Optimizing Optimizers

summarized by Michael Scherbela

Deep Learning Seminar
March 22, 2023

universität
wien

Let‘s take our usual optimizers…

Gradient-based, first-order optimizers:

General optimizer:

```
def get_update(g, state, hyperp):
  return update, state
```

e.g. SGD:

```
def sgd_momentum(g, m, lr, beta):
  m = m * beta + g * (1-beta)
  return -m*lr, m
```

… and „optimize“ them

detailed next

A **Learning to learn by gradient descent by gradient descent**
Google, NeurIPS 2016

- Replace **get_update** by a Neural Network and learn it

B **Symbolic Discovery of Optimization Algorithms**
Google, Feb 2023 (arxiv)

- Let **get_update** be a “normal”, simple function
- Find optimal function using evolutionary algorithms

C **Gradient Descent: The Ultimate Optimizer**
MIT, NeurIPS 2022

- Keep standard optimizers (Adam, SGD, …)
- **Optimize hyper-parameters** during training using a hyper-optimizer

Neural Network as Optimizer

$f(\theta)$ ... loss function

θ ... model parameters
ϕ ... optimizer parameter

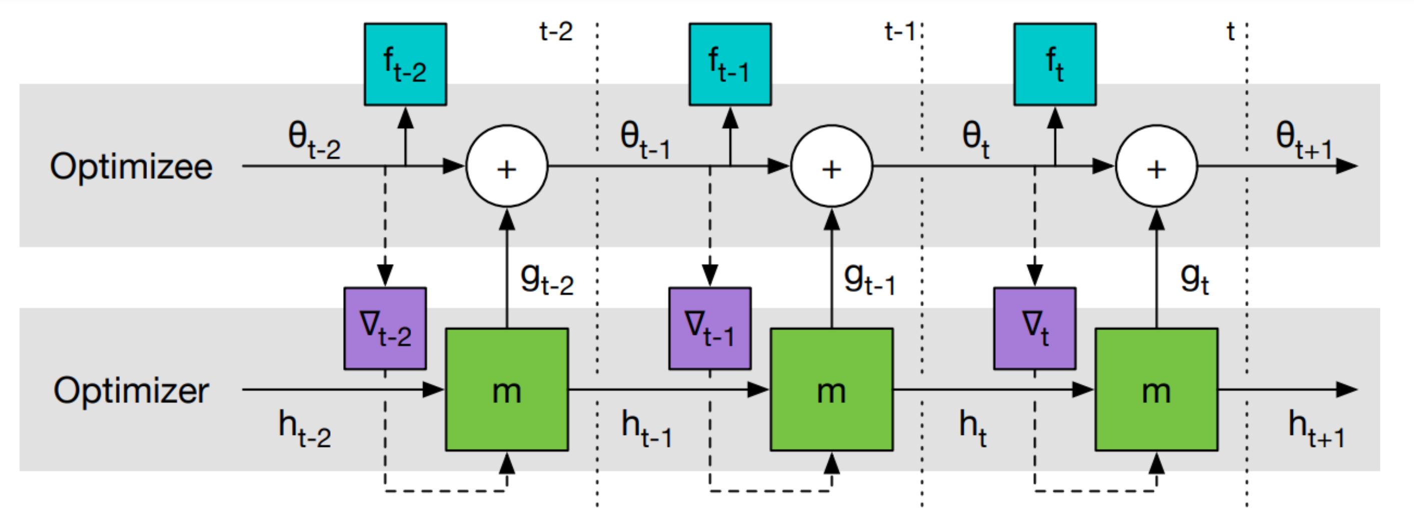
Goal: Minimize loss of final model parameters

