深度学习平台与应用

第三讲：正则化与优化

范琦

fanqi@nju.edu.cn

2024年9月18日

大　　纲

- 正则化
- 模型优化

图像分类

- 图像分类是最核心的计算机视觉任务
- 问题：语义鸿沟（Semantic Gap）
- 挑战：
  - 视角差异（Viewpoint Variation）
  - 光照变化（Illumination）
  - 杂乱的背景（Background Clutter）
  - 遮挡（Occlusion）
  - 形变（Deformation）
  - 类内差异（Intra-Class Variation）
  - 类间混淆（Inter-Class Similarity）
  - 环境干扰（Context Disturbance）
- KNN 最近邻分类器

线性分类器

输入图像

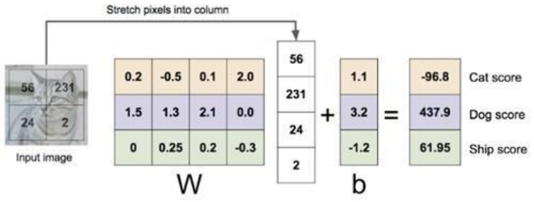
$$f(x,W) = Wx + b$$

$\longrightarrow f(x,W) \longrightarrow$

571.3	猫
1.25	狗
-132.2	船

Algebraic Viewpoint

$f(x,W) = Wx$

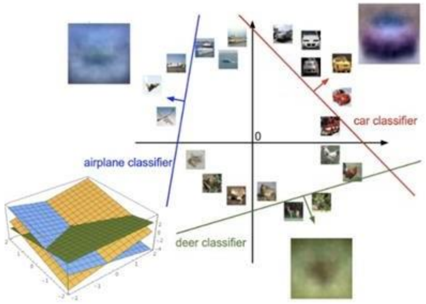


Visual Viewpoint

One template
per class

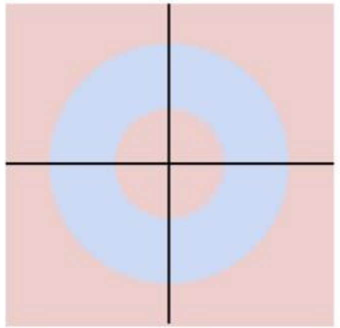
Geometric Viewpoint

Hyperplanes
cutting up space



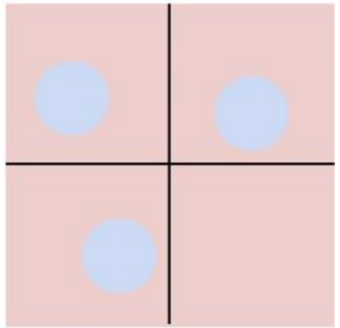
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

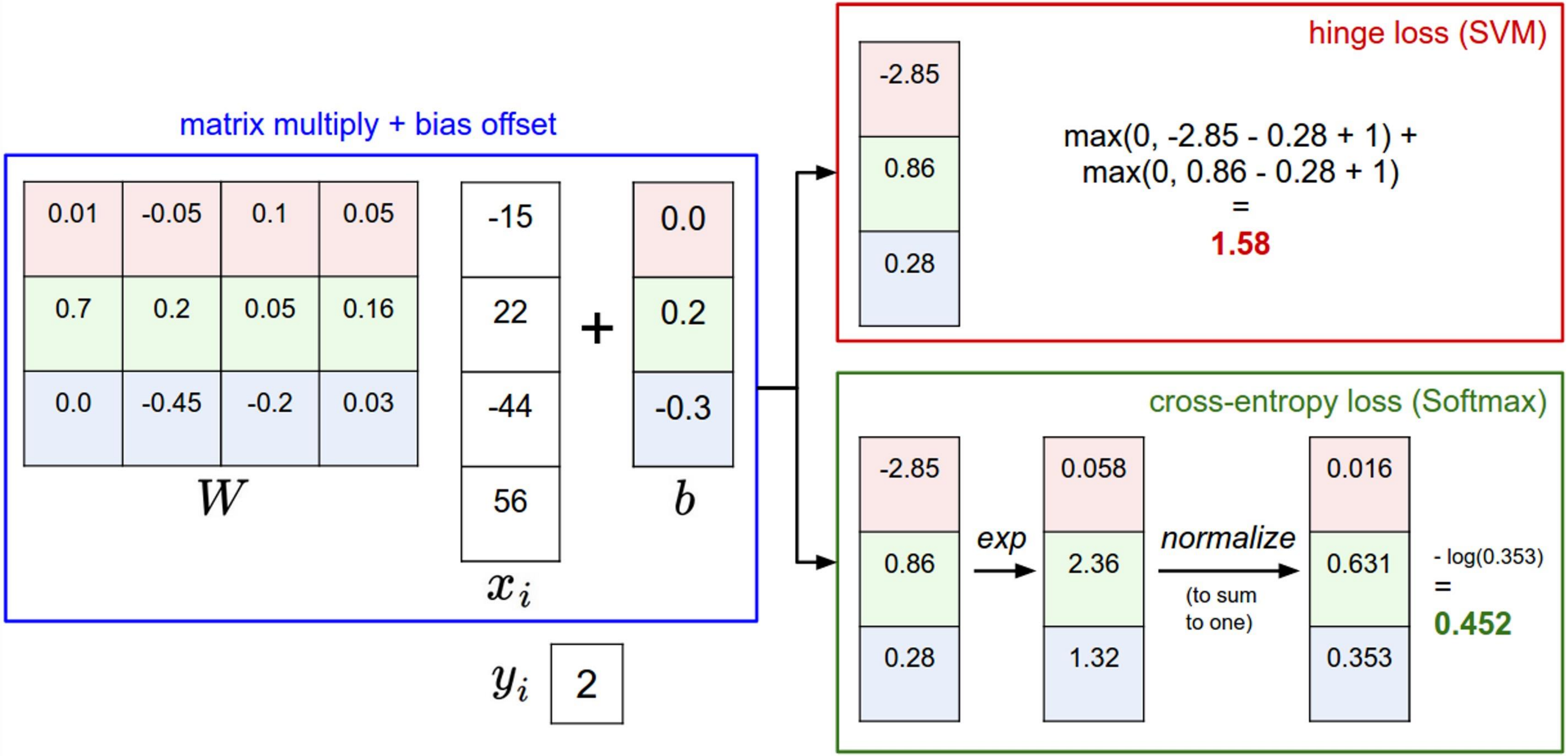
Class 2:
Everything else



线性分类器

- **Softmax *v.s.* SVM Loss**

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



线性分类器

- **SVM Loss** $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
- **Softmax Loss** $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$
  - 损失函数的最小和最大值分别可能是多少?
  - 如果我们将 SVM Loss 中的 sum 替换为 mean，会出现什么情况?
  - 训练初始时，L_i 会是什么样的?
  - 如果对 SVM Loss 中所有类别求和，会怎样? （包括 $j = y_i$ 的情况）
  - 如果正确类别的预测得分 s_{y_i} 发生轻微扰动， L_i 会发生什么?

线性分类器

- 假设我们找到一个 W，使得 L=0。这个 W 是独一无二的吗?
- 不是的！2W 依然能使得 L=0

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

线性分类器

■ 多类别 SVM Loss

$f(x, W) = Wx$

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss:	**2.9**	**0**	

■ **分类器参数为W时:**

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

■ **分类器参数为2W时:**

= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

线性分类器

- 假设我们找到一个 W，使得 L=0。这个 W 是独一无二的吗？
- 不是的！3W 依然能使得 L=0
- 那我们选择 W 还是 2W 呢？

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

正则化

模型损失

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{}$$

模型损失: 模型预测结果应该
与训练数据标签一致

正则化

- 模型损失 + 正则化

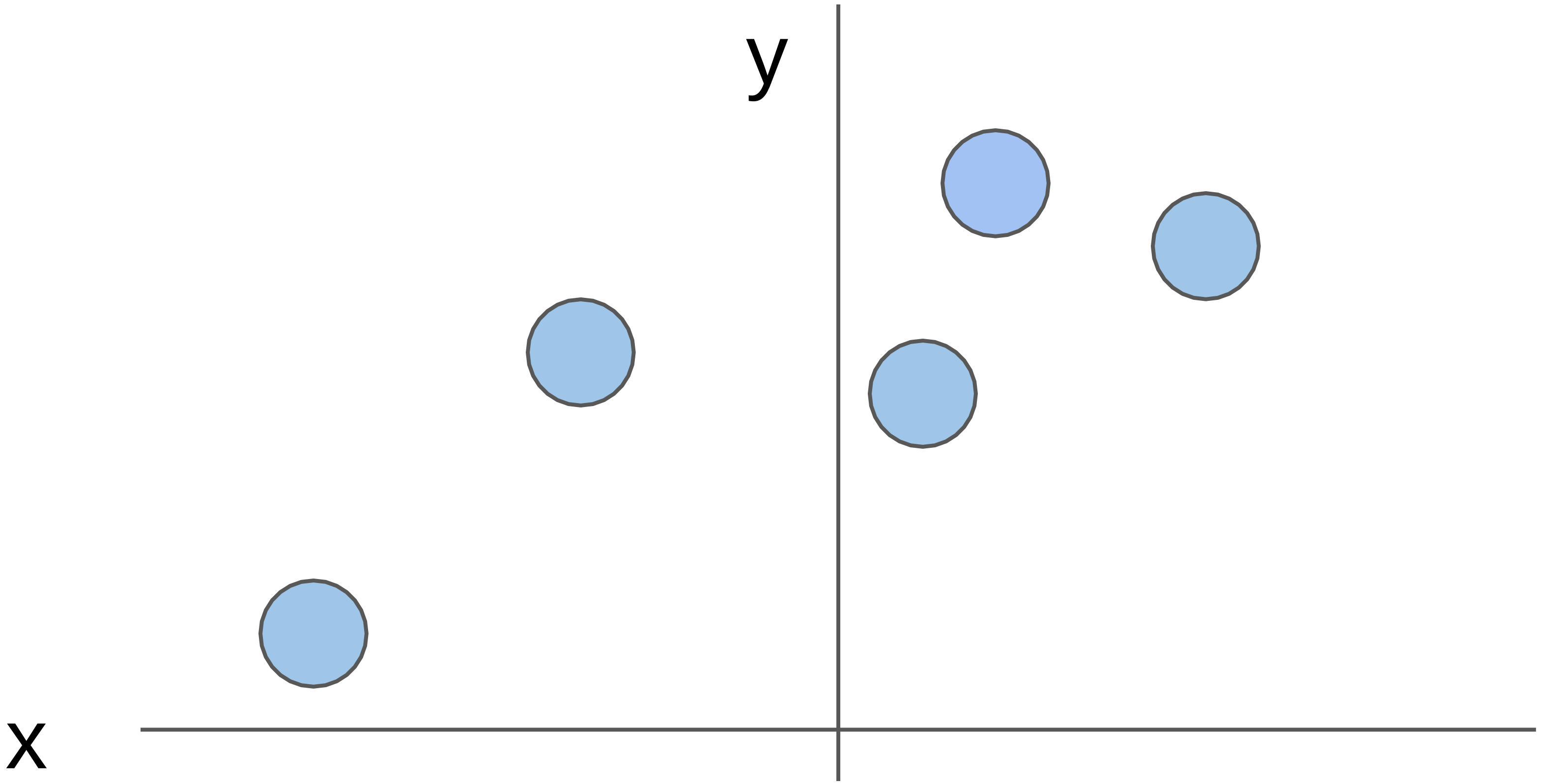
$$L(W)=\underbrace{\frac{1}{N}\sum_{i=1}^{N}L_i(f(x_i,W),y_i)}_{\text{模型损失}}+\underbrace{\lambda R(W)}_{\text{正则化}}$$

