HyperAdam

A Learnable Task-Adaptive Adam

Shipeng Wang, Jian Sun and Zongben Xu

School of Mathematics and Statistics

Xi'an Jiaotong University

wangshipeng8128@stu.xjtu.edu.cn

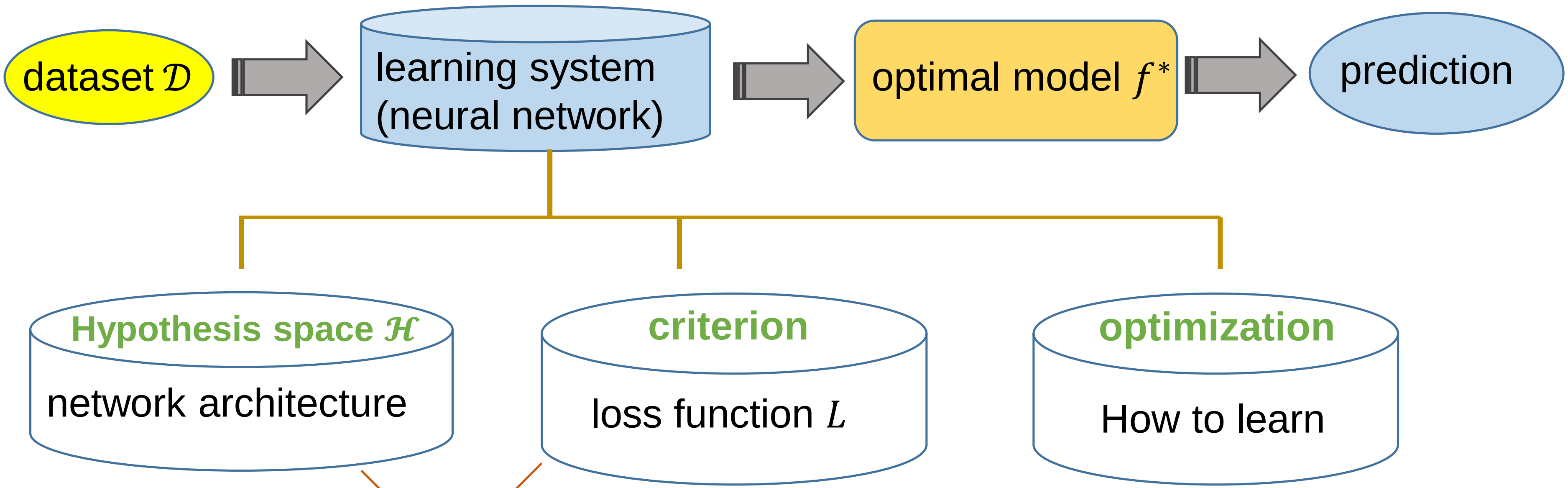
November, 2018

Contents

Introduction

Introduction

An effective optimization algorithm is essential to network training:



$$f^* = \arg\min_{f \in \mathcal{H}} L[f(\mathcal{D})] \quad \Longrightarrow \quad w^* = \arg\min_{w} L[f(\mathcal{D}; w)]$$

$\mathcal{H} = \{f(\mathcal{D}; w)\}$, w : parameters

Introduction

The optimization model for training neural network:

$$w^* = \arg\min_{w} \sum_{i=1}^{n} l(y_i, f(w, x_i))$$

f: learner
w: parameter
l: loss function
$\{x_i, y_i\}_{i=1}^{n}$: dataset

Optimizee: the loss for network training

Optimizer: the optimization algorithm to minimize optimizee.

$$d_t(\Theta) = O(g_t, \mathcal{H}_t, \Theta)$$

g_t: gradient
$\mathcal{H}_t$: historical gradient information
Θ: hyperparameter

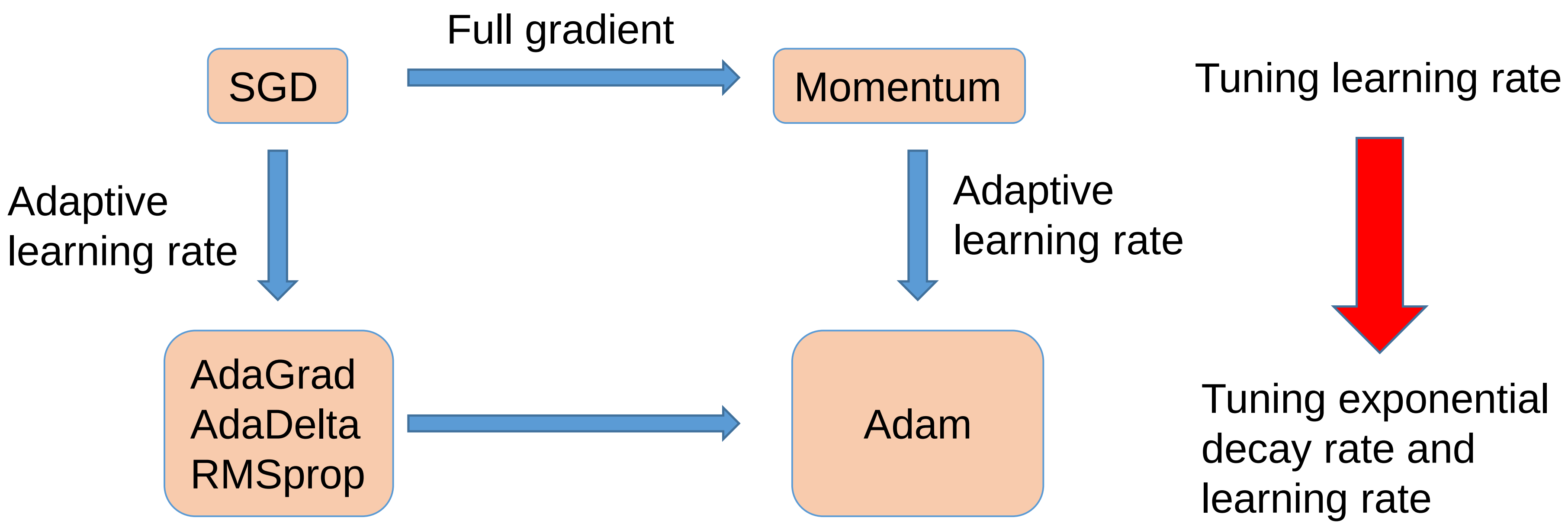
The optimizer O maps the gradient to parameter update d_t.

The optimizer is required to be generalizable to varying network architectures, e.g. network type, depth, width, non-linear activation functions.

Backgrounds

Stochastic Gradient Descent

SGD and its variants are the most popular optimizers in network training.



