Understanding Deep Architectures with Reasoning Layer

Xinshi Chen[1], Yufei Zhang[2], Christoph Reisinger[2], Le Song[1]

[1]Georgia Institute of Technology, [2]University of Oxford

NeurIPS 2020

Unrolled Algorithm As A Layer

$$\mathbf{y}^* = \mathbf{argmin}_{\boldsymbol{y}}\, \boldsymbol{E}^*(\boldsymbol{y})$$

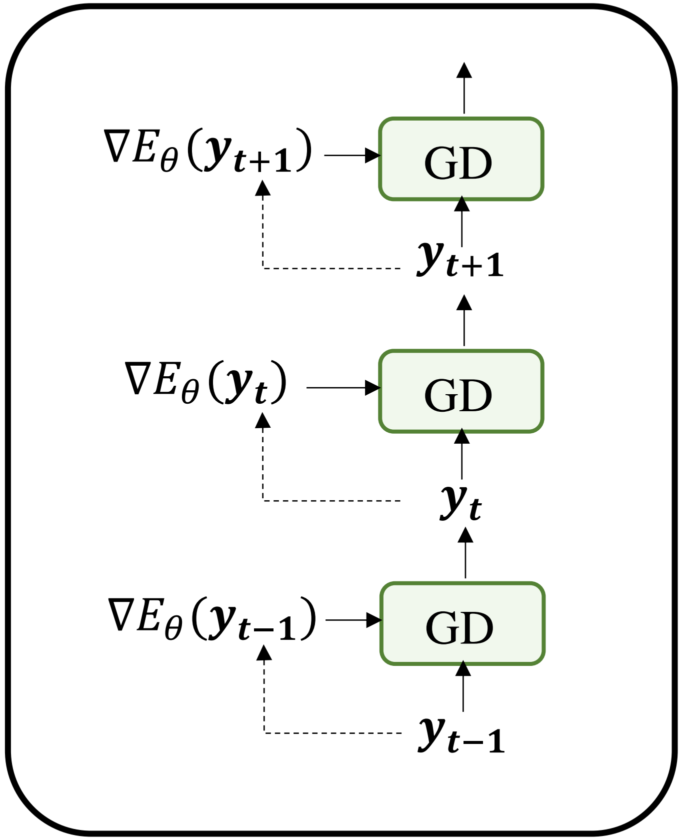
Iterative Algorithm (Gradient Descent)

For $k = 1,2,\ldots$ do

$$\boldsymbol{y}_{t+1} \leftarrow \boldsymbol{y}_k - s\nabla E(\boldsymbol{y}_k)$$

Done

Unrolling



An algorithm can be unrolled and truncated and then used as a specialized layer in the deep learning model.

Ex 1: MAML (Model-Agnostic Meta-Learning)

$k \rightarrow \infty$ implicit-MAML

θ_k (output)

$\theta_2 = \theta_1 - s\nabla_\theta \mathcal{L}(\theta_1, images)$

$\theta_1 = \theta_0 - s\nabla_\theta \mathcal{L}(\theta_0, images)$

θ_0

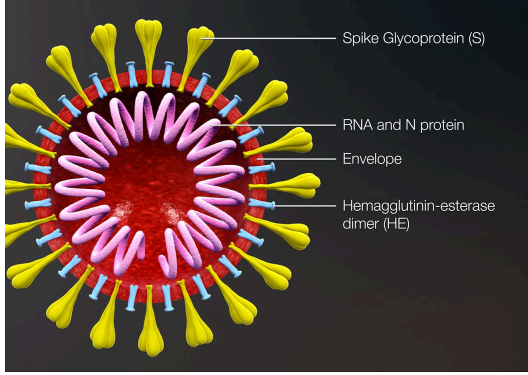
k-step Gradient Descent

$\mathcal{L}(\theta, images)$

(input)

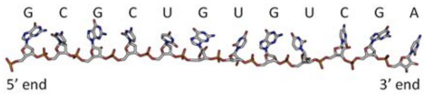
Convolutional Neural Network

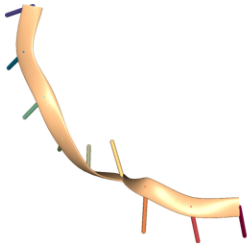
Ex 2: RNA Secondary Structure Prediction

