



《人工智能数学原理与算法》
第 3 章：神经网络基础

3.2 神经网络优化

连德富

liandefu@ustc.edu.cn

目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

神经网络训练优化要点与技巧

06

参数初始化

目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

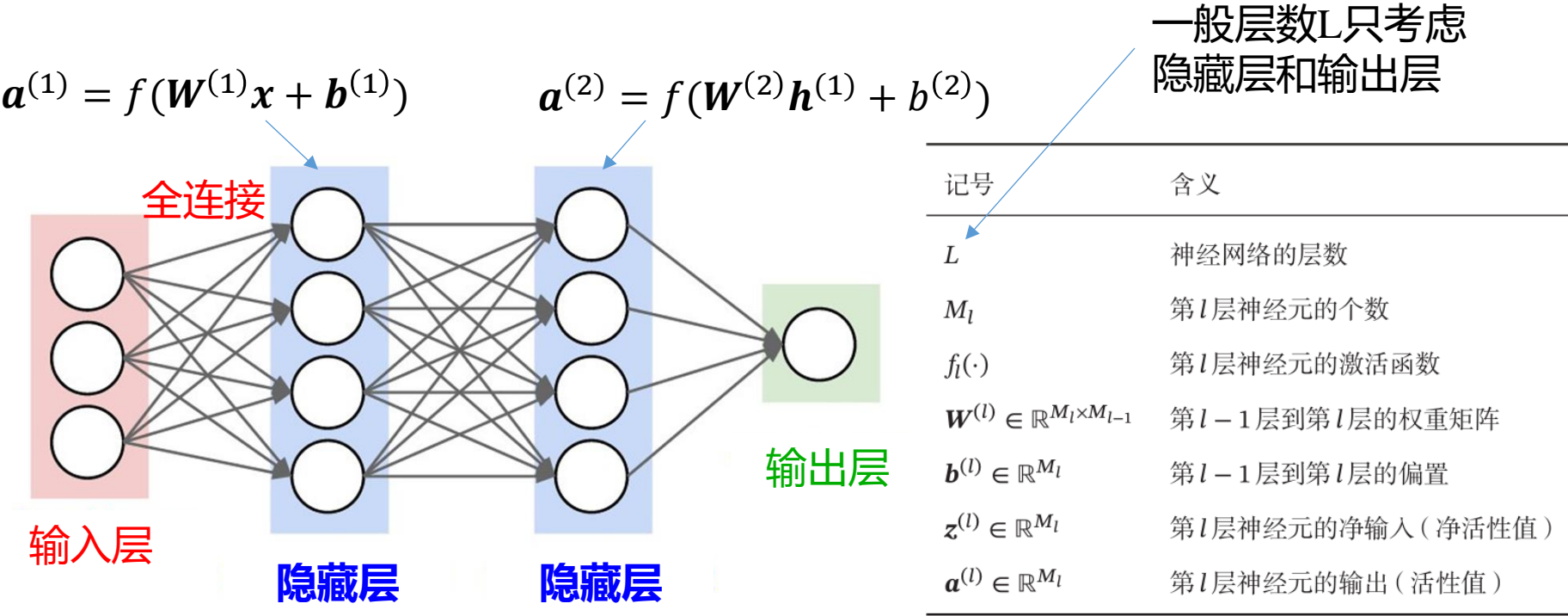
神经网络训练优化要点与技巧

06

参数初始化

前馈神经网络（多层感知机）

- 各神经元分别属于不同的层，层内无连接；相邻两层之间的神经元全部两两连接
- 整个网络中无反馈，信号从输入层向输出层单向传播



梯度下降 (Gradient Descent)

□ 给定训练集为 $D = \{(x^{(n)}, y^{(n)})\}_{n=1}^N$ ，将每个样本 $x^{(n)}$ 输入给前馈神经网络，得到网络输出为 $\hat{y}^{(n)}$ ，其在数据集 D 上的结构化风险函数为：

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(y^{(n)}, \hat{y}^{(n)}) + \frac{1}{2} \lambda \|\mathbf{W}\|_F^2$$

□ 梯度下降 (Gradient Descent, GD)

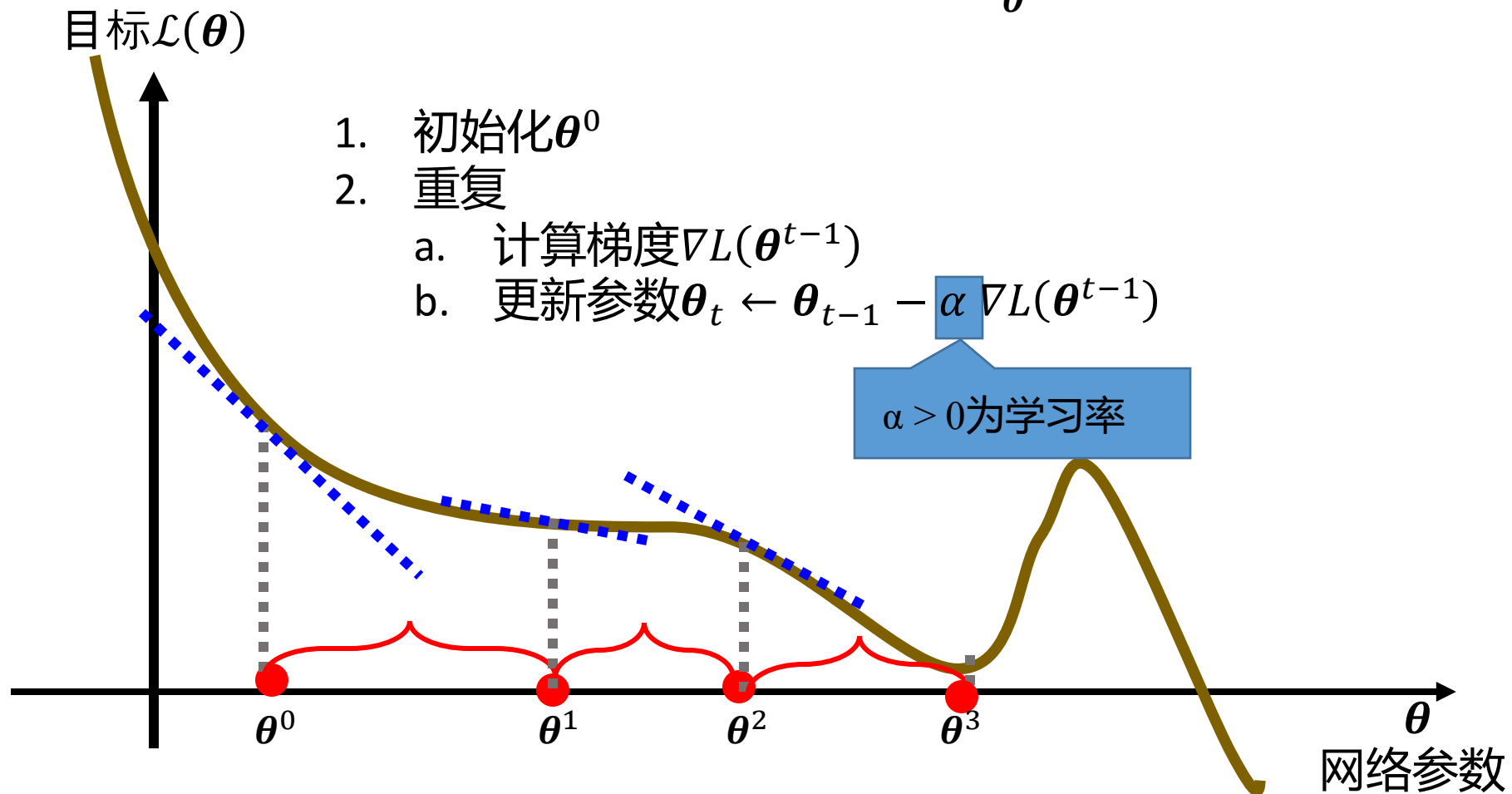
$$\theta^{(l+1)} \leftarrow \theta^{(l)} - \alpha \frac{\partial \mathcal{L}(\theta)}{\partial \theta^{(l)}}$$

↑
学习率

思考：为什么梯度下降能保证损失是非递增的？有没有什么前提条件？

梯度下降

优化算法解决 $\theta^* = \arg \min_{\theta} \mathcal{L}(\theta)$



随机梯度下降 (SGD)

□ 由于数据集可能很大，无法全部放入内存计算梯度

□ 一般采用小批量随机梯度下降法，每次从数据集中采样一部分样本（称为batch），计算batch上的梯度，并进行参数更新。

□ 给定训练集为 $D = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$ ，每次采样B个样本

梯度下降

$$\mathcal{L}_D(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2} \lambda \|\mathbf{W}\|_F^2$$

$$\frac{\partial \mathcal{L}_D(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(l)}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{W}^{(l)}}$$

小批量随机梯度下降

$$\mathcal{L}_B(\mathbf{W}, \mathbf{b}) = \frac{1}{B} \sum_{i=1}^B \mathcal{L}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) + \frac{1}{2} \lambda \|\mathbf{W}\|_F^2$$

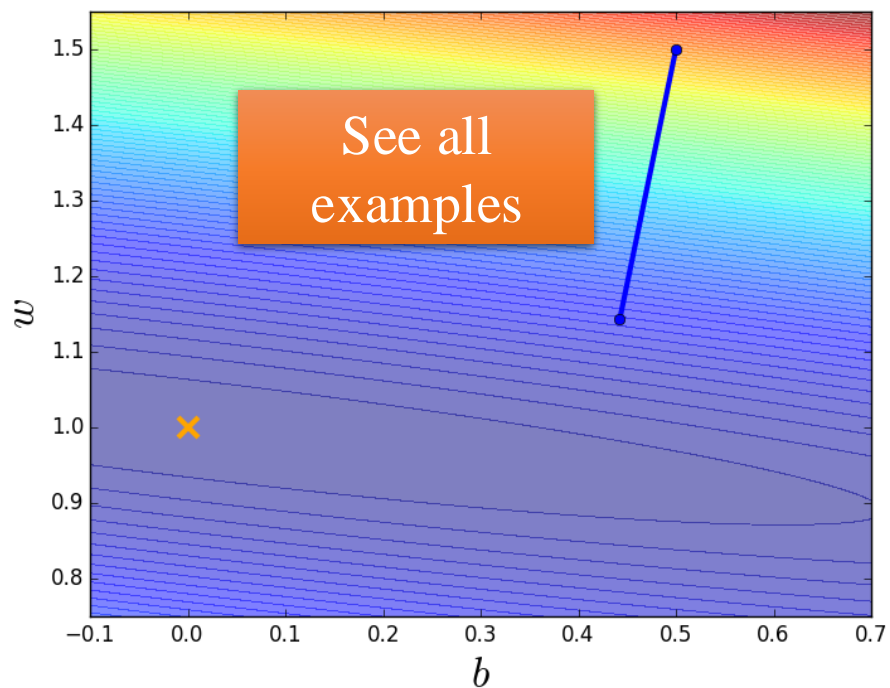
$$\frac{\partial \mathcal{L}_B(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(l)}} = \frac{1}{B} \sum_{i=1}^B \frac{\partial \mathcal{L}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})}{\partial \mathbf{W}^{(l)}}$$

