(\boldsymbol{\lambda}))$
Exact Offline only	Bayesian Optimization [19, 20, 7] $\approx$	-	-	$\frac{\partial \mathbb{E}[\mathcal{L}^*_V]}{\partial \boldsymbol{\lambda}}$ where $\mathcal{L}^*_V \sim$ Gaussian-Process$(\{\boldsymbol{\lambda}_i, \mathcal{L}_V(\boldsymbol{\lambda}_i, \mathbf{w}^*(\boldsymbol{\lambda}_i))\})$

Current practice

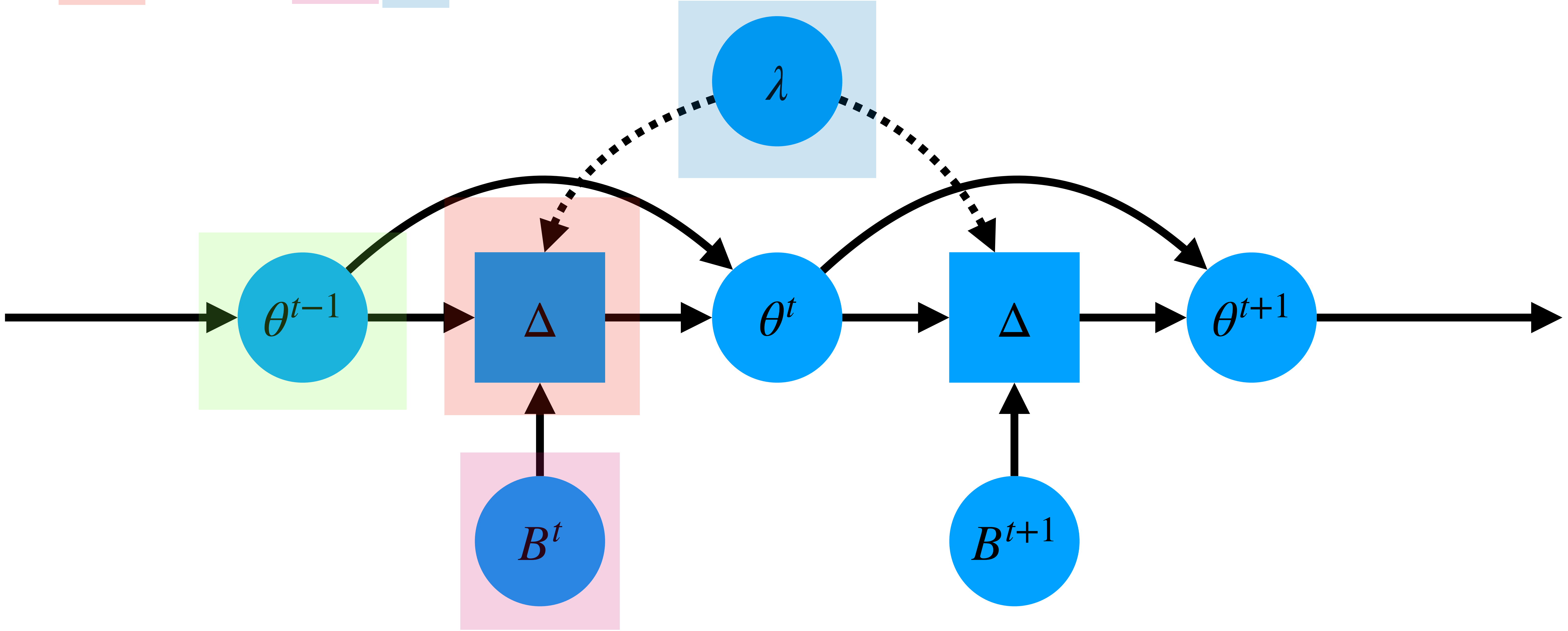
Lorraine et al. [2020]

