Review

Previously we learned…

Optimization & Deep Learning

- Optimization algorithms are important for deep learning
  - training can take hours, days, or even weeks
  - algorithms directly affect the model's efficiency

$$\underset{\theta}{\text{minimize}}\ L(f_\theta(\mathbf{x}), \mathbf{y}), \qquad (\mathbf{x}, \mathbf{y}) \in \mathcal{D}$$

$\mathcal{D}$ → data

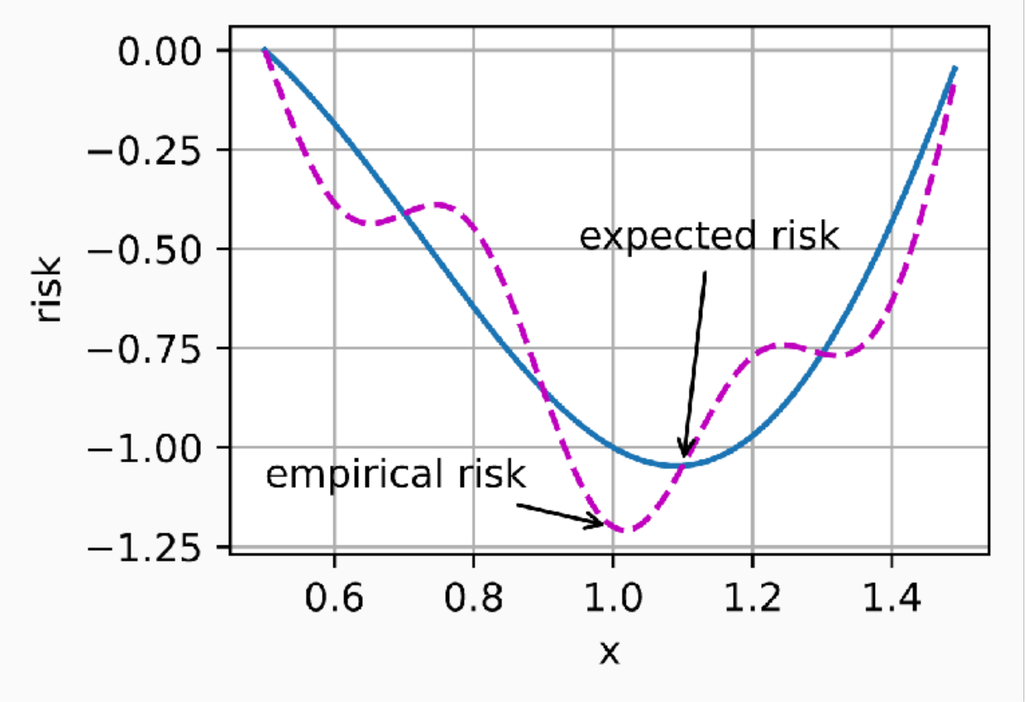
θ → parameters

L → loss function

f_θ → model (neural net)

Optimization ≠ Deep Learning

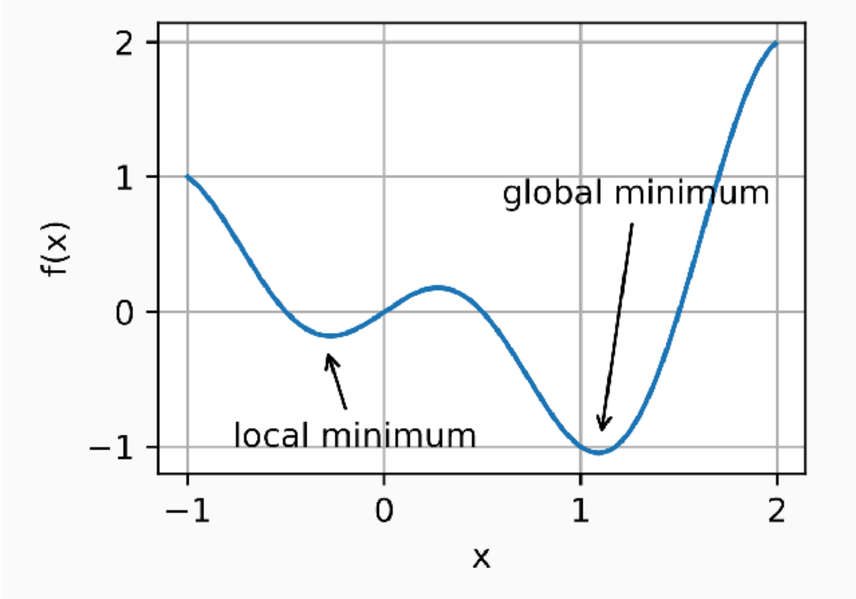
- Almost all optimization problems arising in DL are non-convex
- Optimization ≠ Deep Learning
  - goal of optimization is to reduce the training error
  - goal of statistical inference is to reduce the generalization error

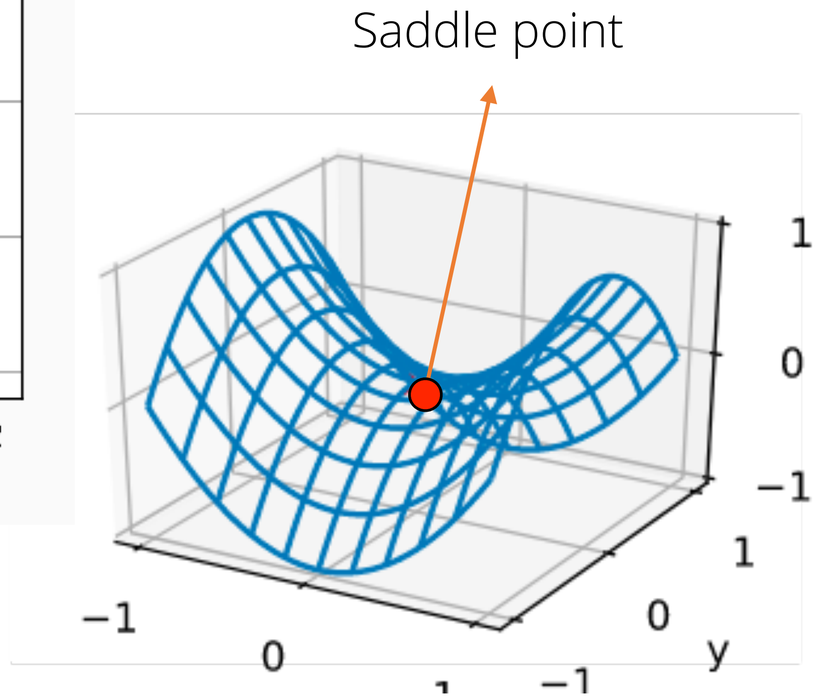


$$L_{\text{train}} \neq L_{\text{test}}$$

Optimization challenges in DL

- There are many challenges in deep learning optimizations
  - Local minima
  - Saddle points
  - Vanishing gradients
  - Gradient explosion