思考：数据集规模、模型大小以及GPU内存大小之间的关系

随机梯度下降 vs. 梯度下降

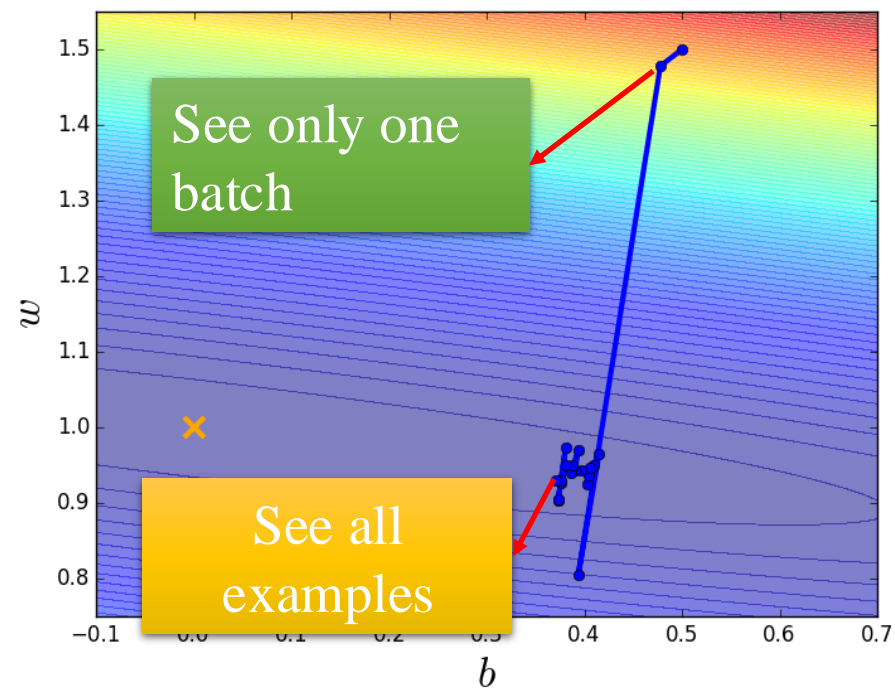
梯度下降

用所有样本对参数更新



随机梯度下降

每个批都对参数更新
更新次数更多



神经网络随机梯度下降的终止条件

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

1 随机初始化 θ ;

2 **repeat**

3 对训练集 \mathcal{D} 中的样本随机重排序;

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

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

 // 更新参数

6 $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta; \mathbf{x}^{(n)}, y^{(n)})}{\partial \theta}$;

7 **end**

8 **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

目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

神经网络训练优化要点与技巧

06

参数初始化

如何计算梯度？

□神经网络为一个复杂的复合函数

$$y = f^4(f^3(f^2(f^1(x, \theta^{(1)}), \theta^{(2)}), \theta^{(3)}), \theta^{(4)})$$

链式求导法则



$$\frac{\partial y}{\partial \theta^{(1)}} = \frac{\partial f^1}{\partial \theta^{(1)}} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial y}{\partial f^3}$$

$$\frac{\partial y}{\partial \theta^{(2)}} = \frac{\partial f^2}{\partial \theta^{(2)}} \frac{\partial f^3}{\partial f^2} \frac{\partial y}{\partial f^3}$$

$$\frac{\partial y}{\partial \theta^{(3)}} = \frac{\partial f^3}{\partial \theta^{(3)}} \frac{\partial y}{\partial f^3}$$

$$\frac{\partial y}{\partial \theta^{(4)}} = \frac{\partial f^4}{\partial \theta^{(4)}}$$

$\frac{\partial f^l}{\partial f^{l-1}}$ 在计算 y 对于每一层的参数的导数时重复出现，可以**存储复用**。

□反向传播算法

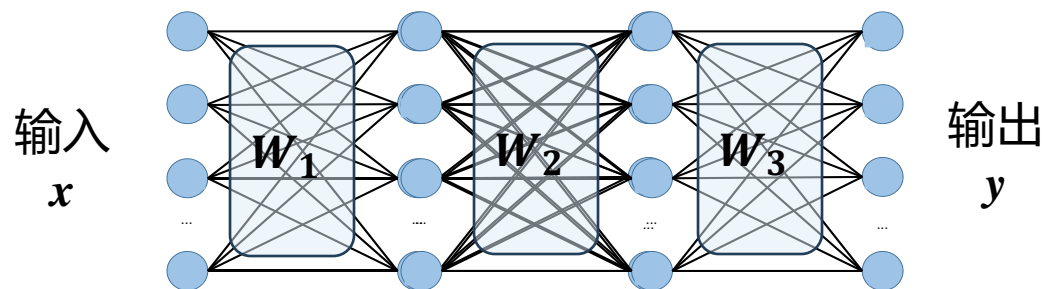
根据前馈网络的特点而设计的高效方法:由最深层开始向浅层依次根据链式法则计算导数值。

□一个更加通用的计算方法

自动微分 (Automatic Differentiation, AD)

如何计算梯度?

神经网络为一个复杂的复合函数



$$y = \sigma(W_3 \sigma(W_2 \sigma(W_1 x)))$$

(σ 为激活函数, 不考虑偏置项 b)
我们定义每一层“激活”:

$$\begin{aligned} z_1 &= W_1 x; a_1 = \sigma(z_1) \\ z_2 &= W_2 a_1; a_2 = \sigma(z_2) \\ z_3 &= W_3 a_2; a_3 = \sigma(z_3) = y \end{aligned}$$

计算梯度 $\frac{\partial y}{\partial W_i}$

链式求导法则

$$\begin{aligned} \frac{\partial y}{\partial W_3} &= \frac{\partial y}{\partial z_3} \frac{\partial z_3}{\partial W_3} \\ \frac{\partial y}{\partial W_2} &= \frac{\partial y}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial W_2} \\ \frac{\partial y}{\partial W_1} &= \frac{\partial y}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial W_1} \end{aligned}$$

$\frac{\partial y}{\partial z_3}$, $\frac{\partial a_2}{\partial z_2}$ 等在计算 y 对于第二层和第一层的参数的导数时重复出现, 可以**存储复用**。

观察1: 计算梯度过程中需要计算神经网络各层激活值。(前向传播)

观察2: 计算梯度过程中的部分中间结果 (如 $\frac{\partial a_3}{\partial z_3}$, $\frac{\partial a_2}{\partial z_2}$) 可以复用。(后向传播)