New

Learning
as a recurrent network

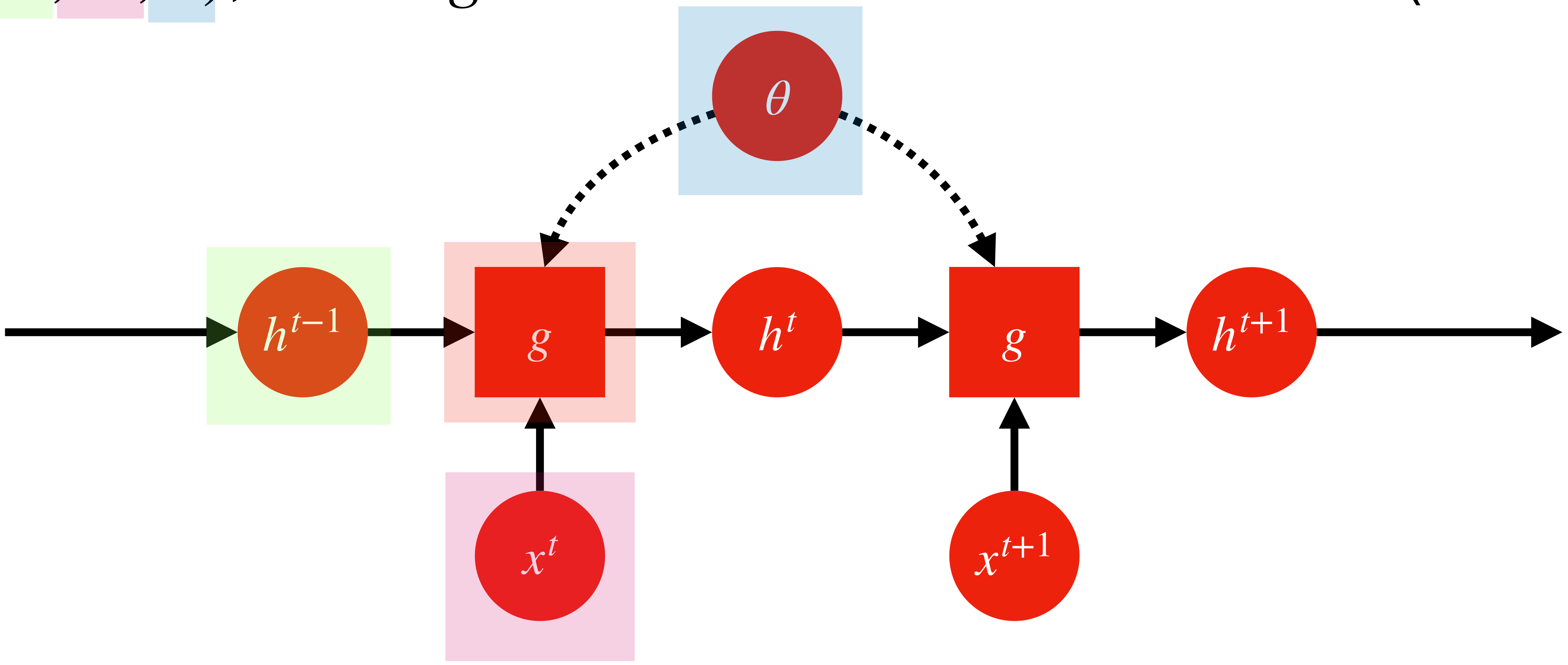
Learning as a recurrent network

- Inner optimization
  - $\theta^t = \theta^{t-1} + \Delta(\theta^{t-1}, B^t; \lambda)$ and $\theta^0 \sim \mathrm{Init.}$



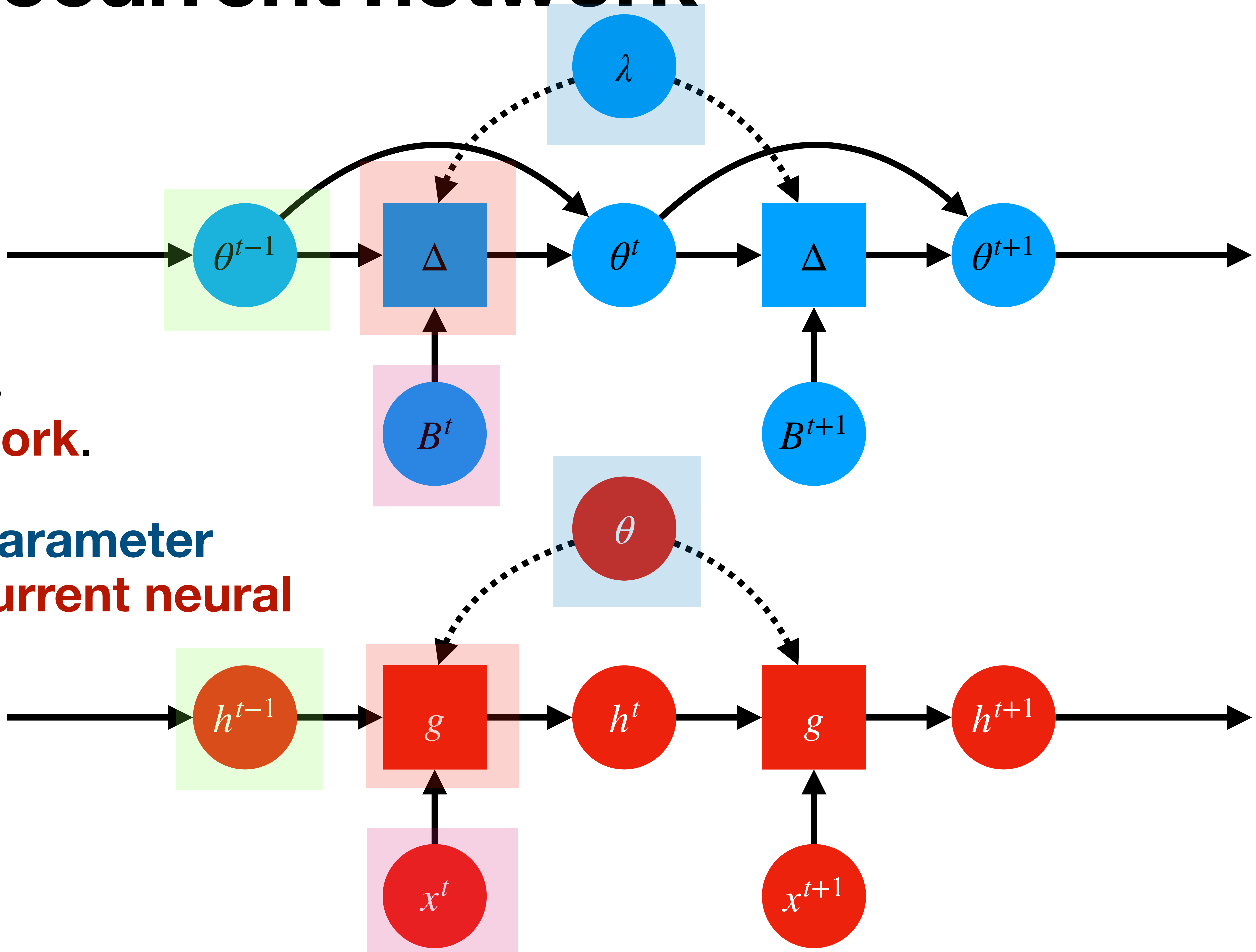
Learning as a recurrent network

- A recurrent neural network
  - $h^t = g(h^{t-1}, x^t; \theta)$, where g is a recurrent transition function (LSTM & GRU)



Learning as a recurrent network

- **Learning a neural net** is a **recurrent neural network**.
- **Gradient-based hyperparameter tuning** is **learning a recurrent neural networks**



Quick detour:
real-time recurrent learning

See [Marschall et al., 2020 JMLR] for an overview of real-time recurrent learning.