模型损失: 模型预测结果应该与训练数据标签一致

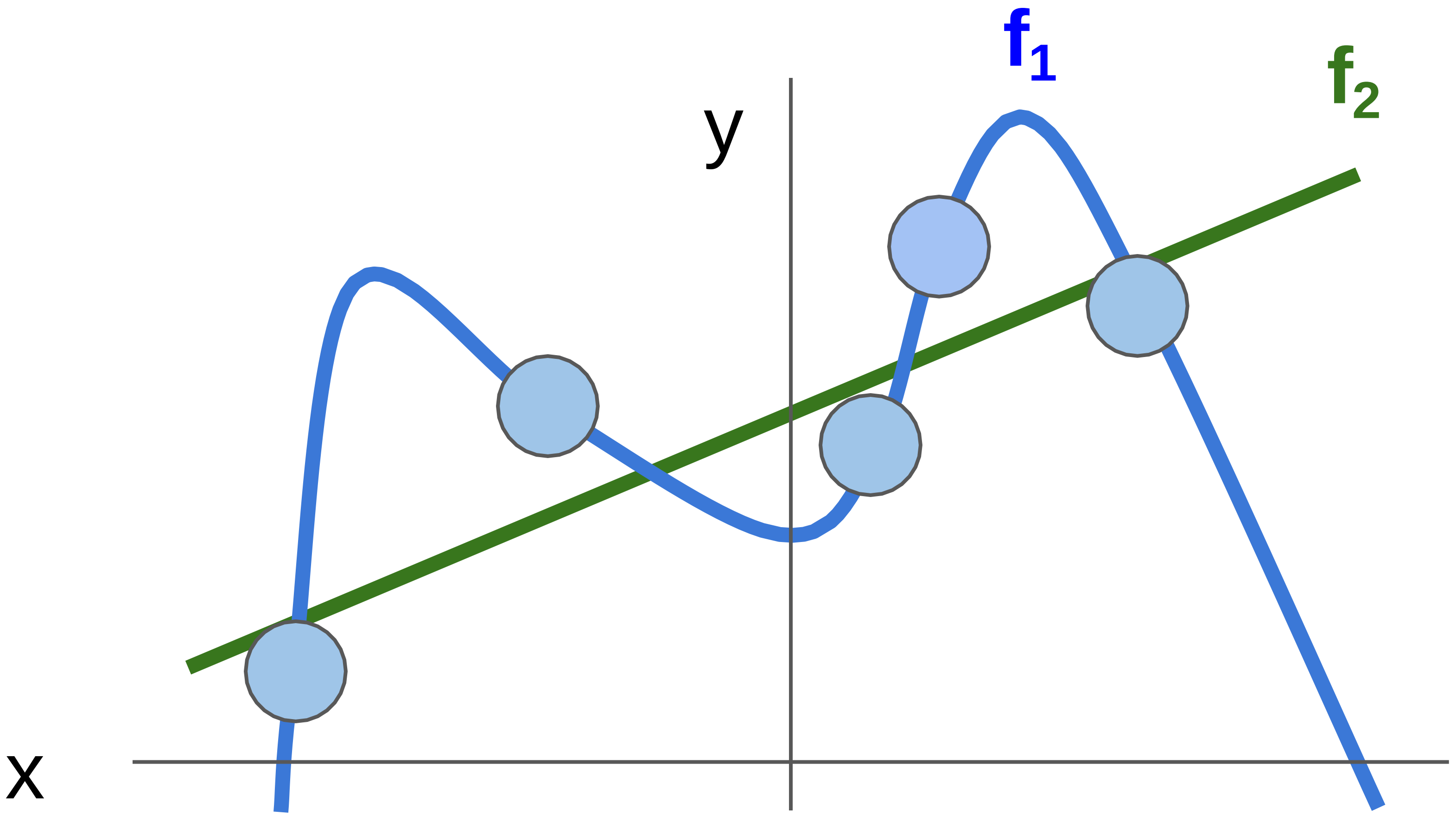
正则化：避免模型过拟合训练数据

正则化

正则化例子

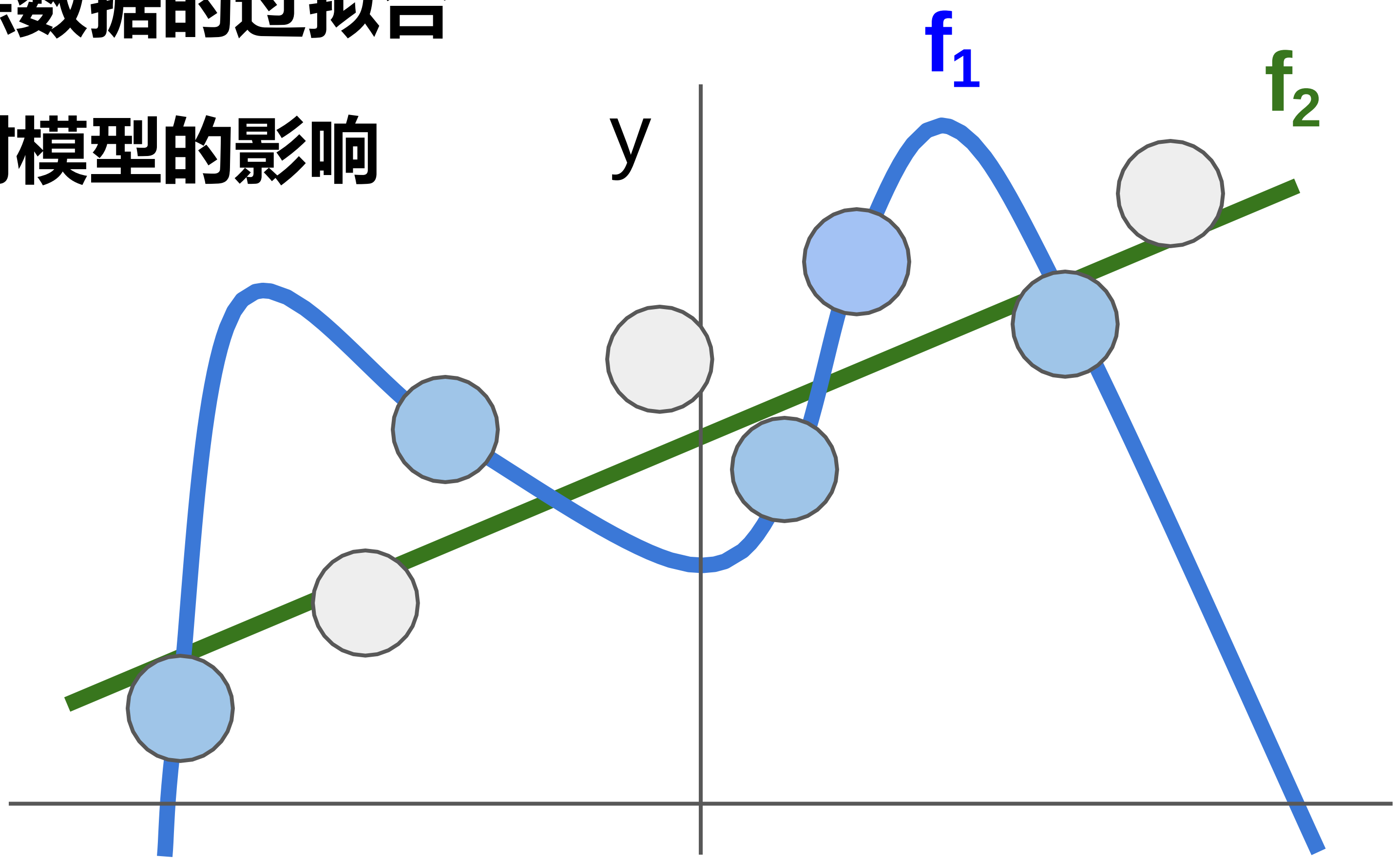


- 正则化例子



正则化

- 正则化：偏好简单的模型
  - 降低模型对训练数据的过拟合
  - 降低训练噪声对模型的影响



正则化

- 模型损失 + 正则化
- 奥卡姆剃刀原则（Occam’s Razor）：
  - 如无必要，勿增实体（Among multiple competing hypotheses, the simplest is the best)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{模型损失: 模型预测结果应该与训练数据标签一致}} + \underbrace{\lambda R(W)}_{\text{正则化：避免模型过拟合训练数据}}$$

正则化

■ 模型损失 + 正则化

λ = 正则化强度的超参数

$$L(W)=\underbrace{\frac{1}{N}\sum_{i=1}^{N}L_i(f(x_i,W),y_i)}_{\text{模型损失: 模型预测结果应该与训练数据标签一致}}+\underbrace{\lambda R(W)}_{\text{正则化：避免模型过拟合训练数据}}$$

正则化

- **模型损失 + 正则化**

λ = 正则化强度的超参数

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

- **正则化项例子：**

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

正则化

- 模型损失 + 正则化

λ = 正则化强度的超参数

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

- 正则化项例子：

  L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

  L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

  Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- 其他正则化方法：Dropout，Batch normalization，Stochastic depth, fractional pooling, 等等

正则化

南京大學 NANJING UNIVERSITY

- 模型损失 + 正则化

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

- 为什么使用正则化呢?
  - 表达对模型参数的偏好
  - 使模型简单，使其适用于测试数据
  - 改进模型优化

正则化

- 正则化：表达对模型参数的偏好

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

L2 正则化更偏好 w1 还是 w2?

正则化

- 正则化：表达对模型参数的偏好

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

$$w_1^T x = w_2^T x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

L2 正则化更偏好 w1 还是 w2?

L2 正则化偏好“分散”权重

正则化

- **正则化：表达对模型参数的偏好**

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

$$w_1^T x = w_2^T x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

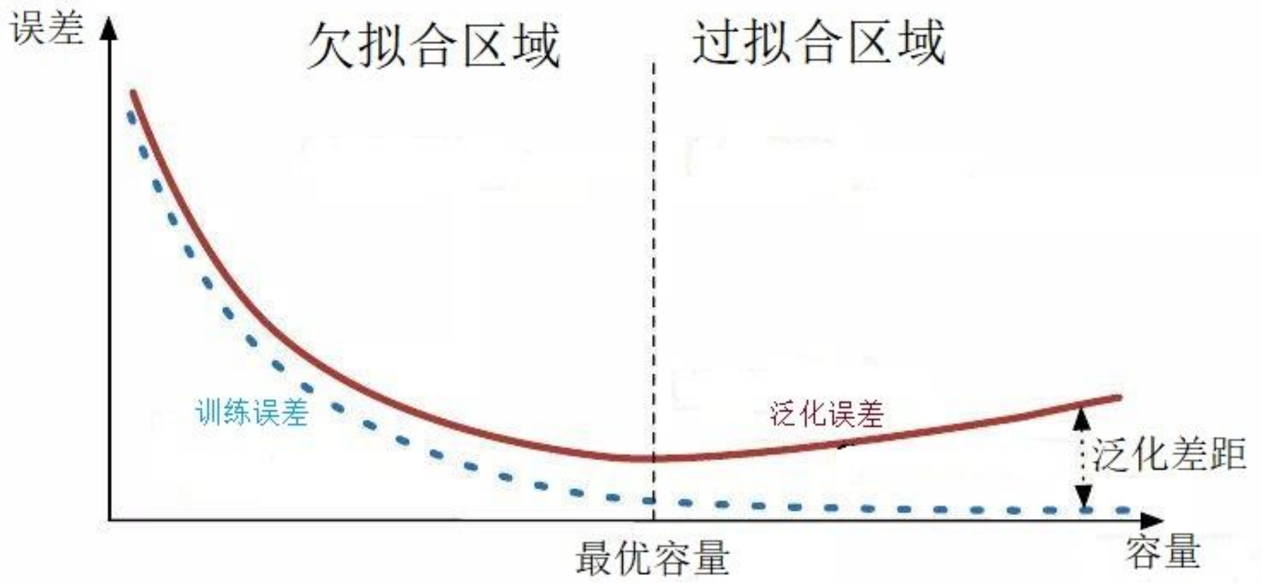
L2 正则化更偏好 w1 还是 w2?

L2 正则化偏好“分散”权重

L1 正则化偏好什么样的权重?

正则化

- **我们有训练数据：** $\{(x_i, y_i)\}_{i=1}^{N}$
- **我们有模型预测结果：** $s = f(x; W) = Wx$
- **我们有损失函数：** $L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$
- **我们怎么优化模型来降低模型在数据上的损失？**