Adam suffers from unsatisfactory convergence due to the constant decay rates (Reddi, Kale and Kumar, 2018)

$$m_t = \beta m_{t-1} + (1-\beta) g_t$$
$$v_t = \gamma v_{t-1} + (1-\gamma) g_t^2$$
$$w_{t+1} = w_t - \alpha \frac{\operatorname{diag}\left(\frac{v_t}{(1-\gamma)^t}\right)^{-\frac{1}{2}}}{(1-\beta)^t} m_t$$

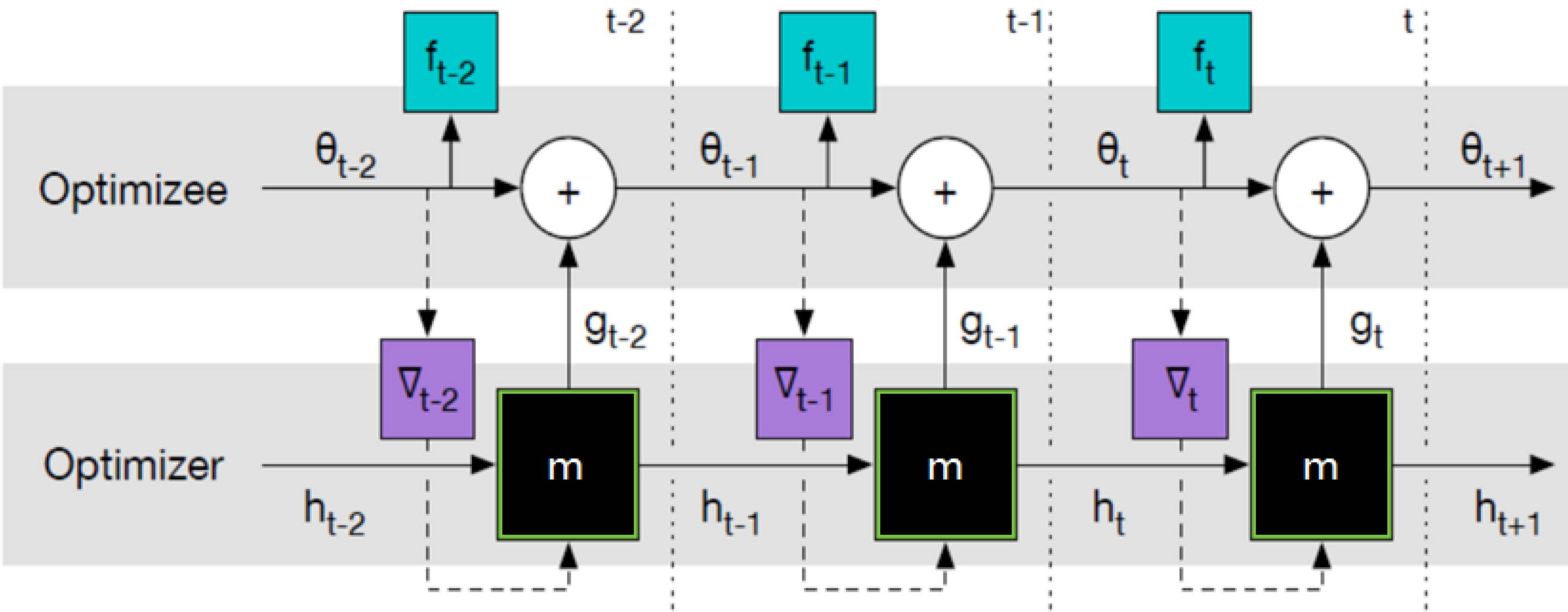
β, γ are exponential decay rates

Reddi, S. J.; Kale, S.; and Kumar, S. On the Convergence of Adam and Beyond. In ICLR, 2018

Learnable Optimizer for Network Training

— black box optimizer

RNN as optimizer (poor generalization ability):



meta-loss

$$\mathcal{L}(\phi) = \mathbb{E}_f\left[\sum_{t=1}^{T} f(w_t)\right]$$

with

$$w_{t+1} = w_t + g_t$$

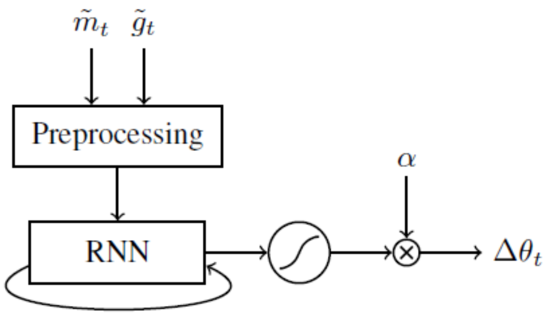
$$\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi)$$

Andrychowicz, Marcin, et al， Learning to Learn by Gradient Descent by Gradient Descent. In NIPS，2016

Learnable Optimizer for Network Training

— black box optimizer

Two training tricks to improve generalization (Kaifeng Lv, et al., 2017):



- Random Scaling (data augmentation)

loss function $f(w)$

Random to scale the parameters

$f_c(w) = f(cw)$

- Combination with Convex Function (large training set)

$F(w, \theta) = f(w) + g(\theta)$

$g(\theta)$ is a convex function, which helps accelerating the training process

The generalization ability of the RNN optimizer is still limited !

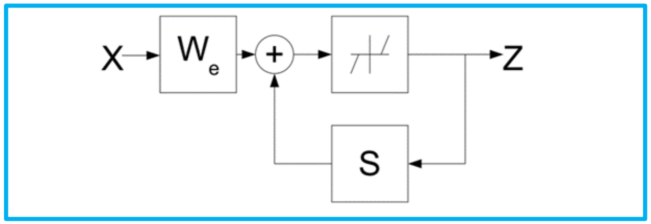
Kaifeng Lv, Shunhua Jiang, Jian Li, Learning Gradient Descent: Better Generalization and Longer Horizons. In ICML, 2017

Unfolding Optimizers as CNN

Unfolding ISTA as LISTA (ICML, 2010)

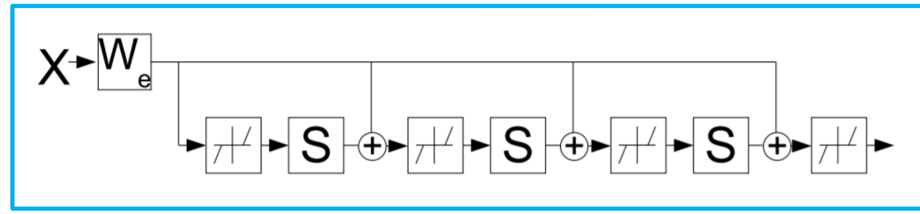
$$E_{W_d}(X,Z) = \frac{1}{2}||X - W_d Z||_2^2 + \alpha ||Z||_1$$ (Sparse Coding)