Gradient-based learning of a recurrent network

Gradient of the total loss w.r.t. the parameters

- A recurrent network works with a sequence: $l(y, F(x;\theta)) = \sum_{t=1}^{T} l(y^t, f(h^t;\theta))$
    - $F(x;\theta)$: a recurrent network that consumes x and outputs y
    - $f(h^t;\theta)$: a readout function
    - $h^t = g(h^{t-1}, x^t;\theta)$: a recurrent transition
- The gradient of the total loss w.r.t. the parameters: $\frac{\partial l(y, F(x;\theta))}{\partial \theta}$

Gradient-based learning of a recurrent network

Future-facing view of gradient computation

- $$\frac{\partial l(y, F(x;\theta))}{\partial \theta} = \sum_{s=1}^{T} \boxed{\sum_{t=s}^{T}} \frac{\partial l(y^t, f(h^t;\theta^s))}{\partial \theta^s} \boxed{\cancel{\frac{\partial \theta^s}{\partial \theta}}}$$
  - θ^s is the use of the parameters θ at time $s \rightarrow \partial\theta^s / \partial\theta = I$.
  - θ^s only influences the loss values at $t \geq s$.
- It is **future facing**, because $\frac{\partial l(y, f(x;\theta))}{\partial \theta^s} = \sum_{t=s}^{T} \frac{\partial l(y^t, f(h^t;\theta^s))}{\partial \theta^s}$:
  - For each application of θ at time s, consider its impact on the future losses $t = s, \ldots, T$

Gradient-based learning of a recurrent network

Past-facing view of gradient computation

- $$\frac{\partial l(y, f(x;\theta))}{\partial \theta} = \sum_{t=1}^{T} \boxed{\sum_{s=1}^{t}} \frac{\partial l(y^t, f(x_{\leq t}; \theta^s))}{\partial \theta^s} \cancel{\frac{\partial \theta^s}{\partial \theta}}$$
  - θ^s is the use of the parameters θ at time $s \rightarrow \partial\theta^s \big/ \partial\theta = I$
  - θ^s only influences the loss values at $t \geq s$.
- It is **past facing**, because $\frac{\partial l(y^t, f(h^t;\theta))}{\partial \theta} = \boxed{\sum_{s=1}^{t}} \frac{\partial l(y^t, f(h^t;\theta^s))}{\partial \theta^s}$:
  - Consider the impact of the earlier applications of the parameters $\theta^{\leq t}$ on the loss at time t.

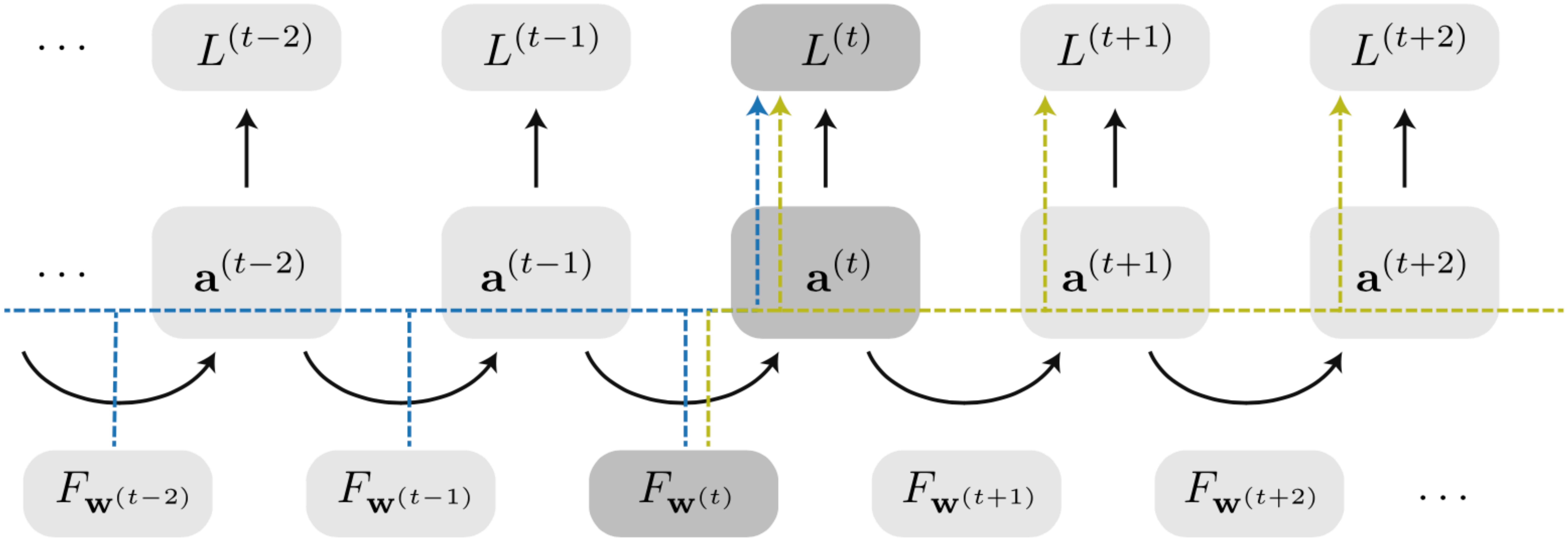
Gradient-based learning of a recurrent network

Summary

- Both future-facing & past-facing views are exact with a fixed θ.

Past-facing: $\frac{\partial L^{(t)}}{\partial \mathbf{w}} = \sum_{t' \leq t} \frac{\partial L^{(t)}}{\partial \mathbf{w}^{(t')}}$

Future-facing: $\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{(t)}} = \sum_{t' \geq t} \frac{\partial L^{(t')}}{\partial \mathbf{w}^{(t)}}$



Gradient-based learning of a recurrent network

Past-facing view admits online learning

- At each step t, we can compute one term in the gradient without $x^{>t}$:
  - $$\frac{\partial l(y^t, f(h^t;\theta))}{\partial \theta} = \sum_{s=1}^{t} \frac{\partial l(y^t, f(h^t;\theta^s))}{\partial \theta^s}$$
- We then update the parameters immediately by $\theta \leftarrow \theta - \eta \frac{\partial l(y^t, f(h^t;\theta))}{\partial \theta}$
  - with small $\eta \ll 1$

Gradient-based learning of a recurrent network

Past-facing view admits online learning

- Let's rearrange terms:

1. $$\frac{\partial l(y^t, f(h^t;\theta))}{\partial \theta} = \underbrace{\frac{\partial l^t}{\partial h^t}}_{=c^t} \frac{\partial h^t}{\partial \theta}$$

