Data-Driven Modeling *MSDM 5055*

Deep Learning for Modeling: Concepts, Tools, and Techniques

Week 6: Momentum-based Gradient Descent and Learning Rates

Li Shuo-Hui

@HKUST | Spring 2024

Norm layers

Parameter initialization principle:

(1). Activation values should have a mean of zero

(2). Activation values should have the same variance across different layers

(3). Weights should be independent and identically distributed (i.i.d)

(4). Weights distribution should have a mean of zero

(5). Bias can be initialized to zeros

Norm layers

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

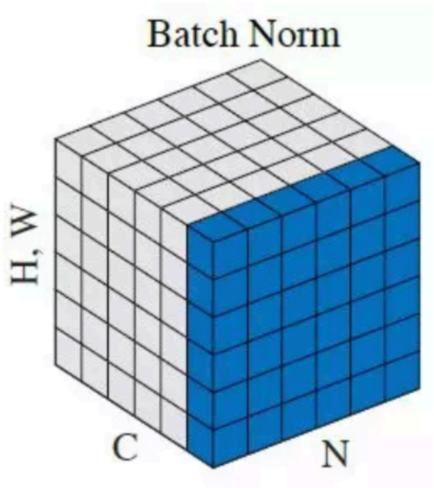
$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

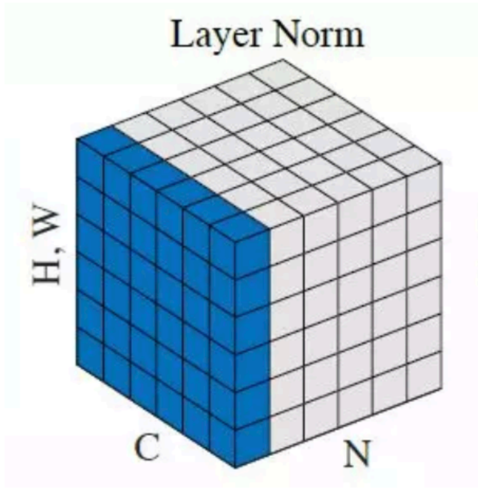
$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Norm layers





Usually seen in CV tasks

Usually seen in NLP tasks

Gradient-based optimization: revisit

$$\min \mathcal{L}(\mathbf{x})$$

Update: $$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta_k \mathbf{p}_k$$

Optim direction

Learning rate/step size

$$\mathbf{p}_k = -\mathbf{B}_k^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_k}$$

Local geometry

Gradient descent: revisit

Gradient descent

Repeat

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \boldsymbol{\theta}}$$

Till $\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x})$ *is small enough*

$$\eta_k = \eta$$

$$\mathbf{p}_k = -\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

$$\mathbf{B}_k^{-1} = \mathbf{I}$$

Newton's method: revisit

Newton's method

Solve $\underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = 0 \longrightarrow \frac{\partial f}{\partial \mathbf{x}} + \delta \mathbf{x} \frac{\partial^2 f}{\partial \mathbf{x}^2} = 0$$