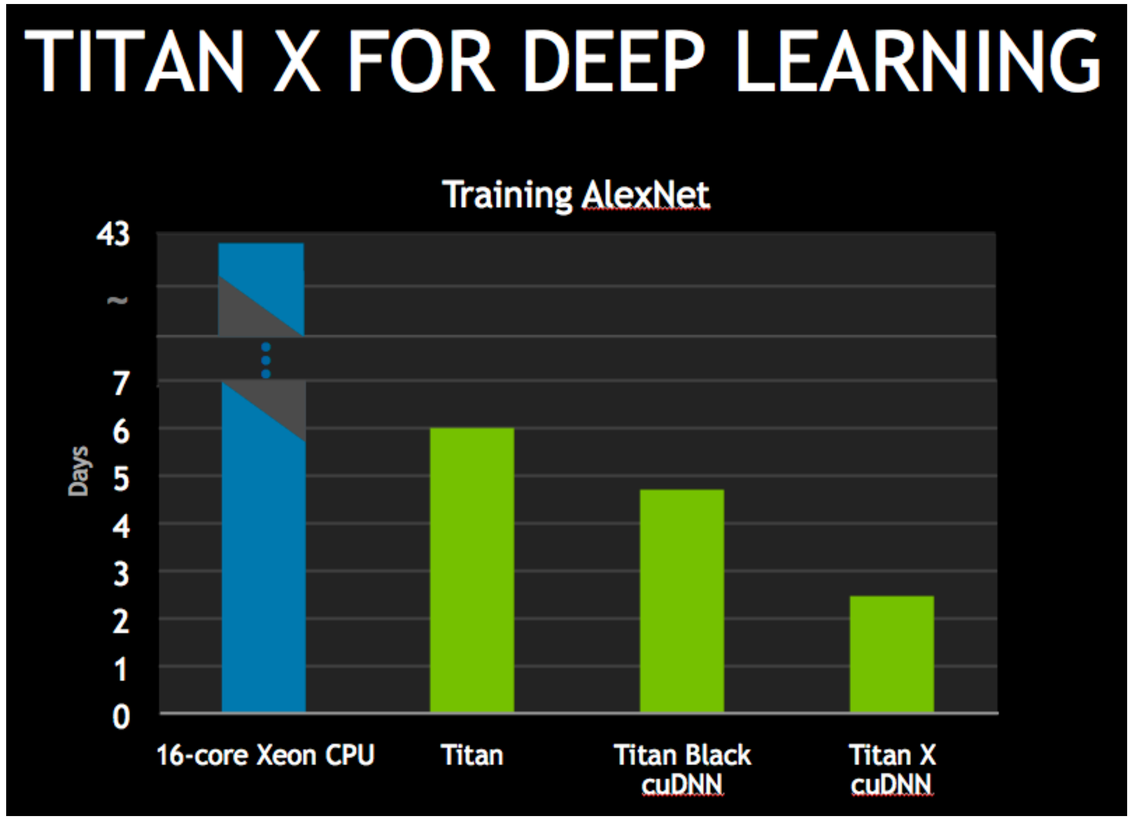


Heart of Deep Learning

WARNING: Don't do mining using lab GPUs

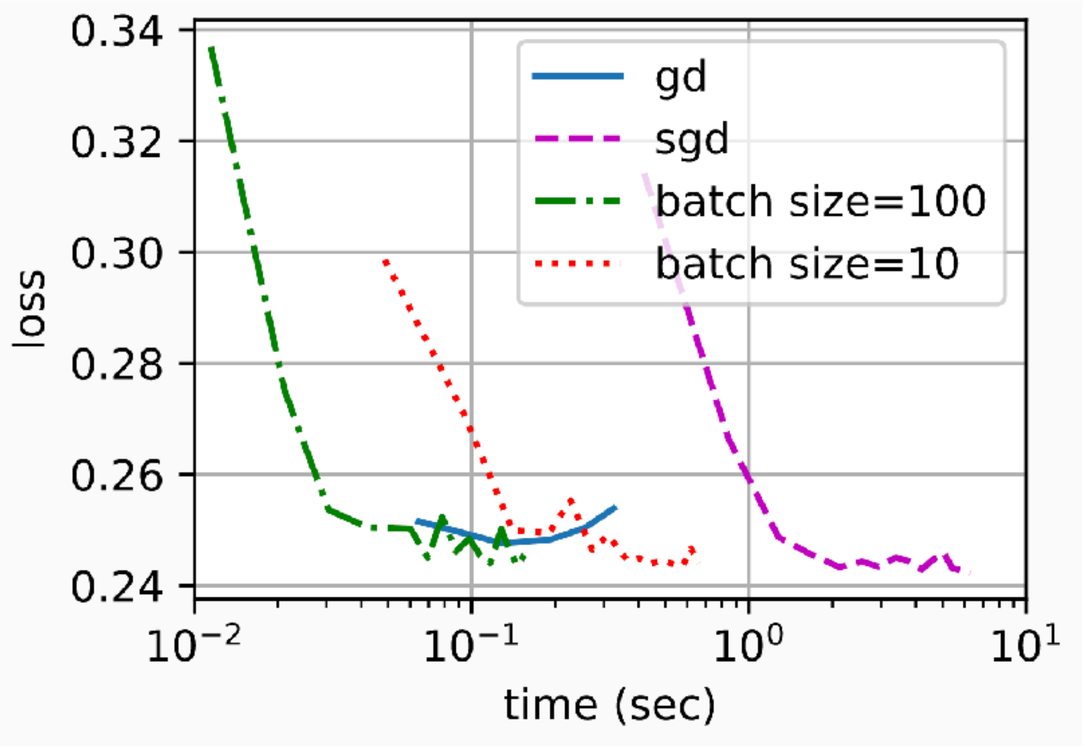
I can also earn some Bitcoins...

- What is the heart of deep learning?
  - parallelization
    - to multiple GPUs
    - to multiple servers



Heart of Deep Learning

- What is the heart of deep learning?
  - parallelization
    - to multiple GPUs
    - to multiple servers
- Use a hierarchy of CPU caches to supply the processor with data
- Mini-batch SGD reduces overhead when updating parameters



oops.. but the mini batch size is another hyperparameter!