2. $$\underbrace{\frac{\partial h^t}{\partial \theta}}_{=M^t} = \sum_{s=1}^{t} \frac{\partial h^t}{\partial \theta^s} = \frac{\partial h^t}{\partial \theta^t} + \sum_{s=1}^{t-1} \frac{\partial h^t}{\partial h^{t-1}} \frac{\partial h^{t-1}}{\partial \theta^s} = \underbrace{\frac{\partial h^t}{\partial \theta^t}}_{=\bar{M}^t} + \underbrace{\frac{\partial h^t}{\partial h^{t-1}}}_{=J^t} \underbrace{\frac{\partial h^{t-1}}{\partial \theta}}_{=M^{t-1}}$$

- The computation is decomposed into two terms; immediate gradient and temporal gradient:
  - $$\frac{\partial l(y^t, f(h^t;\theta))}{\partial \theta} = c^t M^t = c^t \left(\bar{M}^t + J^t M^{t-1}\right)$$

Real-time recurrent learning

An alternative to backpropagation through time [Williams & Zipser, 1989]

- At each step t,
  - Update the influence matrix: $M^t = \bar{M}^t + J^t M^{t-1}$
  - Compute the immediate credit: $c^t = \partial l^t \big/ \partial h^t$
  - Update the parameters: $\theta \leftarrow \theta - \eta c^t M^t$

Real-time recurrent learning

An alternative to backpropagation through time [Williams & Zipser, 1989]

- Not widely used in practice, because it's quite expensive
  - $O(N^3)$ memory complexity, where N is the # of neurons
  - # of parameters $\approx O(N^2) \rightarrow$ the size of the influence matrix $M^t \approx N \times N^2$
- But, what if # of parameters $\approx O(1)$?

Online hyperparameter tuning

Real-time recurrent learning for hyperparameter tuning

- The gradient of the validation loss after t updates w.r.t. the hyperparameters:

1. $$\frac{\partial L(D_{\text{val}}, \theta^t)}{\partial \lambda} = \underbrace{\frac{\partial L^t}{\partial \theta^t}}_{=c^t} \frac{\partial \theta^t}{\partial \lambda}$$

2. $$\underbrace{\frac{\partial \theta^t}{\partial \lambda}}_{=M^t} = \sum_{s=1}^{t} \frac{\partial \theta^t}{\partial \lambda^s} = \frac{\partial \theta^t}{\partial \lambda^t} + \sum_{s=1}^{t-1} \frac{\partial \theta^t}{\partial \theta^{t-1}} \frac{\partial \theta^{t-1}}{\partial \lambda^s} = \underbrace{\frac{\partial \theta^t}{\partial \lambda^t}}_{=\bar{M}^t} + \underbrace{\frac{\partial \theta^t}{\partial \theta^{t-1}}}_{=J^t} \underbrace{\frac{\partial \theta^{t-1}}{\partial \lambda}}_{=M^{t-1}}$$

- Just like RTRL for training a recurrent neural network.

Real-time recurrent learning for hyperparameter tuning

- The gradient of the validation loss after t updates w.r.t. the hyperparameters:
- $$\frac{\partial L(D_{\text{val}}, \theta^t)}{\partial \lambda} = \underbrace{\frac{\partial L^t}{\partial \theta^t}}_{=c^t} \left(\underbrace{\frac{\partial \theta^t}{\partial \lambda^t}}_{=\bar{M}^t} + \underbrace{\frac{\partial \theta^t}{\partial \theta^{t-1}}}_{=J^t} \underbrace{\frac{\partial \theta^{t-1}}{\partial \lambda}}_{=M^{t-1}} \right)$$
- $\frac{\partial \theta^t}{\partial \theta^{t-1}}$ still contains an expensive Hessian: $I + \nabla_\theta^2 L(\theta^{t-1}, B; \lambda)$
  - Stochastic approximation to the Hessian-vector product (Perlmutter's trick)
    - See the blog post by Domke.

Real-time recurrent learning for hyperparameter tuning

- Computationally feasible, because
  - the size of the influence matrix $M \approx O(|\lambda|^2 \times N)$ instead of $O(N^3)$
  - # of hyperparameters $|\lambda| \ll |\theta|$ # of parameters
- Computationally cheaper unbiased approximations exist:
  - UORO (unbiased online recurrent optimization; Tallec & Ollivier, 2017)

Real-time recurrent learning for hyperparameter tuning

It's this straightforward!

- At each step t,
  - Update the influence matrix: $M^t = \bar{M}^t + J^t M^{t-1}$
  - Compute the immediate credit: $c^t = \partial L^t \big/ \partial \theta^t$
  - Update the **hyperparameters**: $\lambda \leftarrow \lambda - \eta_\lambda c^t M^t$

Real-time recurrent learning for hyperparameter tuning

Two major properties

- **Fully online**: we update θ and λ simultaneously.
- **Almost exact**: with a small enough meta-learning rate η_{λ}

Obviously, we weren't the first to come up with it (!!)

Luca Franceschi et al. (2017)

Luca Franceschi et al. (2017)

Forward and Reverse Gradient-Based Hyperparameter Optimization

As simple example of these dynamics occurs when training a neural network by gradient descent with momentum (GDM), in which case $s_t = (v_t, w_t)$ and

$$\begin{aligned} v_t &= \mu v_{t-1} + \nabla J_t(w_{t-1}) \\ w_t &= w_{t-1} - \eta(\mu v_{t-1} - \nabla J_t(w_{t-1})) \end{aligned} \tag{2}$$

where J_t is the objective associated with the t-th mini-batch, μ is the rate and η is the momentum. In this example, $\lambda = (\mu, \eta)$.