大　　　纲

- 正则化
- 模型优化

模型优化

■ Idea 1: 模型参数随机搜索

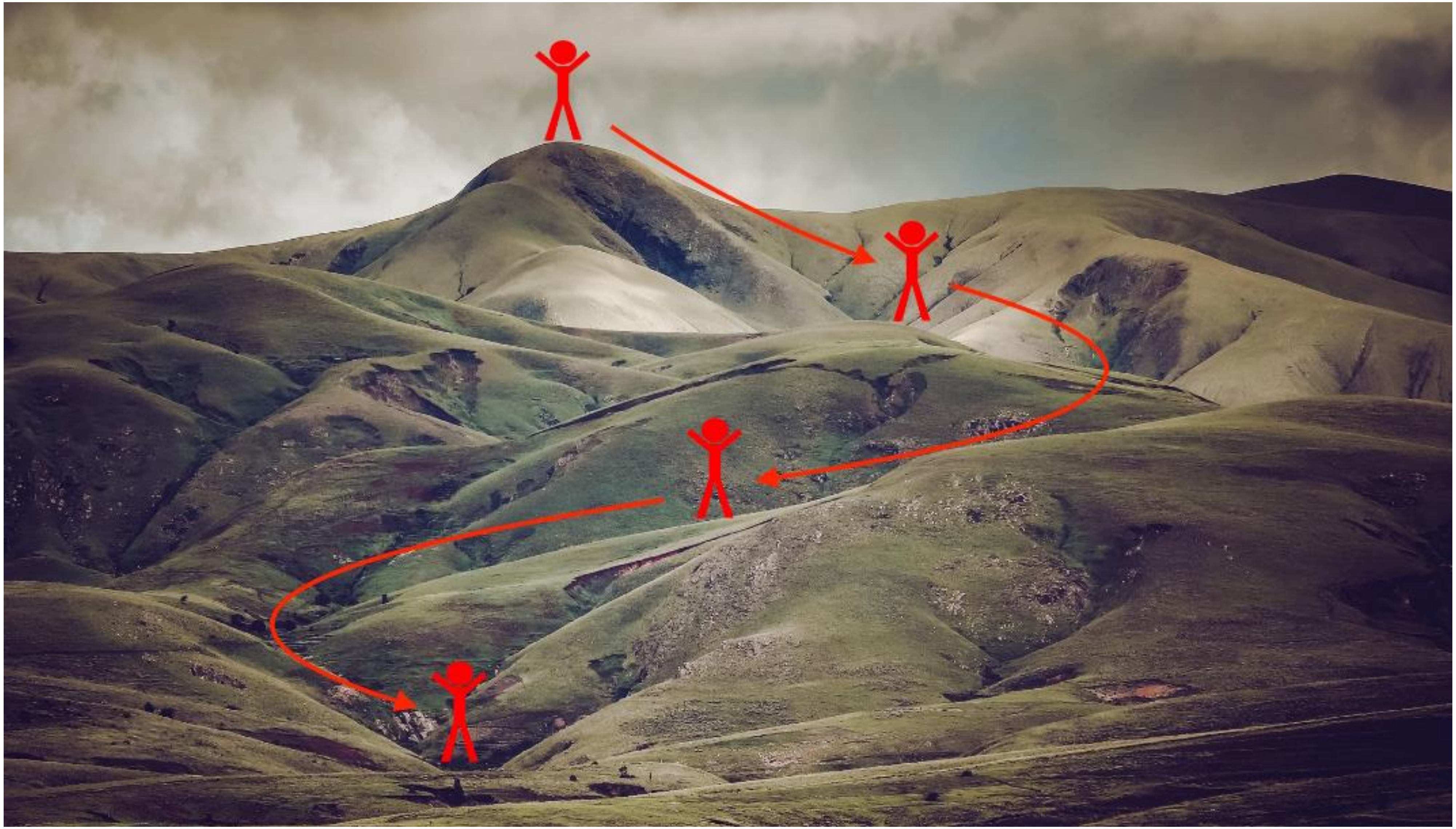
```python
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (trunctated: continues for 1000 lines)
```

模型优化

- Idea 1: 模型参数随机搜索
- 效果非常差！15.55%的准确率，SOTA 为 > 99.7%准确率

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

模型优化



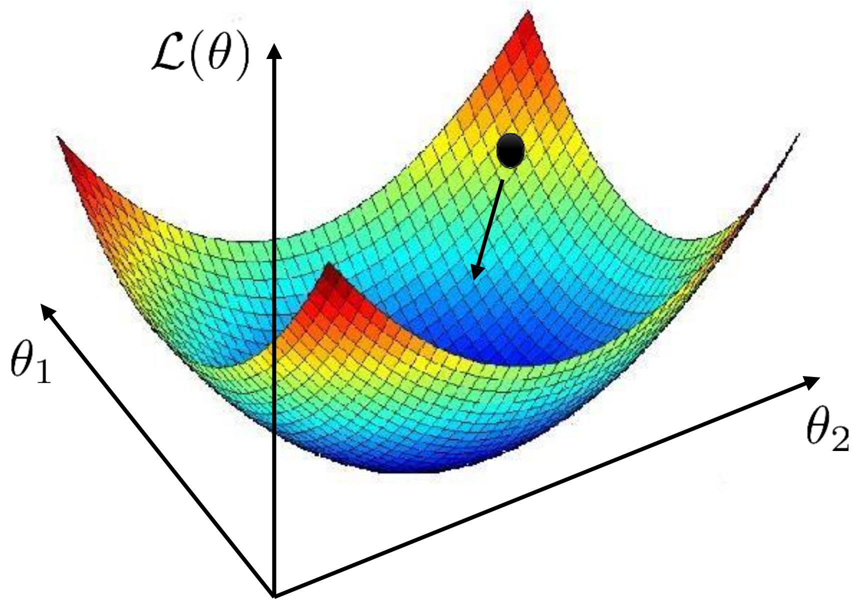
模型优化

- **Idea 2: 沿着最陡下降方向进行模型参数优化**

$$\theta^\star \leftarrow \arg\min_\theta \underbrace{-\sum_i \log p_\theta(y_i|x_i)}_{\mathcal{L}(\theta)}$$

假设 $\boldsymbol{\theta}$ 是二维的

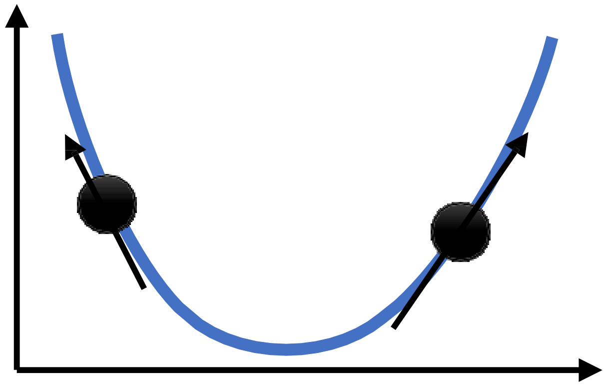
- **找到一个方向 $\boldsymbol{v}$ 使得损失 $\mathcal{L}(\boldsymbol{\theta})$ 降低**
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\alpha v}$



模型优化

$$\theta^\star \leftarrow \arg\min_\theta \underbrace{-\sum_i \log p_\theta(y_i|x_i)}_{\mathcal{L}(\theta)}$$

- **找到一个方向 $\boldsymbol{v}$ 使得损失 $\mathcal{L}(\boldsymbol{\theta})$ 降低**
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
- **哪个方向能使 $\mathcal{L}(\boldsymbol{\theta})$ 降低呢?**
- **梯度向量:**
  - **梯度方向:函数值最大增长的方向**
  - **梯度值:函数在这个方向上的增长率**



模型优化

- 找到一个方向 $\boldsymbol{v}$ 使得损失 $\mathcal{L}(\boldsymbol{\theta})$ 降低 $\theta^\star \leftarrow \arg\min_\theta \underbrace{-\sum_i \log p_\theta(y_i|x_i)}_{\mathcal{L}(\theta)}$
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
- 哪个方向能使 $\mathcal{L}(\boldsymbol{\theta})$ 降低呢?
- 对于每个维度，沿着该维度的梯度的相反方向更新模型参数，更新的幅度与梯度大小有关

$$\frac{df(x)}{dx} = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

$$v_1 = -\frac{d\mathcal{L}(\theta)}{d\theta_1}$$

$$v_2 = -\frac{d\mathcal{L}(\theta)}{d\theta_2}$$

$$\nabla_\theta \mathcal{L}(\theta) = \begin{pmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{pmatrix}$$

模型优化

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

梯度 dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

模型优化

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (第一维):

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25322

梯度 dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

模型优化

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (第一维):

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25322

梯度 dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

?,
?,…]

模型优化

南京大學 NANJING UNIVERSITY

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (第二维):

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25353

梯度 dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,…]

模型优化

NANJING UNIVERSITY

当前的 W:	W + h (第二维):	梯度 dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	**0.6**,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

(1.25353 - 1.25347)/0.0001
= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

模型优化

南京大学 NANJING UNIVERSITY

当前的 W:	W + h (第三维):	梯度 dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

模型优化

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (第三维):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

梯度 dW:

[-2.5,
0.6,
0,
?,
?,…]

(1.25347 - 1.25347)/0.0001
= 0

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

模型优化

NANJING UNIVERSITY

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (第三维):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

梯度 dW:

[-2.5,
0.6,
0,
?,

数值法梯度计算:

- **太慢了！需要对所有维度进行循环计算**
- **求得的是一个近似值**
- **是一种“笨”方法**

模型优化

- 损失函数是模型参数 W 的函数
- 使用微积分计算解析梯度 $\nabla_W L$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

模型优化

当前的 W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

dW = ...
(数据和模型参数的函数)

梯度 dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

模型优化

- 数值法梯度计算：计算慢，近似，但易于实现
- 解析法梯度计算：快速，准确，但实现较为复杂
- **梯度检查**：使用解析法计算梯度，使用数值法检查梯度
- **根据梯度更新模型参数，全量数据训练？**

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

NANJING UNIVERSITY

- 随机梯度下降（Stochastic Gradient Descent, SGD）
  - 计算资源与成本

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

  - 使用小批量（mini batch）样本计算

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

模型优化

■ 随机梯度下降（Stochastic Gradient Descent, SGD）

算法 2.1: 随机梯度下降法

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α

随机初始化 θ;

repeat

  对训练集 $\mathcal{D}$ 中的样本随机重排序;

  for $n = 1 \cdots N$ **do**

    从训练集 $\mathcal{D}$ 中选取样本 $(\mathbf{x}^{(n)}, y^{(n)})$;

    // 更新参数

    $$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}\left(\theta; x^{(n)}, y^{(n)}\right)}{\partial \theta};$$

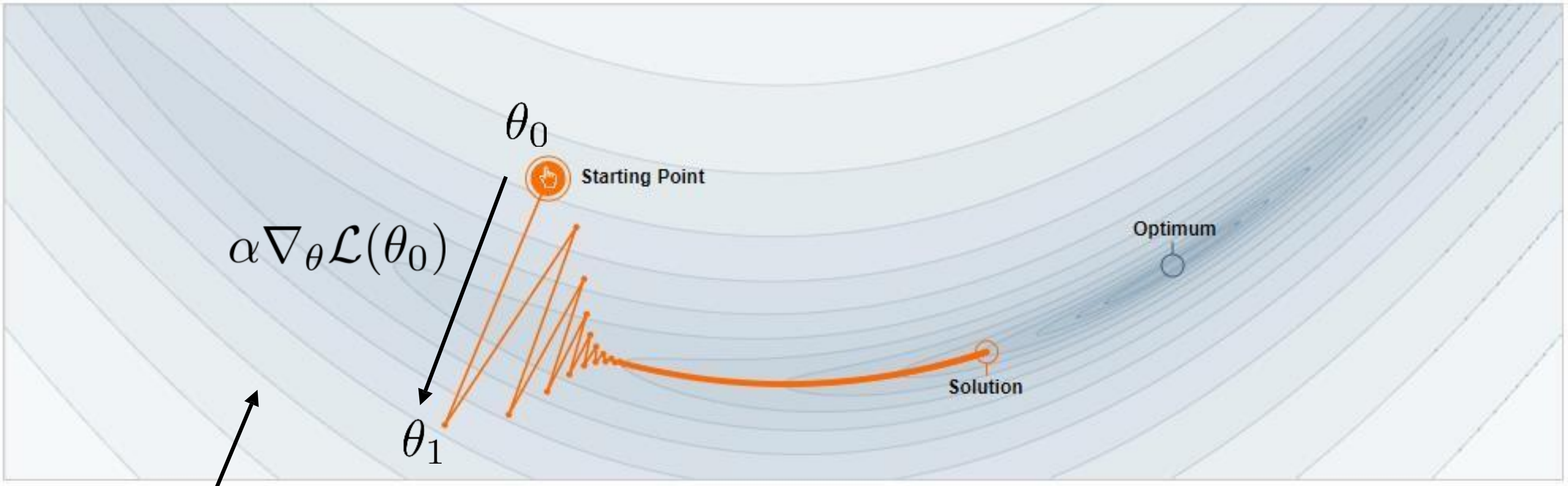
  end

until 模型 $f(\mathbf{x}; \theta)$ 在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: θ