Repeat $\mathbf{x}' = \mathbf{x} - \delta \mathbf{x}$ *till converge*

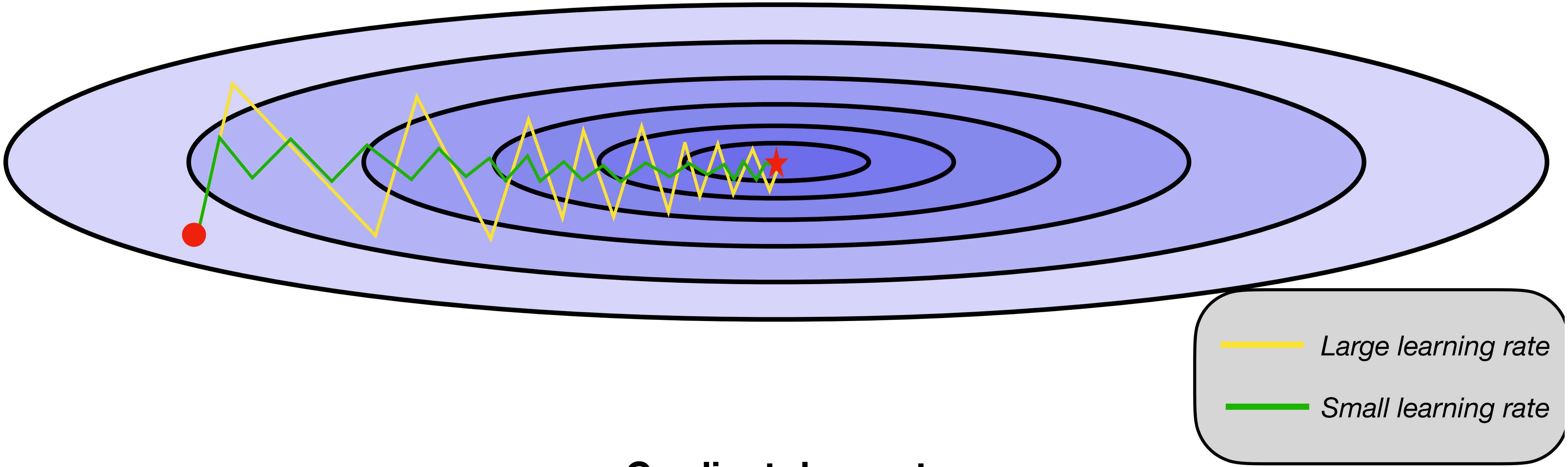
$$\eta_k = 1$$

$$\mathbf{p}_k = -\left(\frac{\partial^2 f}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial f}{\partial \mathbf{x}}$$

$$\mathbf{B}_k^{-1} = \left(\frac{\partial^2 f}{\partial \mathbf{x}^2}\right)^{-1}$$