Note that the iterates $s_1, \ldots, s_T$ implicitly depend on the vector of hyperparameters λ. Our goal is to optimize the hyperparameters according to a certain error function E evaluated at the last iterate s_T. Specifically, we wish to solve the problem

$$\min_{\lambda \in \Lambda} f(\lambda) \tag{3}$$

where the set $\Lambda \subset \mathbb{R}^m$ incorporates constraints on the hyperparameters, and the response function $f : \mathbb{R}^m \to \mathbb{R}$ is defined at $\lambda \in \mathbb{R}^m$ as

$$f(\lambda) = E(s_T(\lambda)). \tag{4}$$

rule we have, for every $t \in \{1, \ldots, T\}$, that

$$\frac{ds_t}{d\lambda} = \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial s_{t-1}} \frac{ds_{t-1}}{d\lambda} + \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial \lambda}. \tag{13}$$

Defining $Z_t = \frac{ds_t}{d\lambda}$ for every $t \in \{1, \ldots, T\}$ and recalling Eq. (11), we can rewrite Eq. (13) as the recursion

$$Z_t = A_t Z_{t-1} + B_t, \quad t \in \{1, \ldots, T\}. \tag{14}$$

Using Eq. (14), we obtain that

$$\begin{aligned} \nabla f(\lambda) &= \nabla E(s_T) Z_T \\ &= \nabla E(s_T)(A_T Z_{T-1} + B_T) \\ &= \nabla E(s_T)(A_T A_{T-1} Z_{T-2} + A_T B_{T-1} + B_T) \\ &\vdots \\ &= \nabla E(s_T) \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} A_s \right) B_t. \end{aligned} \tag{15}$$

Note that the recurrence (14) on the Jacobian matrix is structurally identical to the recurrence in the RTRL procedure described in (Williams & Zipser, 1989, eq. (2.10)).

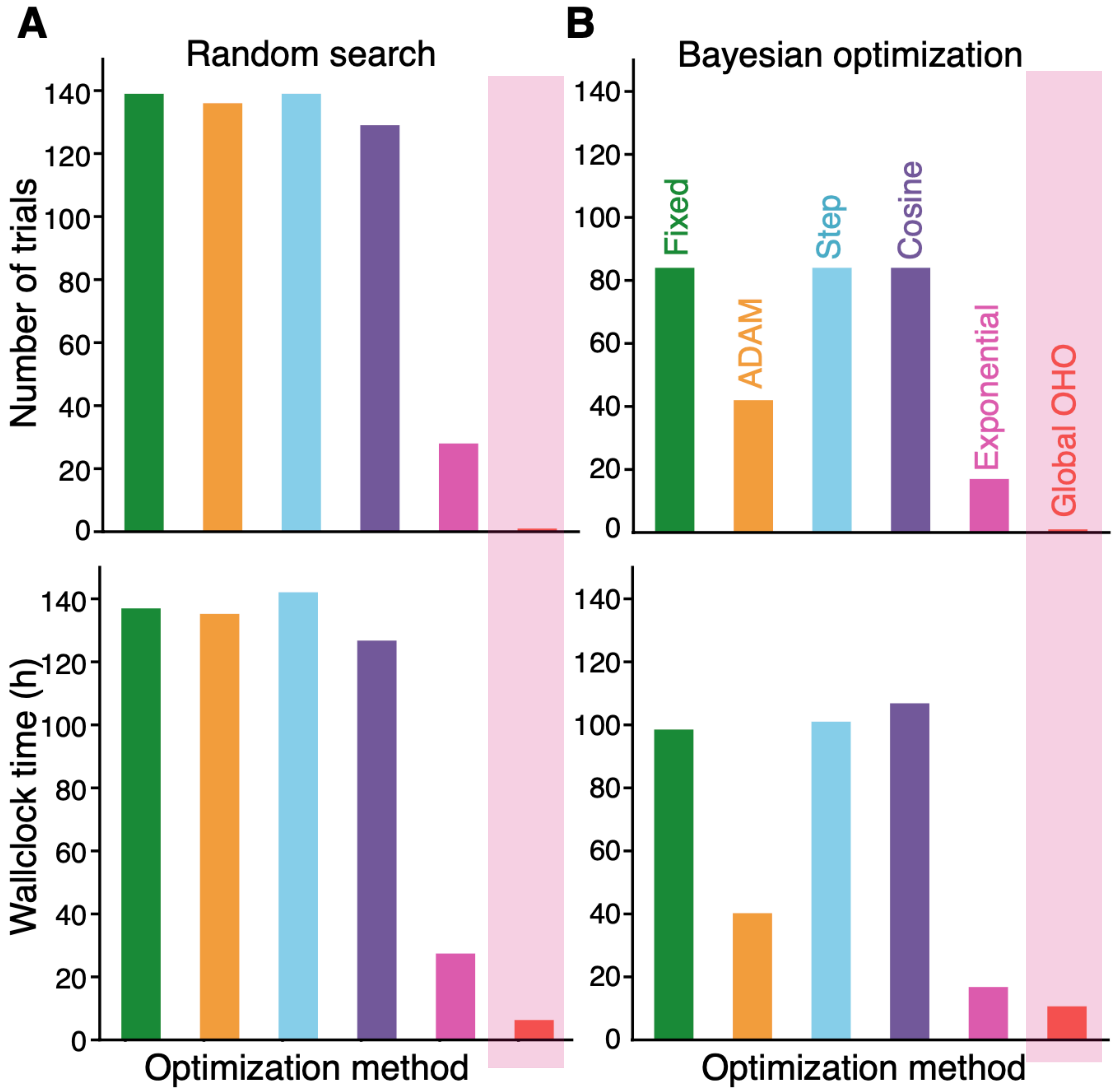
How well does it work?

Setup

- Datasets: MNIST and CIFAR-10
- Networks:
  - MNIST: a 4-layer fully-connected neural network
  - CIFAR-10: ResNet-18 [He et al., 2017]
- Optimizer: Adam [Kingma & Ba, 2015]