模型优化

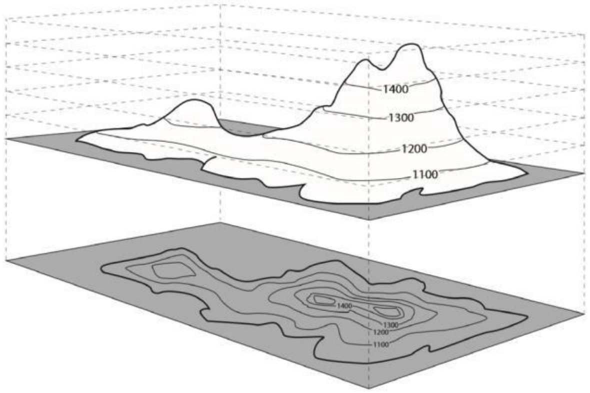
■ SGD 梯度下降可视化



水平集轮廓

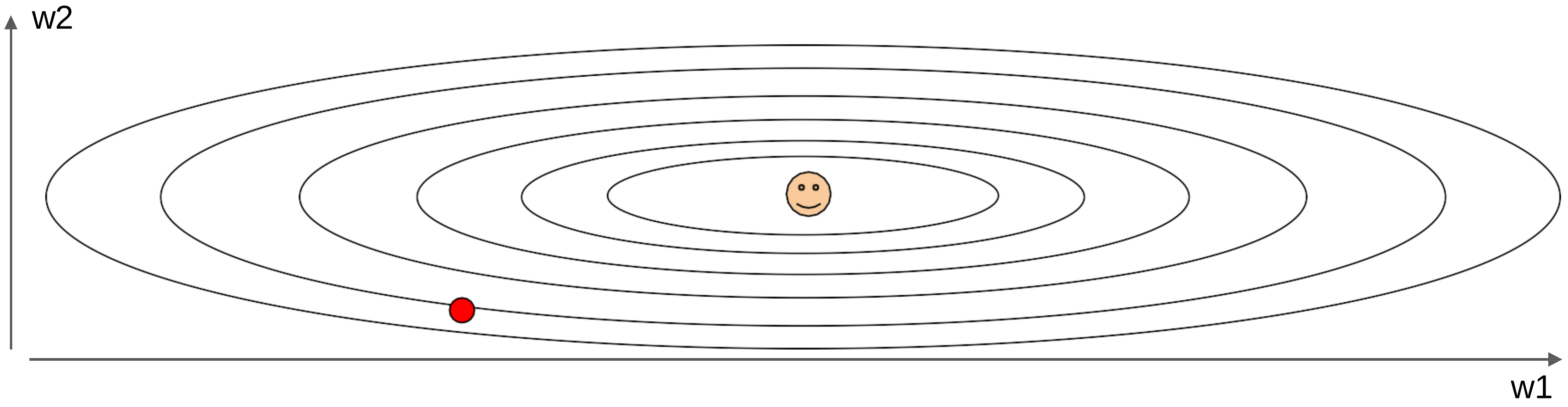
for all θ values along a line

$\mathcal{L}(\theta)$ takes on the same value



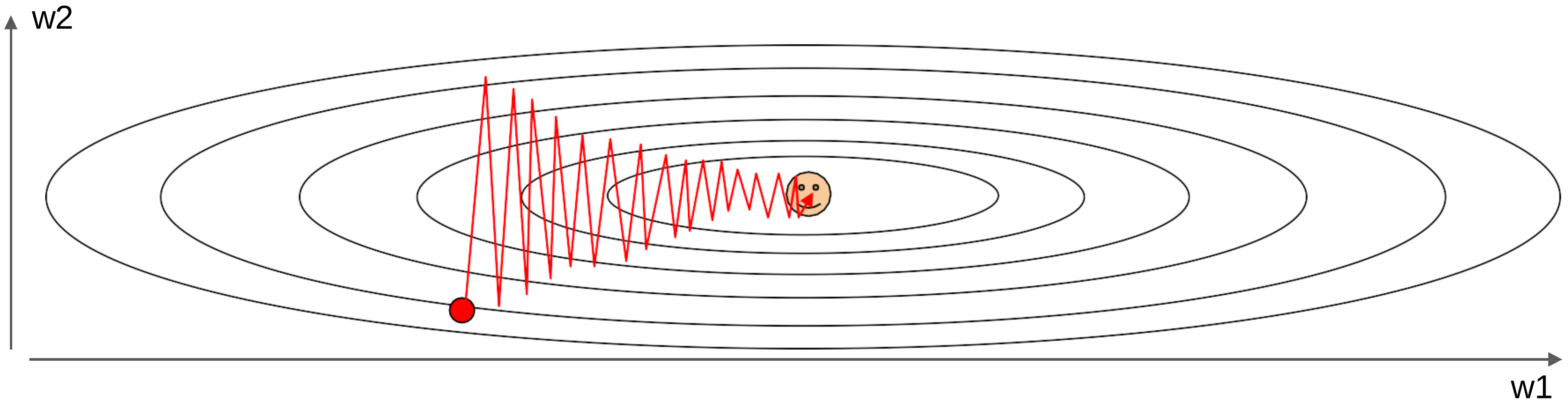
模型优化

- **SGD 问题1: 如果损失在一个方向上变化很快，在另一个方向变化很慢怎么办？梯度下降法会怎样表现？**



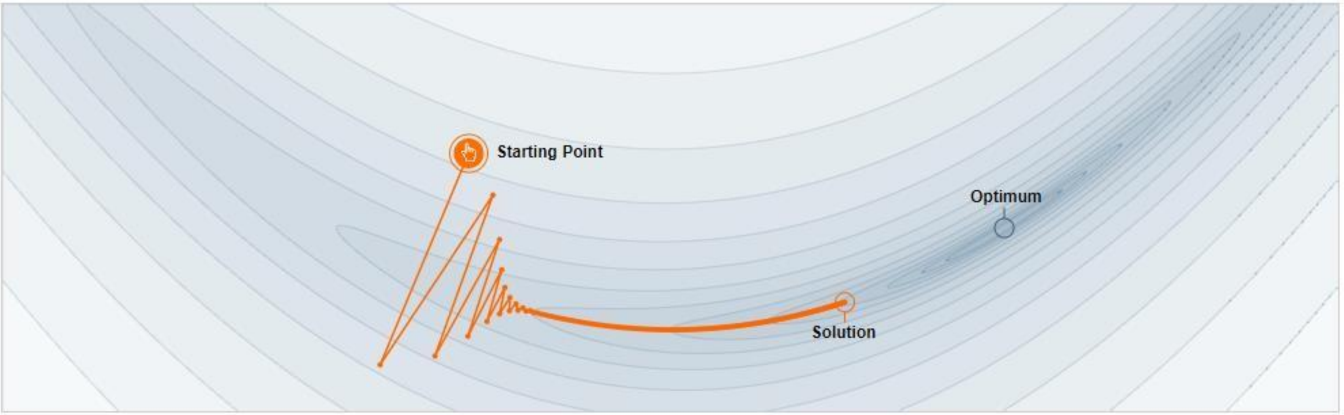
模型优化

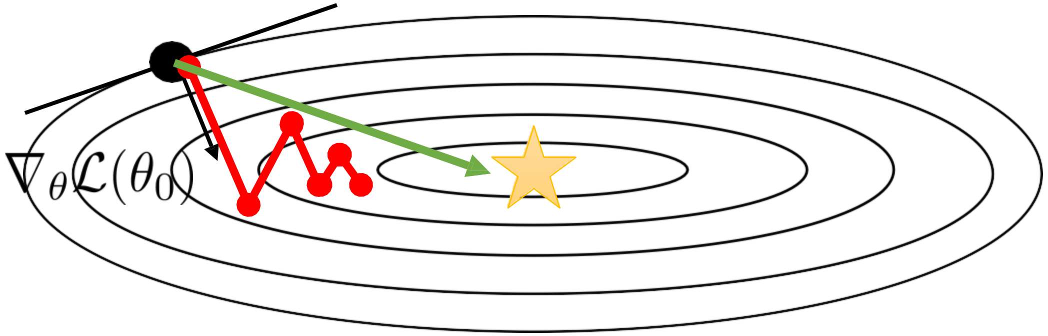
- SGD 问题1: 如果损失在一个方向上变化很快，在另一个方向变化很慢怎么办？梯度下降法会怎样表现？
- 沿 w1 方向下降非常缓慢，沿 w2 方向抖动



模型优化

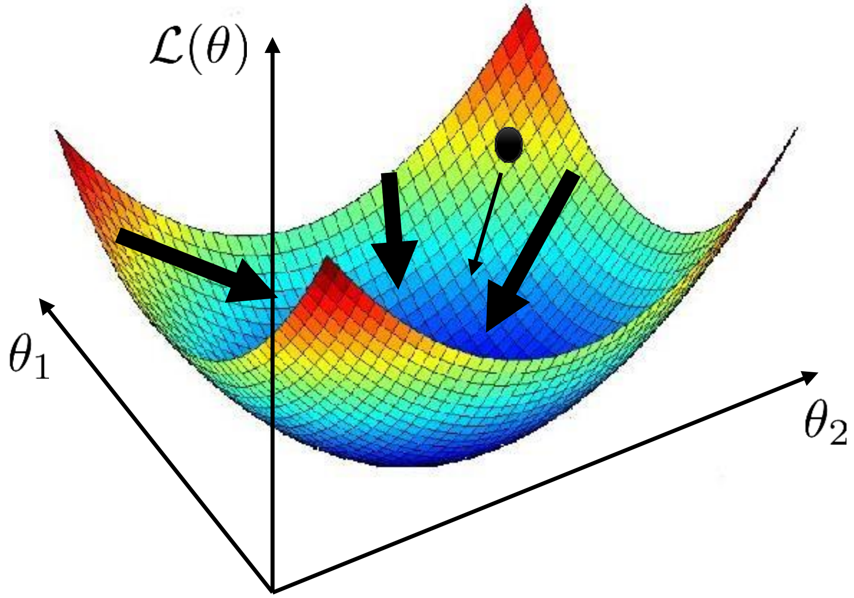
- SGD 问题1: 我们并不总是朝着全局最优点进行优化
- 当前的最陡方向并不总是最优的方向



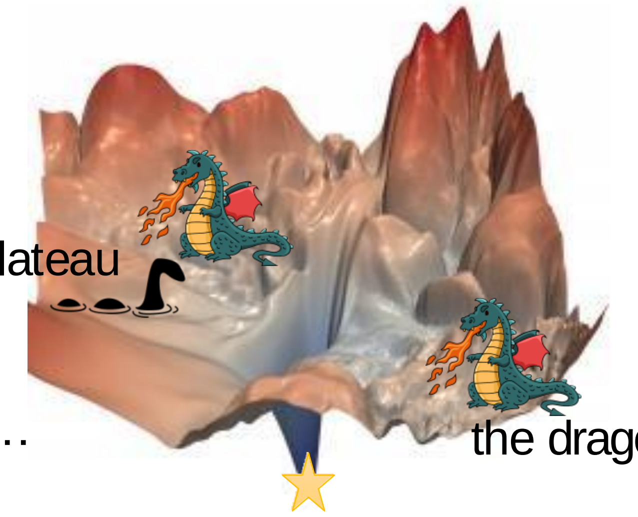


模型优化

- Loss landscape 可视化
- 神经网络的 Loss landscape 非常难以可视化，因为神经网络有非常多的参数



the monster of the plateau



Oh no…

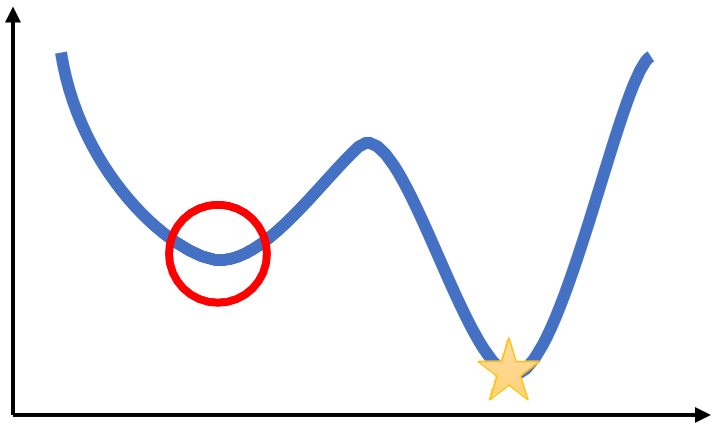
the dragon of local optima

Visualizing the Loss Landscape of Neural Nets

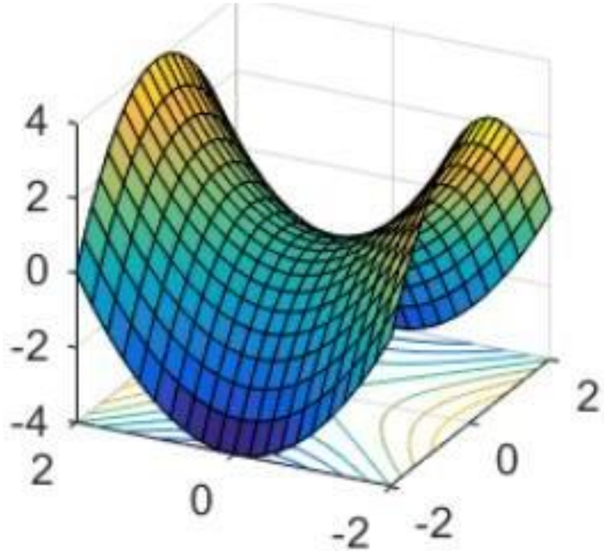
Hao Li[1], Zheng Xu[1], Gavin Taylor[2], Christoph Studer[3], Tom Goldstein[1]
[1]University of Maryland, College Park [2]United States Naval Academy [3]Cornell University
{haoli,xuzh,tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu

南京大学 NANJING UNIVERSITY

- **SGD 问题2: 如果损失函数有局部最优值（local optimum），鞍点（saddle point）怎么办?**



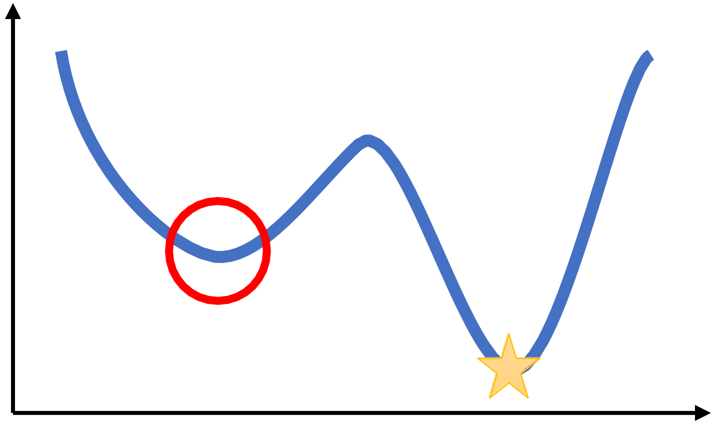
the local optimum

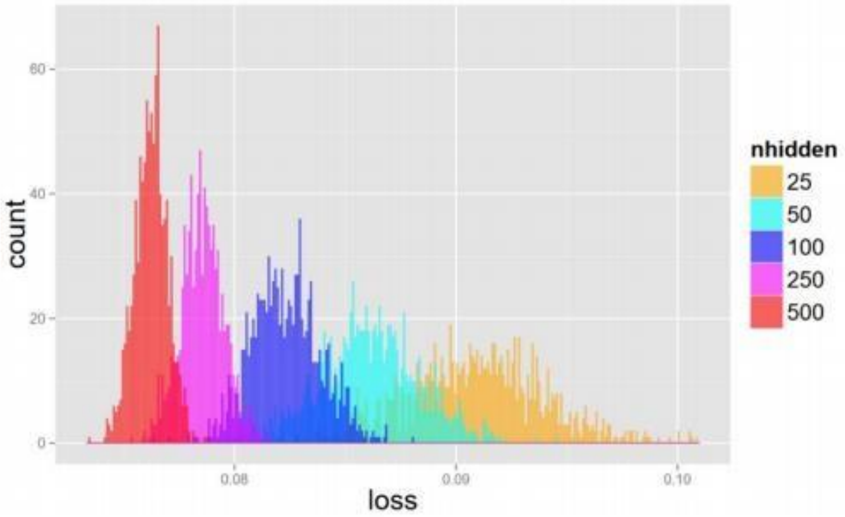


the saddle point

模型优化

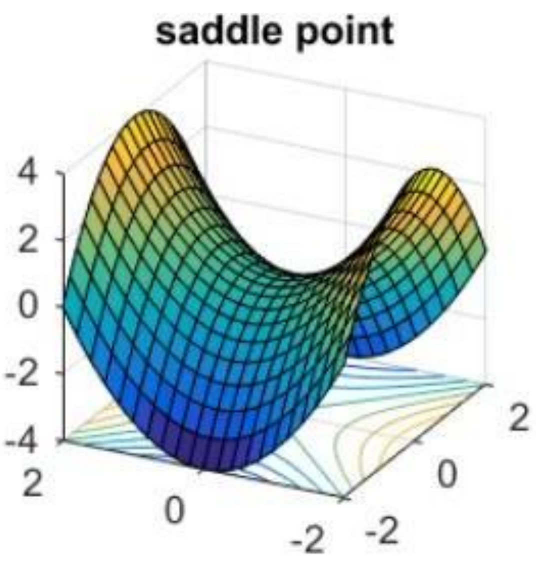
- 局部最优值（local optimum）
  - 此处梯度为 0，理论上可怕（非凸损失函数）
  - 实际上，随着网络参数量的增加，这个问题的影响会很小
  - 对于大型网络，局部最优值通常与全局最优值很接近

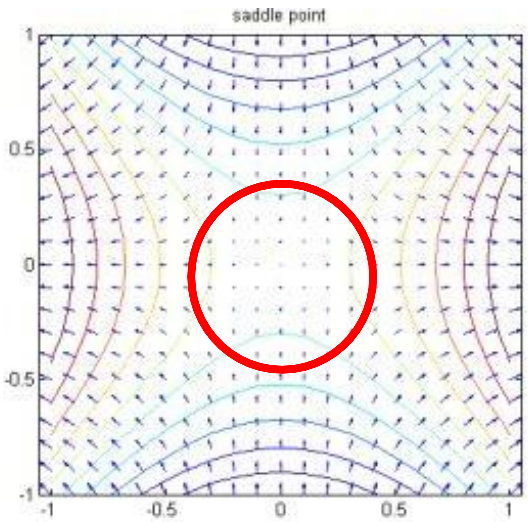




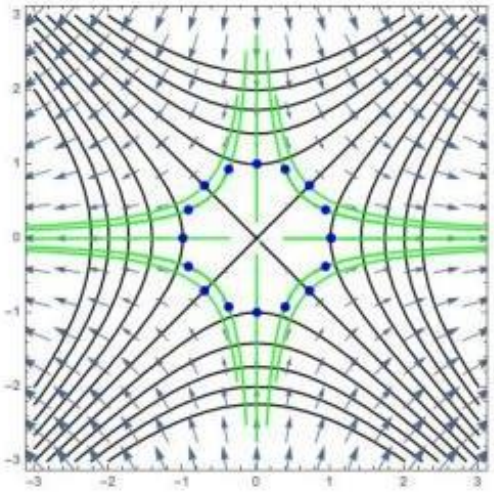
模型优化

- 鞍点（saddle point）：
  - 鞍点处的梯度很小（或为 0 ），跳出鞍点需要很长时间
  - 神经网络的 loss landscape 中的大多数关键点都是鞍点





Gradient vectors



Gradient flows (green)

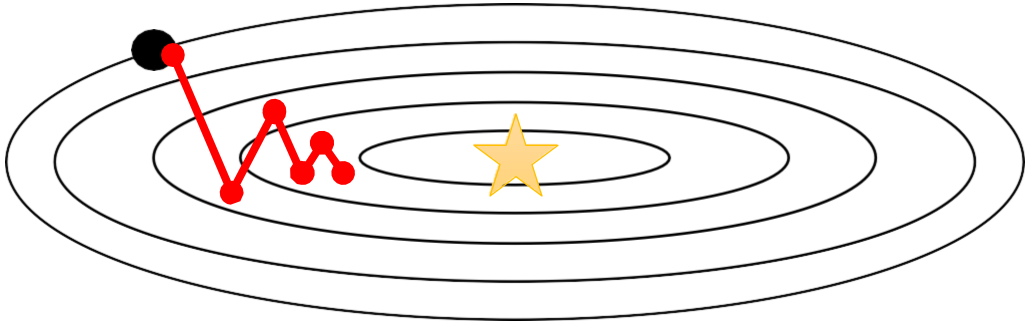
模型优化

- 区分鞍点（saddle point）和局部最优值（local optimum）？
- 损失函数的 Hessian 矩阵的特征值
  - 鞍点：Hessian 矩阵的特征值有正也有负
  - 局部最优值：Hessian 矩阵的特征值全部为正或全部为负

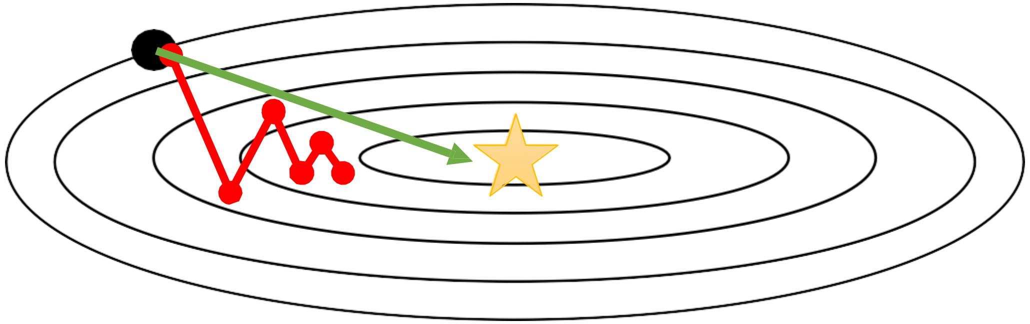
$$\begin{bmatrix} \frac{d^2\mathcal{L}}{d\theta_1 d\theta_1} & \frac{d^2\mathcal{L}}{d\theta_1 d\theta_2} & \frac{d^2\mathcal{L}}{d\theta_1 d\theta_3} \\ \frac{d^2\mathcal{L}}{d\theta_2 d\theta_1} & \frac{d^2\mathcal{L}}{d\theta_2 d\theta_2} & \frac{d^2\mathcal{L}}{d\theta_2 d\theta_3} \\ \frac{d^2\mathcal{L}}{d\theta_3 d\theta_1} & \frac{d^2\mathcal{L}}{d\theta_3 d\theta_2} & \frac{d^2\mathcal{L}}{d\theta_3 d\theta_3} \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

模型优化

- SGD 的模型优化方向：

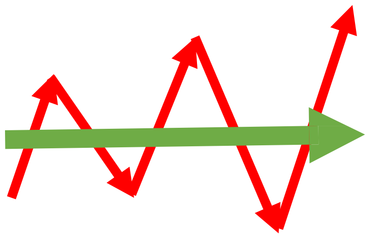


- 更优的模型优化方向：

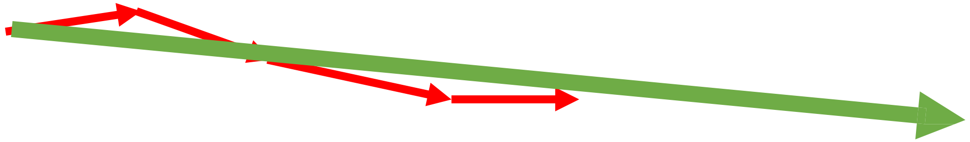


模型优化

- 想法一：
  - 如果连续的梯度优化步指向不同的方向，我们应该对优化方向进行修正

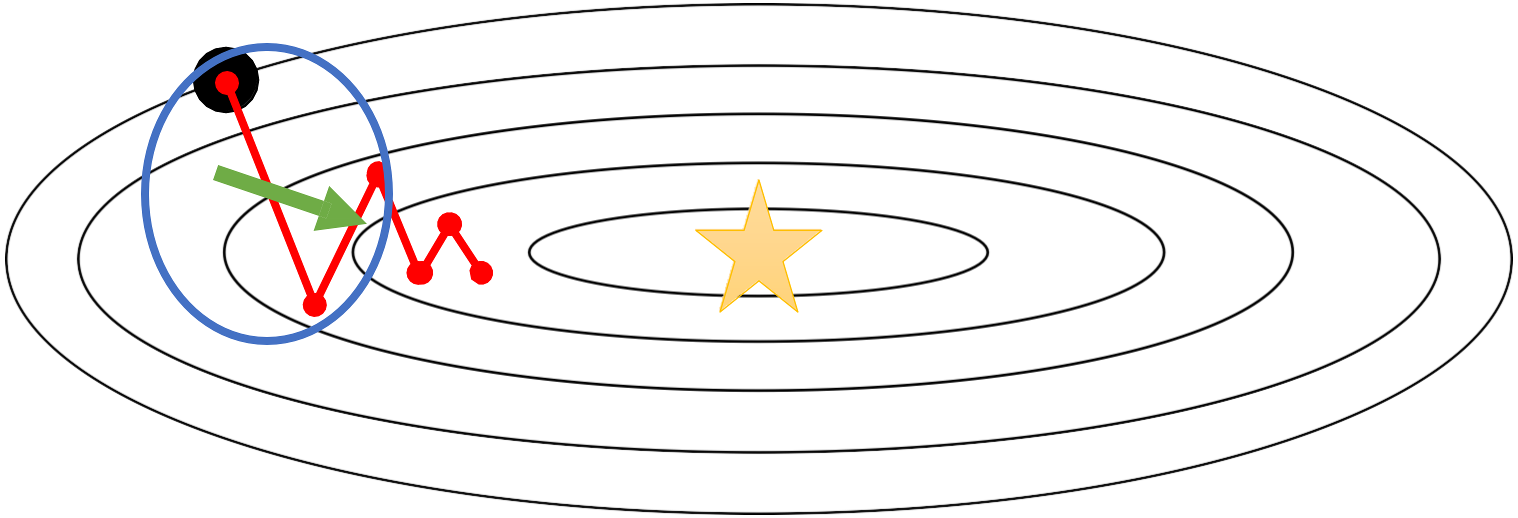


  - 如果连续的梯度优化步指向相似的方向，我们应该朝那个方向走得更快



模型优化

- SGD + Momentum
  - 将连续梯度平均在一起似乎会产生更好的方向！



模型优化

南京大学 NANJING UNIVERSITY

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

rho 一般为 0.9 或 0.99

模型优化

- 对于损失函数的每个维度（模型参数）：
  - 梯度的正负影响优化的方向
  - 梯度的大小影响参数更新的幅度
  - 梯度的值可能会发生巨大变化，使得学习率难以调整
  - 想法二：对每个维度的梯度大小进行“归一化”

RMSProp (Root Mean Squared Propagation)

SGD + Momentum

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

根据每个维度的梯度值历史平方和进行梯度缩放（有衰减）

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

模型优化

南京大學 NANJING UNIVERSITY

- RMSProp (Root Mean Squared Propagation)

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

- 问题：RMSProp 做了什么?
  - 沿着“陡峭”方向的优化变慢
  - 沿着“平缓”方向的优化加快

模型优化

南京大学 NANJING UNIVERSITY

AdaGrad (Adaptive Gradient Algorithm)

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

模型优化

- AdaGrad *v.s.* RMSProp
- AdaGrad 更适合解决凸优化问题
  - 学习率会随着时间的推移而有效地降低
  - 由于不断在分母中快速累计梯度，导致学习率不断变小，因此需要在学习率快速衰减之前找到最优值
- RMSProp 往往更适合深度学习（以及大多数非凸问题）