Variants of stochastic gradient descent

speed

angle

GD → Stochastic GD (Robbins & Monto. 1951)

Stochastic GD → Adagrad (Duchi et al. 2011)

Adagrad → AdaDelta (Zeiler. 2012)

Adagrad → RMSProp (Tieleman & Hinton. 2012)

RMSProp → Adam (Kingma & Ba. 2014)

Adam → RAdam (Liu. 2019)

Stochastic GD → Momentum (Polyak. 1964)

Momentum → Adam

Momentum → NAG (Nesterov. 1983)

Adam → NAdam (Dozat. 2016)

NAG → NAdam

which one is the best?

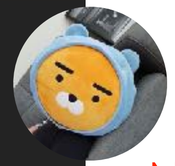
How to implement?

- ex) Adam use `torch.optim.Adam`

```
20  criterion = torch.nn.CrossEntropyLoss()
21  # optimizer = torch.optim.SGD(model.parameters(), lr=0.03)
22  optimizer = torch.optim.
```

Adadelta
Adagrad
Adam
Adamax
AdamW
ASGD
LBFGS
{} lr_scheduler
Optimizer
RMSprop

Adam Optimizer



There is no NAG! Do I have to implement by myself?

Nope. use `torch.optim.SGD`

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.03,
                            momentum=0.9,
                            nesterov=True)
```

How to implement?

- ex) Adam use `torch.optim.Adam`

```
?torch.optim.Adam
```

$$\min_{\theta} \left(L(f_\theta(x), y) \right)$$

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}.$$

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$
$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2.$$

Let's compare docs and algorithms

Step4) parameter update

Step3) gradient computation

Step2) renormalization

Step1) EWMA of gradient and second moment of gradient

```
Init signature: torch.optim.Adam(*args, **kwargs)
Docstring:
Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic
Optimization`_.

Arguments:
    params (iterable): iterable of parameters to optimize or
dicts defining
        parameter groups
    lr (float, optional): learning rate (default: 1e-3)
    betas (Tuple[float, float], optional): coefficients used for
computing
        running averages of gradient and its square (default:
(0.9, 0.999))
    eps (float, optional): term added to the denominator to
improve
        numerical stability (default: 1e-8)
    weight_decay (float, optional): weight decay (L2 penalty)
(default: 0)
    amsgrad (boolean, optional): whether to use the AMSGrad
variant of this
        algorithm from the paper `On the Convergence of Adam and
Beyond`_
        (default: False)

.. _Adam\: A Method for Stochastic Optimization:
    https://arxiv.org/abs/1412.6980
.. _On the Convergence of Adam and Beyond:
    https://openreview.net/forum?id=ryQu7f-RZ
File:           /usr/local/lib/python3.6/dist-
packages/torch/optim/adam.py
Type:           type
```

How to implement?

- ex) Adam use `torch.optim.Adam`

```
?torch.optim.Adam
```

$$\min_{\theta} \left(L(f_\theta(x), y) \right)$$

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}.$$

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$
$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2.$$

Step4) parameter update

Step3) gradient computation

Step2) renormalization

Step1) EWMA of gradient and second moment of gradient

```
Init signature: torch.optim.Adam(*args, **kwargs)
Docstring:
Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic
Optimization`_.

Arguments:
    params (iterable): iterable of parameters to optimize or
dicts defining
        parameter groups
    lr (float, optional): learning rate (default: 1e-3)
    betas (Tuple[float, float], optional): coefficients used for
computing
        running averages of gradient and its square (default:
(0.9, 0.999))
    eps (float, optional): term added to the denominator to
improve
        numerical stability (default: 1e-8)
    weight_decay (float, optional): weight decay (L2 penalty)
(default: 0)
    amsgrad (boolean, optional): whether to use the AMSGrad
variant of this
        algorithm from the paper `On the Convergence of Adam and
Beyond`_
        (default: False)

.. _Adam\: A Method for Stochastic Optimization:
    https://arxiv.org/abs/1412.6980
.. _On the Convergence of Adam and Beyond:
    https://openreview.net/forum?id=ryQu7f-RZ
File:           /usr/local/lib/python3.6/dist-
packages/torch/optim/adam.py
Type:           type
```

How to implement?

- ex) Adam use `torch.optim.Adam`

```
?torch.optim.Adam
```

$$\min_{\theta} \Big(L(f_\theta(x), y) \Big)$$

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}.$$

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$
$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2.$$

Step4) parameter update

Step3) gradient computation

Step2) renormalization

Step1) EWMA of gradient and second moment of gradient

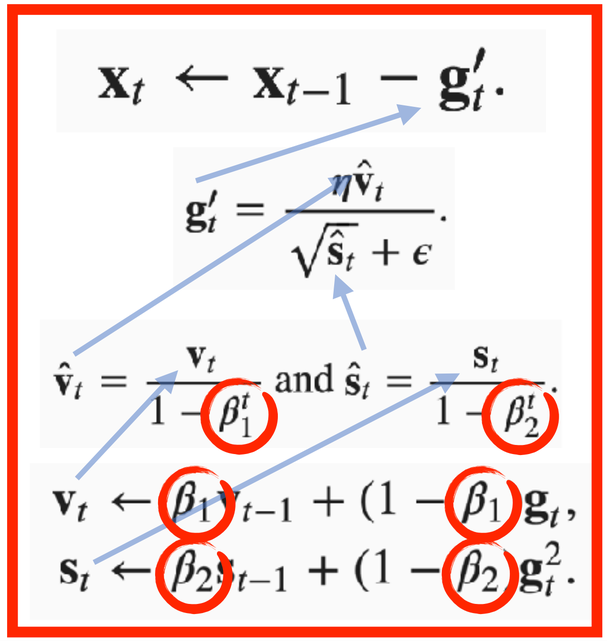
```
Init signature: torch.optim.Adam(*args, **kwargs)
Docstring:
Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic
Optimization`_.