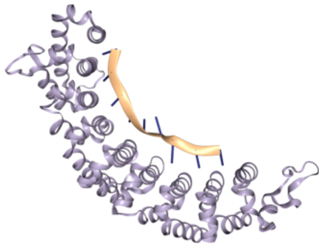


RNA secondary structure prediction

Primary Structure → Secondary Structure → 3D Structure → Function



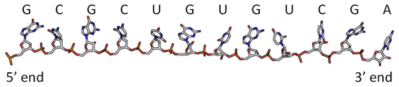




$\boldsymbol{x} = (x_1, x_2, \dots, x_L)$ $\boldsymbol{A}^* \in \{0,1\}^{L \times L}$

Ex 2: E2Efold -- Constrained Optimization Solver as a Layer

$\boldsymbol{x} = (x_1, x_2, \dots, x_L)$

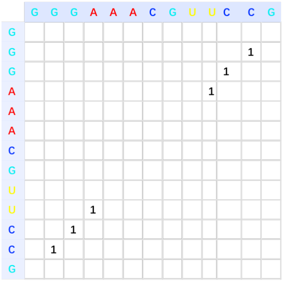


Transformer
& Convolution

$\boldsymbol{E_\theta(x, A)}$

Unrolled Algorithm

$\boldsymbol{A^*}$



$\boldsymbol{A}^* \approx \operatorname{argmin}_{\boldsymbol{A} \in \mathcal{A}} \boldsymbol{E_\theta(x, A)}$

Highly **expressive** module

Encode complex sequence information and dependency

Highly **structured** module

- Enforces the constraints
- Restrict the output space

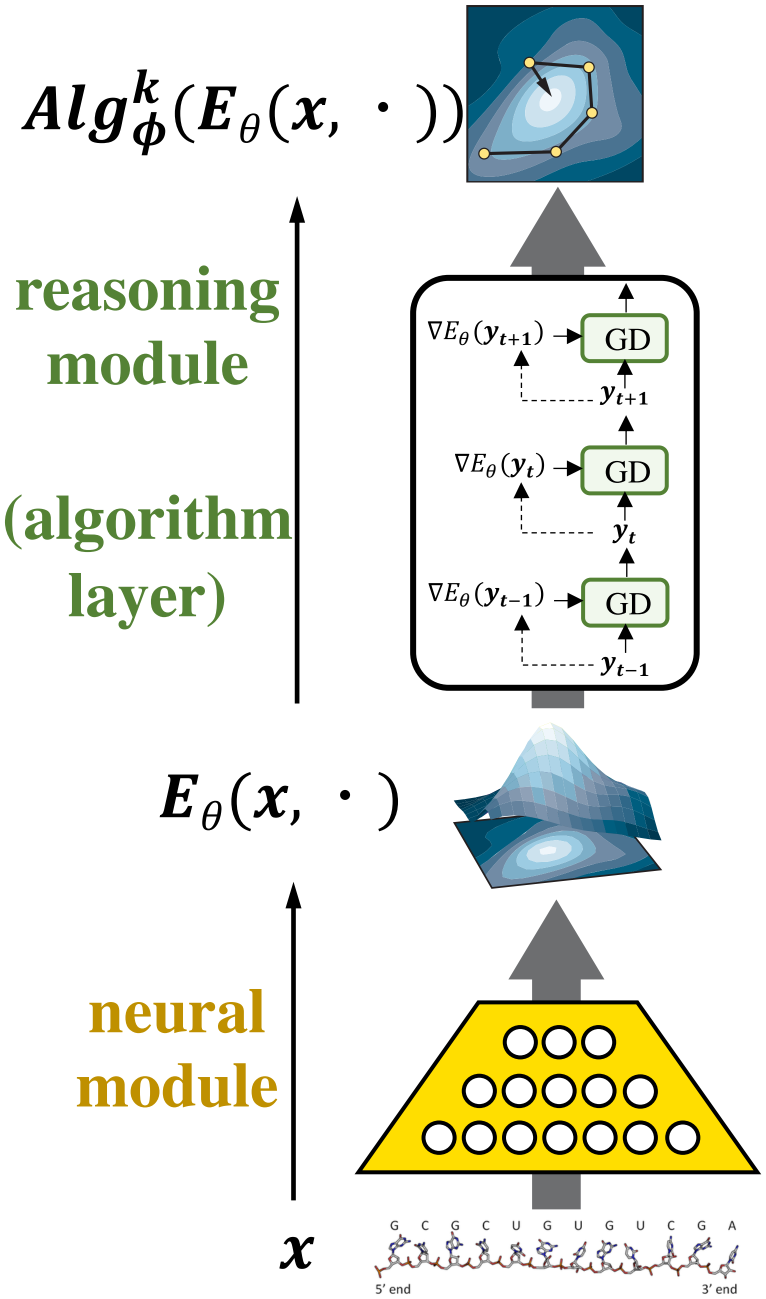
$$\min_{\boldsymbol{A} \in [\boldsymbol{0},\boldsymbol{1}]^{\boldsymbol{L} \times \boldsymbol{L}}} \boldsymbol{E_\theta(x, A)} + \boldsymbol{\rho} \|\boldsymbol{A}\|_{\boldsymbol{1}}$$