ISTA



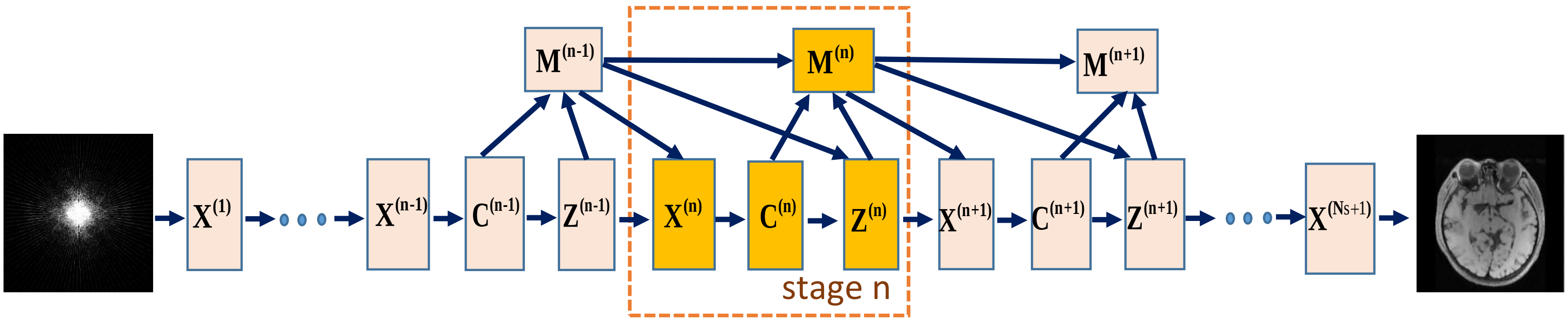
$W_e = W_d^T, S = W_d^T W_d$

LISTA



W_e, S are learned

Unfolding ADMM as ADMM-Net (NIPS, 2016)



Reconstruction layer: $\mathbf{X}^{(n)}$ Convolution layer: $\mathbf{C}^{(n)}$ Nonlinear transform layer: $\mathbf{Z}^{(n)}$ Multiplier update layer: $\mathbf{M}^{(n)}$

Karol Gregor and Yann LeCun, Learning Fast Approximations of Sparse Coding. In ICML, 2010;
Yang Y, Sun J, Li HB & Xu ZB, Deep ADMM-Net for Compressive Sensing MRI. In NIPS, 2016

Summary

Human-designed Optimizer:

Universal but hard to tune the hyper-parameters

Black box Optimizer:

No need to tune the hyper-parameter but not universal

LISTA & ADMM-Net Optimizer:

Model-driven, but fix-step, not for network training

Motivation

Our HyperAdam is proposed based on the following motivation:

Motivation:

- The update rule of optimizer should be adaptive to the task
- The learnable optimizer should generalize well
- For network training, the learnable optimizer should generalize to long horizons.

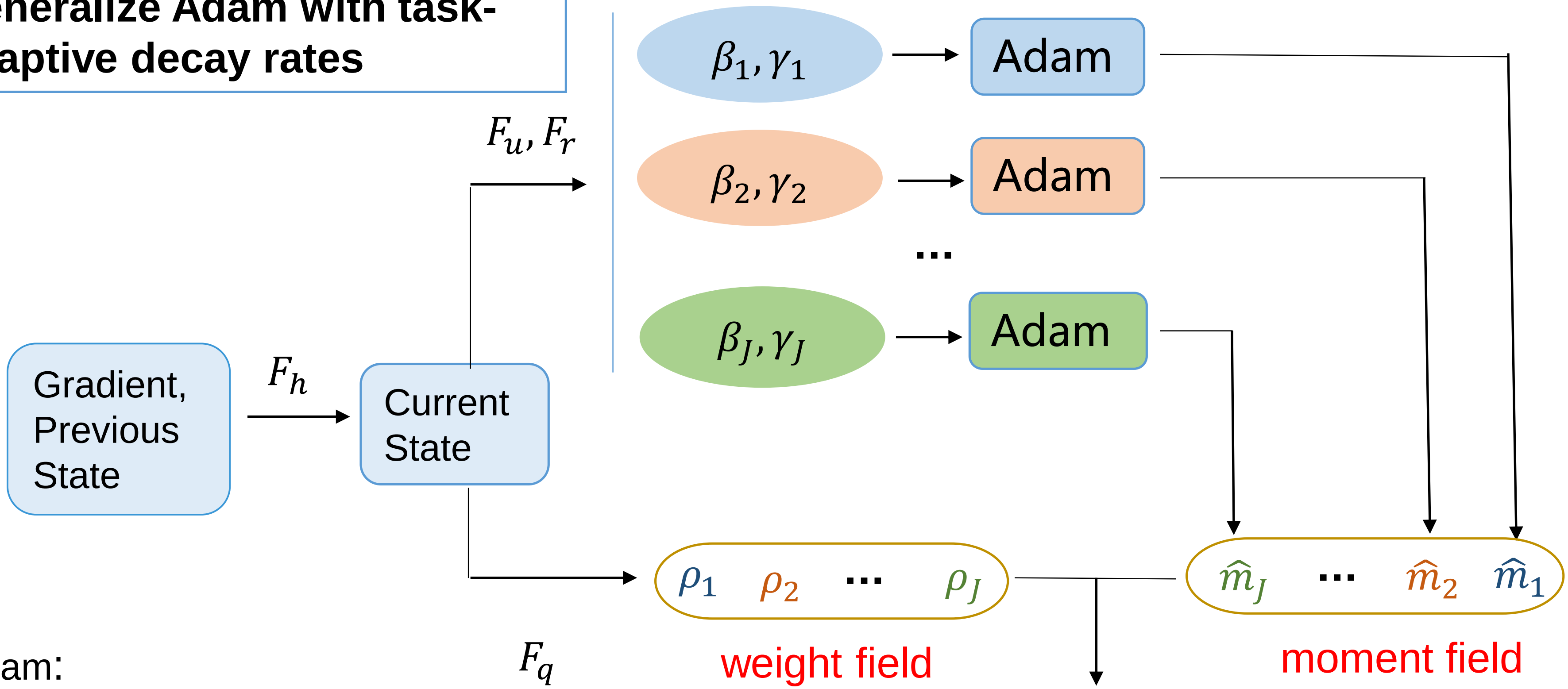
✓ HyperAdam is a first task-adaptive optimizer taking merits of Adam (human-designed optimizer) and learning-based approach in a single framework.

✓ Extensive experiments justify that HyperAdam works well for network training.