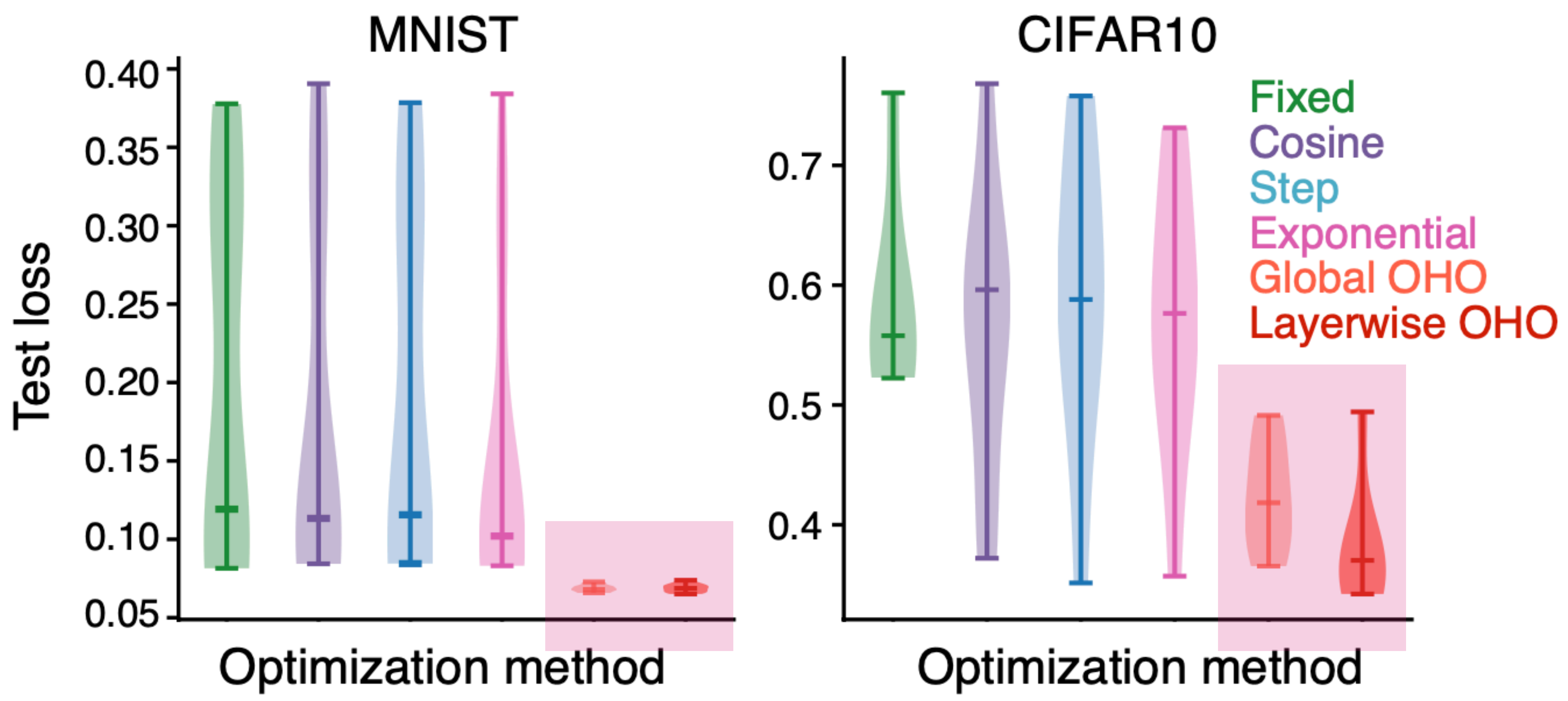
vs. blackbox optimization

- On CIFAR-10
- Baselines
  - Random search
  - Bayesian optimization: scikit-opt
- Run until the target test loss ≤ 0.3
- The proposed approach (OHO) is substantially more efficient, because almost always a single run is enough.



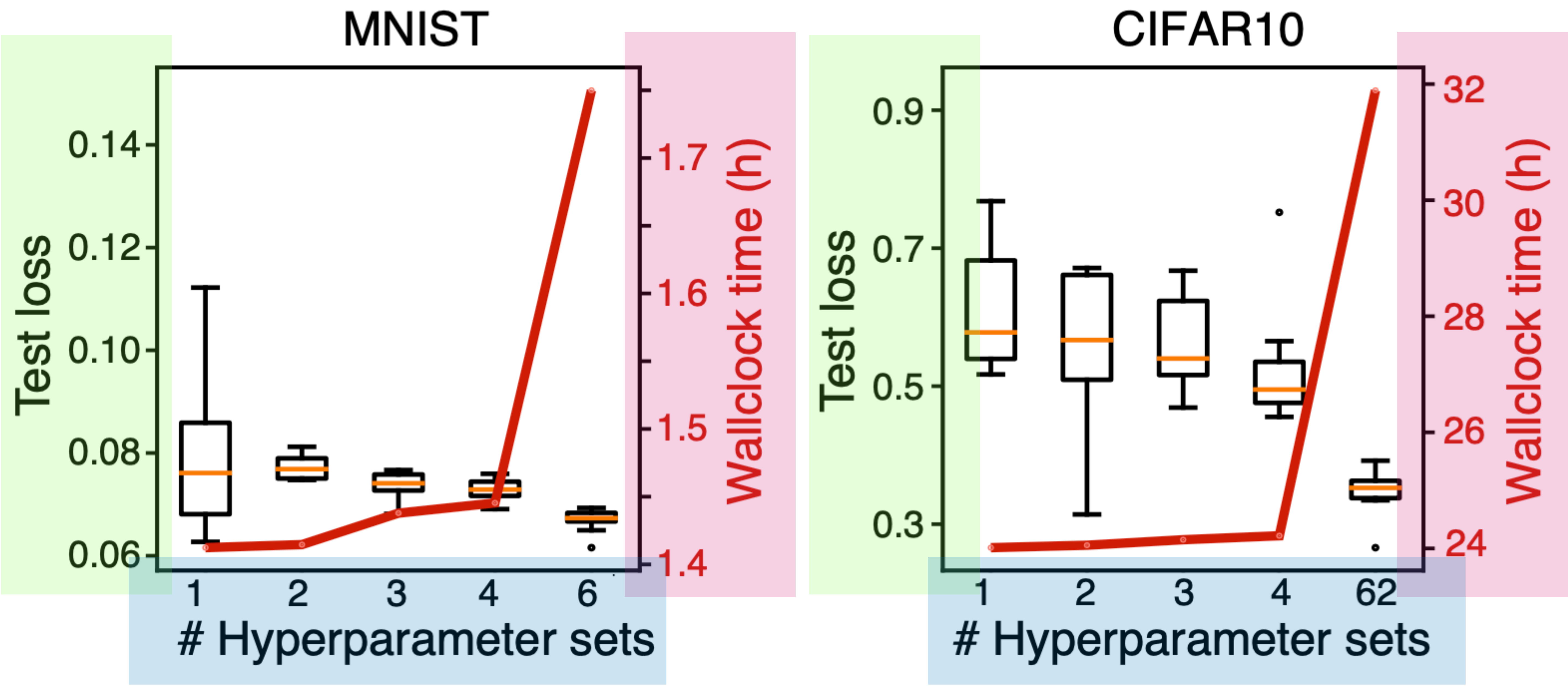
Stability

- On MNIST & CIFAR-10
- Randomly varying
    - an initial learning rate
    - a regularization coefficient
    - [scheduling coefficients]



- OHO rarely (if ever) fails to find a good set of hyperparameters.