反向传播算法

$f_2(\cdot)$: 输出层激活函数

$w_{hj}^{(2)}$: 隐层与输出层神经元之间的连接权重

$b_j^{(2)}$: 输出层神经元的偏置

$f_1(\cdot)$: 隐层激活函数

$w_{ih}^{(1)}$: 输入层与隐层神经元之间的连接权重

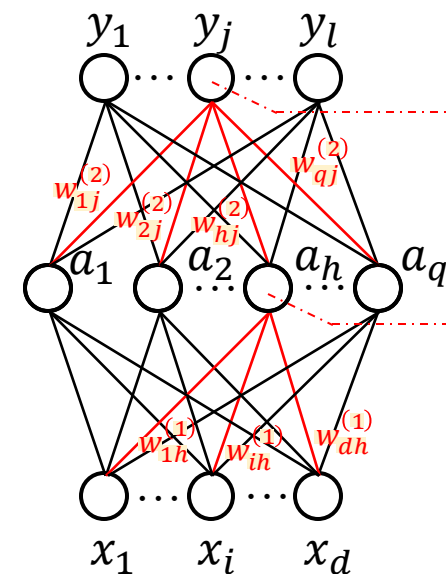
$b_h^{(1)}$: 隐层神经元的偏置

网络中需要 $(d + l + 1)q + l$ 个参数
需要优化

BP是一个迭代学习算法, 在迭代的每一轮中采用广义的感知机学习规则对参数进行更新估计, 任意的参数 v 的更新估计式为

$$v \leftarrow v + \Delta v$$

输出 l 维实值向量 \mathbf{y}

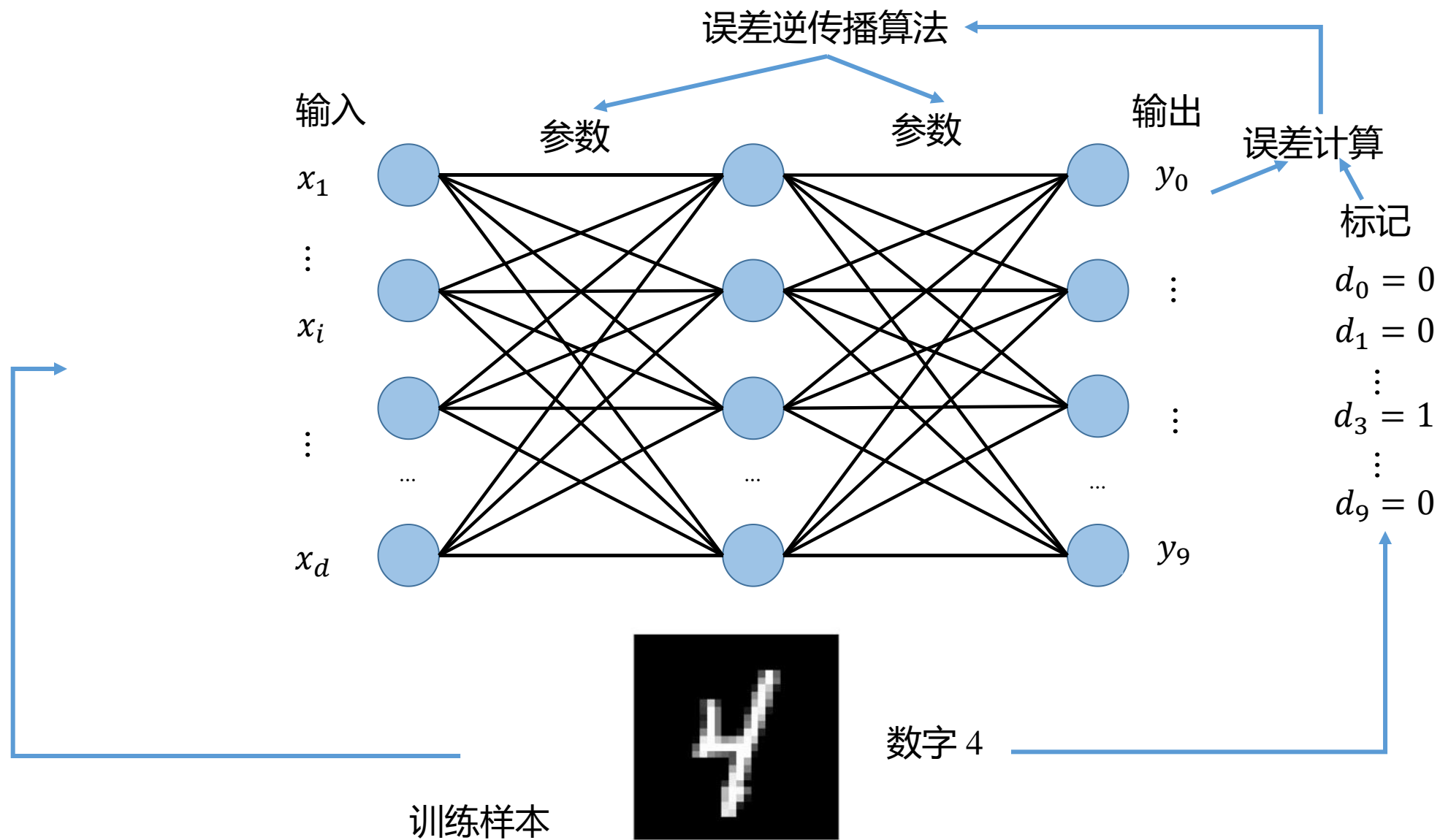


第 j 个输出神经元:
输入: $z_j^{(2)} = \sum_{h=1}^q w_{hj}^{(2)} a_h + b_j^{(2)}$
输出: $y_j = f_2(z_j^{(2)})$

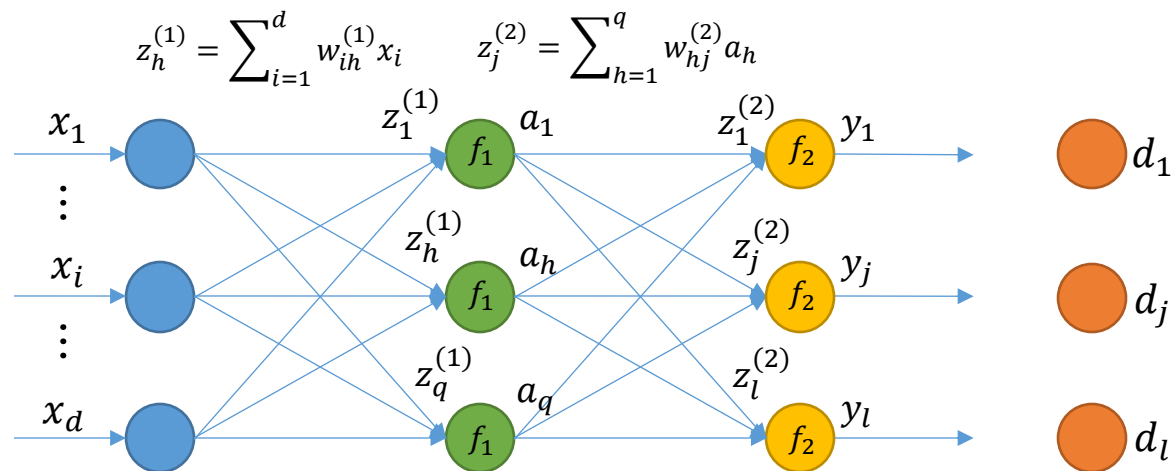
第 h 个隐层神经元:
输入: $z_h^{(1)} = \sum_{i=1}^d w_{ih}^{(1)} x_i + b_h^{(1)}$
输出: $a_h = f_1(z_h^{(1)})$

输入示例 \mathbf{x} 由 d 个属性描述

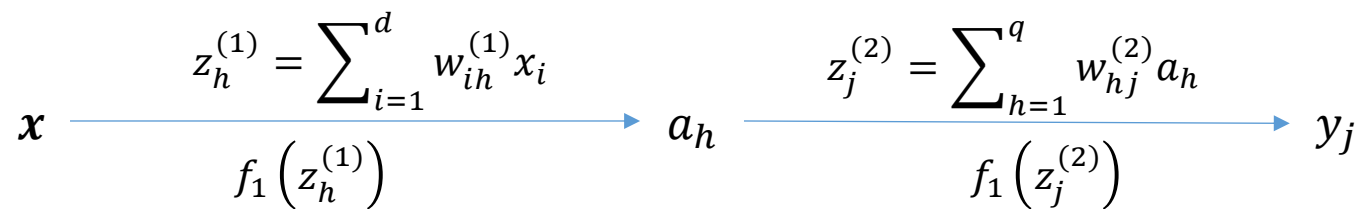
反向传播算法



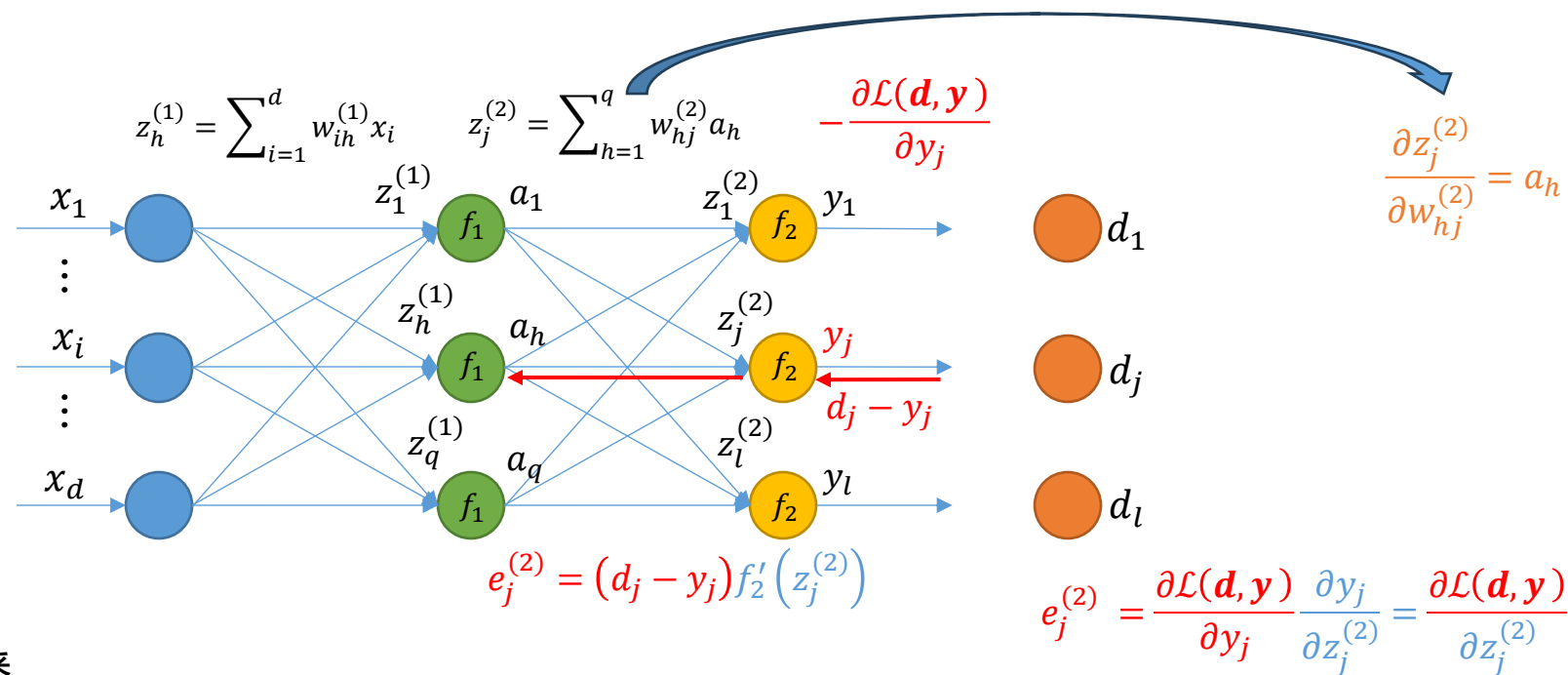
反向传播算法—前向



前向预测



反向传播算法—后向

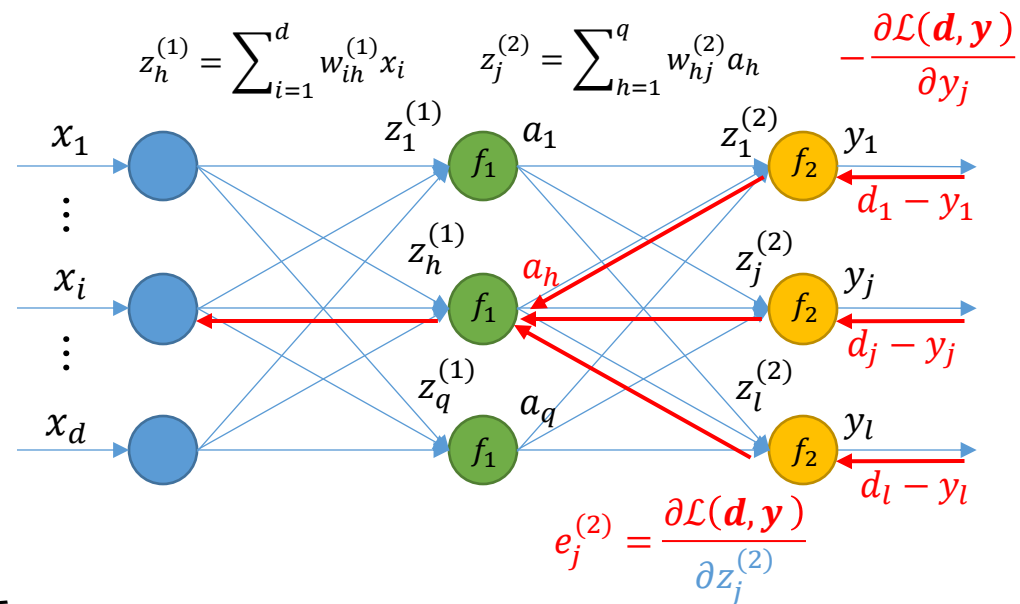


后向传播

$$w_{hj}^{(2)} = w_{hj}^{(2)} + \Delta w_{hj}^{(2)} \quad \Delta w_{hj}^{(2)} = \eta e_j^{(2)} a_h = \eta \text{Error}_j \text{Output}_h \quad \mathcal{L}(\mathbf{d}, \mathbf{y}) = \frac{1}{2} \sum_{j=1}^l (y_j - d_j)^2$$

$$\Delta w_{hj}^{(2)} = -\eta \frac{\partial \mathcal{L}(\mathbf{d}, \mathbf{y})}{\partial w_{hj}^{(2)}} = -\eta \frac{\partial \mathcal{L}(\mathbf{d}, \mathbf{y})}{\partial y_j} \frac{\partial y_j}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial w_{hj}^{(2)}} = \eta (d_j - y_j) f_2'(z_j^{(2)}) a_h = \eta e_j^{(2)} a_h$$