HyperAdam

HyperAdam: Generalized Adam

Generalize Adam with task-adaptive decay rates



$$d_t = \Sigma_{j=1}^{J} \rho_j \odot \hat{m}_j$$

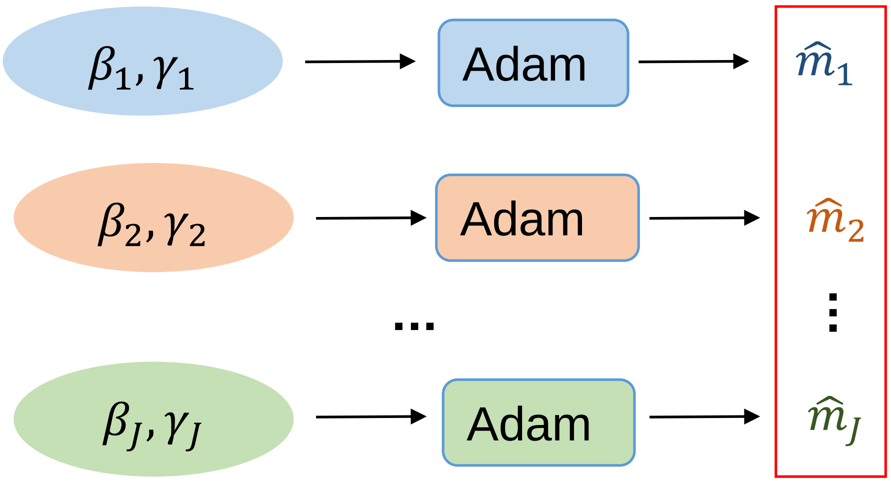
update vector

Adam:

$$m_t = \beta m_{t-1} + (1-\beta) g_t$$
$$v_t = \gamma v_{t-1} + (1-\gamma) g_t^2$$
$$w_{t+1} = w_t - \alpha \frac{\mathrm{diag}\left(\frac{v_t}{(1-\gamma)^t}\right)^{-\frac{1}{2}}}{(1-\beta)^t} m_t$$

HyperAdam: Moment Field

Moment field is inspired by the ensemble technique.

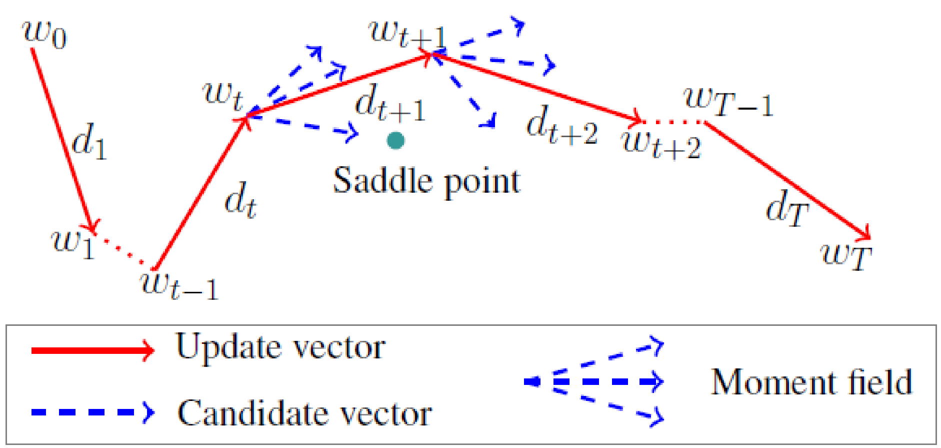


$\widehat{m}_i (i = 1, \dots, J)$: candidate vector

Moment Field: the set containing these candidate vectors.

The parameter update is an ensemble of these updates.

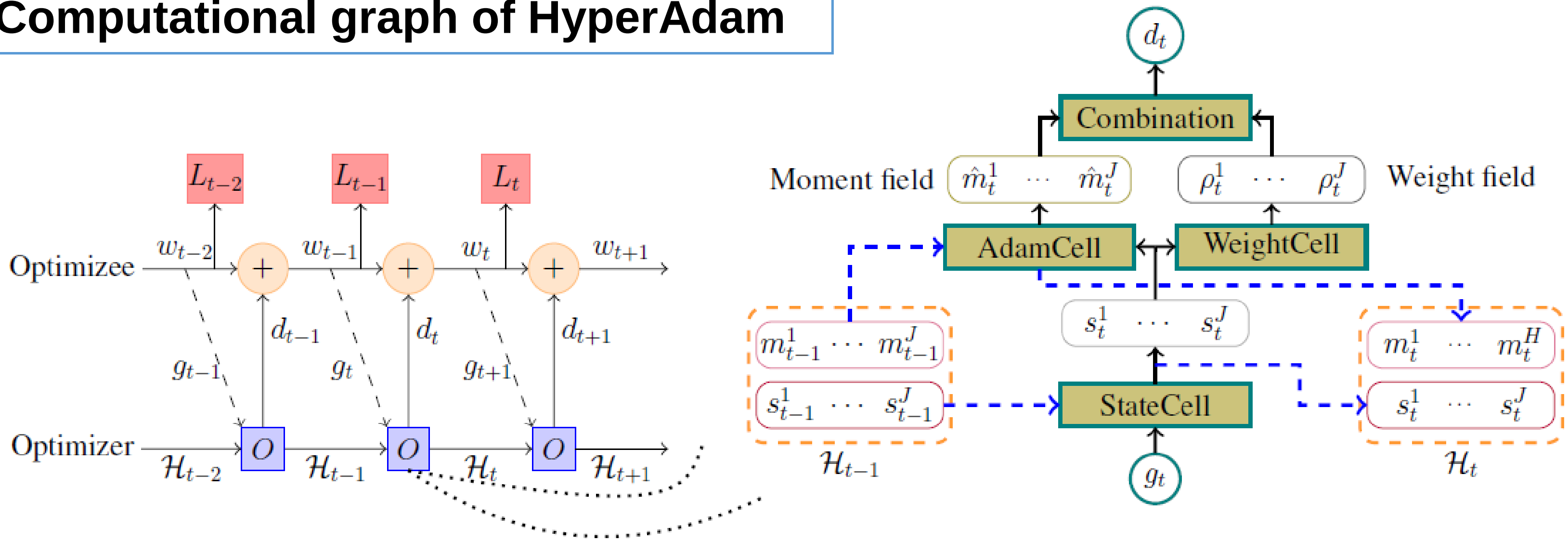
Why moment field?



An adaptive combination of several candidate vectors may potentially relieve the possibility of getting stuck in saddle point.