Arguments:
    params (iterable): iterable of parameters to optimize or
dicts defining
        parameter groups
    lr (float, optional): learning rate (default: 1e-3)
    betas (Tuple[float, float], optional): coefficients used for
computing
        running averages of gradient and its square (default:
(0.9, 0.999))
    eps (float, optional): term added to the denominator to
improve
        numerical stability (default: 1e-8)
    weight_decay (float, optional): weight decay (L2 penalty)
(default: 0)
    amsgrad (boolean, optional): whether to use the AMSGrad
variant of this
        algorithm from the paper `On the Convergence of Adam and
Beyond`_
        (default: False)

.. _Adam\: A Method for Stochastic Optimization:
    https://arxiv.org/abs/1412.6980
.. _On the Convergence of Adam and Beyond:
    https://openreview.net/forum?id=ryQu7f-RZ
File:           /usr/local/lib/python3.6/dist-
packages/torch/optim/adam.py
Type:           type
```

How to implement?

- ex) Adam use `torch.optim.Adam`

```
?torch.optim.Adam
```

$$\min_{\theta} \Big(L(f_\theta(x), y) \Big)$$



Step4) parameter update

Step3) gradient computation

Step2) renormalization

Step1) EWMA of gradient and second moment of gradient

```
Init signature: torch.optim.Adam(*args, **kwargs)
Docstring:
Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic
Optimization`_.

Arguments:
    params (iterable): iterable of parameters to optimize or
dicts defining
        parameter groups
    lr (float, optional): learning rate (default: 1e-3)
    betas (Tuple[float, float], optional): coefficients used for
computing
        running averages of gradient and its square (default:
(0.9, 0.999))
    eps (float, optional): term added to the denominator to
improve
        numerical stability (default: 1e-8)
    weight_decay (float, optional): weight decay (L2 penalty)
(default: 0)
    amsgrad (boolean, optional): whether to use the AMSGrad
variant of this
        algorithm from the paper `On the Convergence of Adam and
Beyond`_
        (default: False)

.. _Adam\: A Method for Stochastic Optimization:
    https://arxiv.org/abs/1412.6980
.. _On the Convergence of Adam and Beyond:
    https://openreview.net/forum?id=ryQu7f-RZ
File:           /usr/local/lib/python3.6/dist-
packages/torch/optim/adam.py
Type:           type
```

How to implement?

- ex) Adam use `torch.optim.Adam`

```
?torch.optim.Adam
```

$$\min_{\theta} \Big(L(f_\theta(x), y) \Big)$$

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}.$$

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$
$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2.$$

Step4) parameter update

Step3) gradient computation

Step2) renormalization

Step1) EWMA of gradient and second moment of gradient

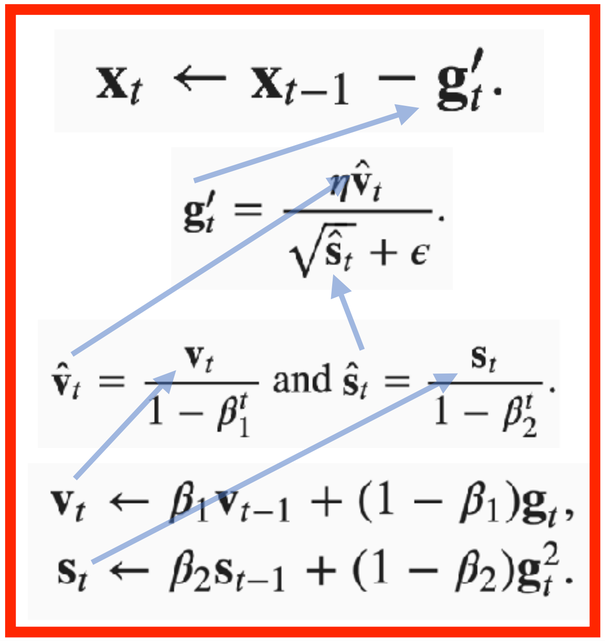
```
Init signature: torch.optim.Adam(*args, **kwargs)
Docstring:
Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic
Optimization`_.

Arguments:
    params (iterable): iterable of parameters to optimize or
dicts defining
        parameter groups
    lr (float, optional): learning rate (default: 1e-3)
    betas (Tuple[float, float], optional): coefficients used for
computing
        running averages of gradient and its square (default:
(0.9, 0.999))
    eps (float, optional): term added to the denominator to
improve
        numerical stability (default: 1e-8)
    weight_decay (float, optional): weight decay (L2 penalty)
(default: 0)
    amsgrad (boolean, optional): whether to use the AMSGrad
variant of this
        algorithm from the paper `On the Convergence of Adam and
Beyond`_
        (default: False)

.. _Adam\: A Method for Stochastic Optimization:
    https://arxiv.org/abs/1412.6980
.. _On the Convergence of Adam and Beyond:
    https://openreview.net/forum?id=ryQu7f-RZ
File:           /usr/local/lib/python3.6/dist-
packages/torch/optim/adam.py
Type:           type
```

How to implement?

- ex) Adam use `torch.optim.Adam`

```
?torch.optim.Adam
```

$$\min_{\theta}\left(L(f_\theta(x), y) + \lambda\|\theta\|_2^2\right)$$



$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1-\beta_2^t}.$$

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\mathbf{g}_t,$$
$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2.$$

Step4) parameter update

Step3) gradient computation

Step2) renormalization

Step1) EWMA of gradient and second moment of gradient

```
Init signature: torch.optim.Adam(*args, **kwargs)
Docstring:
Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic
Optimization`_.

Arguments:
    params (iterable): iterable of parameters to optimize or
dicts defining
        parameter groups
    lr (float, optional): learning rate (default: 1e-3)
    betas (Tuple[float, float], optional): coefficients used for
computing
        running averages of gradient and its square (default:
(0.9, 0.999))
    eps (float, optional): term added to the denominator to
improve
        numerical stability (default: 1e-8)
    weight_decay (float, optional): weight decay (L2 penalty)
(default: 0)
    amsgrad (boolean, optional): whether to use the AMSGrad
variant of this
        algorithm from the paper `On the Convergence of Adam and
Beyond`_
        (default: False)