$$\text{s.t.} \quad M(x) \circ A = A$$
$$A^\top = A$$
$$A\mathbf{1} \leq \mathbf{1}$$
$$A \geq 0$$

Chen, Xinshi, et al. "RNA Secondary Structure Prediction By Learning Unrolled Algorithms." ICLR 2019

Hybrid Architecture

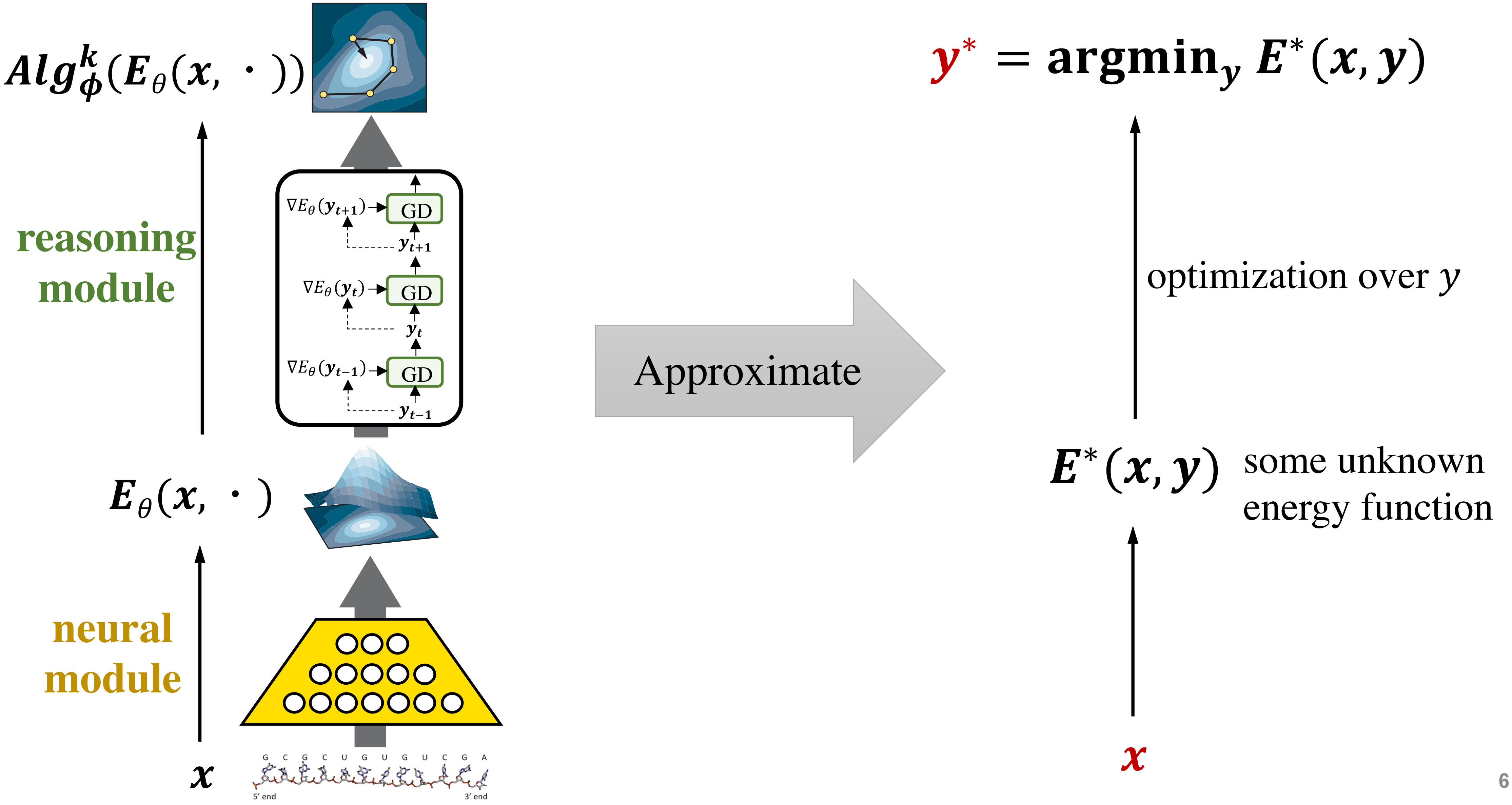
- Unrolled iterative algorithms
- Executes prescribed operations
- Interpretable