HyperAdam: Framework

Computational graph of HyperAdam



O: optimizer; g_t: gradient of the optimizee L; d_t: update vectors

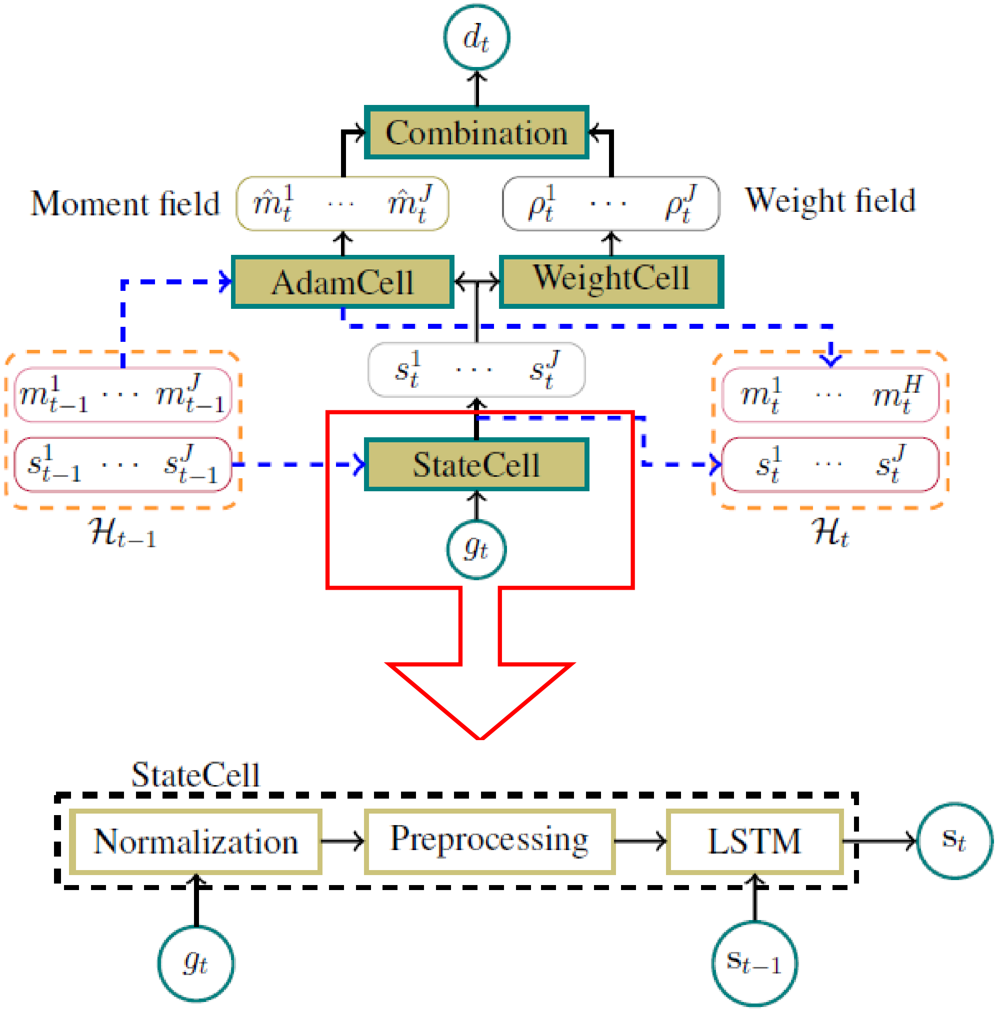
StateCell: encoding the current state $S_t = [s_t^1, \ldots, s_t^J]$

AdamCell: outputting moment field that contains multiple candidate update vectors

WeightCell: outputting weight field that contains multiple weight vectors

Combination: combining these candidate vectors to give the final update vector

HyperAdam: StateCell



Output the current state

Normalization

To achieve the scale invariance property same as traditional Adam.

Preprocessing

Fully connected layer + ELU

LSTM

Current state is determined by the current gradient and previous state.

HyperAdam: AdamCell

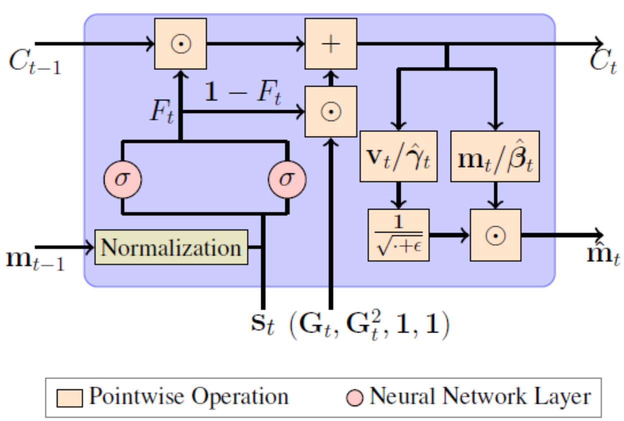
Implementing Adam with different hyper-parameters in parallel.

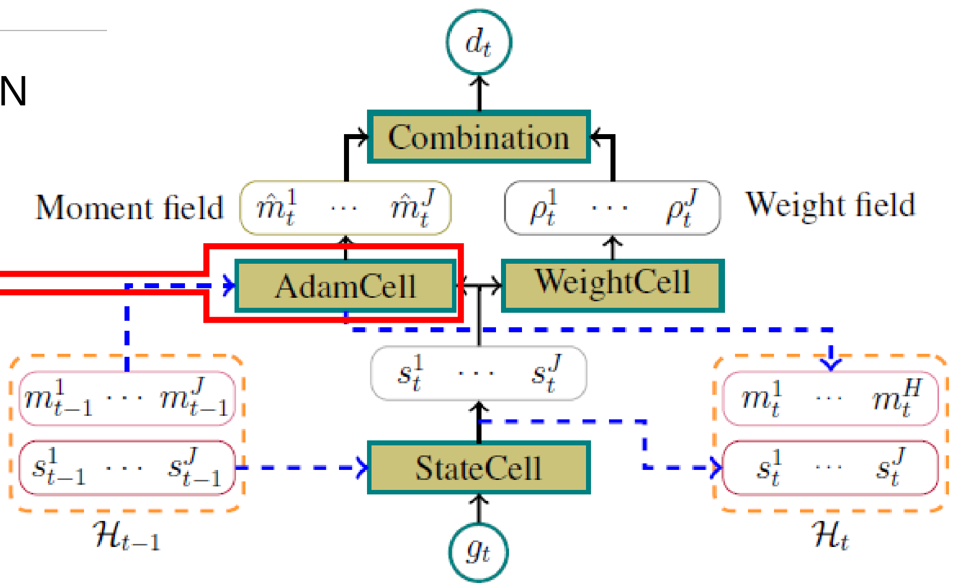
$$\boldsymbol{\beta}_t = \sigma([\mathbf{m}'_{t-1}, \mathbf{s}_t]\boldsymbol{\theta}_u + \mathbf{b}_u),$$
$$\boldsymbol{\gamma}_t = \sigma([\mathbf{m}'_{t-1}, \mathbf{s}_t]\boldsymbol{\theta}_r + \mathbf{b}_r),$$
$$\boxed{C_t = F_t \odot C_{t-1} + (\mathbf{1} - F_t) \odot \tilde{C}_t}$$
$$\tilde{\mathbf{m}}_t = \mathbf{m}_t / \hat{\boldsymbol{\beta}}_t,\ \tilde{\mathbf{v}}_t = \mathbf{v}_t / \hat{\boldsymbol{\gamma}}_t,$$
$$\hat{\mathbf{m}}_t \triangleq [\hat{m}_t^1, \ldots, \hat{m}_t^J] = \frac{\tilde{\mathbf{m}}_t}{\sqrt{\tilde{\mathbf{v}}_t + \varepsilon}}$$

$$F_t = [\boldsymbol{\beta}_t, \boldsymbol{\gamma}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma}_t]$$
$$C_t = [\mathbf{m}_t, \mathbf{v}_t, \hat{\boldsymbol{\beta}}_t, \hat{\boldsymbol{\gamma}}_t]$$
$$\tilde{C}_t = [\mathbf{G}_t, \mathbf{G}_t^2, \mathbf{1}, \mathbf{1}]$$

$$\mathbf{m}_t = \boldsymbol{\beta}_t \odot \mathbf{m}_{t-1} + (\mathbf{1} - \boldsymbol{\beta}_t) \odot \mathbf{G}_t$$
$$\mathbf{v}_t = \boldsymbol{\gamma}_t \odot \mathbf{v}_{t-1} + (\mathbf{1} - \boldsymbol{\gamma}_t) \odot \mathbf{G}_t^2$$
$$\hat{\boldsymbol{\beta}}_t = \boldsymbol{\beta}_t \odot \hat{\boldsymbol{\beta}}_{t-1} + (\mathbf{1} - \boldsymbol{\beta}_t) \odot \mathbf{1}$$
$$\hat{\boldsymbol{\gamma}}_t = \boldsymbol{\gamma}_t \odot \hat{\boldsymbol{\gamma}}_{t-1} + (\mathbf{1} - \boldsymbol{\gamma}_t) \odot \mathbf{1}$$