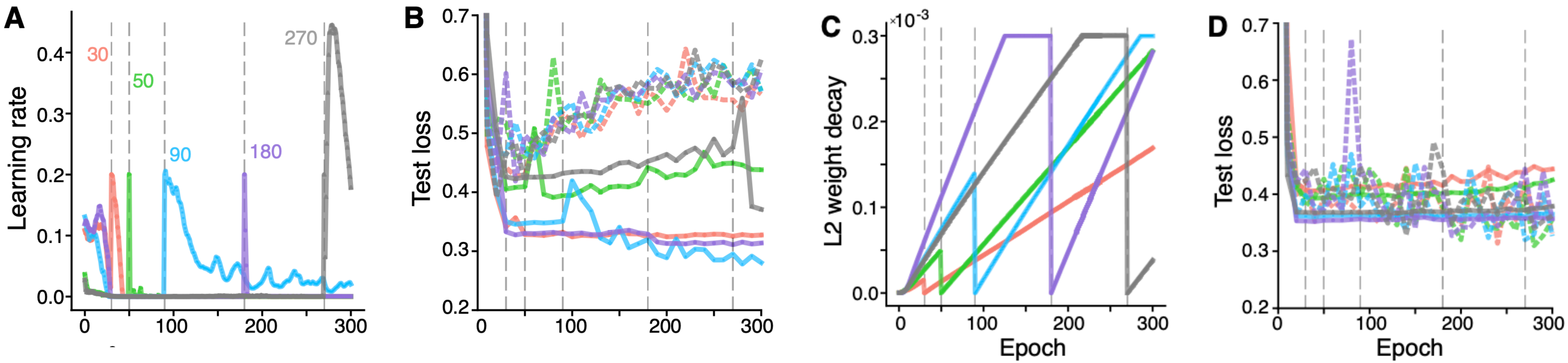
Scalability



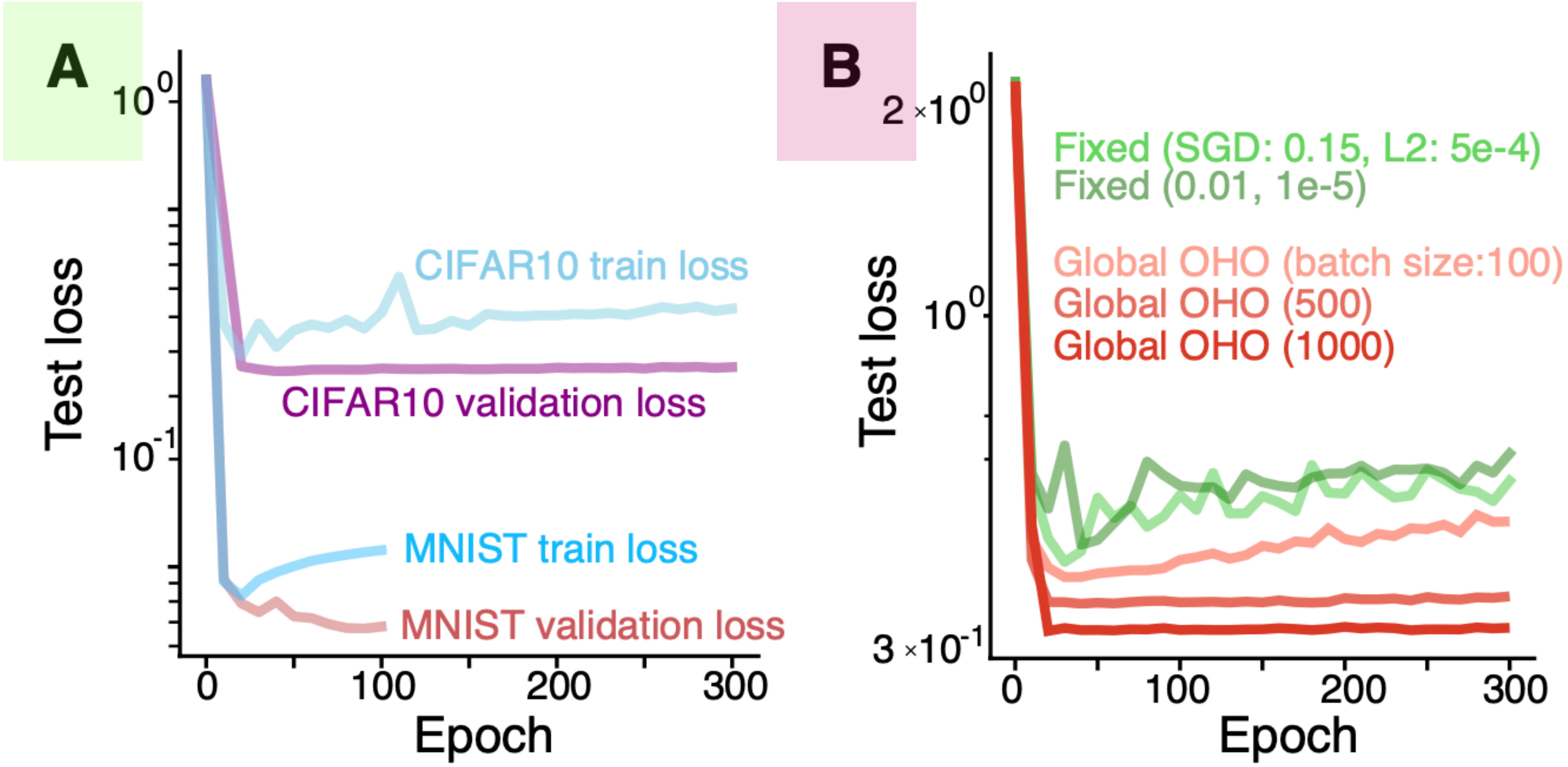
- On MNIST & CIFAR-10
- Increasing the number of hyperparameters by
  - Layer-wise learning rates
  - Layer-wise weight decay coefficient
- The test loss reliably decreases as the # of hyperparameters increases.
- The computational complexity grows rapidly: $|M| = O(|\lambda|^2 \times |\theta|)$