.. _Adam\: A Method for Stochastic Optimization:
    https://arxiv.org/abs/1412.6980
.. _On the Convergence of Adam and Beyond:
    https://openreview.net/forum?id=ryQu7f-RZ
File:           /usr/local/lib/python3.6/dist-
packages/torch/optim/adam.py
Type:           type
```

Learning Rate Scheduling

Beyond constant learning rate

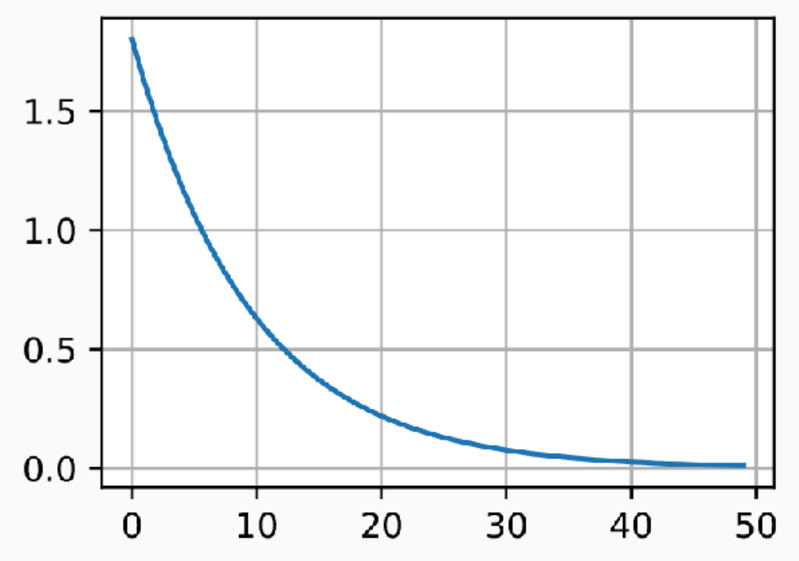
Learning rate selection is tricky

- There are number of aspects to consider in LR selection
  - the magnitude of the learning rate
  - initialization & decay rate
- Popular methods
  - factor scheduler
  - cosine scheduler
  - warmup

Factor Scheduler

Use `torch.optim.lr_scheduler.LambdaLR` in 

- Multiplicative decay $\eta_{t+1} \leftarrow \max(\eta_{\min}, \eta_t \cdot \alpha), \quad \alpha \in (0, 1)$
  - basic and frequently used



CLASS `torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda, last_epoch=-1)` [SOURCE]

Sets the learning rate of each parameter group to the initial lr times a given function. When last_epoch=-1, sets initial lr as lr.

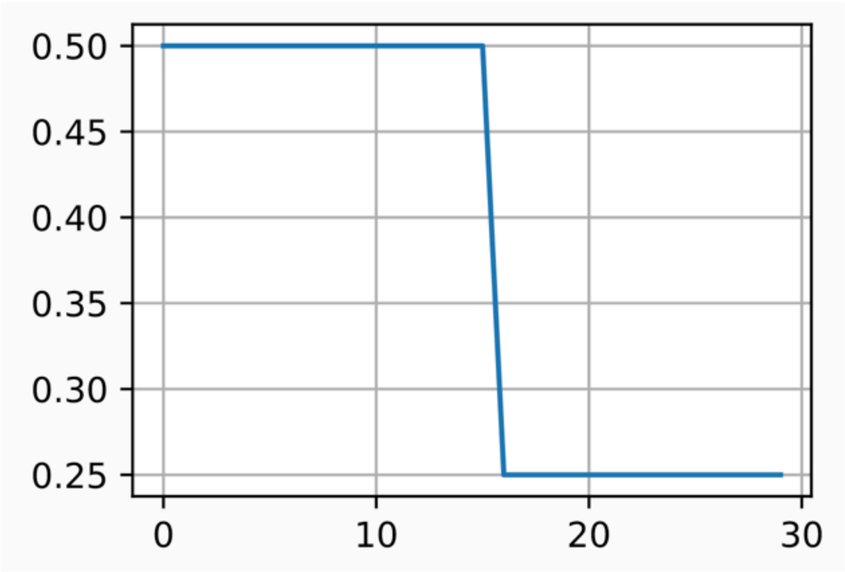
Parameters

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **lr_lambda** (*function or list*) – A function which computes a multiplicative factor given an integer parameter epoch, or a list of such functions, one for each group in optimizer.param_groups.
- **last_epoch** (*python:int*) – The index of last epoch. Default: -1.

Multi-factor Scheduler

Use `torch.optim.lr_scheduler.MultiStepLR` in 

- Keep the LR piecewise constant $\eta_{t+1} \leftarrow \eta_t \cdot \alpha, \quad t \in \{t_1, t_2, \ldots\}$
  - decrease it by fixed amount for given a set of times



CLASS `torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.1, last_epoch=-1)` [SOURCE]

Decays the learning rate of each parameter group by gamma once the number of epoch reaches one of the milestones. Notice that such decay can happen simultaneously with other changes to the learning rate from outside this scheduler. When last_epoch=-1, sets initial lr as lr.

Parameters

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **milestones** (*list*) – List of epoch indices. Must be increasing.