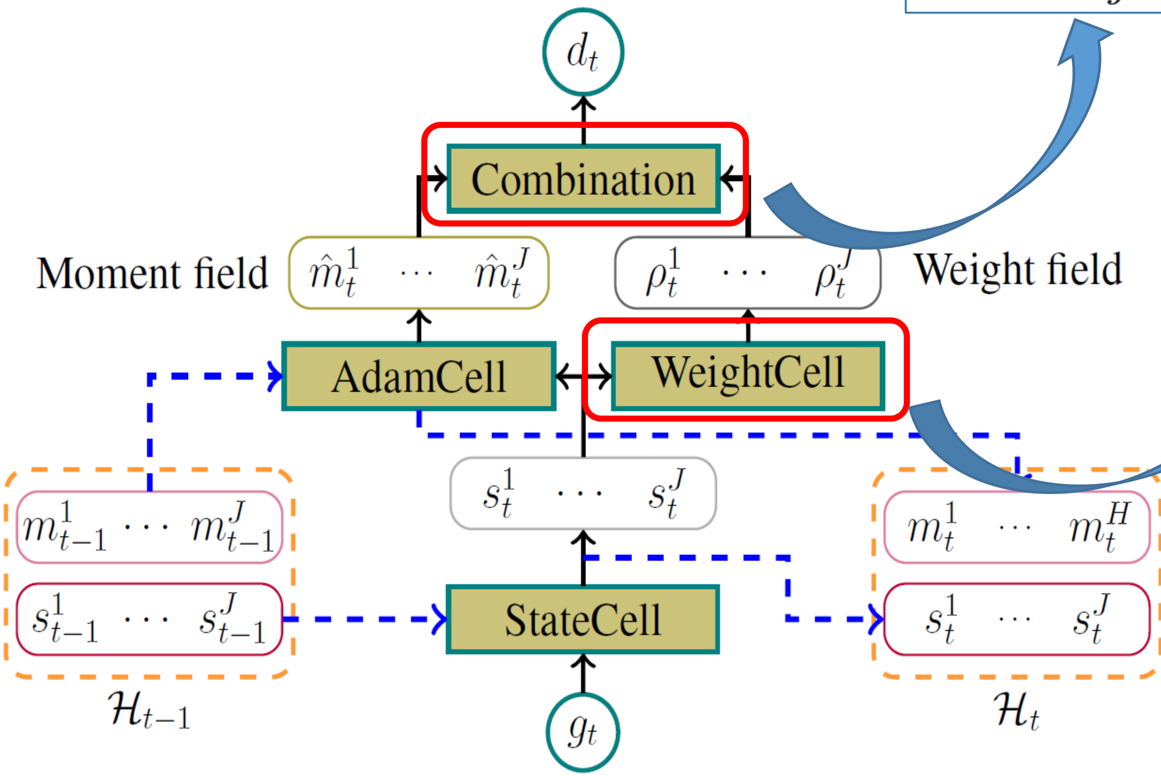
Redesigned as RNN





HyperAdam: WeightCell and Combination

$$d_t = \Sigma_{j=1}^{J} \rho_j \odot \hat{m}_j$$



$$\boldsymbol{\rho}_t = \mathrm{ELU}(\mathbf{s}_t \boldsymbol{\theta}_q + \mathbf{b}_q)$$

Evaluation and Discussion

Learning and Testing HyperAdam

Learning HyperAdam:

Meta-train set: $\{\text{learner: } f, \text{dataset: } \mathcal{D}, \text{loss: } L\}$

Meta-loss: $\mathcal{L}(\Theta) = \mathbb{E}_L[\frac{1}{T}\Sigma_{t=1}^{T} L(f(X; w_t(\Theta)), Y)]$

$$w_t(\Theta) = w_{t-1}(\Theta) - \alpha d_t(g_t, \Theta)$$

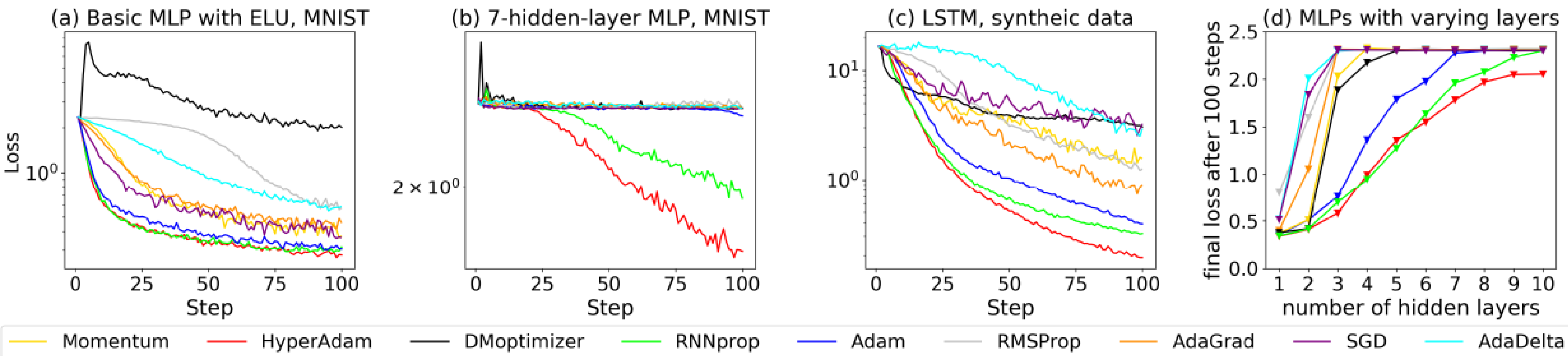
meta-train phase:
f: 1-hidden-layer MLP
$\mathcal{D}$: MNIST
L: cross-entropy
T: 100

Testing HyperAdam:

- Generalization to different activation functions: ReLU, ELU, tanh
- Generalization to different depth: # hidden layer = 2,3,……,10
- Generalization to different structure: CNN, LSTM
- Generalization to different dataset: CIFAR-10
- Generalization to longer horizons: $T = 2000, 10000$