Adaptation

- On CIFAR-10
- Artificially perturb the hyperparameter during training
- OHO rapidly recovers from perturbation



Design choices

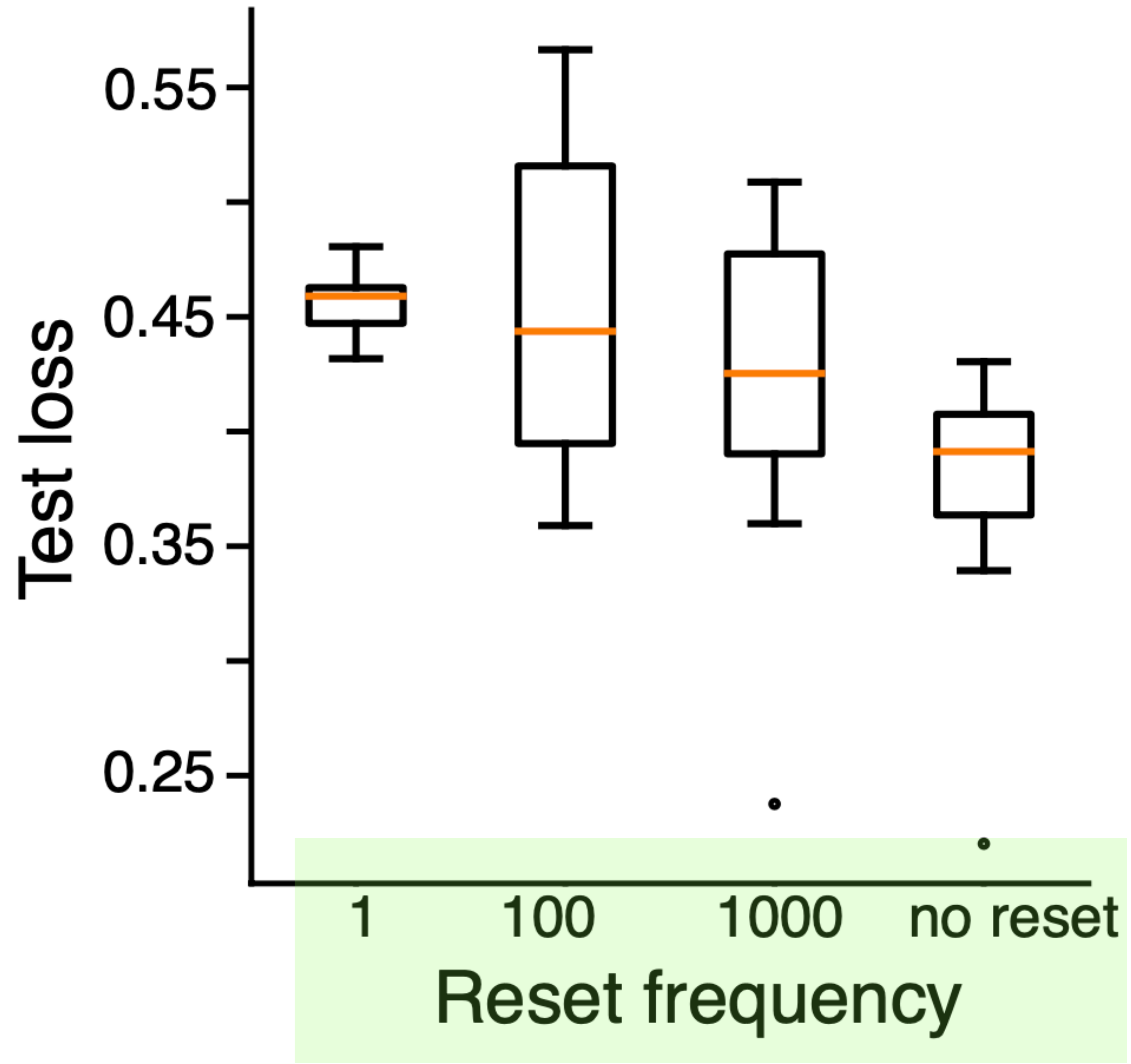


- On MNIST & CIFAR-10
- Validation set vs. training set for hyperparameter tuning
  - Important to use the validation set in order to avoid overfitting.
  - Surprisingly, we do not observe overfitting to the validation set.
- Minibatch size of computing the validation gradient
  - The mini batch size must be sufficiently large but not too large.

Does it actually matter?

Long-term dependency

- Does the choice of hyperparameters affect the loss in the future?
- We can test the long-term influence by manually resetting M regularly
- $$M^t = \begin{cases} \bar{M}^t, & \text{if } t \mod \tau = 0 \\ \bar{M}^t + J^t M^{t-1}, & \text{otherwise} \end{cases}$$
- Frequent resets degrade the final test loss.
- OHO captures long-term influence of hyperparameters on learning.



Findings from the experiments

It looks *very* promising!

- OHO is **efficient**
    - It can find a good solution by adapting the hyperparameter **on-the-fly**.
- OHO is **effective**
    - It often finds a better solution (test loss) than offline hyperparameter tuning.
    - It suggests beneficial co-evolution of network and hyper-parameters.
- OHO is **resilient**
    - It can quickly recover from perturbation to the hyperparameters

Wrap-up

Summary

- How we arrived at **online hyperparameter optimization**:

  1. Model selection and training can be folded into a single optimization problem, when training is done with gradient-based learning.

  2. Gradient-based learning is a recurrent neural network (RNN).

  3. An RNN can be trained online by real-time recurrent learning (RTRL)

Considerations left for the future

- **Overfitting** to the validation loss
- Application to **non-stationary environments.**
- **Inexact RTRL**

Thank you!