- **gamma** (*python:float*) – Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** (*python:int*) – The index of last epoch. Default: -1.

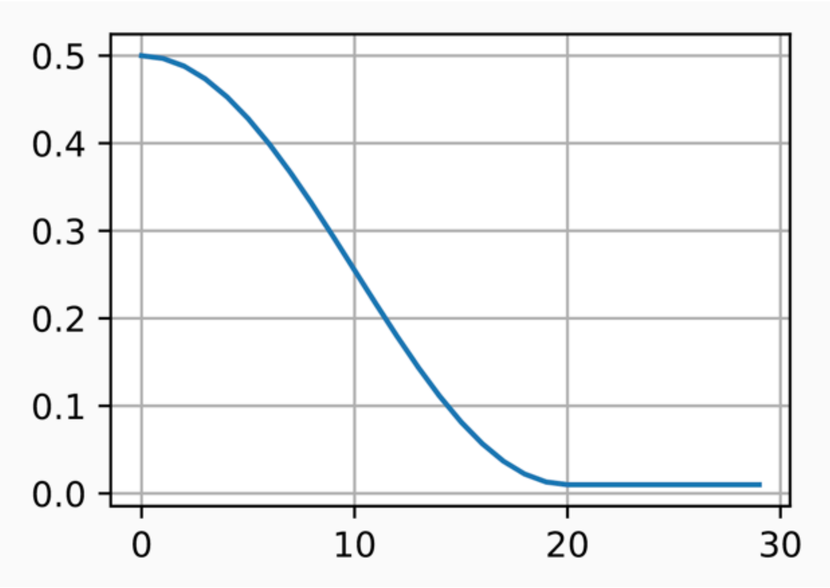
Cosine Scheduler

Use `torch.optim.lr_scheduler.CosineAnnealingLR` in

- Cosine-like decay

$$\eta_t = \eta_T + \frac{\eta_0 - \eta_T}{2}(1 + \cos(\pi t/T))$$

  - heuristic but it works well for visual domain



CLASS `torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max, eta_min=0, last_epoch=-1)` [SOURCE]

Set the learning rate of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial lr and T_{cur} is the number of epochs since the last restart in SGDR:

Parameters

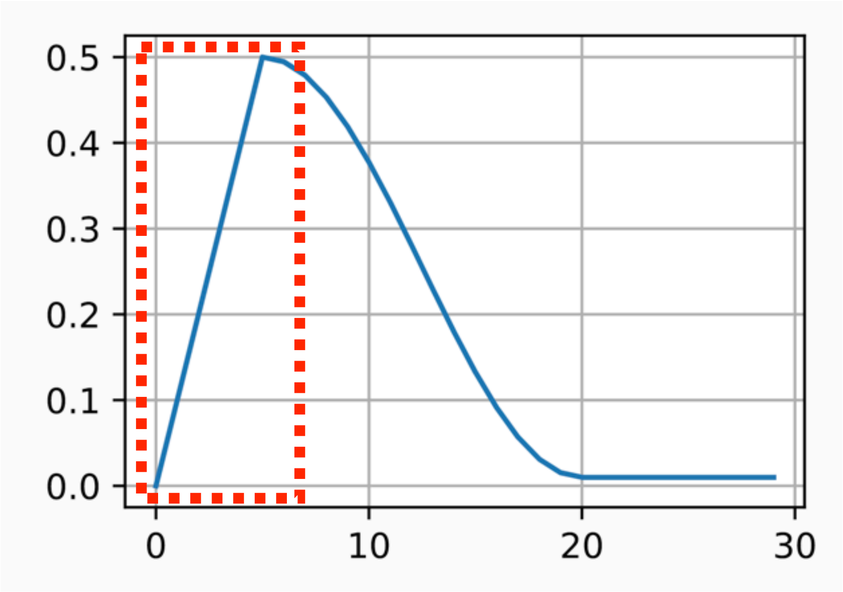
- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **T_max** (*python:int*) – Maximum number of iterations.
- **eta_min** (*python:float*) – Minimum learning rate. Default: 0.
- **last_epoch** (*python:int*) – The index of last epoch. Default: -1.

SGDR: Stochastic Gradient Descent with Warm Restarts, Loshchilov & Hutter, ICLR 2017

Cosine Scheduler + Warmup

Use `torch.optim.lr_scheduler.CosineAnnealingLRWarmRestarts` in

- Large LR initially leads to divergence
  - increase LR gradually to maximum and cool down the rate



CLASS `torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0, T_mult=1, eta_min=0, last_epoch=-1)` [SOURCE]

Set the learning rate of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial lr, T_{cur} is the number of epochs since the last restart and T_i is the number of epochs between two warm restarts in SGDR:

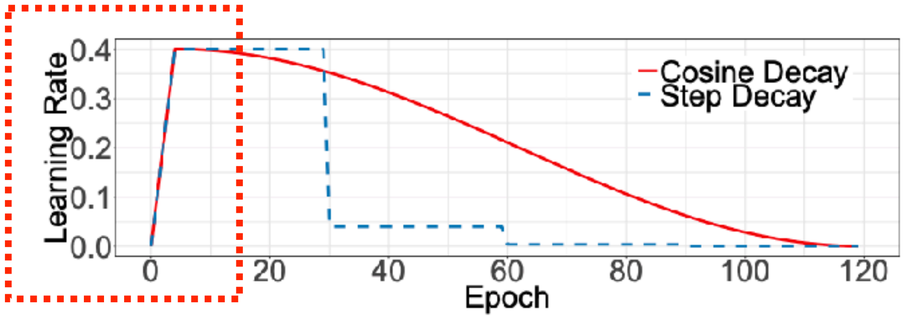
Parameters

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **T_0** (*python:int*) – Number of iterations for the first restart.
- **T_mult** (*python:int, optional*) – A factor increases T_i after a restart. Default: 1.
- **eta_min** (*python:float, optional*) – Minimum learning rate. Default: 0.
- **last_epoch** (*python:int, optional*) – The index of last epoch. Default: -1.

SGDR: Stochastic Gradient Descent with Warm Restarts, Loshchilov & Hutter, ICLR 2017

RAdam

warmup is necessary
for Adam optimizers



(a) Learning Rate Schedule