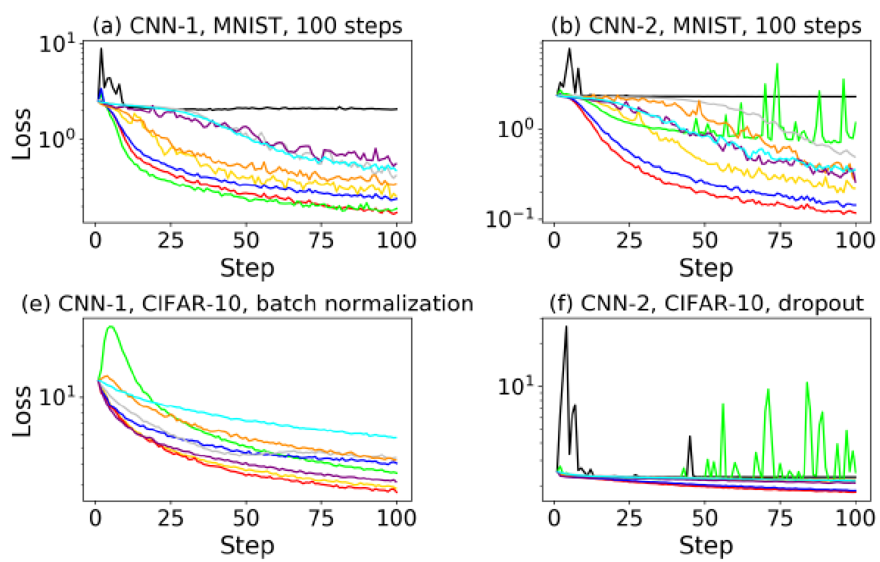
Experiments

Generalization with fixed steps



Activation	Adam	DMoptimizer	RNNprop	HyperAdam
sigmoid	0.35	0.38	0.34	**0.33**
ReLU	0.32	1.42	0.31	**0.29**
ELU	0.31	2.02	0.31	**0.28**
tanh	0.34	0.83	**0.33**	0.36

Table 1: Performance for training basic MLP in 100 steps with different activation functions. Each value is the average final loss for optimizing networks in 100 times.



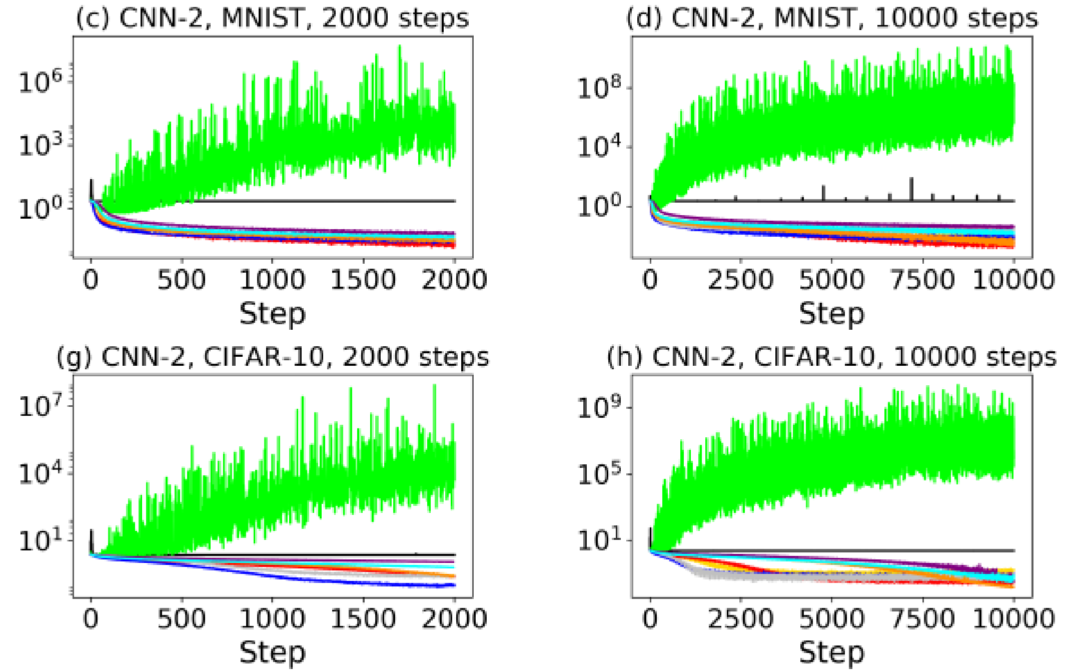
Task	Adam	DMoptimizer	RNNprop	HyperAdam
Baseline	0.65	3.10	0.49	**0.42**
Small noise	0.39	3.06	0.32	**0.19**
2-layer	0.51	2.05	0.27	**0.26**

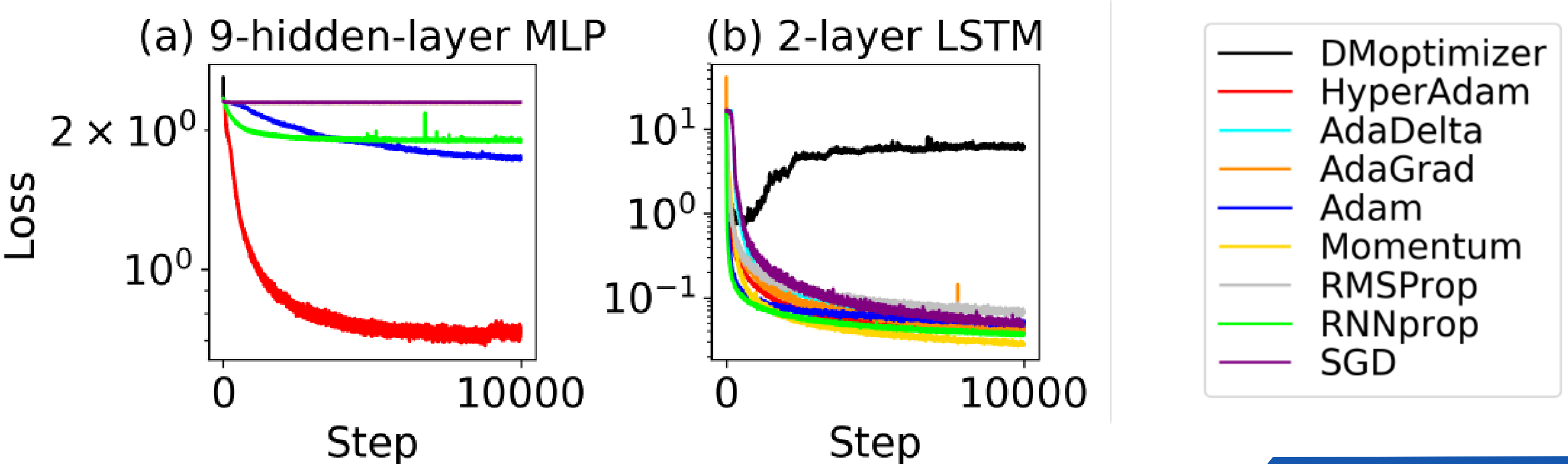
Table 2: Performance on different sequence prediction tasks.