$$\mathcal{L}(\phi) = \mathbb{E}_f\left[f\left(\theta^*(f,\phi)\right)\right]$$

In practice: weighted loss along trajectory

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

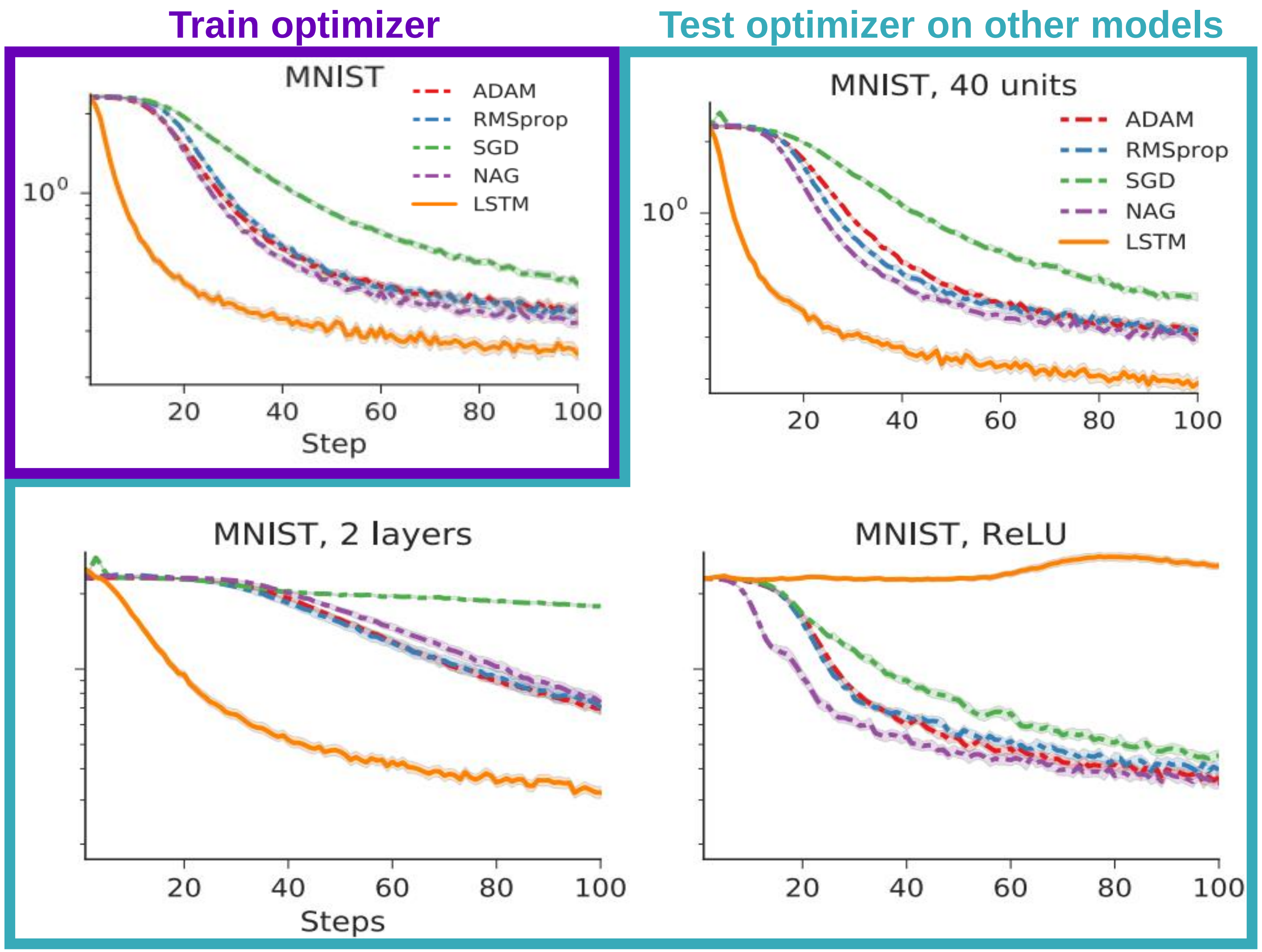
Computational graph



- Optimizer m implemented as LSTM
- Elementwise updates of model parameters

Learning to learn by gradient descent by gradient descent, Andrychowicz et al. 2016

Works well for toy systems, but appears not to generalize



Test system:

- MNIST handwritten digits
- MLP with 1 hidden layer (20 neurons)

Other tested model architectures:

- Quadratic problems
- CNN for CIFAR-10
- Style Transfer

Learning to learn by gradient descent by gradient descent, Andrychowicz et al. 2016

Let‘s take our usual optimizers…

Gradient-based, first-order optimizers:

SGD:

```
def sgd_momentum(g, m, lr, beta):
  m = m * beta + g * (1-beta)
  return -m*lr, m
```

General optimizer:

```
def get_update(g, state, hyperp):
  return update, state
```

… and „optimize“ them

A **Learning to learn by gradient descent by gradient descent**
Google, NeurIPS 2016

- Replace **get_update** by a Neural Network and optimize it

detailed next

B **Symbolic Discovery of Optimization Algorithms**
Google, Feb 2023 (arxiv)

- Let **get_update** be a normal, simple function
- Search for an optimal function using evolutionary algorithms

C **Gradient Descent: The Ultimate Optimizer**
MIT, NeurIPS 2022

- Keep standard optimizers (Adam, SGD, …)
- **Optimizer hyper-parameters** during training using a hyper-optimizer

Evolutionary search for optimizer function

Initial algorithm (AdamW)

```
def train(w, g, m, v, lr):
  g2 = square(g)
  m = interp(g, m, 0.9)
  v = interp(g2, v, 0.999)
  sqrt_v = sqrt(v)
  update = m / sqrt_v
  wd = w * 0.01
  update = update + wd
  lr = lr * 0.001
  update = update * lr
  return update, m, v
```

Allowed statements

- **Unary functions**:
  sqrt, abs, cos, …
- **Binary functions**:
  +, -, *, /, **, …
- **Basic linalg**:
  norm, dot, cosine_sim
- **Linear interpolation**:
  $interp(a, b, \gamma) = (1 - \gamma)a + \gamma b$

Not used

- **Loops**
- **If-conditions**

Random mutations

- Insert new random statement
- Delete random statement
- Change arguments of statement
  - Existing variable
  - New random constant
- Change constants