■ 想法三：结合 Momentum 和 RMSProp

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

模型优化

■ 想法三：结合 Momentum 和 RMSProp

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

RMSProp

- ■ 第一步会发生什么？（beta1 设为 0.9，beta2 设为 0.999)
- ■ first_moment 和 second_moment 初始化时接近 0

模型优化

Adam (Adaptive Moment Estimation)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

■ 想法四：给 Adam 加上 Weight Decay

标准的 Adam 在这里加上 L2 Regularization

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

模型优化

NANJING UNIVERSITY

AdamW

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

AdamW 在这里加上 Weight Decay

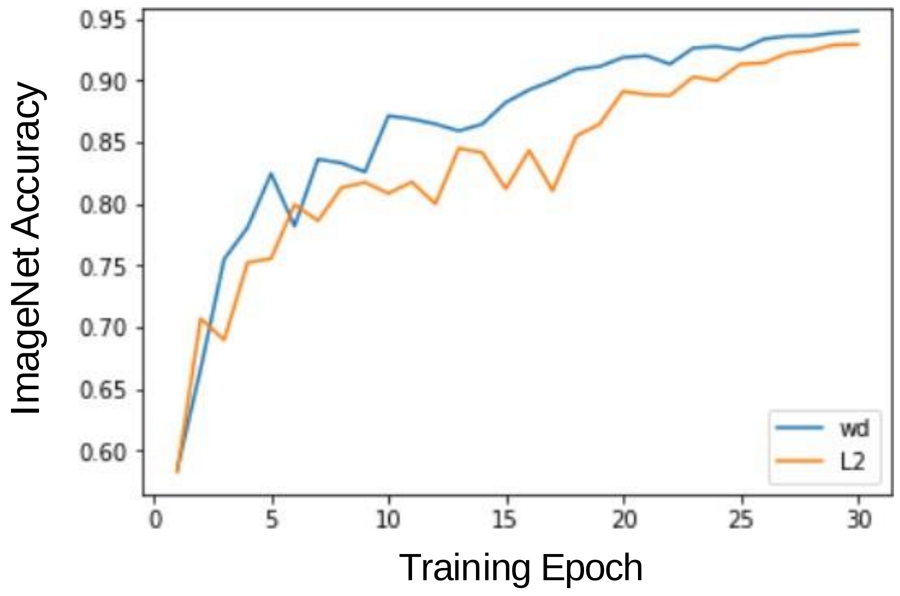
模型优化

■ AdamW

标准的 Adam 在这里加上 L2 Regularization

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

AdamW 在这里加上 Weight Decay



模型优化

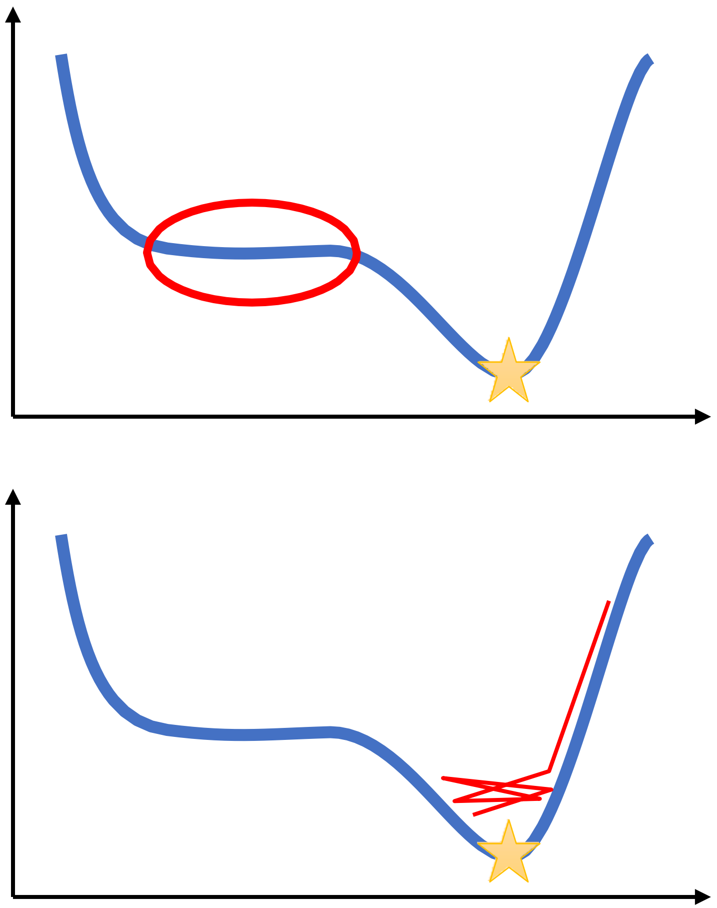
- **学习率 α** $$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

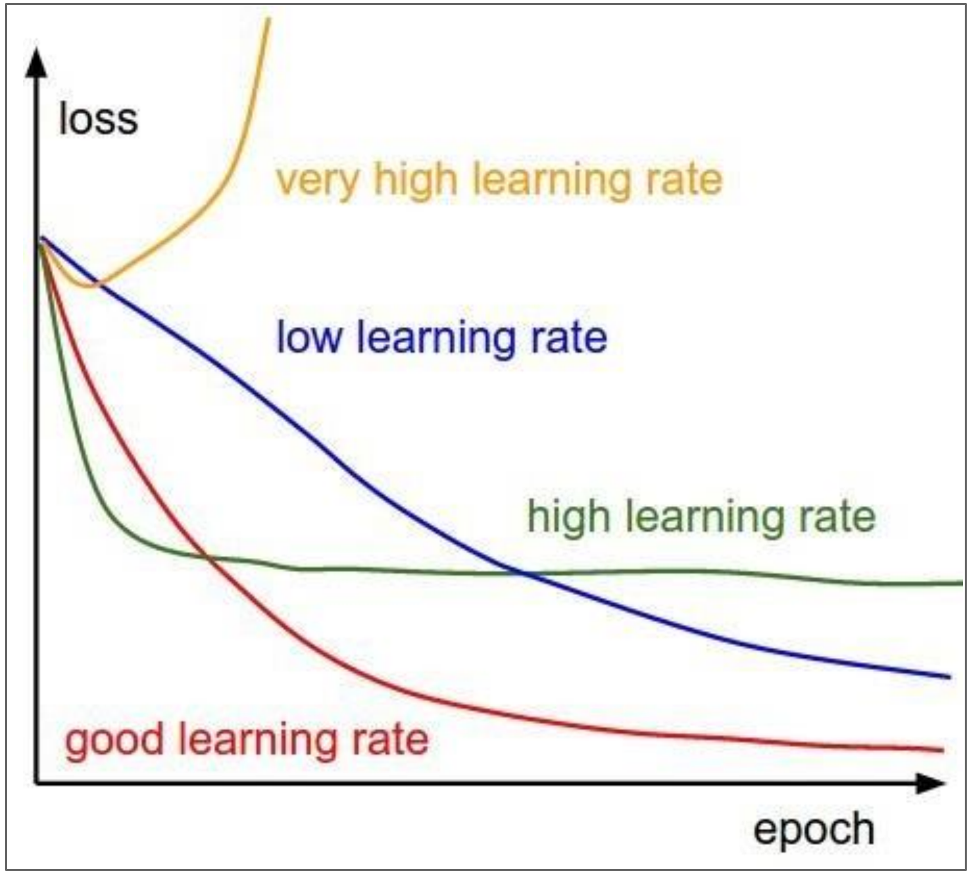
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

学习率 learning rate

模型优化

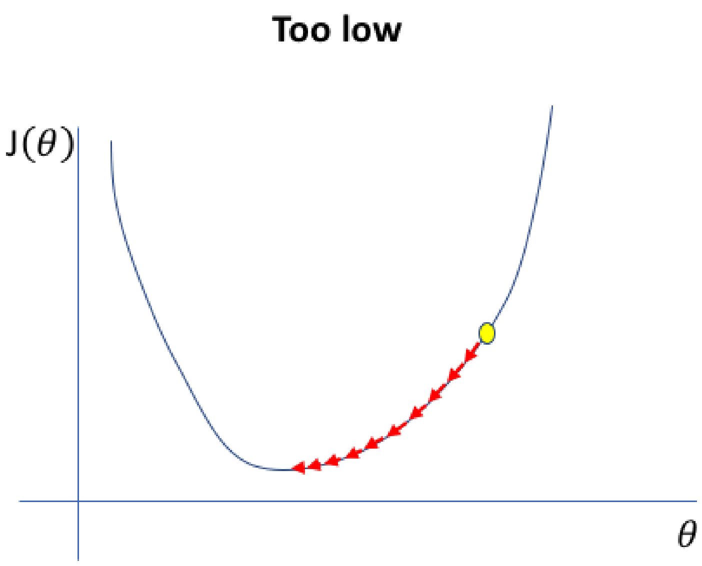
- **学习率 α** $$x_{t+1} = x_t - \alpha \nabla f(x_t)$$



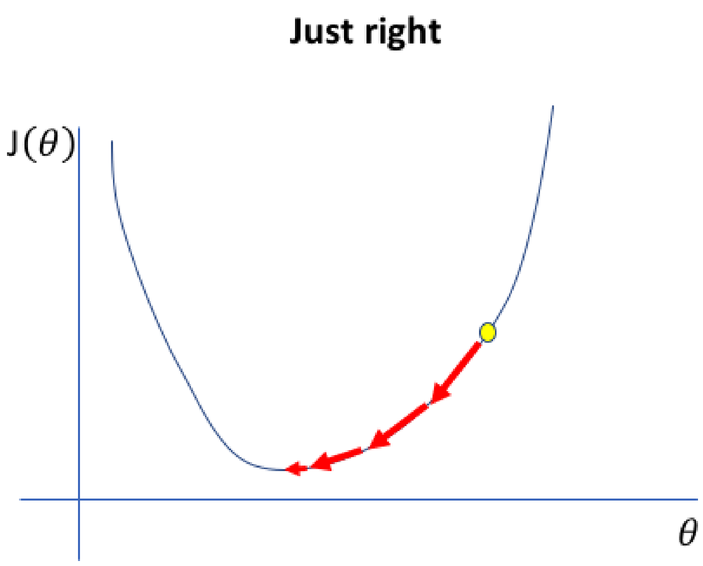


模型优化

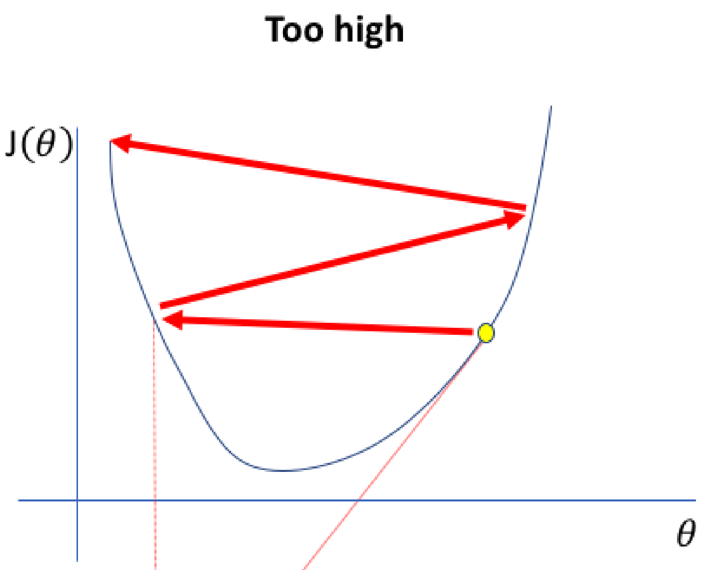
- 学习率 α $$x_{t+1} = x_t - \alpha \nabla f(x_t)$$