Gradient-based optimization: problem one

The local geometry

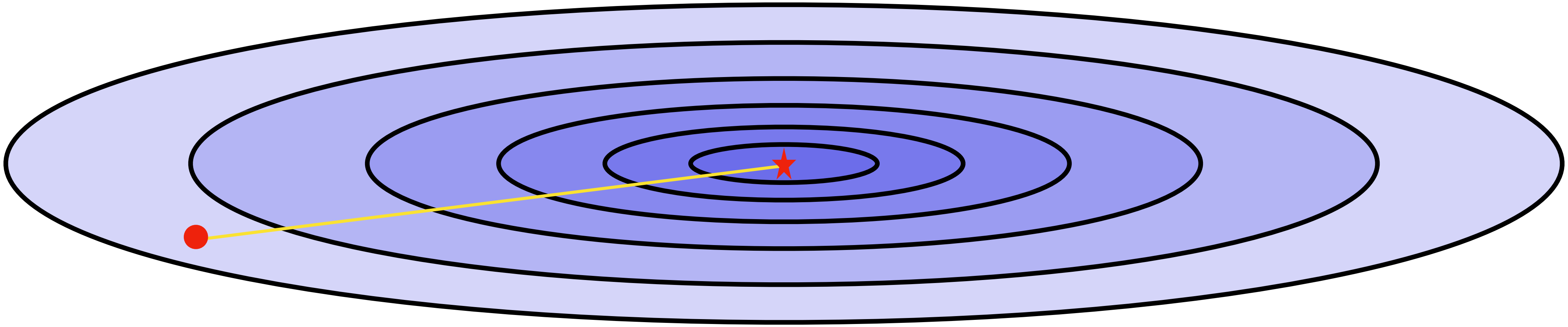


Gradient descent

$$\mathbf{B}_k^{-1} = \mathbf{I}$$

Gradient-based optimization: problem one

The local geometry



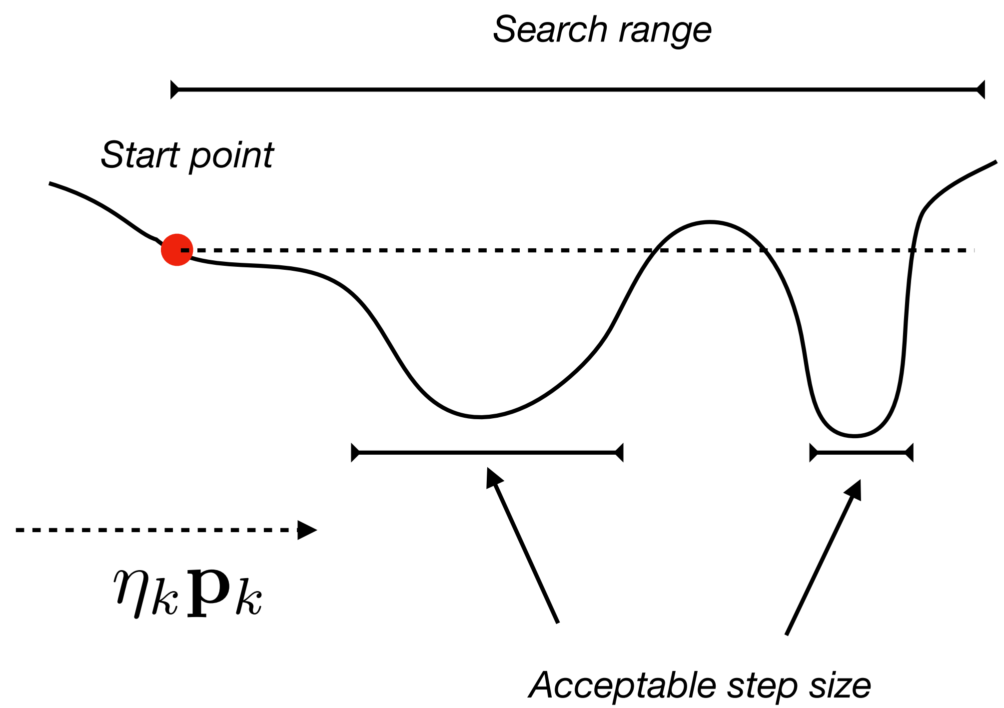
Newton's method

$$\mathbf{B}_k^{-1} = \left(\frac{\partial^2 f}{\partial \mathbf{x}^2}\right)^{-1}$$

Balance update across directions!

Gradient-based optimization: problem two

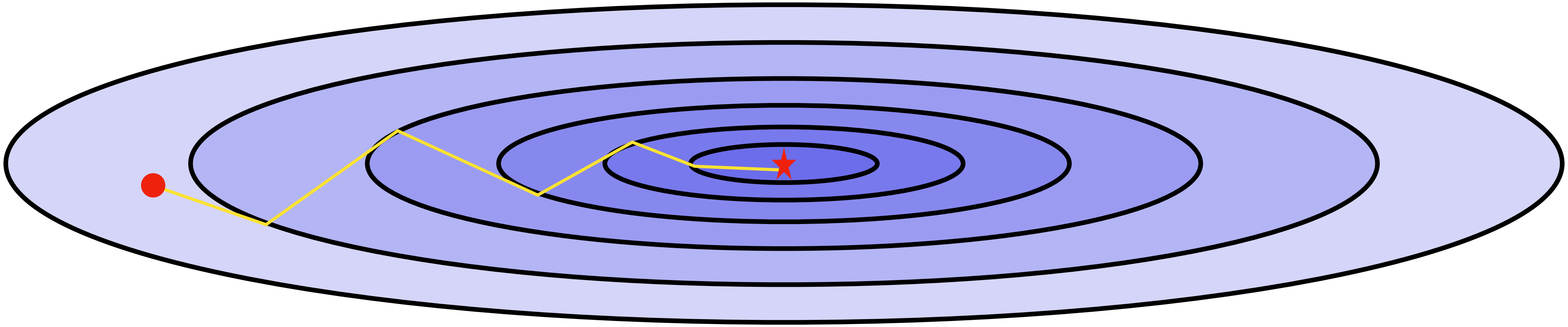
The step size selection



The line search algorithms

Gradient-based optimization: problem one

The step size selection



Steepest Descent:
gradient descent with exact line search

$$\mathbf{B}_k^{-1} = \mathbf{I}$$

"Zig-zag track" tangents to level set

Gradient-based optimization

Comparison

	Line search step size	Local geometry acknowledge
Newton w/ line search	✓	✓
Steepest descent	✓	✗
SGD	✗	✗

These can be fixed with momentum and changing learning rate

Momentum-based gradient descent

Simple momentum

$$\mathbf{p}_k = \boxed{-\boxed{\rho}\mathbf{p}_{k-1}} - \frac{\partial f}{\partial \mathbf{x}_k}$$