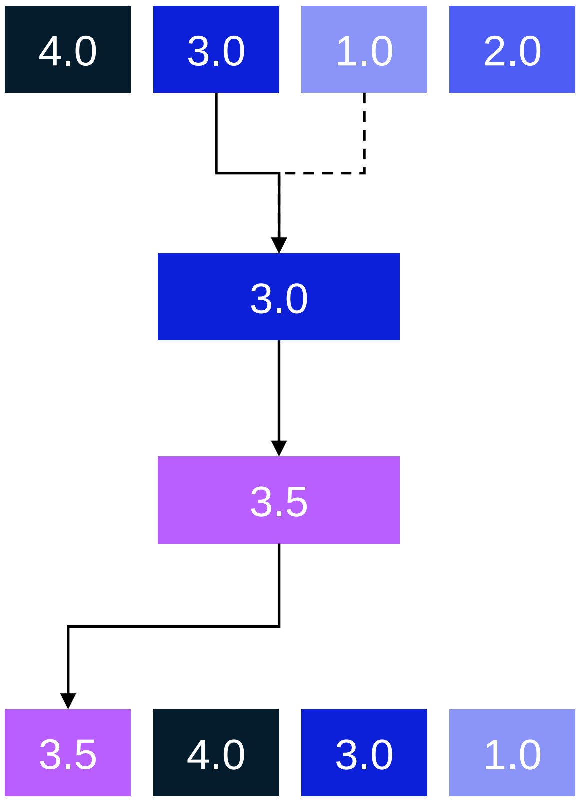
Evolution using Tournament Selection

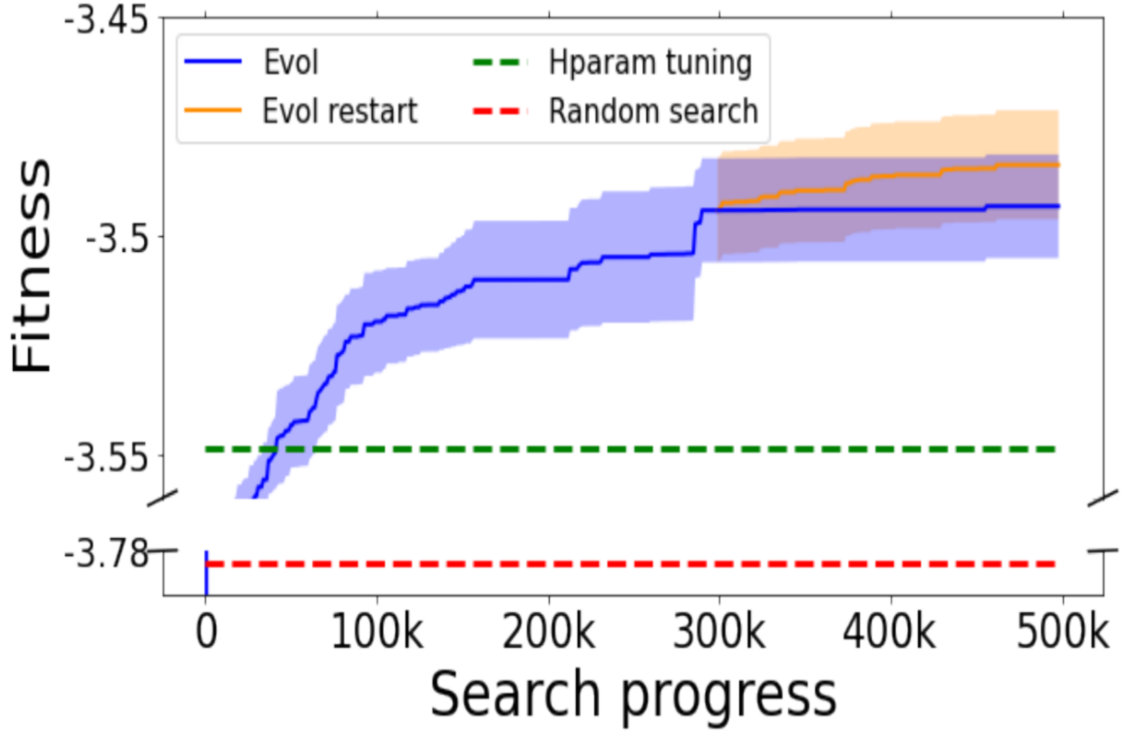
Population of algorithms

Pick 2 random, copy best

Mutate and evaluate[1]

Add to population and drop oldest





1 Evaluation of optimizer = short optimization of small proxy task (~20 min)

Symbolic Discovery of Optimization Algorithms, Chen et al. 2023

Best Optimizer: Lion

EvoLved Sign Momentum

Search output

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  m = clip(m, lr)
  v845 = sqrt(0.6270633339881897)
  v968 = sign(v)
  v968 = v - v
  g = arcsin(g)
  m = interp(g, v, 0.8999999761581421)
  v1 = m * m
  v = interp(g, m, 1.109133005142212)
  v845 = tanh(v845)
  lr = lr * 0.0002171761734643951
  update = m * lr
  v1 = sqrt(v1)
  update = update / v1
  wd = lr * 0.4601978361606598
  v1 = square(v1)
  wd = wd * w
  m = cosh(update)
  lr = tan(1.4572199583053589)
  update = update + wd
  lr = cos(v845)
  return update, m, v
```

After redundant code removal

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  g = arcsin(g)
  m = interp(g, v, 0.899)
  m2 = m * m
  v = interp(g, m, 1.109)
  abs_m = sqrt(m2)
  update = m / abs_m
  wd = w * 0.4602
  update = update + wd
  lr = lr * 0.0002
  m = cosh(update)
  update = update * lr
  return update, m, v
```

After simplification: Lion

```
def train(weight, gradient, momentum, lr):
  update = interp(gradient, momentum, β1)
  update = sign(update)
  momentum = interp(gradient, momentum, β2)
  weight_decay = weight * λ
  update = update + weight_decay
  update = update * lr
  return update, momentum
```

$\beta_1 = 0.9 \ \ \beta_2 = 0.99$

Use full gradient

Keep only sign

Lion Code

```
def train(weight, gradient, momentum, lr):
  update = interp(gradient, momentum, β1)
  update = sign(update)
  momentum = interp(gradient, momentum, β2)
  weight_decay = weight * λ
  update = update + weight_decay
  update = update * lr
  return update, momentum
```