反向传播算法—后向



d_1

d_j

d_l

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{d}, \mathbf{y})}{\partial a_h} &= \sum_j \frac{\partial \mathcal{L}(\mathbf{d}, \mathbf{y})}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial a_h} \\ &= \sum_j e_j^{(2)} \frac{\partial z_j^{(2)}}{\partial a_h} \\ &= \sum_j e_j^{(2)} w_{hj}^{(2)} \end{aligned}$$

后向传播

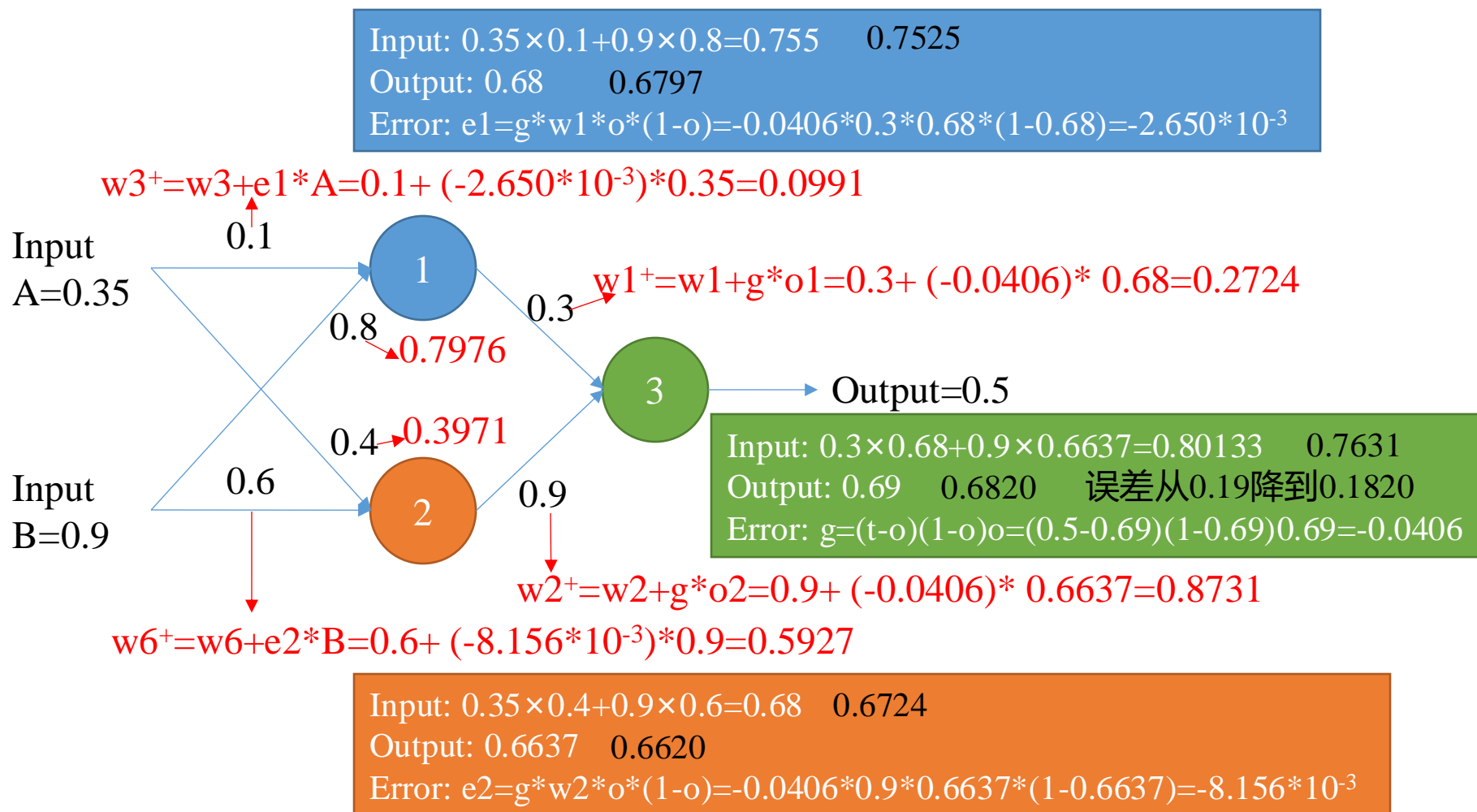
$$w_{ih}^{(1)} = w_{ih}^{(1)} + \Delta w_{ih}^{(1)} \quad \Delta w_{ih}^{(1)} = \eta e_h^{(1)} x_i = \eta \text{Error}_h \text{Output}_i \quad \mathcal{L}(\mathbf{d}, \mathbf{y}) = \frac{1}{2} \sum_{j=1}^l (y_j - d_j)^2$$

$$\Delta w_{ih}^{(1)} = -\eta \frac{\partial \mathcal{L}(\mathbf{d}, \mathbf{y})}{\partial w_{ih}^{(1)}} = -\eta \frac{\partial \mathcal{L}(\mathbf{d}, \mathbf{y})}{\partial a_h} \frac{\partial a_h}{\partial z_h^{(1)}} \frac{\partial z_h^{(1)}}{\partial w_{ih}^{(1)}} = \eta \sum_j e_j^{(2)} w_{hj}^{(2)} f_1'(z_h^{(1)}) x_i = \eta e_h^{(1)} x_i$$

$$e_h^{(1)} = f_1'(z_h^{(1)}) \sum_j e_j^{(2)} w_{hj}^{(2)}$$

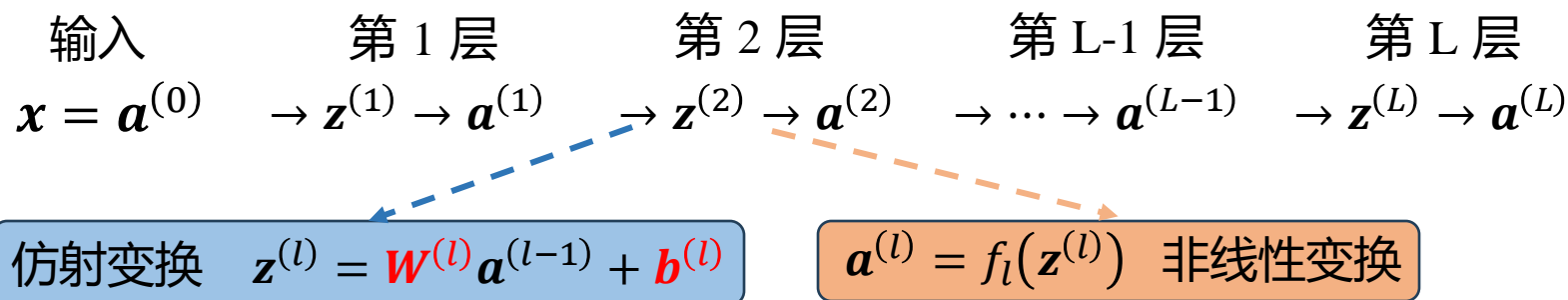
反向传播算法：简单例子

□考虑如下简单网络 假设激活函数为Sigmoid函数



后向传播算法 — 一般情形

前向计算激活过程



后向计算梯度过程

链式法则 $\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}_{ij}^{(l)}} = \left\langle \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}, \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}_{ij}^{(l)}} \right\rangle$

$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \triangleq \mathbf{e}^{(l)}$ 误差项

$\frac{\partial \mathbf{z}_k^{(l)}}{\partial \mathbf{w}_{ij}^{(l)}} = \delta(k=i) \mathbf{a}_j^{(l-1)}$

$\mathbf{W}^{(l)}$ 的第 k 行 $\mathbf{z}_k^{(l)} = \langle \mathbf{w}_k^{(l)}, \mathbf{a}^{(l-1)} \rangle + b_k^{(l)}$

链式法则 $\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \left\langle \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}, \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \right\rangle$ $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{1}_{m^{(l)}} \quad \text{单位阵}$

The diagram illustrates the backward pass of a neural network. It shows the flow of gradients from the output back to the input. The chain rule is used to calculate the gradient of the loss with respect to the weights $\mathbf{w}_{ij}^{(l)}$ and biases $\mathbf{b}^{(l)}$. The error term $\mathbf{e}^{(l)}$ is defined as $\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$. The gradient of the pre-activation $\mathbf{z}_k^{(l)}$ with respect to the weight $\mathbf{w}_{ij}^{(l)}$ is given by $\frac{\partial \mathbf{z}_k^{(l)}}{\partial \mathbf{w}_{ij}^{(l)}} = \delta(k=i) \mathbf{a}_j^{(l-1)}$. The gradient of the pre-activation $\mathbf{z}^{(l)}$ with respect to the bias $\mathbf{b}^{(l)}$ is given by $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{1}_{m^{(l)}}$, where $\mathbf{1}_{m^{(l)}}$ is the identity matrix.