- Model complex information of the inputs

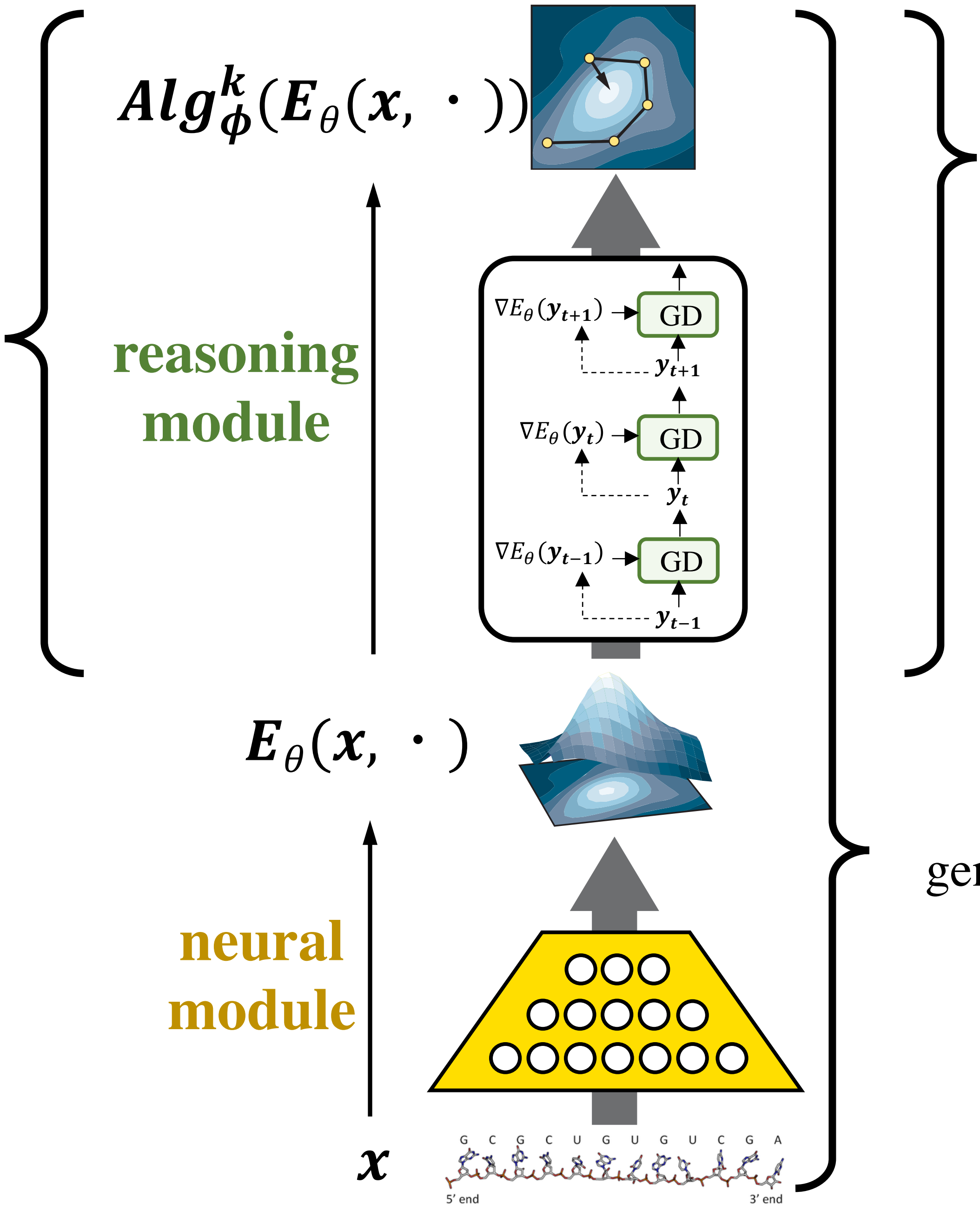
End-to-end differentiable architecture trained with (x, y^*) pairs