$\beta_1 = 0.9 \qquad \beta_2 = 0.99$

Properties

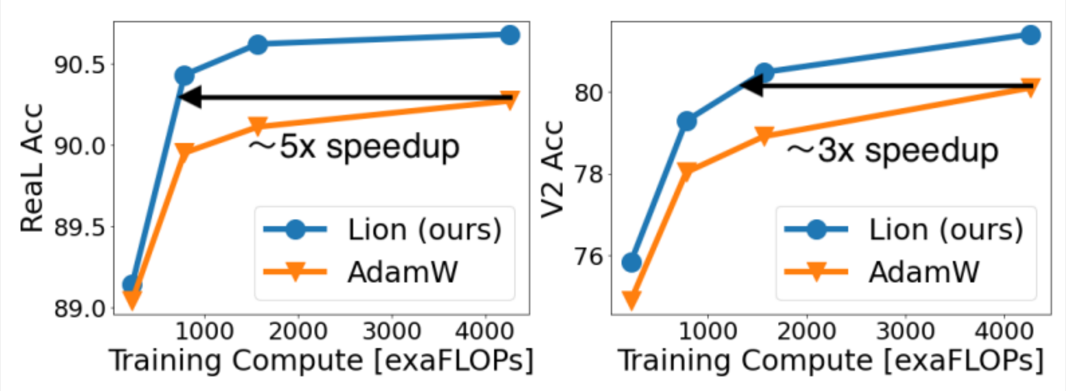
- **Sign update**
  - Adds quantization noise: Regularization?
  - Uniform magnitude
- **Momentum**
  - Long momentum history (0.99 vs 0.9 for Adam)
  - More weight on current gradient (0.1)
  - Single interpolation (either 0.9 or 0.99) much worse
- **Compute**
  - Only 1 momentum: Less memory
  - 2-15% faster than Adam

Symbolic Discovery of Optimization Algorithms, Chen et al. 2023

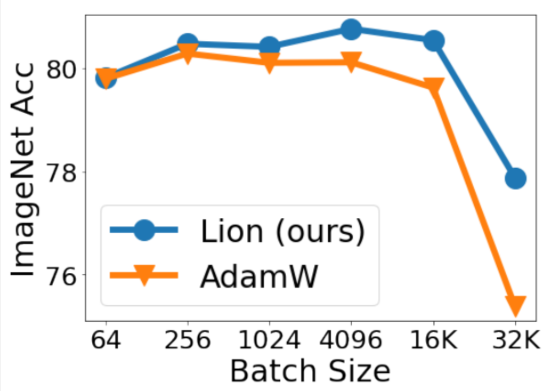
New SOTA on ImageNet

Model	#Params	Optimizer	RandAug + Mixup	ImageNet	ReaL	V2
Train from scratch on ImageNet						
ResNet-50	25.56M	SGD	✗	76.22	82.39	63.93
		AdamW		76.34	**82.72**	**64.24**
		Lion		**76.45**	**82.72**	64.02
Mixer-S/16	18.53M	AdamW	✗	69.26	75.71	55.01
		Lion		**69.92**	**76.19**	**55.75**
Mixer-B/16	59.88M	AdamW	✗	68.12	73.92	53.37
		Lion		**70.11**	**76.60**	**55.94**
ViT-S/16	22.05M	AdamW	✗	76.12	81.94	63.09
		Lion		**76.70**	**82.64**	**64.14**
		AdamW	✓	78.89	84.61	66.73
		Lion		**79.46**	**85.25**	**67.68**
ViT-B/16	86.57M	AdamW	✗	75.48	80.64	61.87
		Lion		**77.44**	**82.57**	**64.81**
		AdamW	✓	80.12	85.46	68.14
		Lion		**80.77**	**86.15**	**69.19**
CoAtNet-1	42.23M	AdamW	✓	83.36 (83.3)	-	-
		Lion		**84.07**	-	-
CoAtNet-3	166.97M	AdamW	✓	84.45 (84.5)	-	-
		Lion		**84.87**	-	-
Pre-train on ImageNet-21K then fine-tune on ImageNet						
ViT-B/16_{384}	86.86M	AdamW	✗	84.12 (83.97)	88.61 (88.35)	73.81
		Lion		**84.45**	**88.84**	**74.06**
ViT-L/16_{384}	304.72M	AdamW	✗	85.07 (85.15)	88.78 (88.40)	75.10
		Lion		**85.59**	**89.35**	**75.84**

Faster training



Larger batch-size



Works across models

- Image classification (ResNet and ViT)
- Text-to-Image Diffusion
- LLMs

Symbolic Discovery of Optimization Algorithms, Chen et al. 2023

Let‘s take our usual optimizers…

Gradient-based, first-order optimizers:

SGD:

```
def sgd_momentum(g, m, lr, beta):
  m = m * beta + g * (1-beta)
  return -m*lr, m
```

General optimizer:

```
def get_update(g, state, hyperp):
  return update, state
```

… and „optimize“ them

A **Learning to learn by gradient descent by gradient descent**
Google, NeurIPS 2016

- Replace **get_update** by a Neural Network and optimize it

B **Symbolic Discovery of Optimization Algorithms**
Google, Feb 2023 (arxiv)