后向传播算法 — 一般情形

前向计算激活过程

$$x = a^{(0)} \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow a^{(2)} \rightarrow \dots \rightarrow a^{(L-1)} \rightarrow z^{(L)} \rightarrow a^{(L)}$$

后向计算梯度过程

$$z^{(l+1)} = W^{(l+1)} a^{(l)} + b^{(l+1)} \quad a^{(l)} = f_l(z^{(l)}) \quad \Rightarrow \quad a_k = f_l(z_k^{(l)})$$

误差项分解 $e^{(l)} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z^{(l)}}$

$$\frac{\partial z^{(l+1)}}{\partial a^{(l)}} = W^{(l+1)}$$

链式求导法则

$$= \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}}$$

$$\frac{\partial a^{(l)}}{\partial z^{(l)}} = \frac{\partial f_l(z^{(l)})}{\partial z^{(l)}} = \text{diag}(\nabla f_l(z^{(l)}))$$

$$= e^{(l+1)} W^{(l+1)} \text{diag}(\nabla f_l(z^{(l)}))$$

$$= \nabla f_l(z^{(l)}) \odot (e^{(l+1)} W^{(l+1)})$$

观察: $e^{(l+1)}$ 可以用来计算 $e^{(l)}$

后向传播算法 — 一般情形

回过头来

$$\frac{\partial z_k^{(l)}}{\partial w_{ij}^{(l)}} = \delta(k = i) a_j^{(l-1)}$$



$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \langle \mathbf{e}^{(l)}, \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \rangle = e_i^{(l)} a_j^{(l-1)} \quad \Rightarrow \quad \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}^{(l)}} = \mathbf{e}^{(l)} (\mathbf{a}^{(l-1)})^\top$$

后向传播算法计算参数梯度的核心公式组

$$\mathbf{e}^{(l)} = \nabla f_l(\mathbf{z}^{(l)}) \odot (\mathbf{e}^{(l+1)} \mathbf{W}^{(l+1)})$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}^{(l)}} = \mathbf{e}^{(l)} (\mathbf{a}^{(l-1)})^\top$$

观察: $\mathbf{e}^{(l+1)}$ 可以用来计算 $\mathbf{e}^{(l)}$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \langle \mathbf{e}^{(l)}, \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \rangle = \mathbf{e}^{(l)}$$

反向传播算法

□前馈神经网络的训练过程可以分为以下步骤

数据准备：从训练集中随机采样 k 个样本，作为一个训练batch

前向计算：从第一层开始计算每一层的状态和激活值，直到最后一层

反向计算：从最后一层开始计算每一层的参数的偏导数

更新参数：使用合适的学习率更新每一层的参数

目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

神经网络训练优化要点与技巧

06

参数初始化

深度学习的三个步骤



深度学习就是这么简单

<https://keras.io/examples/>

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

loss = model.evaluate(X_test, Y_test, batch_size=32)
```

Pytorch快速入门

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```
model = NeuralNet(input_size, hidden_size, num_classes).to(device)
```

Loss and optimizer

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Train the model

```
total_step = len(train_loader)
```

```
for epoch in range(num_epochs):
```

```
    for i, (images, labels) in enumerate(train_loader):
```

Move tensors to the configured device

```
        images = images.reshape(-1, 28*28).to(device)
```

```
        labels = labels.to(device)
```

Forward pass

```
        outputs = model(images)
```

```
        loss = criterion(outputs, labels)
```

Backward and optimize

```
        optimizer.zero_grad()
```

```
        loss.backward()
```

```
        optimizer.step()
```

目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

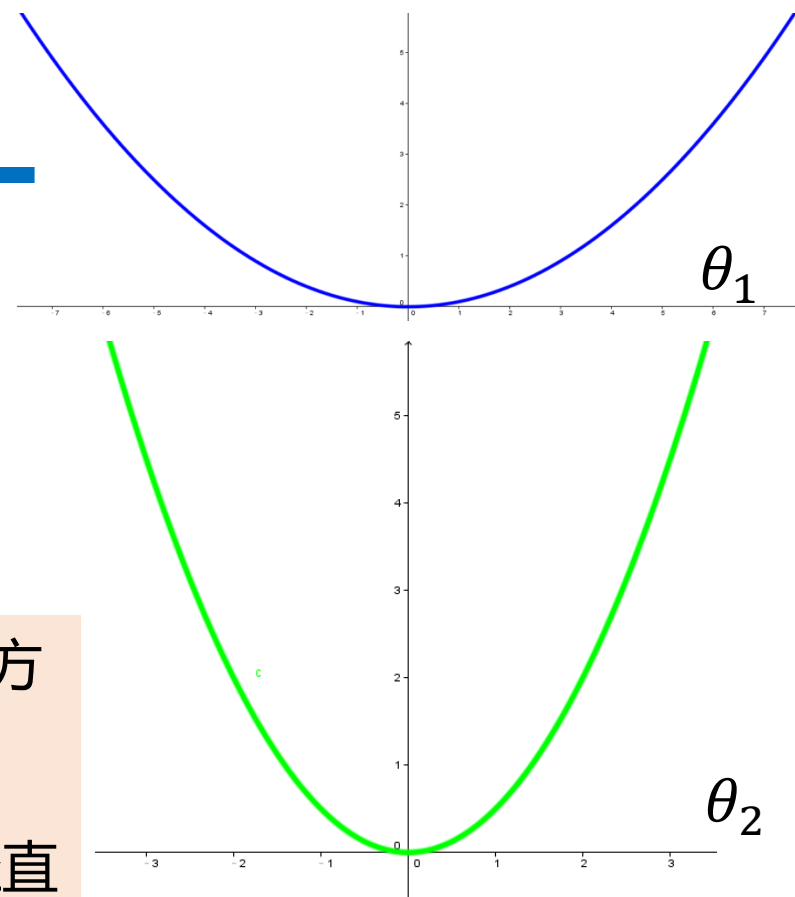
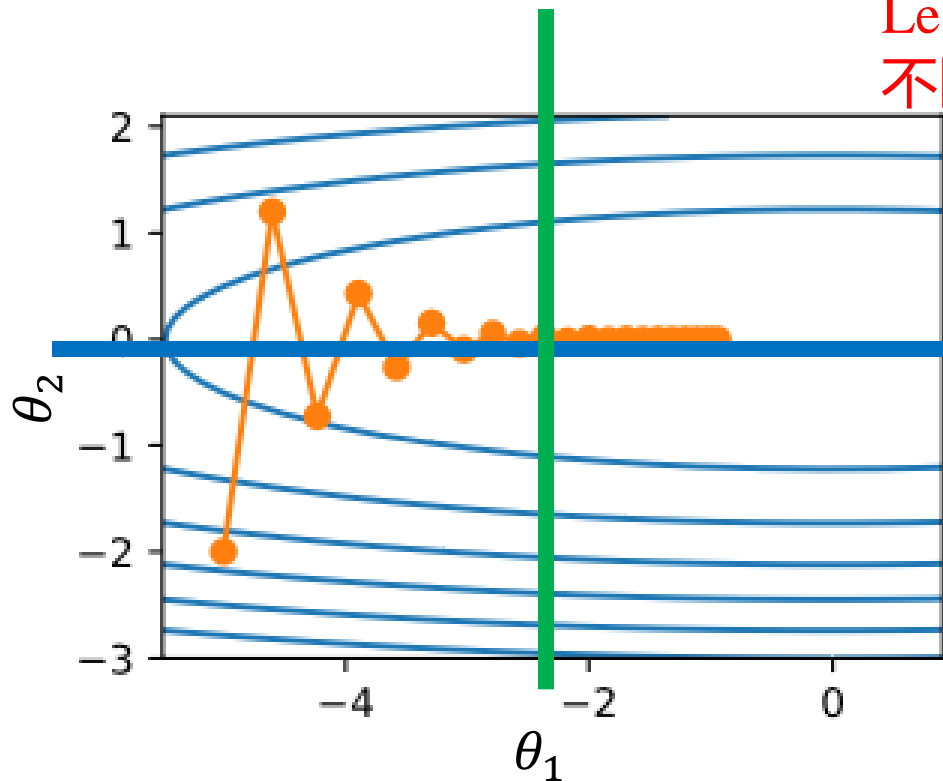
神经网络训练优化要点与技巧