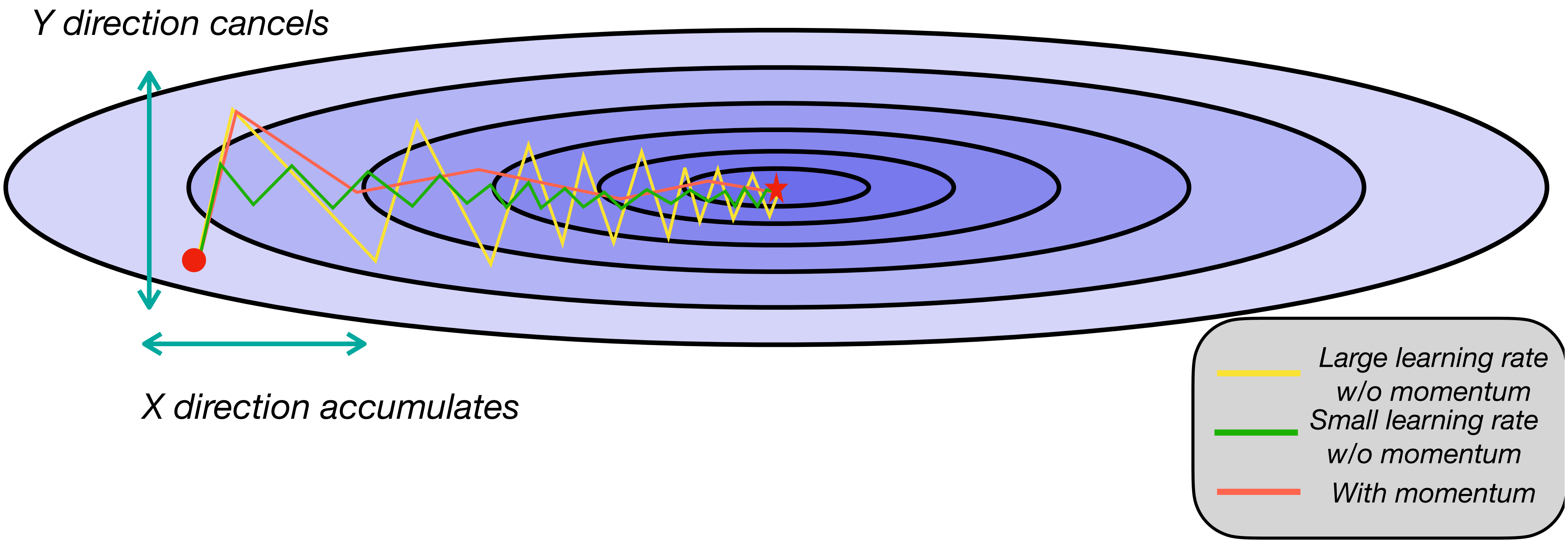
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta \mathbf{p}_k$$

Momentum term

Momentum decay factor
Or "friction"

Momentum-based gradient descent

$$\mathbf{p}_k = -\rho\mathbf{p}_{k-1} - \frac{\partial f}{\partial \mathbf{x}_k}$$



Changing learning rates

Decaying learning rate

Exponential

$$\eta_k = \eta_0 \cdot \gamma^{\max(0, [\frac{k-k_0}{s}])}$$

1/t scheme

$$\eta_k = \frac{\eta_0}{1 + \max(0, [\frac{k-k_0}{s}])}$$

$$\vdots$$

Decaying learning rate to adapt finer optimization at the later stage

Changing learning rates

Adaptive learning rate

Adagrad

$$\eta_{k,i} = \frac{\eta_0}{\sqrt{\sum_j^k p_{j,i}^2}}$$

$$\begin{pmatrix} x_{k+1,1} \\ x_{k+1,2} \\ \vdots \\ x_{k+1,n} \end{pmatrix} = \begin{pmatrix} x_{k,1} \\ x_{k,2} \\ \vdots \\ x_{k,n} \end{pmatrix} + \begin{pmatrix} \frac{\eta_0}{\sqrt{\sum_j^k p_{j,1}}} \\ \frac{\eta_0}{\sqrt{\sum_j^k p_{j,2}}} \\ \vdots \\ \frac{\eta_0}{\sqrt{\sum_j^k p_{j,n}}} \end{pmatrix} \odot \begin{pmatrix} p_{k,1} \\ p_{k,2} \\ \vdots \\ p_{k,n} \end{pmatrix}$$

Adaptively tune the learning rate per parameter using historical updates

Duchi, John, et al. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” 2011.