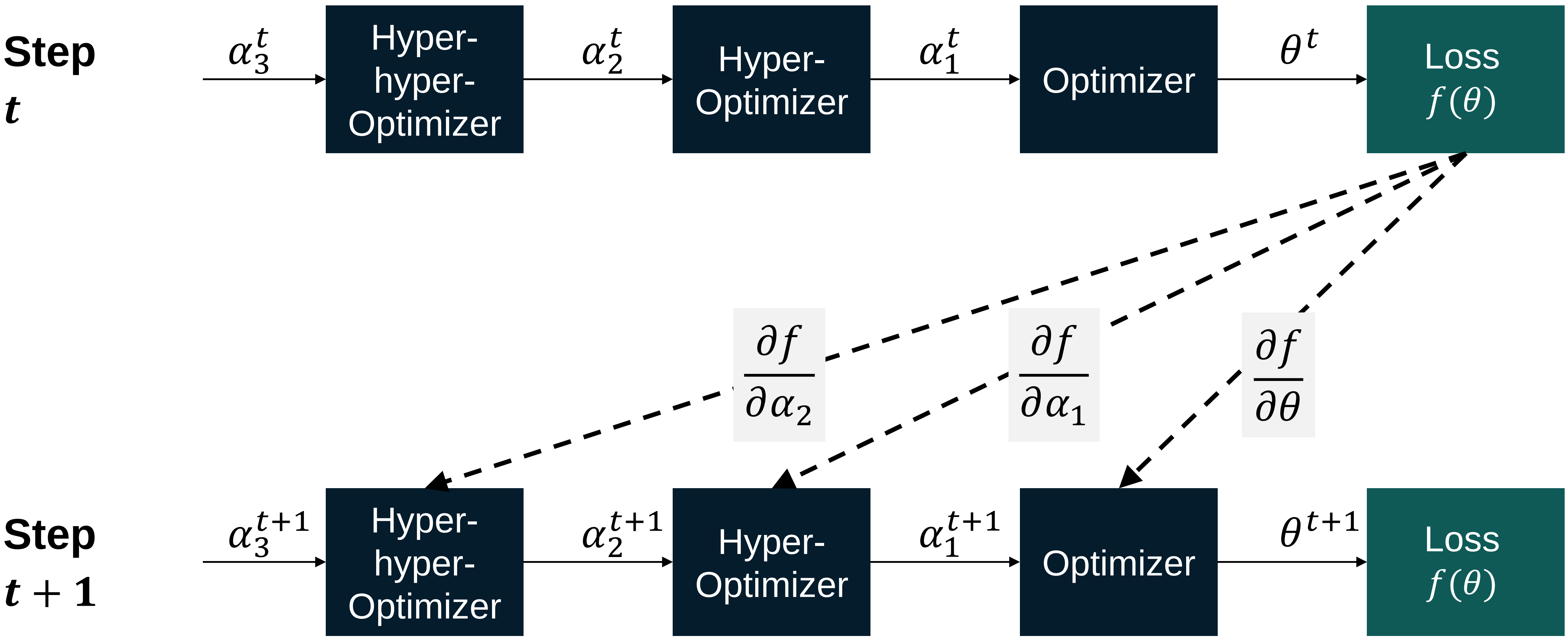
- Let **get_update** be a normal, simple function
- Search for an optimal function using evolutionary algorithms

detailed next

C **Gradient Descent: The Ultimate Optimizer**
MIT, NeurIPS 2022

- Keep standard optimizers (Adam, SGD, …)
- **Optimizer hyper-parameters** during training using a hyper-optimizer