06

参数初始化

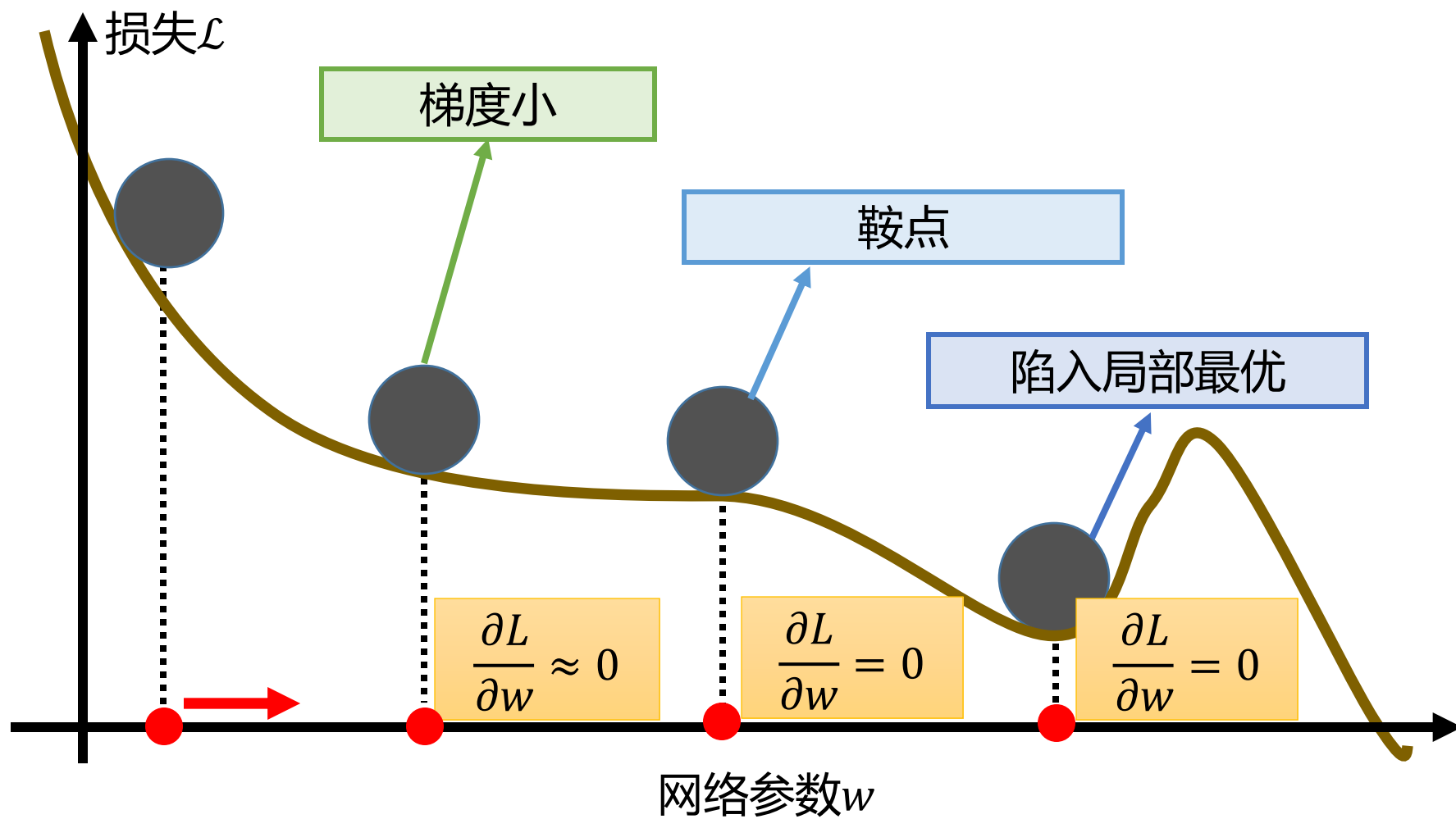
随机梯度下降的问题

Learning rate cannot be one-size-fits-all
不同参数不同学习率

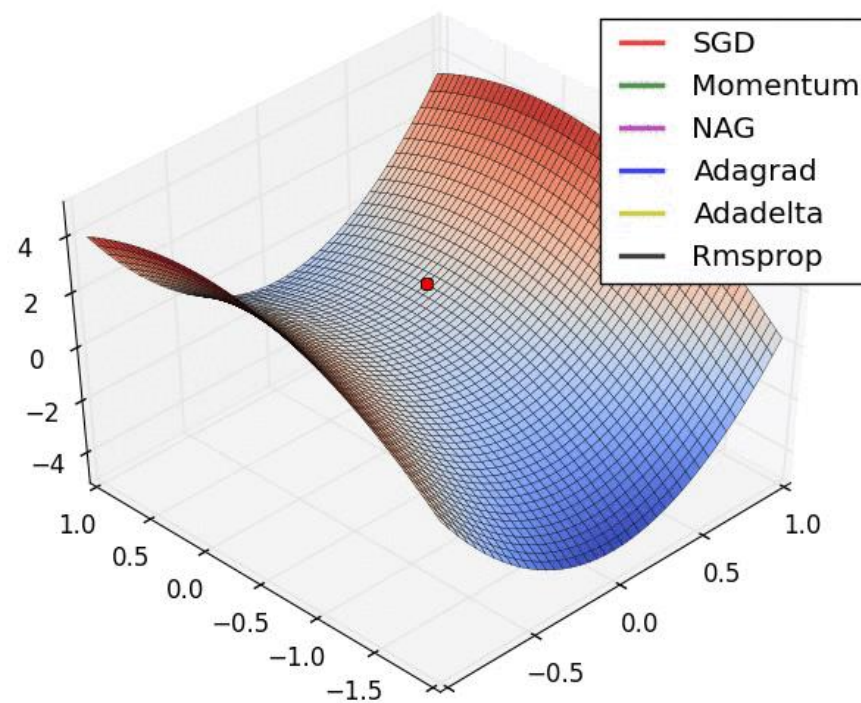
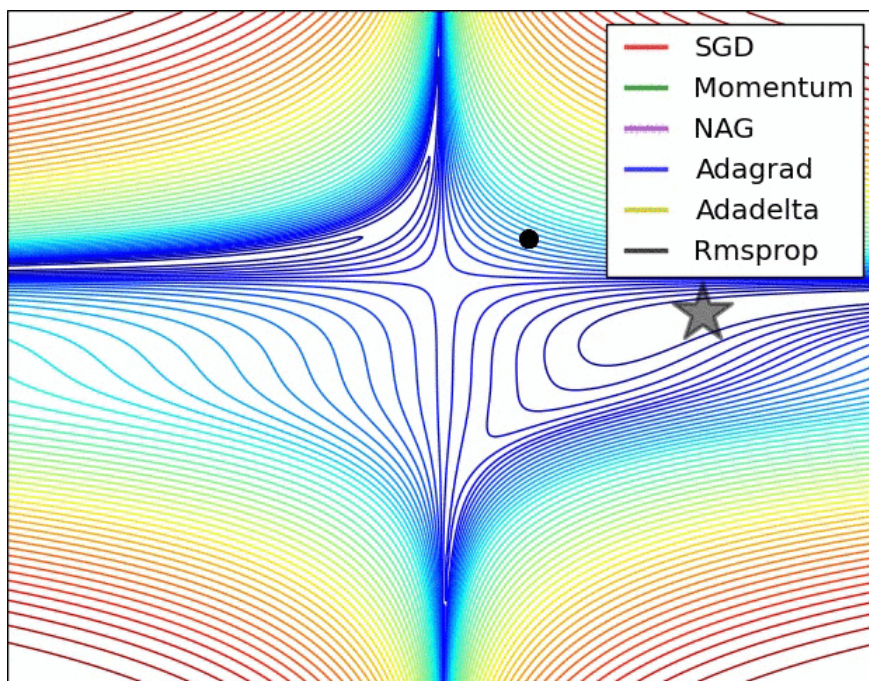


- 同一位置，损失在 θ_2 轴方向比在 θ_1 轴方向的斜率的绝对值更大
- 给定学习率，梯度下降迭代参数时在竖直方向会比在水平方向移动幅度更大

随机梯度下降的问题



优化算法的对比



目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

神经网络训练优化要点与技巧

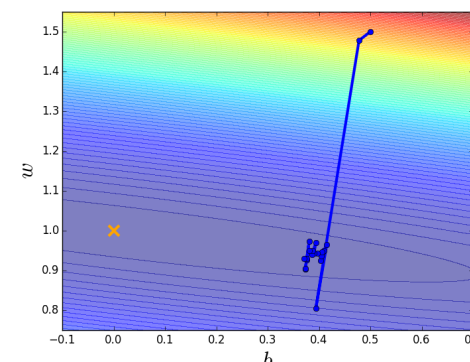
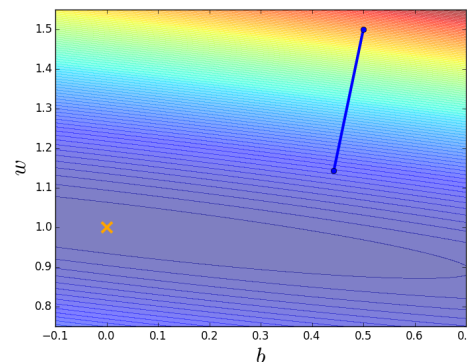
06

参数初始化

样本随机性的影响

□在每次迭代时，随机选择B个样本，这里的随机性非常重要

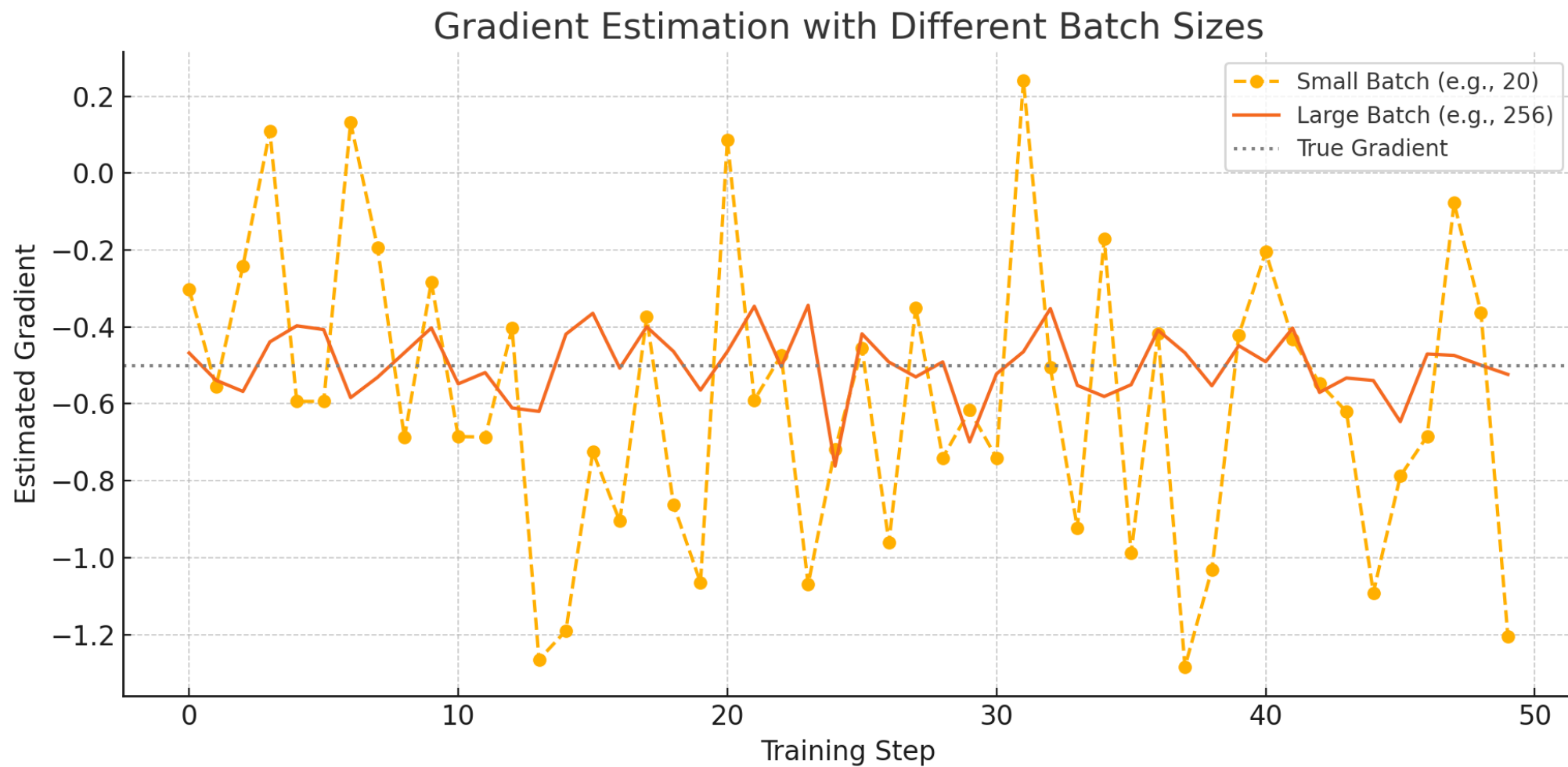
$$\frac{\partial \mathcal{L}_D(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(l)}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{W}^{(l)}} \quad \frac{\partial \mathcal{L}_B(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(l)}} = \frac{1}{B} \sum_{i=1}^B \frac{\partial \mathcal{L}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})}{\partial \mathbf{W}^{(l)}}$$