Experiments

Generalization to longer horizons:

- Structure
- Depth
- Dataset





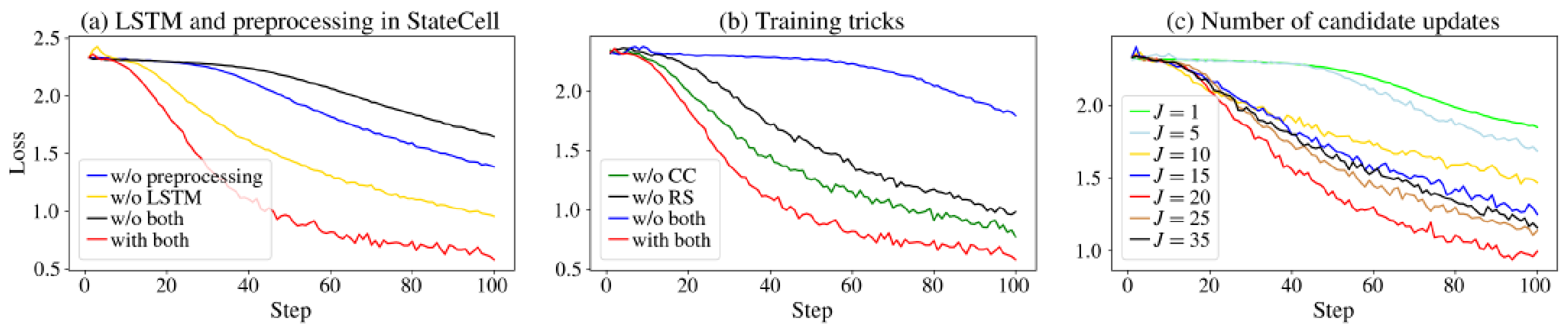
Experiments

Generalization of the Learners

Task	Measure	Adam	DMoptimizer	RNNprop	HyperAdam
CNN-1 (MNIST)	loss	0.10	2.30	0.36	**0.05**
	top-1	**98.50%**	10.10%	96.46%	98.48%
	top-2	99.59%	20.38%	99.03%	**99.63%**
CNN-2 (MNIST)	loss	0.09	2.30	2.30	**0.07**
	top-1	98.98%	11.35%	11.37%	**99.02%**
	top-2	**99.80%**	21.45%	21.69%	99.78%

Table 3: Generalization of the learner trained by Adam, DMoptimizer, RNNprop and HyperAdam for 10000 steps.

Ablation Study



Thanks！

wangshipeng8128@stu.xjtu.edu.cn

HyperAdam: Generalized Adam

Algorithm 2 Task-Adaptive HyperAdam

Require:

Initialized parameter w_0, step size α, batch size N_B.
Dataset $\{(x_i, y_i)\}_{i=1}^{N}$.

Initialize:

$\mathbf{m}_0$, $\mathbf{v}_0$, $\hat{\boldsymbol{\beta}}_0$, $\hat{\boldsymbol{\gamma}}_0$, $\mathbf{s}_0 = \mathbf{0} \in \mathbb{R}^{p \times J}$, $\mathbf{1} \in \mathbb{R}^{p \times J}$, ε=1e-24 .
for all $t = 1, \ldots, T$ **do**
    Draw random batch $\{(x_{i_k}, y_{i_k})\}_{k=1}^{N_B}$ from dataset
    $g_t = \Sigma_{k=1}^{N_B} \nabla l(x_{i_k}, y_{i_k}, w_{t-1})$
    $\mathbf{G}_t = [g_t, \ldots, g_t]$ ▷ $\mathbf{G}_t \in \mathbb{R}^{p \times J}$
    $\mathbf{s}_t = F_h(\mathbf{s}_{t-1}, g_t; \Theta_h)$ ▷ *current state*
    $\boldsymbol{\beta}_t \triangleq [\beta_t^1, \ldots, \beta_t^J] = F_u(\mathbf{s}_t, \mathbf{m}_{t-1}; \Theta_u)$
    $\boldsymbol{\gamma}_t \triangleq [\gamma_t^1, \ldots, \gamma_t^J] = F_r(\mathbf{s}_t, \mathbf{m}_{t-1}; \Theta_r)$
    $\mathbf{m}_t = \boldsymbol{\beta}_t \odot \mathbf{m}_{t-1} + (\mathbf{1} - \boldsymbol{\beta}_t) \odot \mathbf{G}_t$
    $\mathbf{v}_t = \boldsymbol{\gamma}_t \odot \mathbf{v}_{t-1} + (\mathbf{1} - \boldsymbol{\gamma}_t) \odot \mathbf{G}_t^2$
    $\hat{\boldsymbol{\beta}}_t = \boldsymbol{\beta}_t \odot \hat{\boldsymbol{\beta}}_{t-1} + (\mathbf{1} - \boldsymbol{\beta}_t) \odot \mathbf{1}$
    $\hat{\boldsymbol{\gamma}}_t = \boldsymbol{\gamma}_t \odot \hat{\boldsymbol{\gamma}}_{t-1} + (\mathbf{1} - \boldsymbol{\gamma}_t) \odot \mathbf{1}$
    $\tilde{\mathbf{m}}_t = \mathbf{m}_t / \hat{\boldsymbol{\beta}}_t$, $\tilde{\mathbf{v}}_t = \mathbf{v}_t / \hat{\boldsymbol{\gamma}}_t$, ▷ *correcting bias*
    $\hat{\mathbf{m}}_t \triangleq [\hat{m}_t^1, \ldots, \hat{m}_t^J] = \frac{\tilde{\mathbf{m}}_t}{\sqrt{\tilde{\mathbf{v}}_t + \varepsilon}}$ ▷ *moment field*
    $\boldsymbol{\rho}_t \triangleq [\rho_t^1, \ldots, \rho_t^J] = F_q(\mathbf{s}_t; \Theta_q)$ ▷ *weight field*
    $d_t = \Sigma_{j=1}^{J} \rho_t^j \odot \hat{m}_t^j$
    $w_t = w_{t-1} - \alpha d_t$
end for
return final parameter w_T.

Adam:

$$m_t = \beta m_{t-1} + (1-\beta) g_t$$
$$v_t = \gamma v_{t-1} + (1-\gamma) g_t^2$$
$$w_{t+1} = w_t - \alpha \frac{\mathrm{diag}\left(\frac{v_t}{(1-\gamma)^t}\right)^{-\frac{1}{2}}}{(1-\beta)^t} m_t$$

Current State

Determining multiple groups of hyper-parameters

Generating multiple candidate updates with corresponding hyper-parameters in parallel

Combining these updates to get the final update using adaptively learned combination weights