Idea: Stack optimizers to optmizer hyperparameters

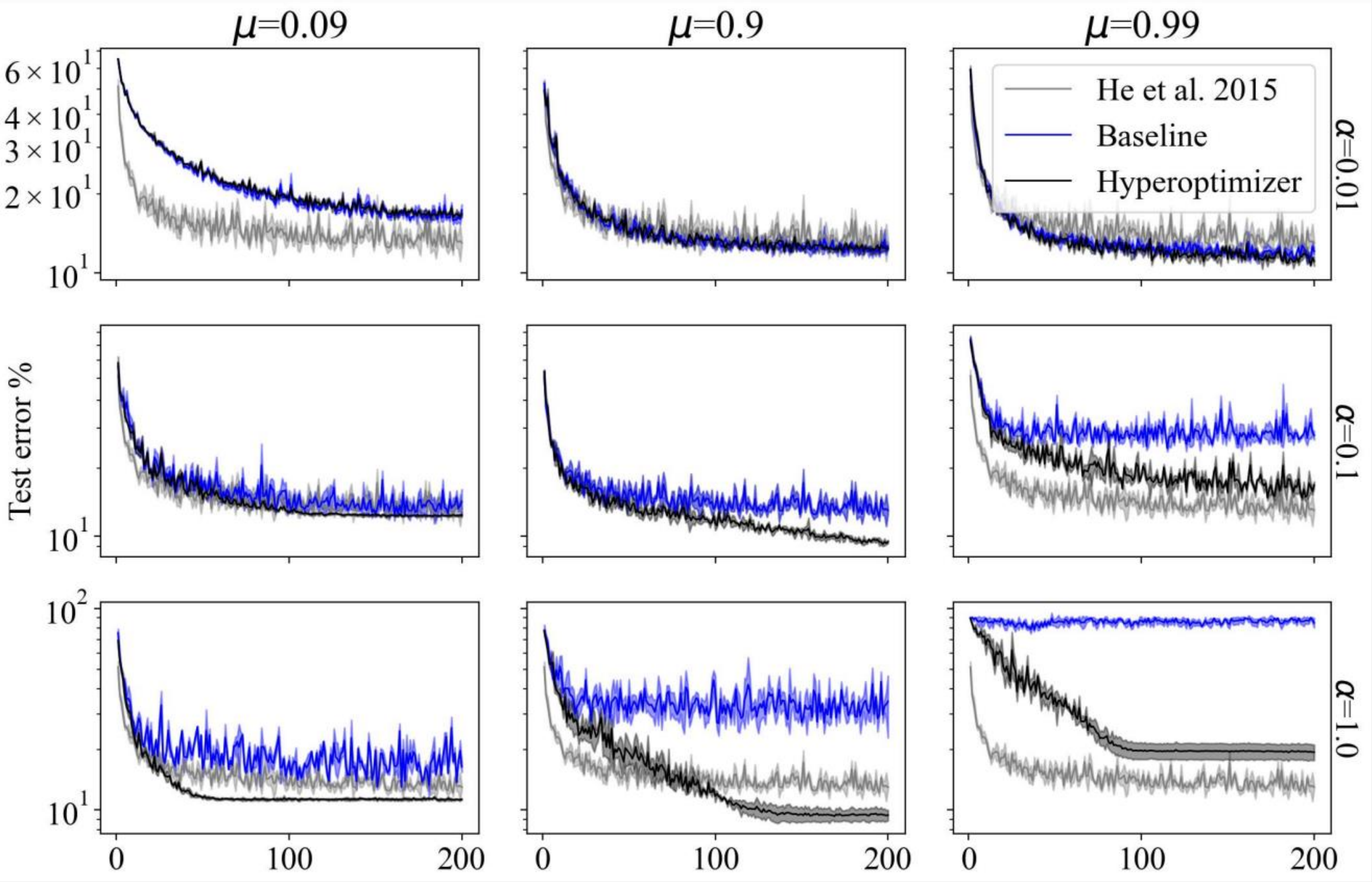


ResNet on CIFAR-10

Hyper-optimizer yields decent results, even with bad initial hyper-params

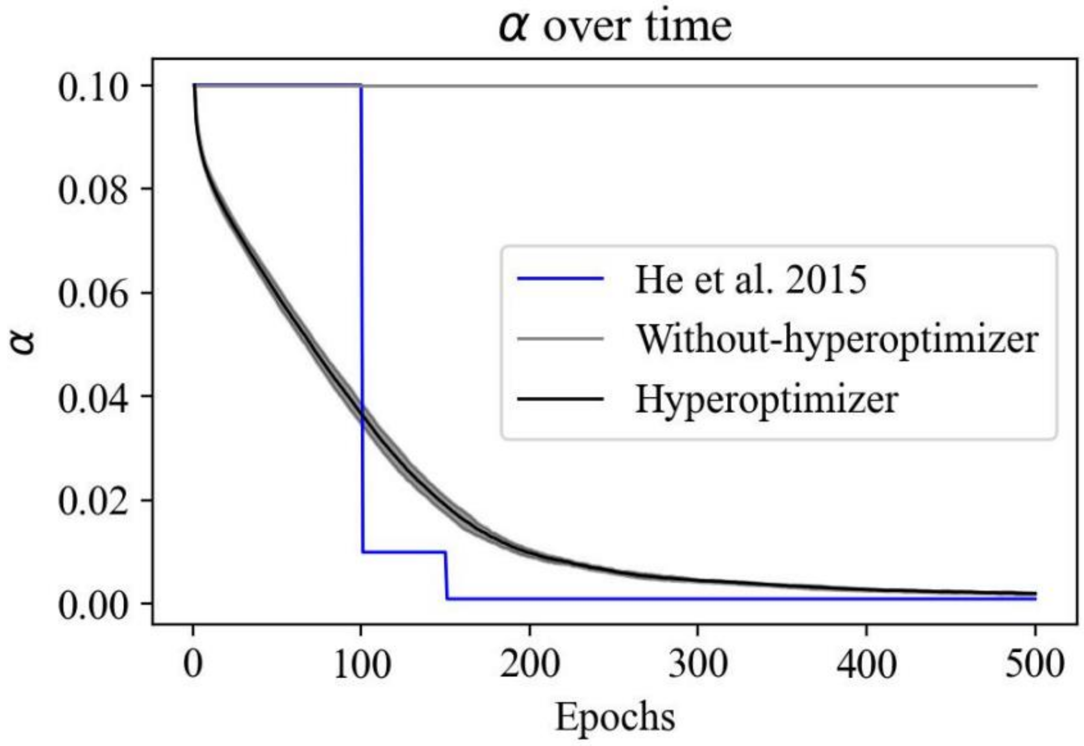
α ... learning rate

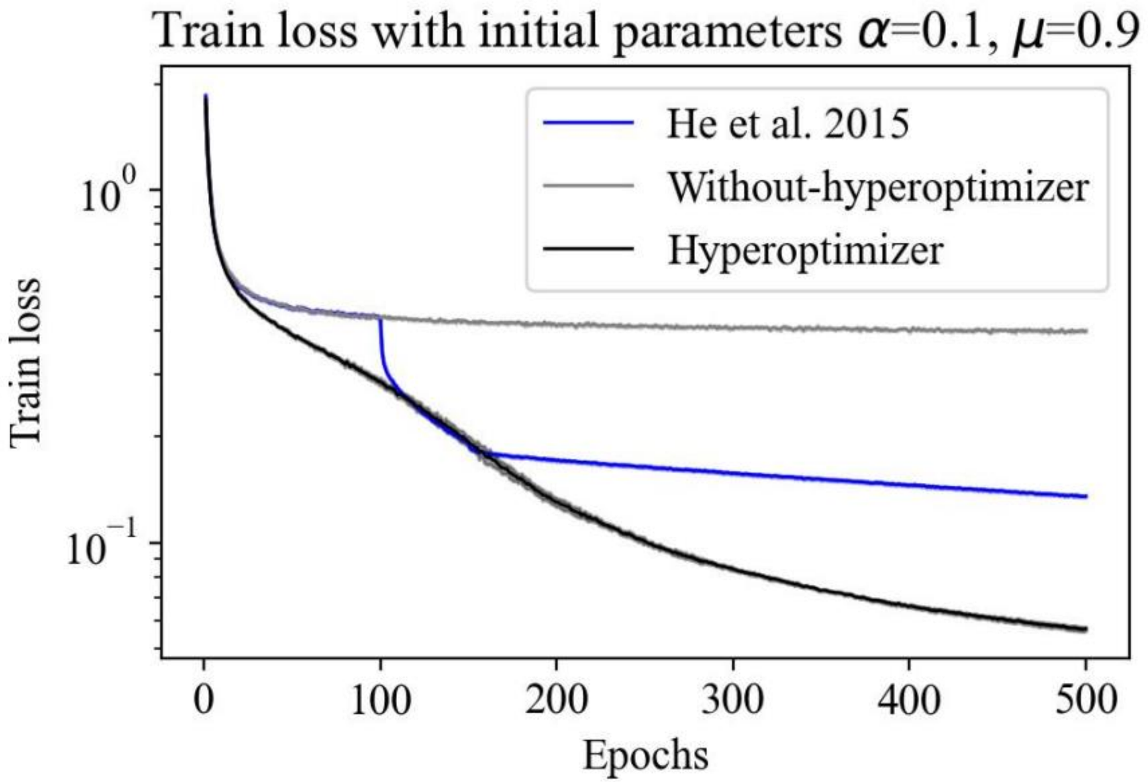
μ ... momentum

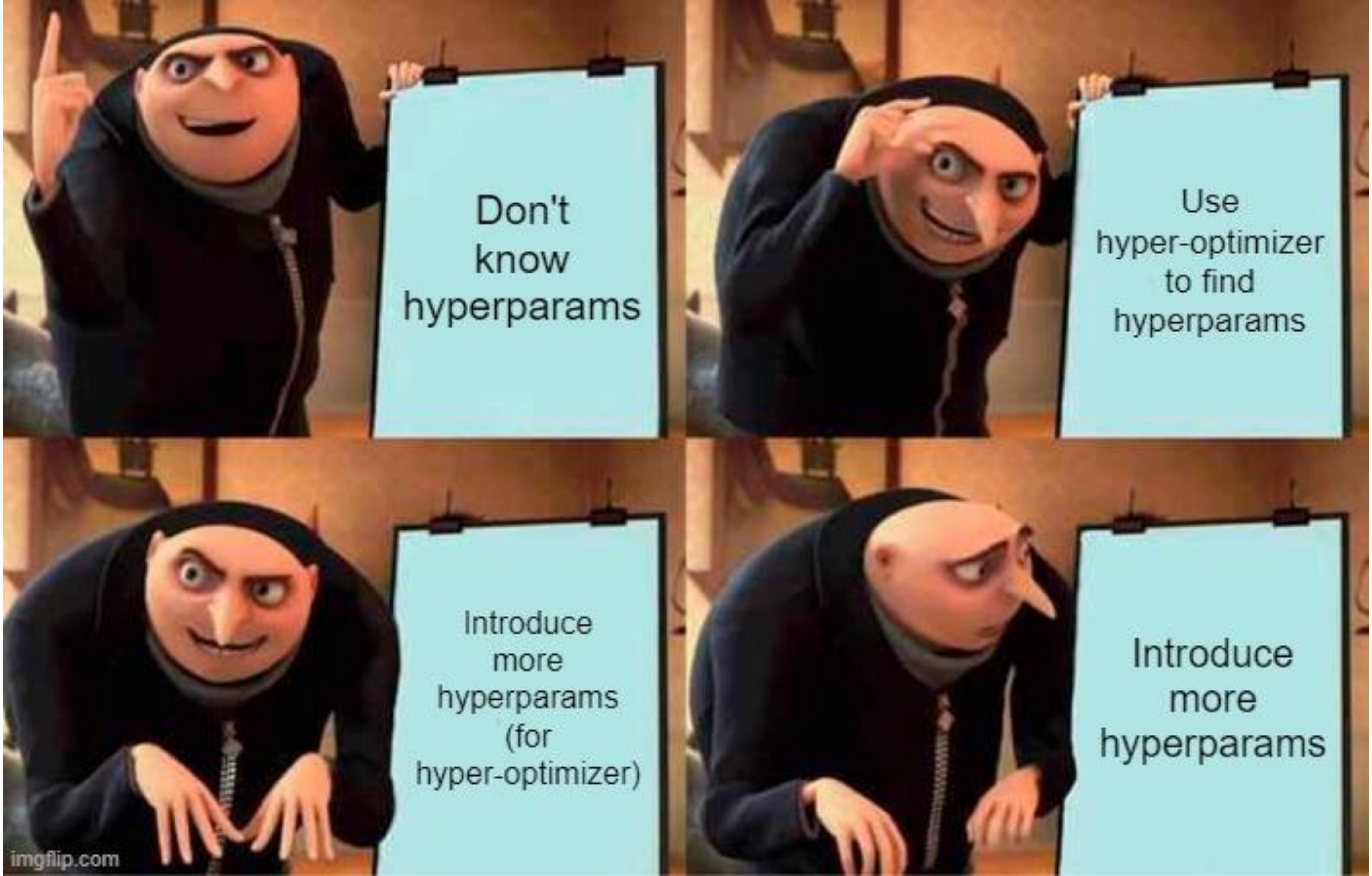


Gradient Descent: The Ultimate Optimizer, Chandra et al. 2022