假设你想知道全校本科生喜欢看什么课外书，但你没法每次都问全校8000人，于是你每次随机找50个人来问，这就是一个 batch

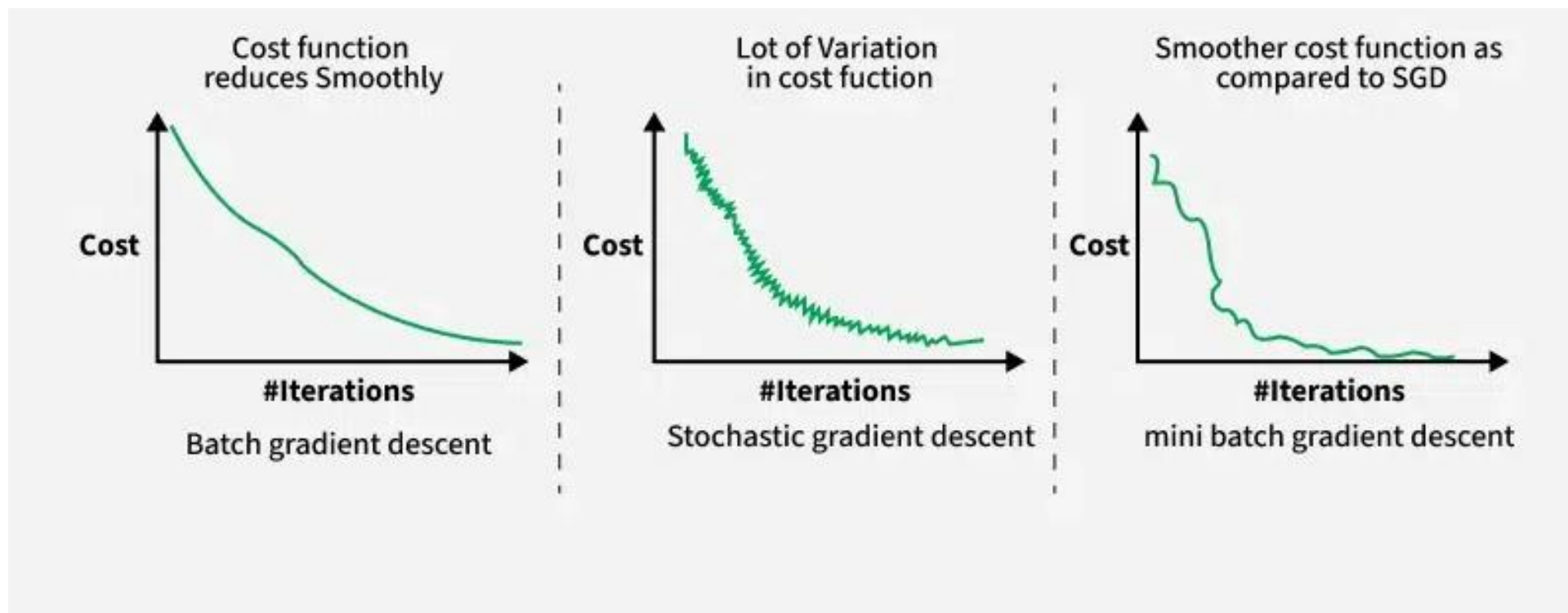
- 如果问“在座的50位同学”，那你很可能得到**偏见的结果**，比如都喜欢《电磁学千题解》，但这不代表全校的喜好。
- 每次你都要**随机从全校抽取**，这样才能保证你的“调查”是**尽可能代表整个学校的意见**。