Hybrid Architecture



Questions

- Different algorithms can solve the SAME reasoning task
- How are they different from each other when used as a reasoning module?



Algorithm properties
- convergence
- stability
- sensitivity

Effects? Relation?

Representation & generalization ability of the overall hybrid model

Problem Setting: Optimization Module + Neural Energy Module

$$\overbrace{\boldsymbol{Alg}_{\boldsymbol{\phi}}^{\boldsymbol{k}}}^{\text{Unrolled } \textbf{Optimization} \text{ Algorithm}}\left(\underbrace{\boldsymbol{E}_{\theta}(\boldsymbol{x},\boldsymbol{y})}_{\text{Neural Energy}}\right)$$

- $\boldsymbol{\theta}$ **:** parameters in the neural module
- ϕ **:** step size in the unrolled algorithm
- k **:** number of unrolled iterations

- We restrict to the case when $\boldsymbol{E}_{\theta}(\boldsymbol{x},\boldsymbol{y})$ is a quadratic function in $\boldsymbol{y}$.
  - $\boldsymbol{E}_{\theta}(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^{\top}\boldsymbol{Q}_{\theta}(\boldsymbol{x})\boldsymbol{y} + \boldsymbol{y}^{\top}\boldsymbol{b}$, where $\boldsymbol{Q}_{\theta}(\boldsymbol{x})$ is a neural network.
- Ground-truth model is $\boldsymbol{y}^* = \mathbf{argmin}_{\boldsymbol{y}}\, \boldsymbol{E}^*(\boldsymbol{x},\boldsymbol{y})$ for some unknown energy function $\boldsymbol{E}^*$
- Training dataset contains n many input-output pairs $(\boldsymbol{x},\boldsymbol{y}^*)$, without intermediate supervision on $\boldsymbol{E}^*$

How To Design The Reasoning Module (Algorithm Layer)?

- Different optimization algorithms, which one is better?

$Alg =$ Gradient Descent

$$GD_{\phi}^{k}(E_{\theta}(x, y))$$

?

$Alg =$ Nesterov's Accelerated Gradient

$$NAG_{\phi}^{k}(E_{\theta}(x, y))$$

- More iterations k, the better?

$$GD_{\phi}^{k}(E_{\theta}(x, y))$$

?

Equilibrium model

$$GD_{\phi}^{\infty}(E_{\theta}(x, y)) = \mathbf{argmin}_y (E_{\theta}(x, y))$$

Algorithm Property

1) Convergence
   - portrays how fast the optimization error decreases ask the number of iterations k grows.

$$\| Alg_{\phi}^{k}(E(x,y)) - \mathrm{argmin}_{y}(E(x,y)) \| \leq \boldsymbol{Cvg(k,\phi)} \| Alg_{\phi}^{0}\big(E(x,y)\big) - \mathrm{argmin}_{y}(E(x,y)) \|$$

2) Stability
   - characterizes its robustness to small perturbations in the optimization objective $\boldsymbol{E_{\theta}(x,y)}$.

$$\| Alg_{\phi}^{k}(E(x,y)) - Alg_{\phi}^{k}(\hat{E}(x,y)) \| \leq \boldsymbol{Stab(k,\phi)} \| E - \hat{E} \|_{\infty}$$

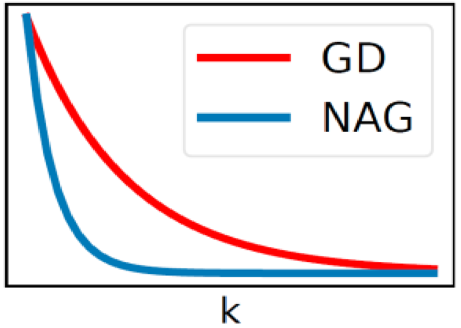
3) Sensitivity
   - characterizes its robustness to small perturbations in the step size ϕ in the algorithm

$$\| Alg_{\phi}^{k}(E(x,y)) - Alg_{\varphi}^{k}(E(x,y)) \| \leq \boldsymbol{Sens(k)} \, | \phi - \varphi |$$