A small learning rate requires many updates before reaching the minimum point



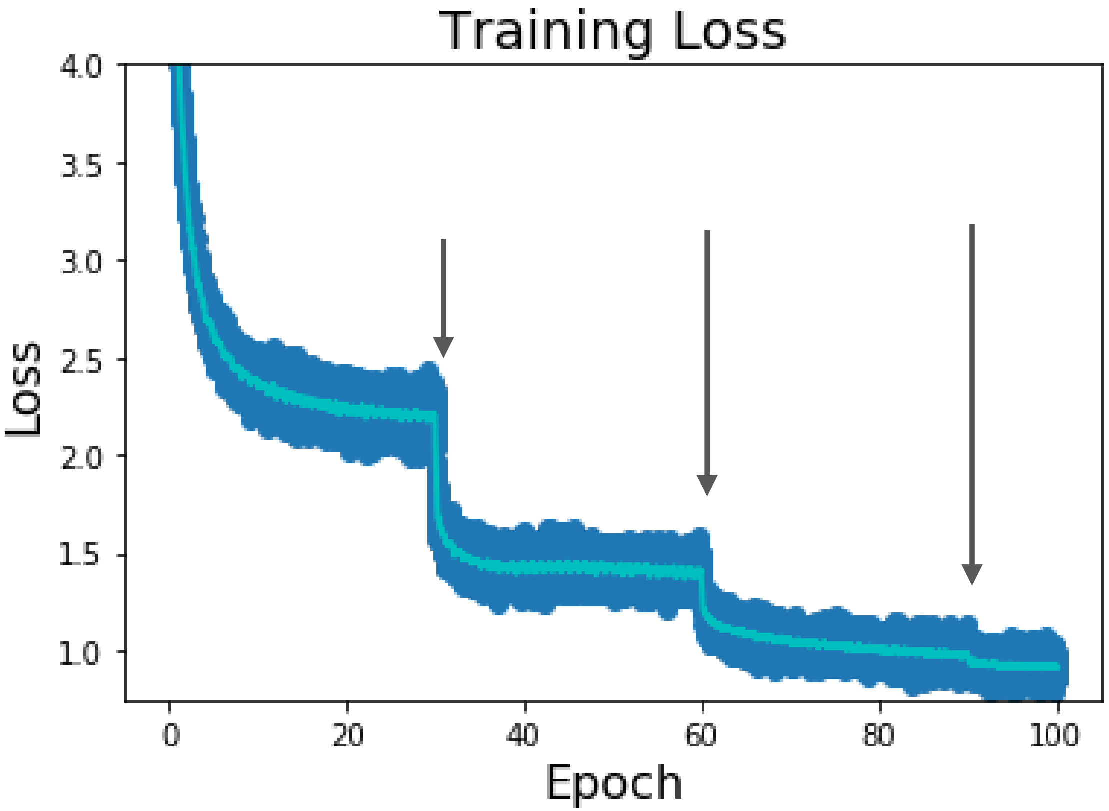
The optimal learning rate swiftly reaches the minimum point



Too large of a learning rate causes drastic updates which lead to divergent behaviors

模型优化

- 随时间改变的学习率
- Step: 训练固定的 epoch 之后，降低学习率
- ResNet 在第 30、60、90个训练 epoch 时，将 lr 乘以0.1

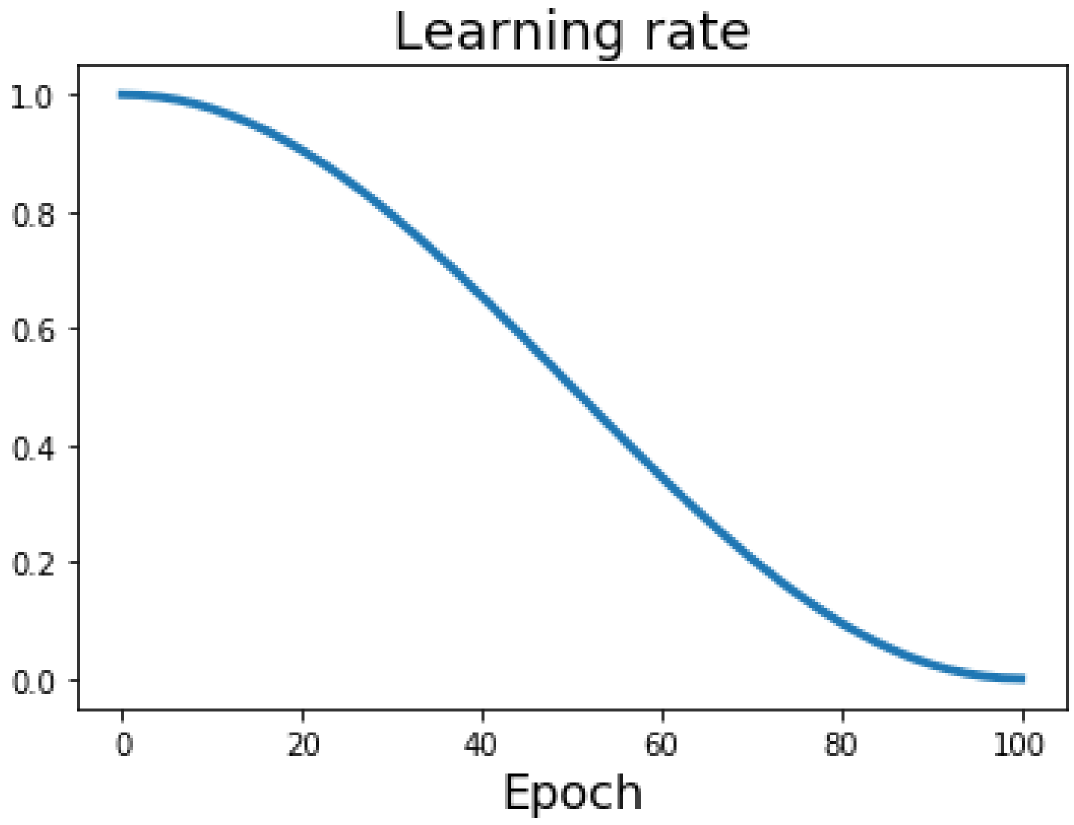


模型优化

- 随时间改变的学习率
- **Step**: 训练固定的 epoch 之后，降低学习率
- **Cosine**:

$$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

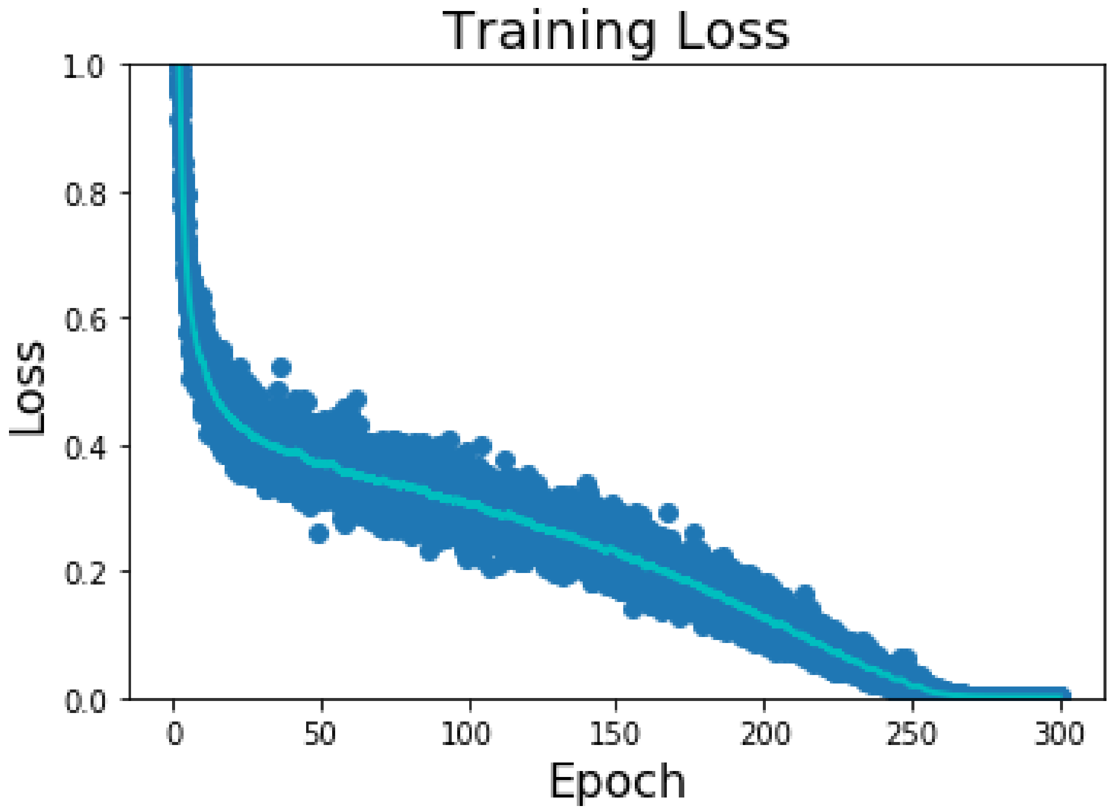
α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs



- 随时间改变的学习率
- **Step**: 训练固定的 epoch 之后，降低学习率
- **Cosine**:

$$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

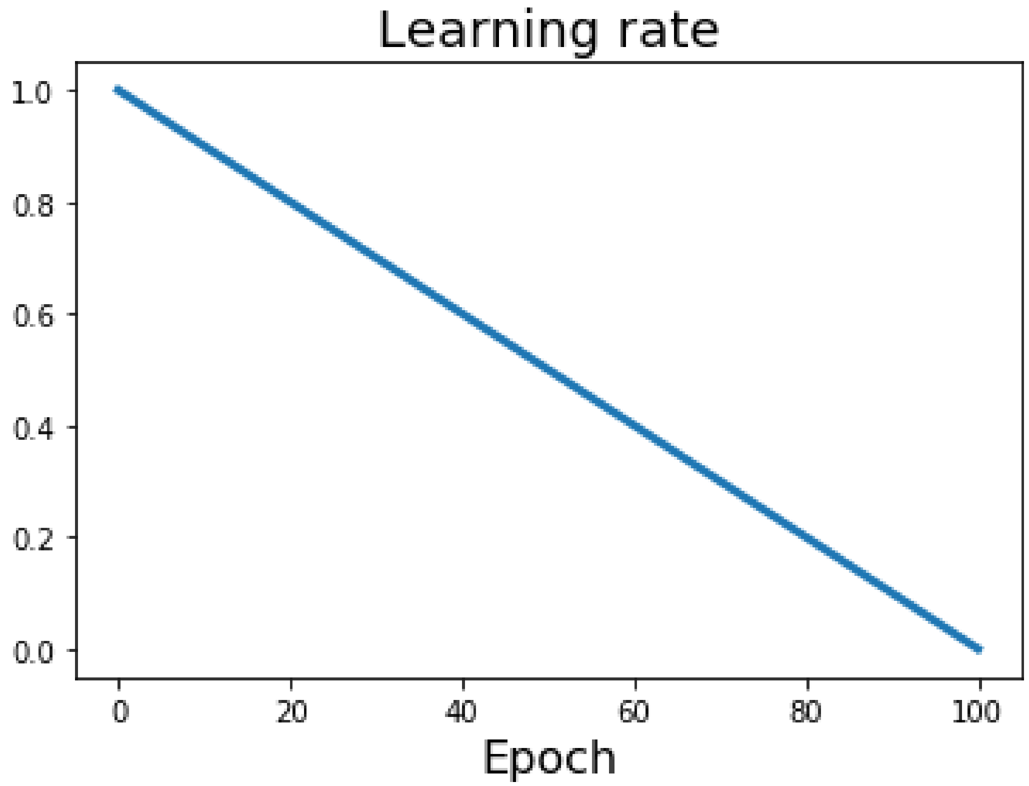


模型优化

- 随时间改变的学习率
- Step: 训练固定的 epoch 之后，降低学习率
- Cosine
- Linear:

$$\alpha_t = \alpha_0(1 - t/T)$$

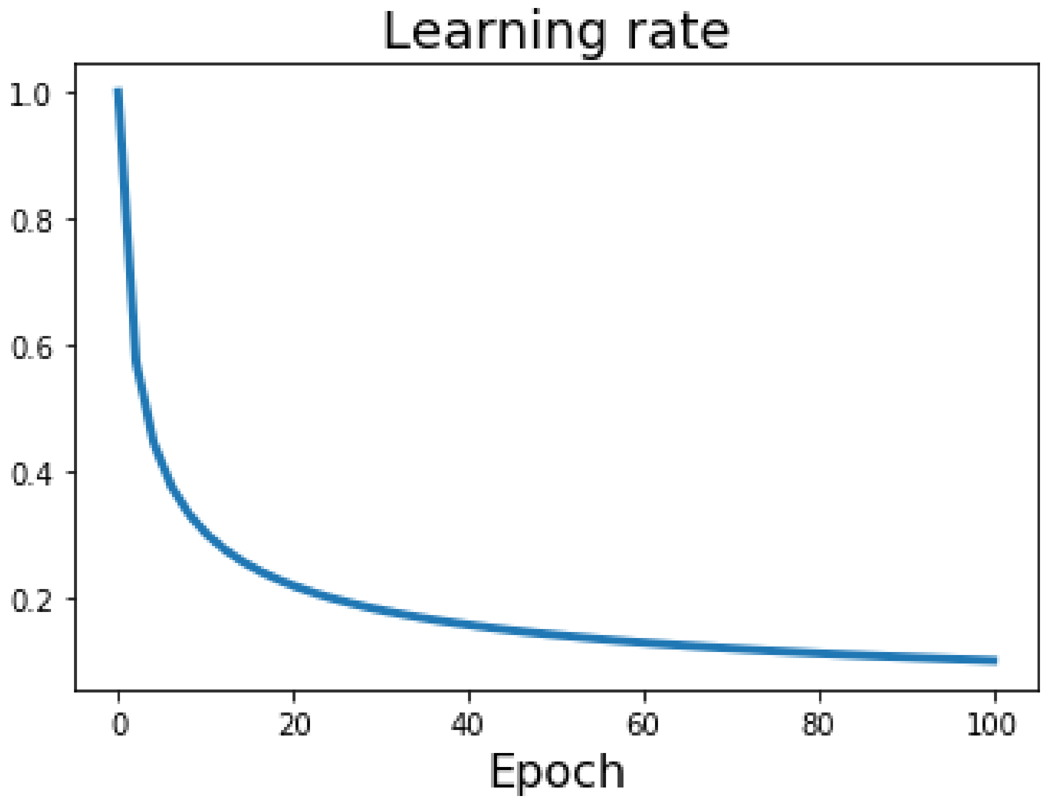
α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs



模型优化

- 随时间改变的学习率
- **Step**: 训练固定的 epoch 之后，降低学习率
- **Cosine**
- **Linear**
- **Inverse:** $\alpha_t = \alpha_0 / \sqrt{t}$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs



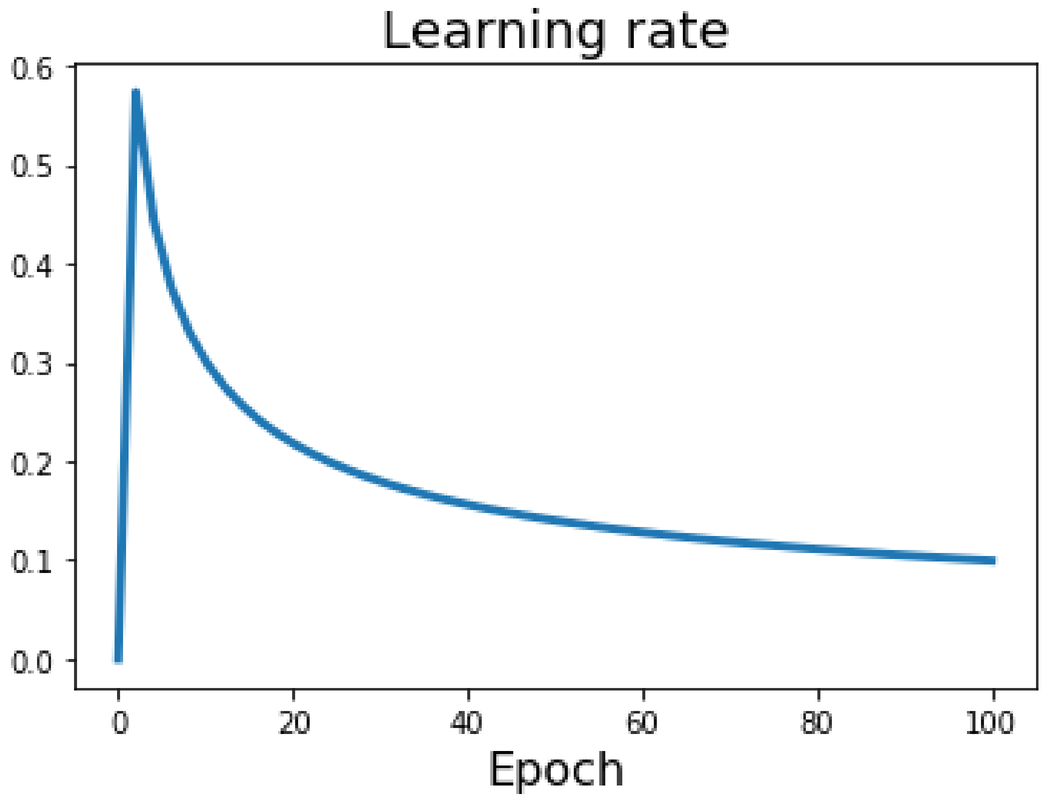
模型优化

- 随时间改变的学习率：

  Linear Warmup

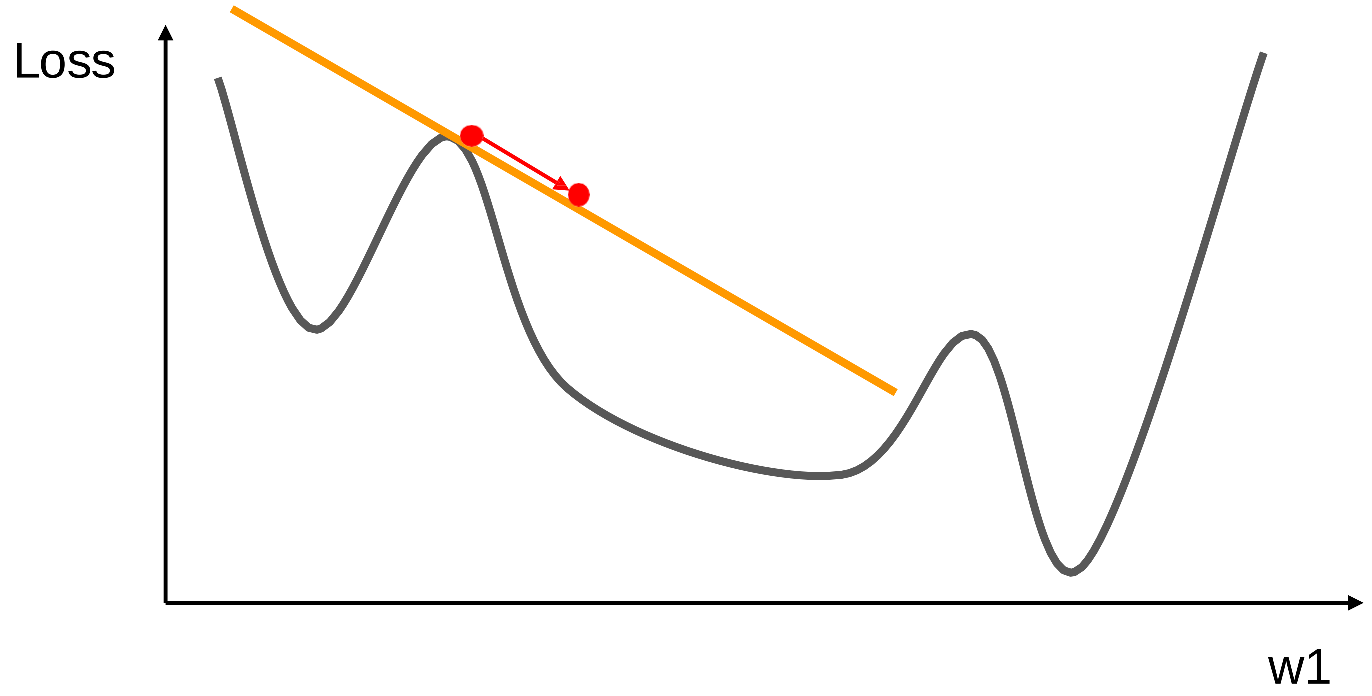
- 高初始学习率会使损失激增。前 5000 次迭代中，从 0 开始线性增加学习率可以防止这种情况

- 经验法则：如果将批处理大小增加N，则初始学习率也按N缩放



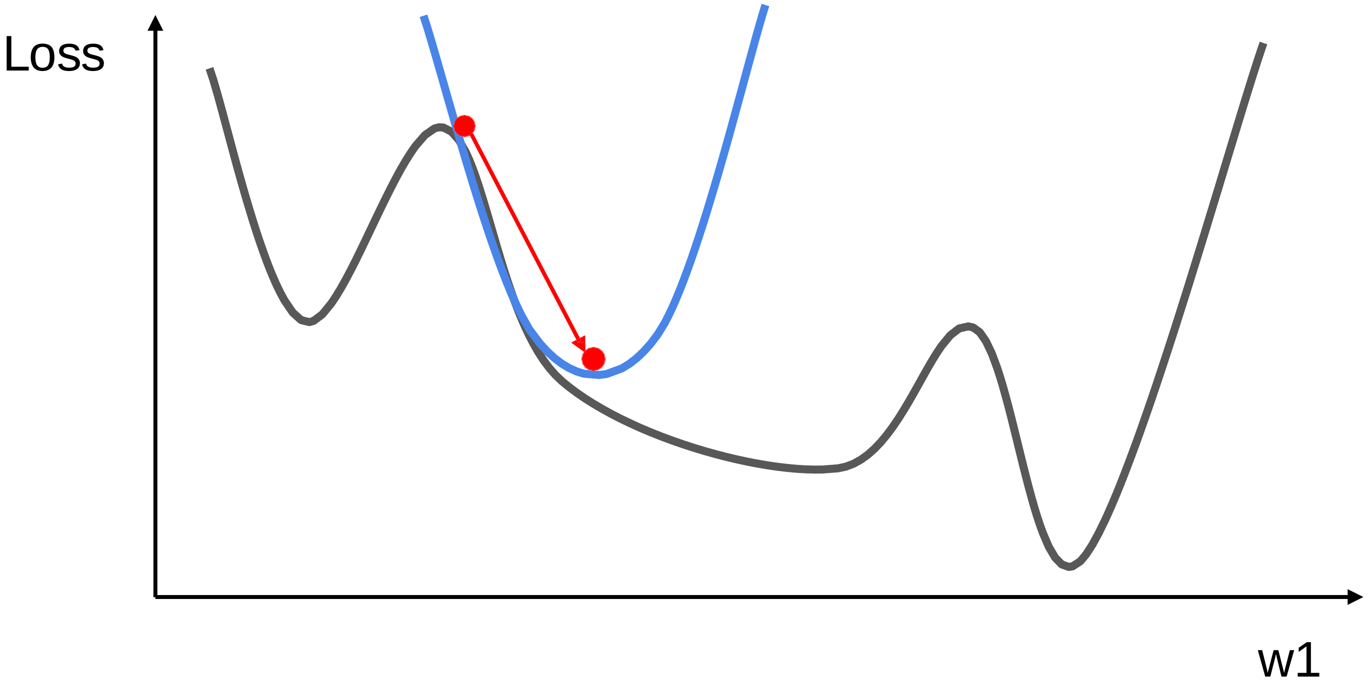
模型优化

- 一阶优化（First-Order Optimization）
  - 使用梯度进行线性近似 $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$
  - 每一步都沿着线性近似降低 loss



模型优化

- 二阶优化（Second-Order Optimization）
  - 使用梯度和 Hessian 进行二次近似
  - 每一步都逼近二次近似的最小值



模型优化

NANJING UNIVERSITY

二阶优化：牛顿法（Newton’s method)

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2$$

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta_0) + \underbrace{\nabla_\theta \mathcal{L}(\theta_0)}_{\text{gradient}}(\theta - \theta_0) + \frac{1}{2}(\theta - \theta_0)^T \underbrace{\nabla^2_\theta \mathcal{L}(\theta_0)}_{\text{Hessian}}(\theta - \theta_0)$$

$$\begin{bmatrix} \frac{d^2\mathcal{L}}{d\theta_1 d\theta_1} & \frac{d^2\mathcal{L}}{d\theta_1 d\theta_2} & \frac{d^2\mathcal{L}}{d\theta_1 d\theta_3} \\ \frac{d^2\mathcal{L}}{d\theta_2 d\theta_1} & \frac{d^2\mathcal{L}}{d\theta_2 d\theta_2} & \frac{d^2\mathcal{L}}{d\theta_2 d\theta_3} \\ \frac{d^2\mathcal{L}}{d\theta_3 d\theta_1} & \frac{d^2\mathcal{L}}{d\theta_3 d\theta_2} & \frac{d^2\mathcal{L}}{d\theta_3 d\theta_3} \end{bmatrix}$$

$$\theta^\star \leftarrow \theta_0 - (\nabla^2_\theta \mathcal{L}(\theta_0))^{-1} \nabla_\theta \mathcal{L}(\theta_0)$$

模型优化

- **一阶优化（First-Order Optimization）**

$$\text{gradient descent: } \theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathcal{L}(\theta_k)$$

$$\text{runtime?} \qquad \mathcal{O}(n)$$

- **二阶优化：牛顿法（Newton’s method）**

$$\theta^\star \leftarrow \theta_0 - (\nabla_\theta^2 \mathcal{L}(\theta_0))^{-1} \nabla_\theta \mathcal{L}(\theta_0)$$

$$\text{runtime?} \qquad \mathcal{O}(n^3)$$

- **二阶方法的计算代价太大，所以我们一般选择一阶方法**

模型优化

- 拟牛顿法（Quasi-Newton methods, e.g., BGFS）
  - 不求 Hessian 矩阵的逆（O(n^3)）
  - 求 Hessian 矩阵的逆的近似解（O(n^2)）
- L-BGFS（Limited memory BFGS）
  - 对 BGFS 的近似，用时间换空间
  - 不存储整个 Hessian 矩阵，存储向量序列，需要时再计算得到 Hessian 矩阵

模型优化

- 实战选择
  - 在许多情况下，Adam(W)是一个很好的默认选择；即使学习率保持不变，它通常也能正常工作
  - SGD+Momentum 有时优于 Adam，但可能需要对 LR 和 LR schedule policy 进行更多调整
  - 如果可以进行全批量更新，那么可以使用高阶优化方法

下节课展望

- 如何优化更复杂的函数?
- 目前关注的函数：线性函数

$$f = Wx$$

$$x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D}$$

下节课展望

- 如何优化更复杂的函数?
- 目前关注的函数：线性函数 $f = Wx$
- 下节课关注的函数：神经网络

$$f = W_2 \max(0, W_1 x)$$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

- 反向传播：优化神经网络