- Rectified Adam
  - utilizes warmup heuristic for adaptive stochastic optimization

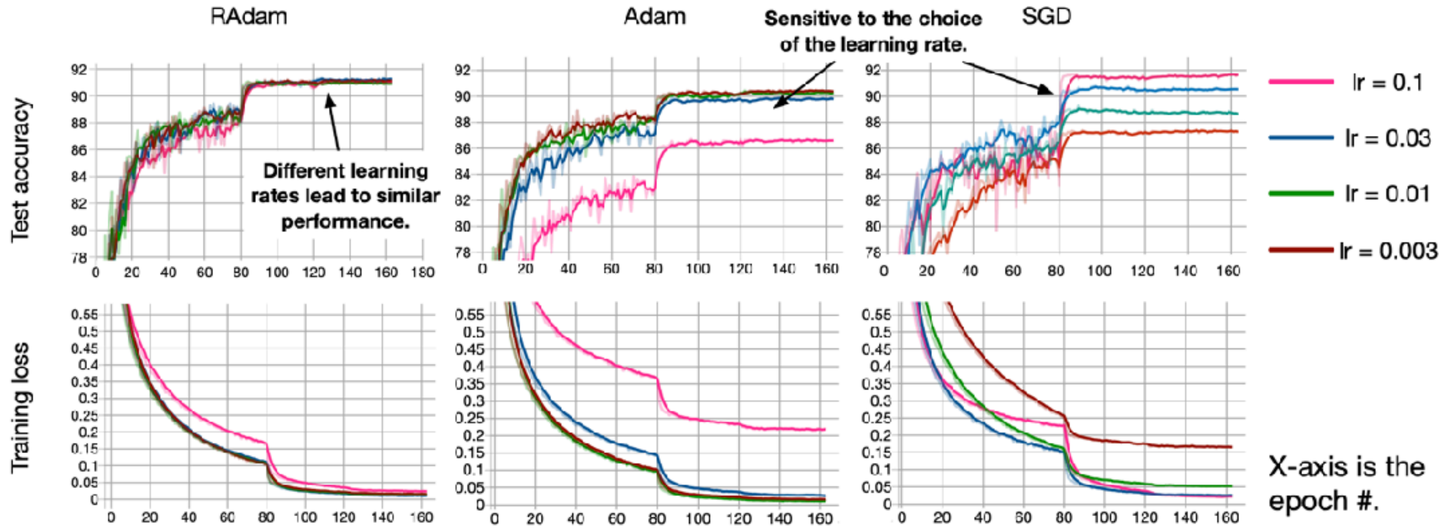


Figure 6: Performance of RAdam, Adam and SGD with different learning rates on CIFAR10.

ON THE VARIANCE OF THE ADAPTIVE LEARNING RATE AND BEYOND

Liyuan Liu *
University of Illinois, Urbana-Champaign
llt2@illinois

Haoming Jiang †
Georgia Tech
jianghm@gatech.edu

Pengcheng He, Weizhu Chen
Microsoft Dynamics 365 AI
{penhe,wzchen}@microsoft.com

Xiaodong Liu, Jianfeng Gao
Microsoft Research
{xiaodl,jfgao}@microsoft.com

Jiawei Han
University of Illinois, Urbana-Champaign
hanj@illinois

On the variance of the adaptive learning rate and beyond, Liu et al., ICLR 2020

Other schedulers

- ReducedLROnPlateau
- CyclicLR
- Population Based Scheduler (multi-GPU)

you can use this if you go to giant companies (kakao)

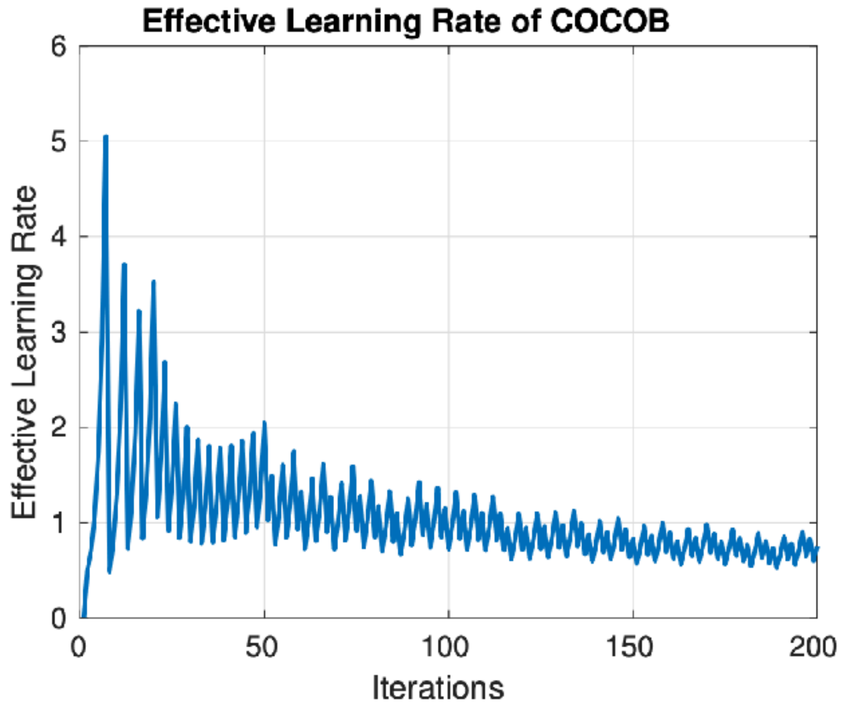
COCOB Algorithm

- Subgradient descent through coin betting
  - learning rate free optimization

Training Deep Networks without Learning Rates Through Coin Betting

Francesco Orabona*
Department of Computer Science
Stony Brook University
Stony Brook, NY
francesco@orabona.com

Tatiana Tommasi*
Department of Computer, Control, and Management Engineering
Sapienza, Rome University, Italy
tommasi@dis.uniroma1.it



Training Deep Networks without Learning Rates Through Coin Betting, Orabona & Tommasi, NIPS 2017

Stochastic Line Search

- Line Search + SGD works fine
  - Theoretically well-established

Visit https://github.com/IssamLaradji/sls

Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates

Sharan Vaswani
Mila, Université de Montréal

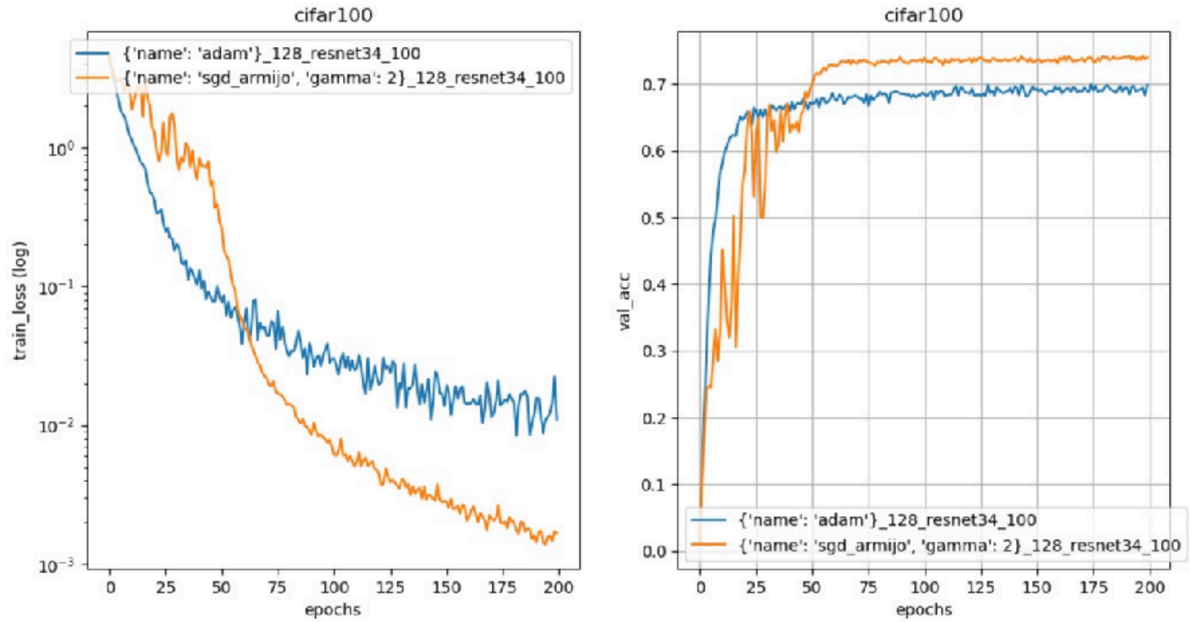
Aaron Mishkin
University of British Columbia

Issam Laradji
University of British Columbia
Element AI

Mark Schmidt
University of British Columbia, 1QBit
CCAI Affiliate Chair (Amii)

Gauthier Gidel
Mila, Université de Montréal
Element AI

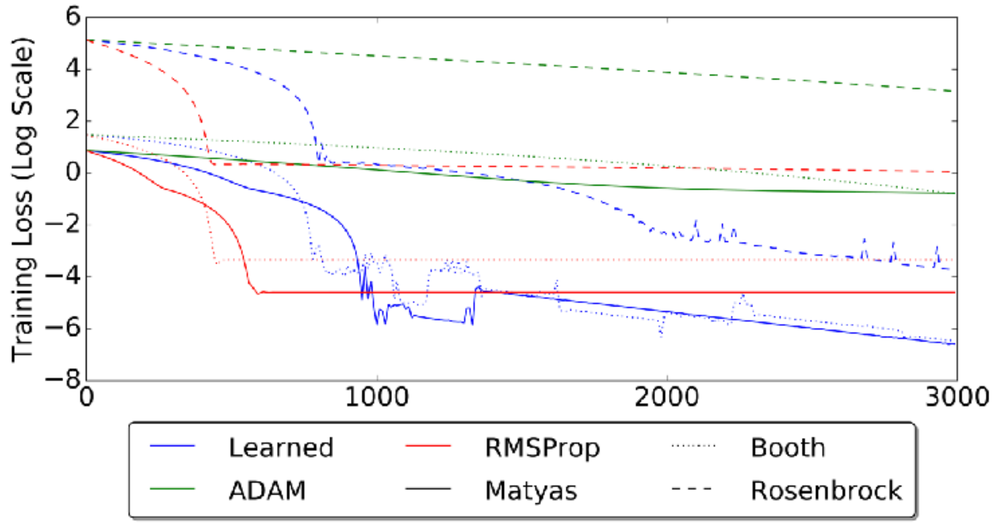
Simon Lacoste-Julien[†]
Mila, Université de Montréal

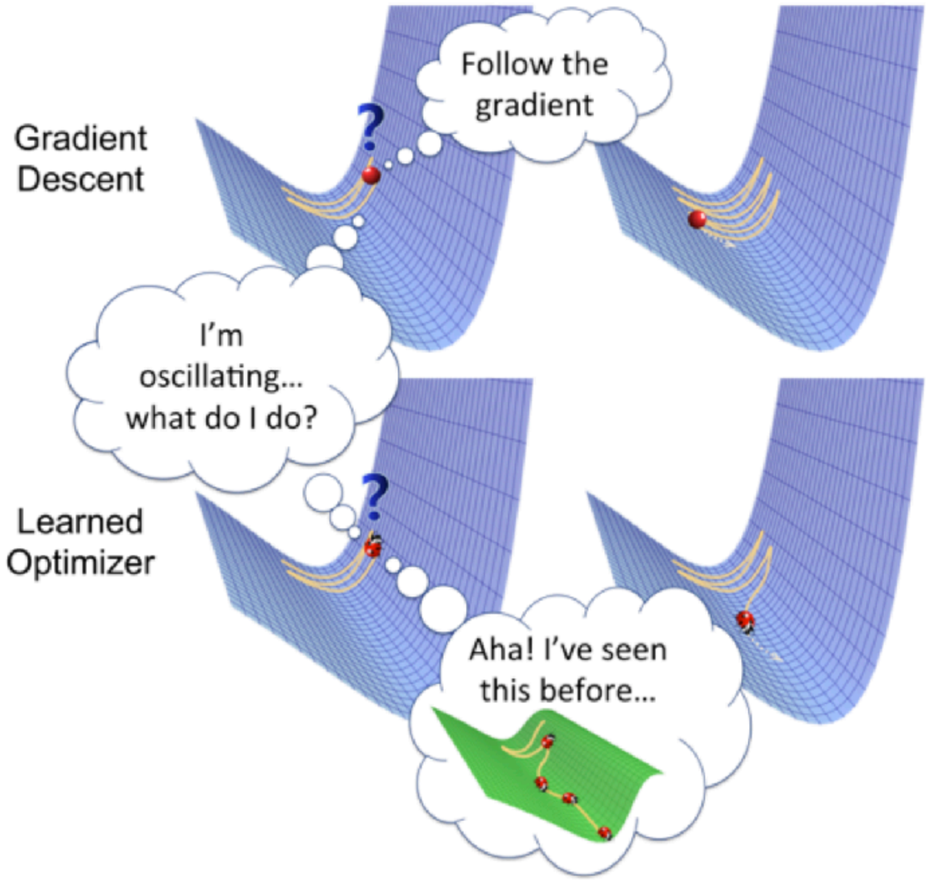


Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates, Vaswani et al., NeurIPS 2019