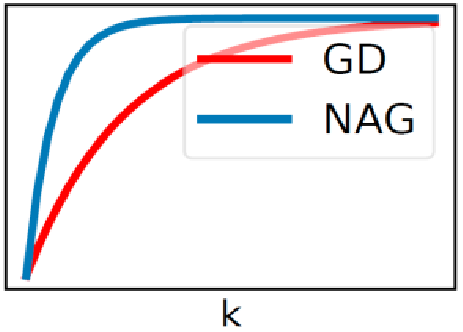
   - Robustness to perturbations in parameters is referred in the deep learning community to "parameter perturbation error" or "sharpness".

GD and NAG: Algorithm Property Comparison

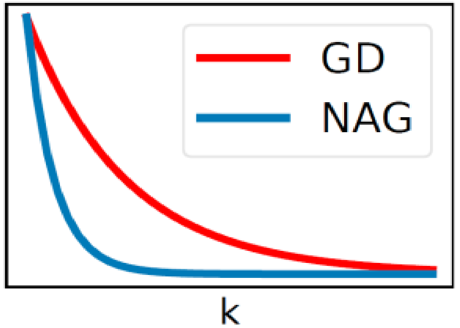
Alg	$\mathit{Cvg}(k,\phi)$	$\mathit{Stab}(k,\phi)$	$\mathit{Sens}(k)$
$\mathtt{GD}_\phi^k$	$\mathcal{O}\left((1-\phi\mu)^k\right)$	$\mathcal{O}\left(1-(1-\phi\mu)^k\right)$	$\mathcal{O}\left(k(1-c_0\mu)^{k-1}\right)$
$\mathtt{NAG}_\phi^k$	$\mathcal{O}\left(k(1-\sqrt{\phi\mu})^k\right)$	$\mathcal{O}\left(1-(1-\sqrt{\phi\mu})^k\right)$	$\mathcal{O}\left(k^3(1-\sqrt{c_0\mu})^k\right)$



NAG converges faster



GD more stable



NAG less sensitive

Faster algorithm less stable

Main Theorem: Local Rademacher Complexity

Local Rademacher complexity of $\boldsymbol{Alg_{\phi}^{k}}\,(\boldsymbol{E}_{\theta}(\boldsymbol{x}, \boldsymbol{y}))$

Theorem 3.1. *Assume the problem setting in Sec 2. Then we have for any $t > 0$, it holds true that*

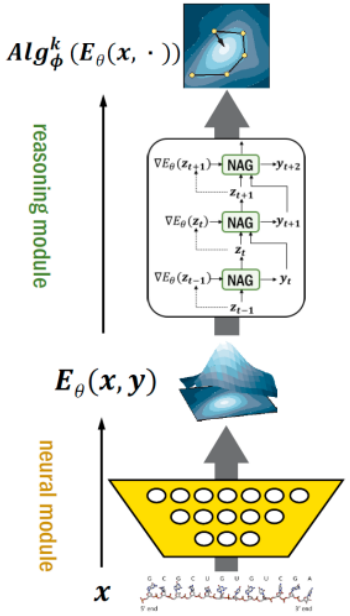
$$\mathbb{E}R_n \ell_{\mathcal{F}}^{loc}(r) \leq \sqrt{2} d n^{-\frac{1}{2}} Stab(k) \left(\sqrt{(Cvg(k)M + \sqrt{r})^2 C_1(n) + C_2(n,t)} + C_3(n,t) + 4 \right) + Sens(k) B_{\Phi}$$