Hyper-optimizer effectively learns LR schedule

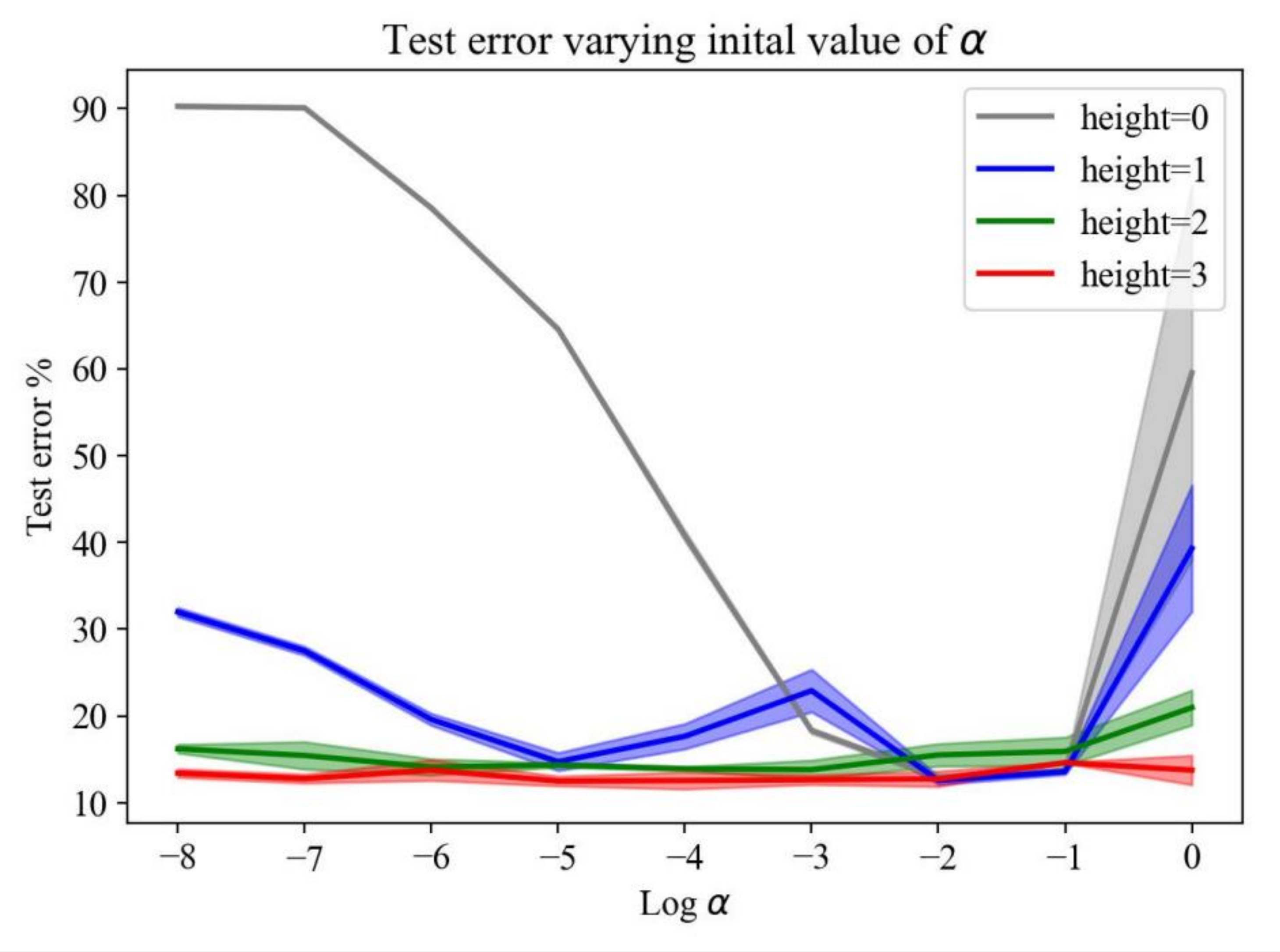





Don't know hyperparams
Use hyper-optimizer to find hyperparams
Introduce more hyperparams (for hyper-optimizer)
Introduce more hyperparams
imgflip.com

Stacking more optimizers reduces sensitivity on initial hyper-parameters

ResNet on CIFAR-10



Gradient Descent: The Ultimate Optimizer, Chandra et al. 2022

References

- **Learning to learn by gradient descent by gradient descent**
  Andrychowicz et al., 2016
  https://proceedings.neurips.cc/paper/2016/hash/fb87582825f9d28a8d42c5e5e5e8b23d-Abstract.html

- **Symbolic Discovery of Optimization Algorithms**
  Chen et al., 2023
  http://arxiv.org/abs/2302.06675

- **Gradient Descent: The Ultimate Optimizer**
  Chandra et al., 2022
  https://openreview.net/forum?id=-Qp-3L-5ZdI