Learned Optimizers

- Can AI optimize itself?
  - Learn-to-Learn (or Meta Learning)





hmm... it could be more interesting
if I can implement speaking optimizers

Learning to Optimize, Li and Malik, ICLR 2017
Learned Optimizers that Scale and Generalize, Wichrowska et al., ICML 2017

Lesson 5: Deep Learning Computation

with PyTorch in a Nutshell

Introduction to Deep Learning

Sungbin Lim (SME)

kakaobrain

PyTorch Tutorials

- Custom DataLoaders how can I *prepare* my data for PyTorch?
- Data Preprocessing how can I *preprocess* my data in PyTorch?
- Custom Functions and AutoGrad how can I *implement* custom functions in PyTorch?
- Layers, Modules, and Sequential Blocks how can I *build* my model efficiently in PyTorch?
- PyTorch and multi-GPU how can I *utilize* multi-GPUs in PyTorch?
- Checkpoints how can I *manage* my models in PyTorch?
- Visualization how can I *visualize* loss functions and the results in PyTorch?

Assignments

- Revise your proposal in detail (until 4/9) → Determine topics (until 4/23)
  - Explain your Data Science problems in detail
    - What is your goal? Why this is important problem?
    - How to evaluate your model empirically? What is your metric?
  - Describe [what is, how to gather, difficulty of] your data concretely
  - Read the issues in your GitHub and revise your proposal

Assignments

these are must-read papers in DL research

- Read papers
  - LeNet (1998): Gradient-Based Learning Applied to Document Recognition
  - AlexNet (2012): ImageNet Classification with Deep Convolutional Neural Networks

  until 4/9
  - ZFNet (2013): Visualizing and Understanding Convolutional Networks
  - VGG (2014): Very Deep Convolutional Networks for Large-Scale Image Recognition
  - Inception (2014): Going Deeper with Convolutions
  - ResNet (2015): Deep Residual Learning for Image Recognition
  - WideResNet (2016): Wide Residual Networks
  - DenseNet (2016): Densely Connected Convolutional Networks

  until 4/14
  - NAS (2018): Learning Transferable Architectures for Scalable Image Recognition
  - EfficientNet (2019): Rethinking Model Scaling for Convolutional Neural Networks

  until 4/16

Q & A