stability **convergence** **sensitivity**

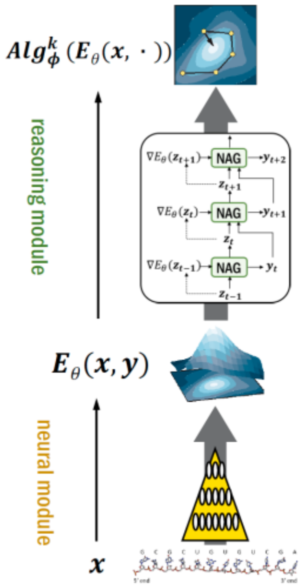
complexity of
neural module

Implication I

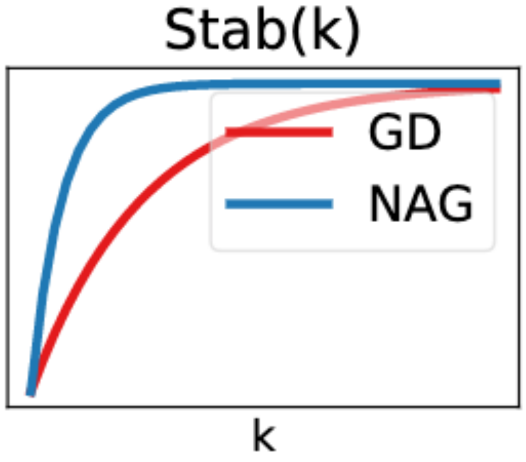
- Bound is dominated by $Stab(k)$
- More iterations ($k \to \infty$), worse generalization
- Fix k, GD generalize better than NAG



Over-parametrization
C_1, C_2, C_3 large

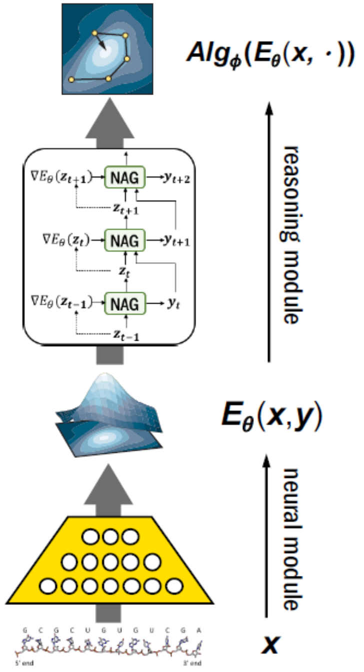


Under-parametrization
C_1, C_2, C_3 small



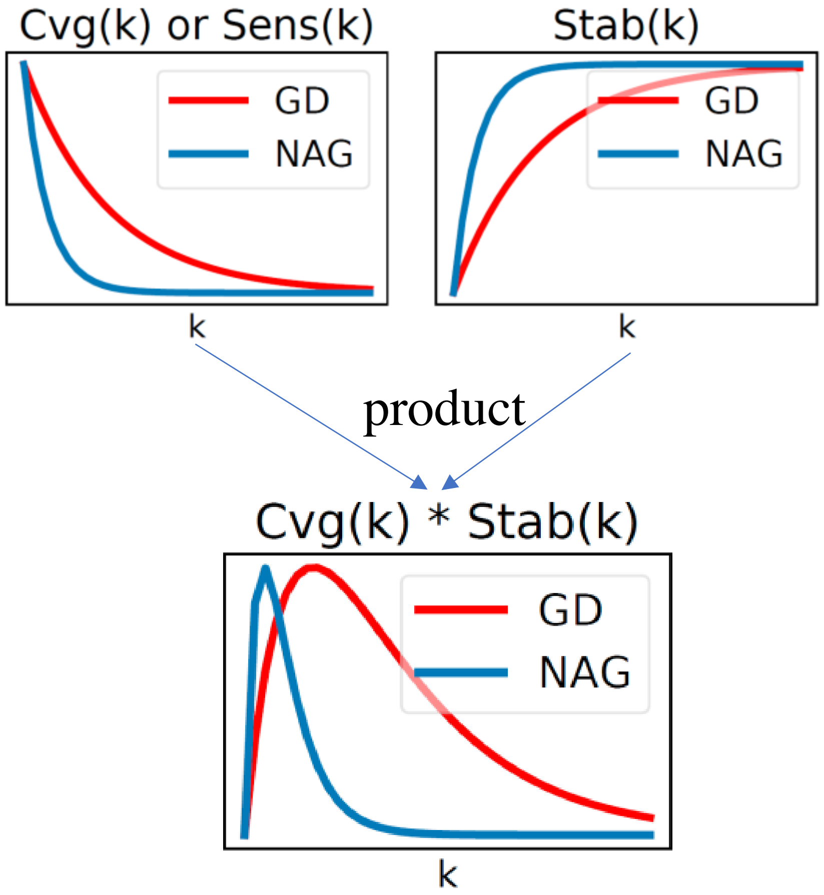
Implication II

- Bound is dominated by the product $Stab(k) * Cvg(k)$
- More iterations (k large) better generalization