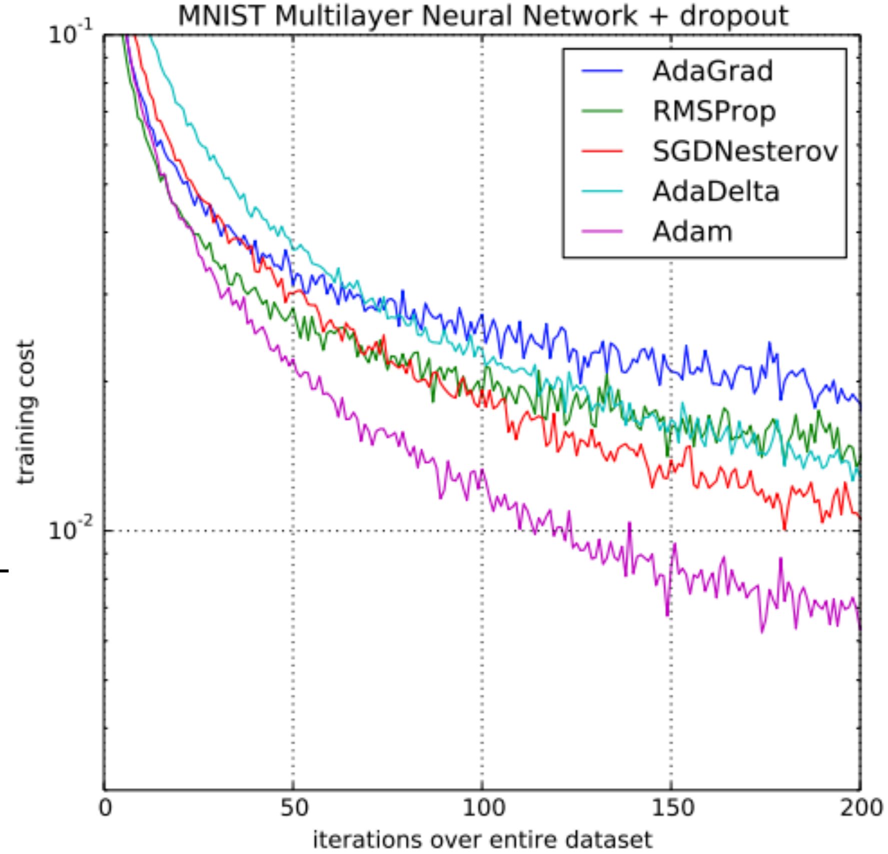
Adam optimizer

$$m_{k+1} = \beta_1 m_k + (1-\beta_1)\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_k}$$

$$v_{k+1} = \beta_2 v_k + (1-\beta_2)\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_k}^2$$