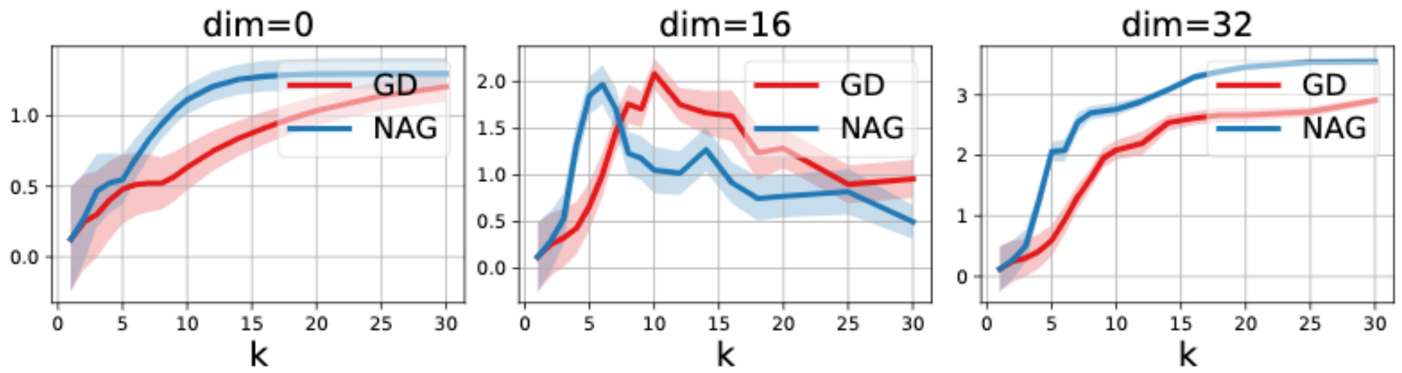
About-right parameterization

C_1 , C_2 , C_3 not large or small

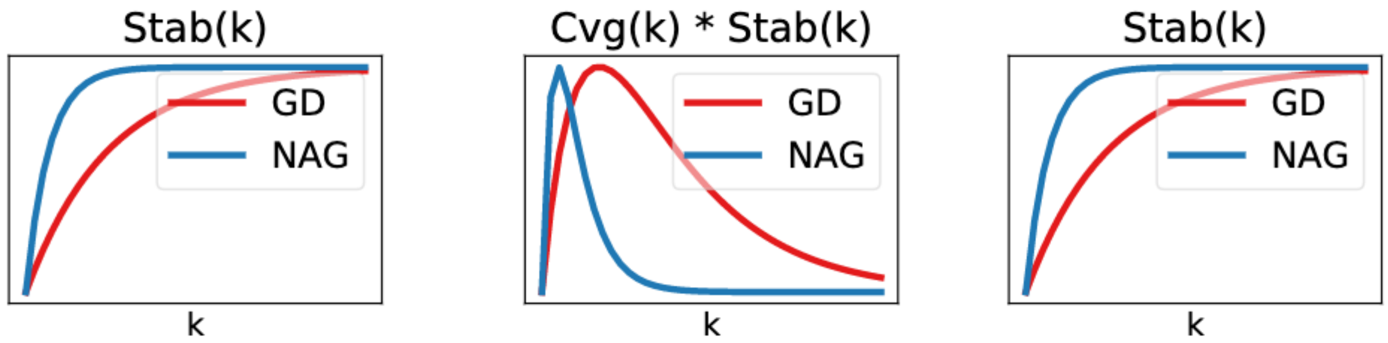


Good Fit between Experiments and Theory

- Generalization gaps, when varying the *hidden dimension* of the neural module.



- Corresponds to the theoretically analyzed algorithm properties:



Align well with the implication of our theorem!

See more details in our paper:

[Understanding Deep Architecture With Reasoning Layer](https://papers.nips.cc/paper/2020/file/0d82627e10660af39ea7eb69c3568955-Paper.pdf)
https://papers.nips.cc/paper/2020/file/0d82627e10660af39ea7eb69c3568955-Paper.pdf

Q&A!