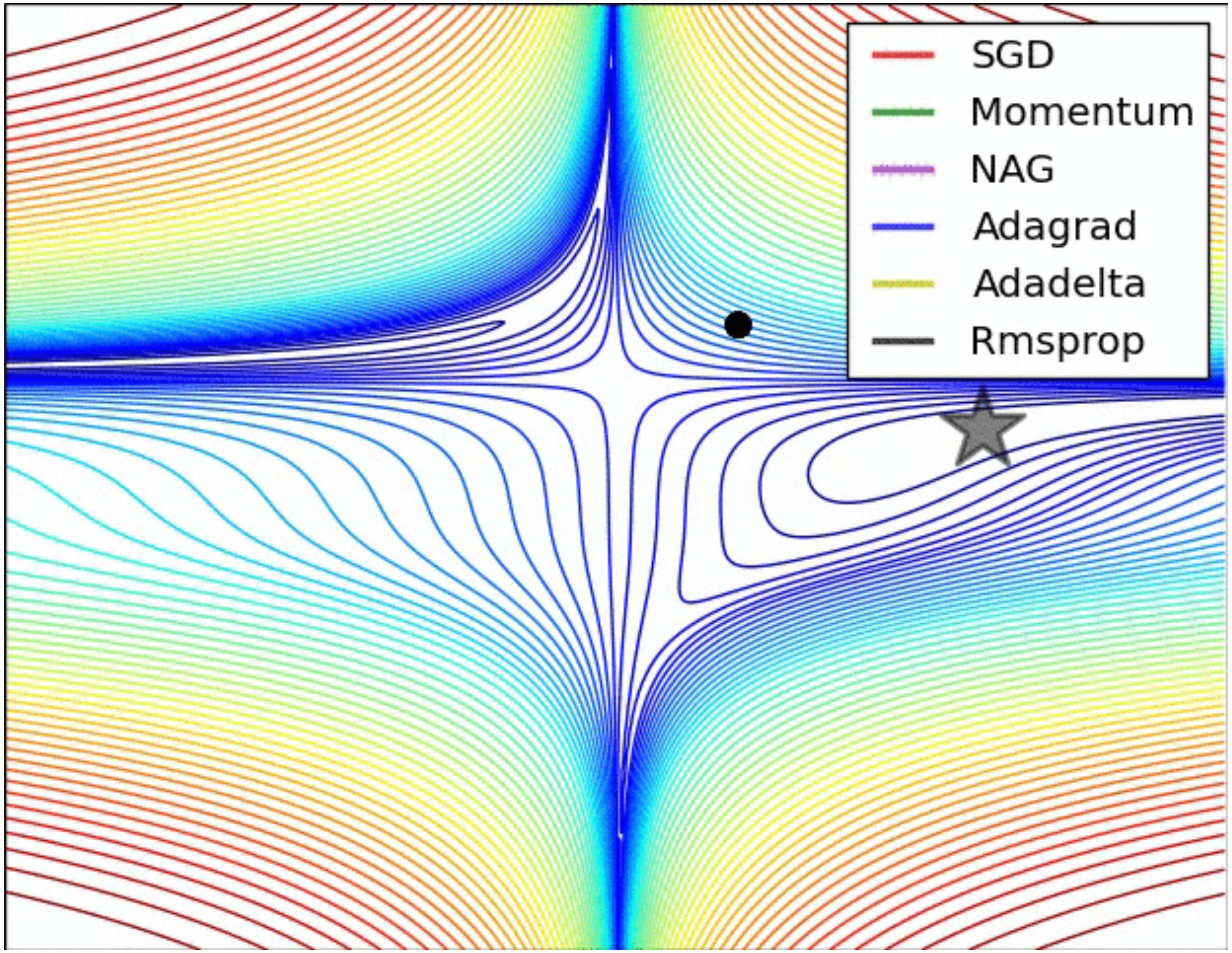
$$\hat{m}_{k+1} = \frac{m_{k+1}}{1-\beta_1^{k+1}} \quad \hat{v}_{k+1} = \frac{v_{k+1}}{1-\beta_2^{k+1}}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1}}}$$



Kingma, Diederik, et al. "Adam: A Method for Stochastic Optimization." 2014.

Some intuitions



Picture from: Alec Radford@twitter

Data-Driven Modeling *MSDM 5055*

Deep Learning for Modeling: Concepts, Tools, and Techniques

Week 6 Tutorial

Li Shuo-Hui

@HKUST | Spring 2024

PyTorch Module

nn.Module

```
NeuralNetwork(
  (hidden): Linear(in_features=10, out_features=30, bias=True)
  (sigmoid): Sigmoid()
  (output): Linear(in_features=30, out_features=2, bias=True)
  (softmax): Softmax()
)
```

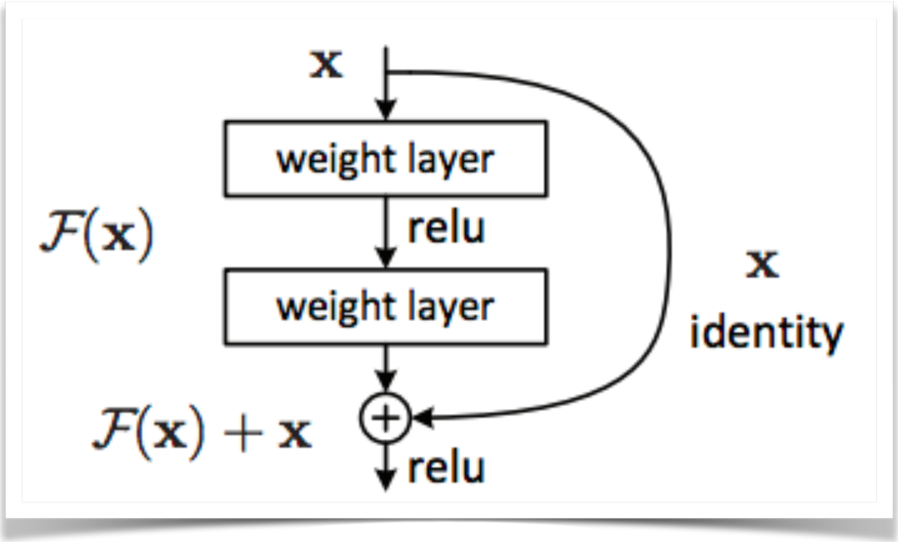
nn.Sequential

```
Sequential(
  (0): Linear(in_features=10, out_features=30, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=30, out_features=2, bias=True)
  (3): Softmax()
)
```

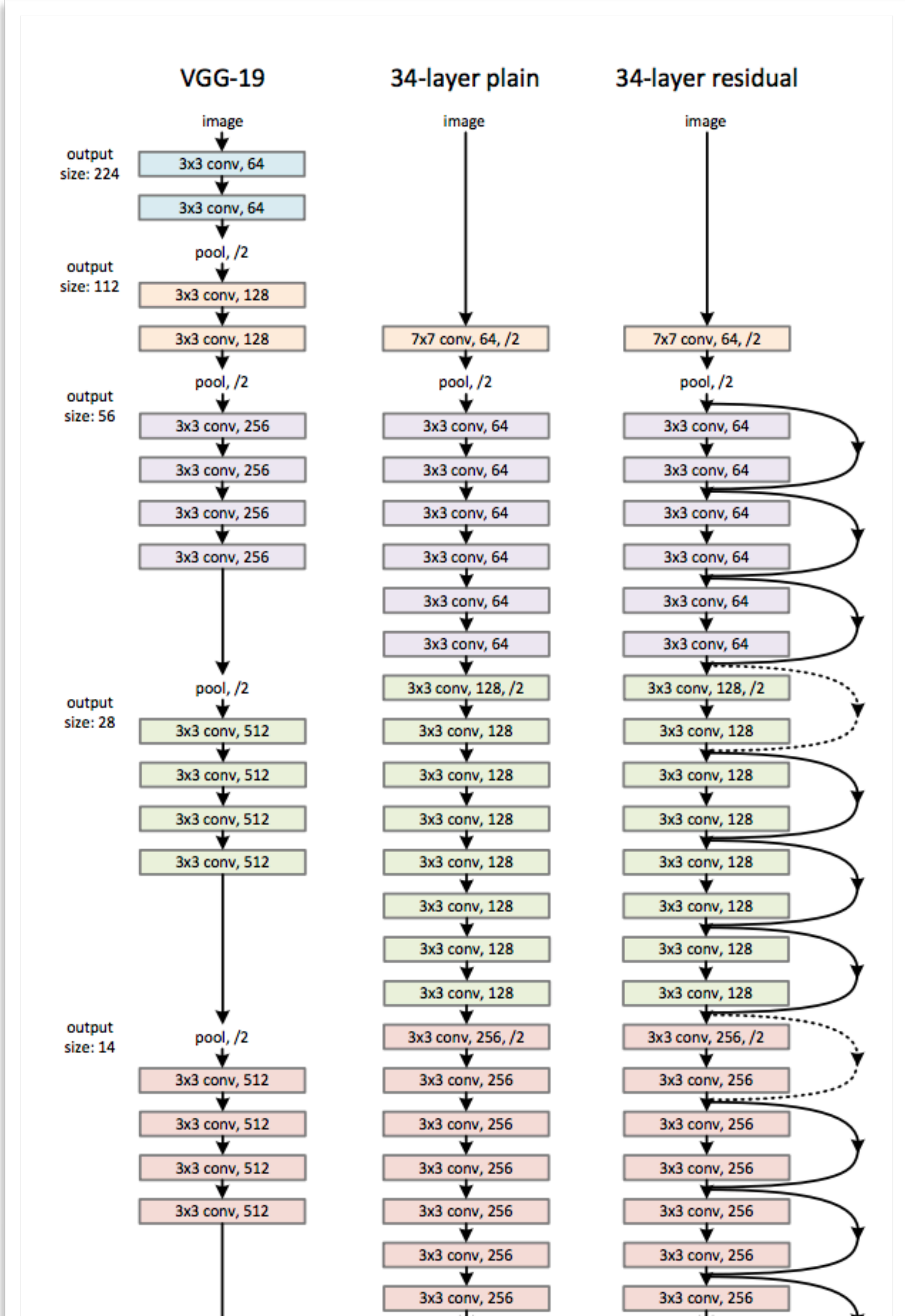
nn.ModuleList

```
NeuralNetwork(
  (module_list): ModuleList(
    (0): Linear(in_features=10, out_features=30, bias=True)
    (1): Sigmoid()
    (2): Linear(in_features=30, out_features=2, bias=True)
    (3): Softmax()
  )
)
```

Resnet

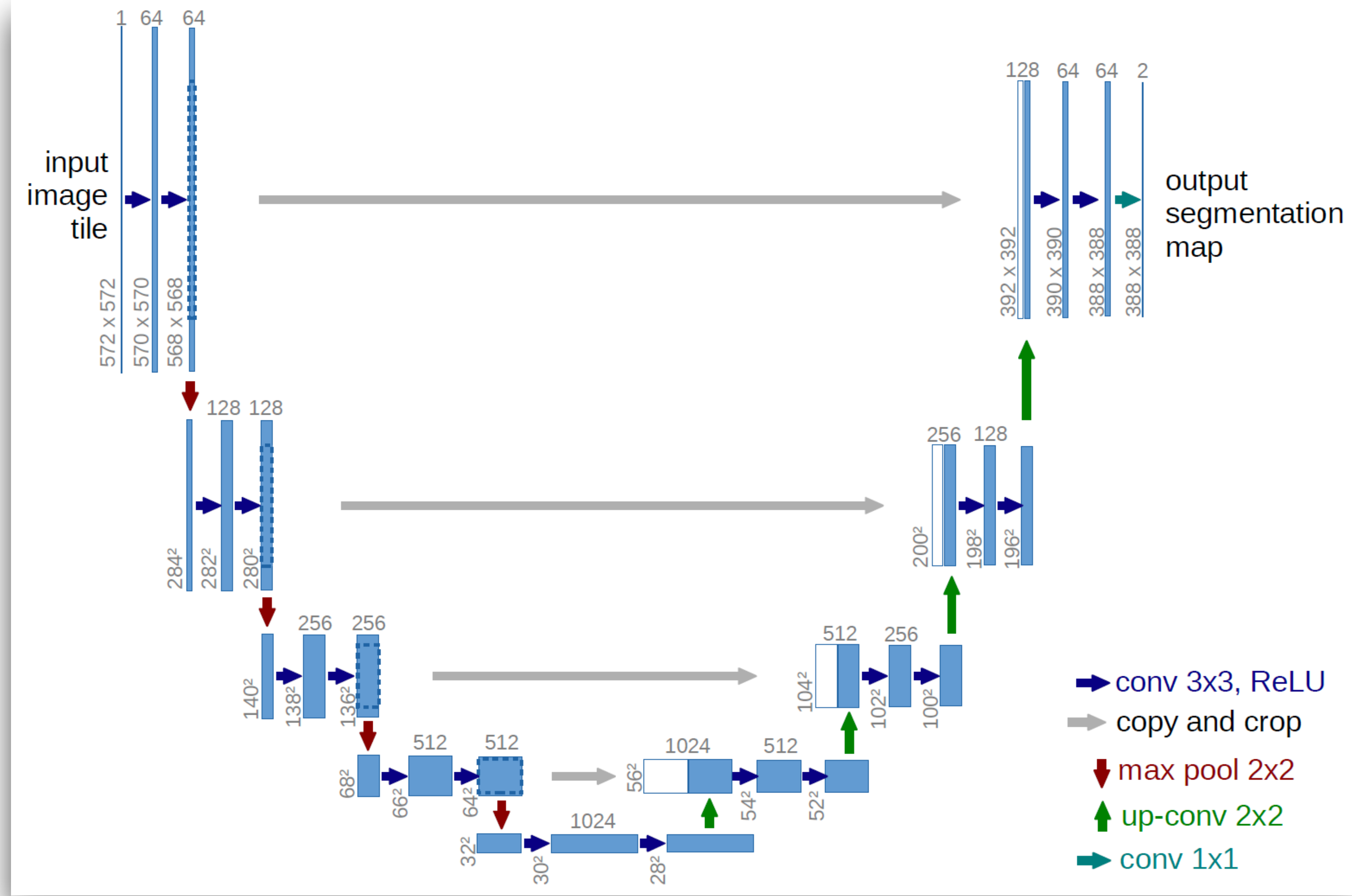


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu



VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
7x7 conv, 64, /2
output size: 56
3x3 conv, 256
3x3 conv, 64
output size: 28
3x3 conv, 512
3x3 conv, 128, /2
3x3 conv, 128
output size: 14
3x3 conv, 256, /2
3x3 conv, 256

Unet



input image tile
output segmentation map
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
1024
32²
30²
28²
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1