样本随机性的影响



批量大小的影响

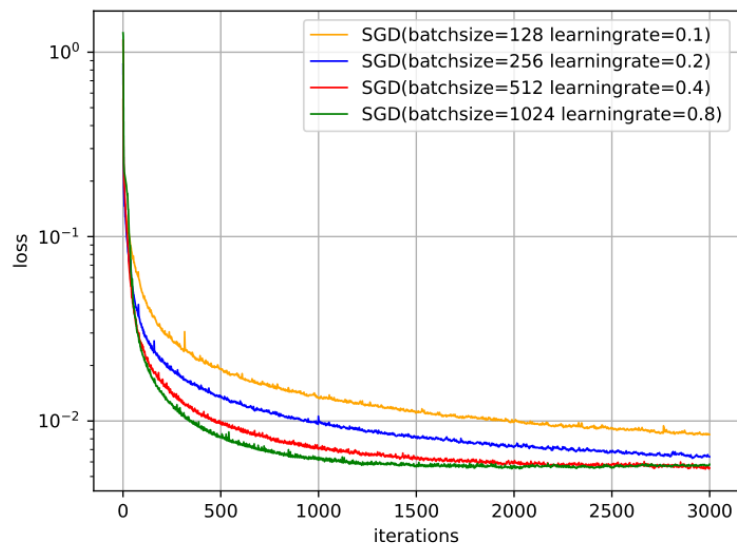
□ 批量大小不影响梯度期望，但会影响梯度方差，一般为2的幂数



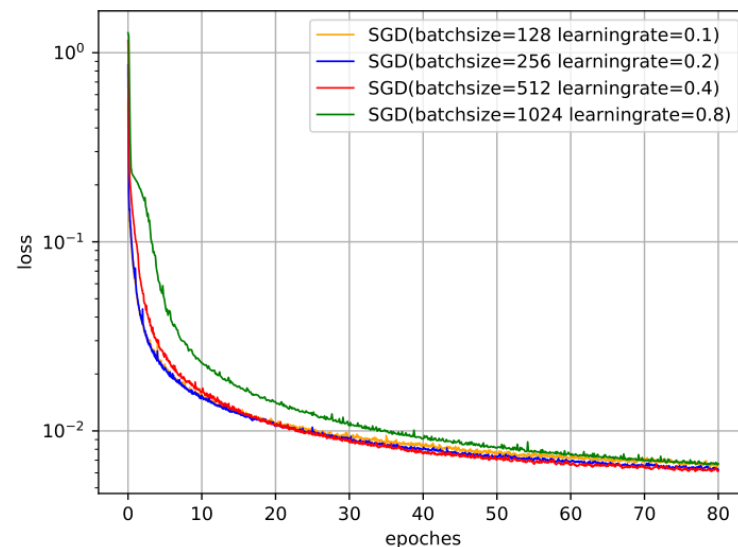
批量大小的影响

□ 批量大小不影响梯度期望，但会影响梯度方差，一般为2的幂数

- 批量越大，随机梯度的方差越小，引入的噪声也越小，训练也越稳定，因此可以设置较大的学习率
- 批量较小时，需要设置较小的学习率，否则模型会不收敛



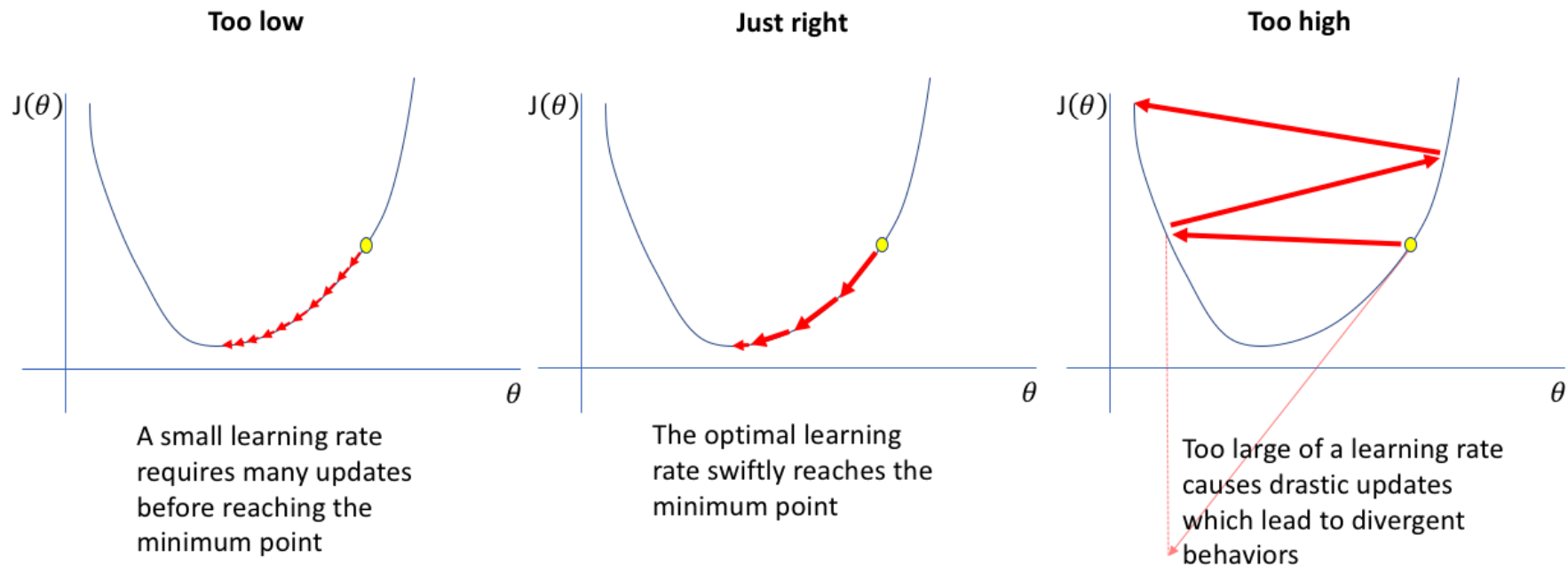
(a) 按 Iteration 的损失变化



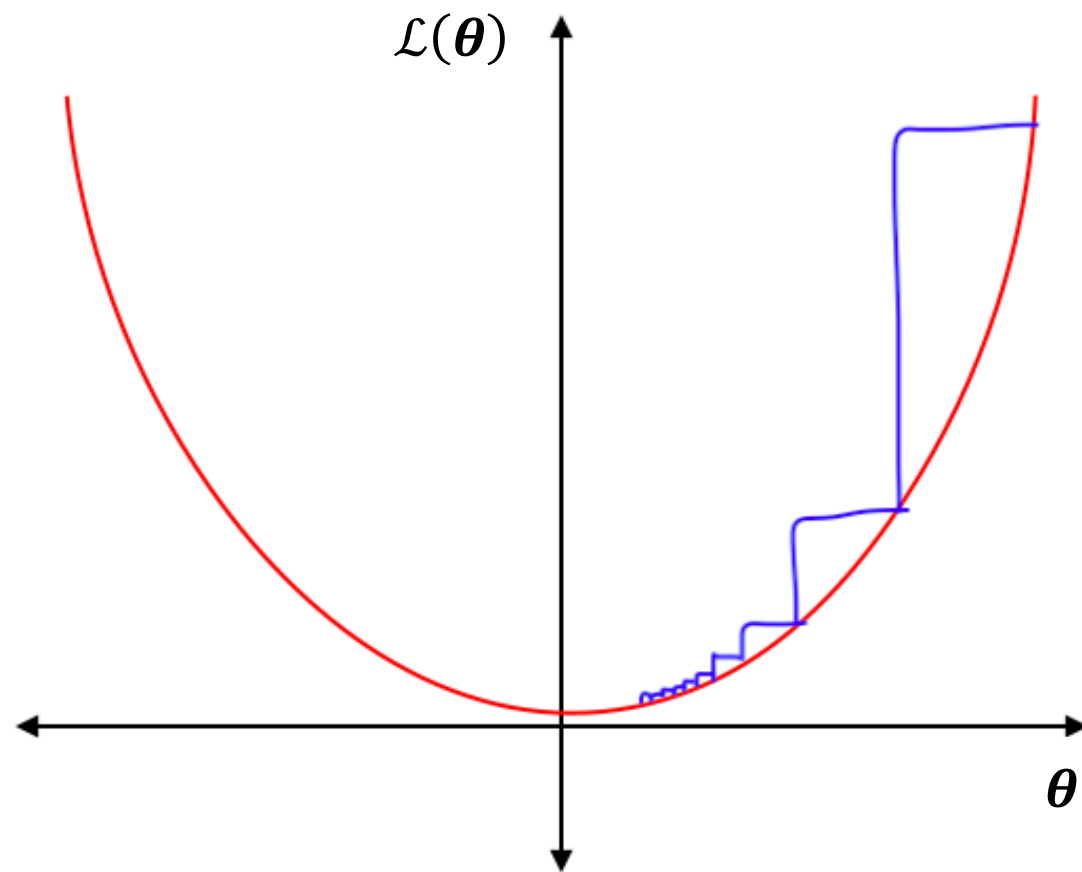
(b) 按 Epoch 的损失变化

小批量梯度下降中，每次选取样本数量对损失下降的影响

学习率的影响

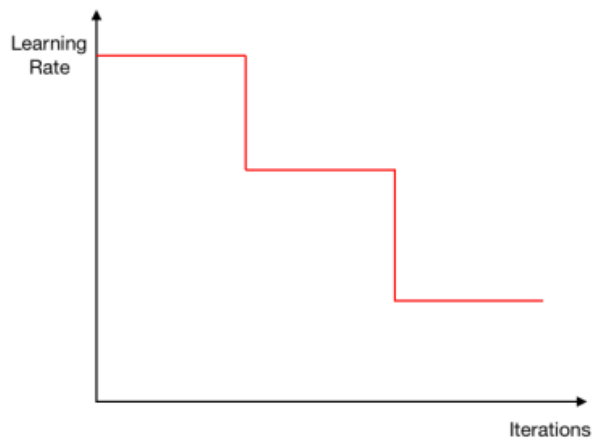


学习率衰减

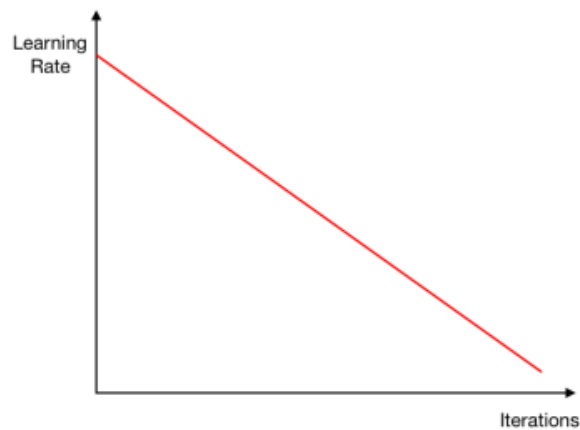


- 学习率一开始要保持大些保证收敛速度
- 在收敛到最优点附近时要小些以避免来回振荡

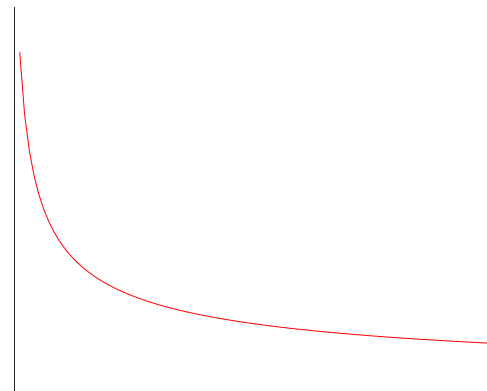
学习率衰减



梯级衰减 (step decay)



线性衰减 (Linear Decay)



1/t衰减 (1/t decay)

正则化 (Regularization)

□新优化目标：不仅最小化损失，而且要让权重尽可能小

$$\mathcal{L}'(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2} \lambda \underbrace{\|\boldsymbol{\theta}\|_2^2}_{\text{正则化项目}} \rightarrow \text{正则化项目}$$

损失，比如平方损失，
交叉熵损失等等

$$\boldsymbol{\theta} = \{w_1, w_2, \dots\}$$

L2 正则:

$$\|\boldsymbol{\theta}\|_2^2 = (w_1)^2 + (w_2)^2 + \dots$$

L1 正则:

$$\|\boldsymbol{\theta}\|_1 = |w_1| + |w_2| + \dots$$

一般不考虑bias

正则化 (Regularization)

L2 正则:

$$\|\boldsymbol{\theta}\|_2^2 = (w_1)^2 + (w_2)^2 + \dots$$

$$\mathcal{L}'(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2}\lambda\|\boldsymbol{\theta}\|_2^2$$

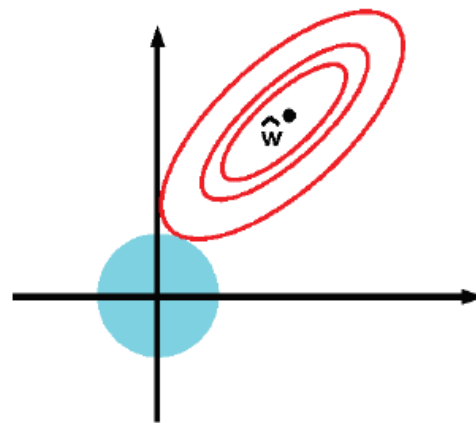
$$\text{梯度: } \nabla \mathcal{L}' = \nabla \mathcal{L} + \lambda \boldsymbol{\theta}$$

梯度下降更新:

$$\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} - \eta \nabla \mathcal{L}' = \boldsymbol{\theta}^{t-1} - \eta \nabla \mathcal{L} - \eta \lambda \boldsymbol{\theta}^{t-1}$$

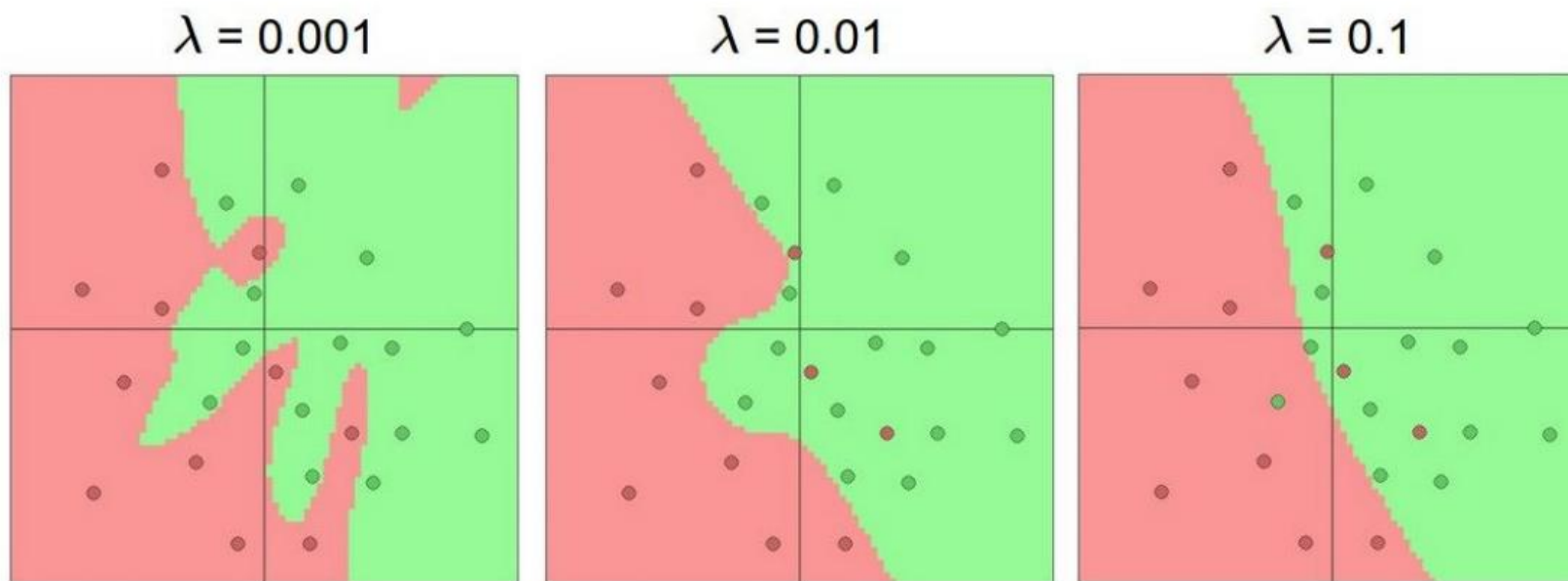
$$\eta < 1, \lambda < 1 \quad = \underbrace{(1 - \eta \lambda)}_{\text{越来越小}} \boldsymbol{\theta}^{t-1} - \eta \nabla \mathcal{L}$$

越来越小，但由于 $-\eta \nabla \mathcal{L}$ 项，使得参数不会变为0



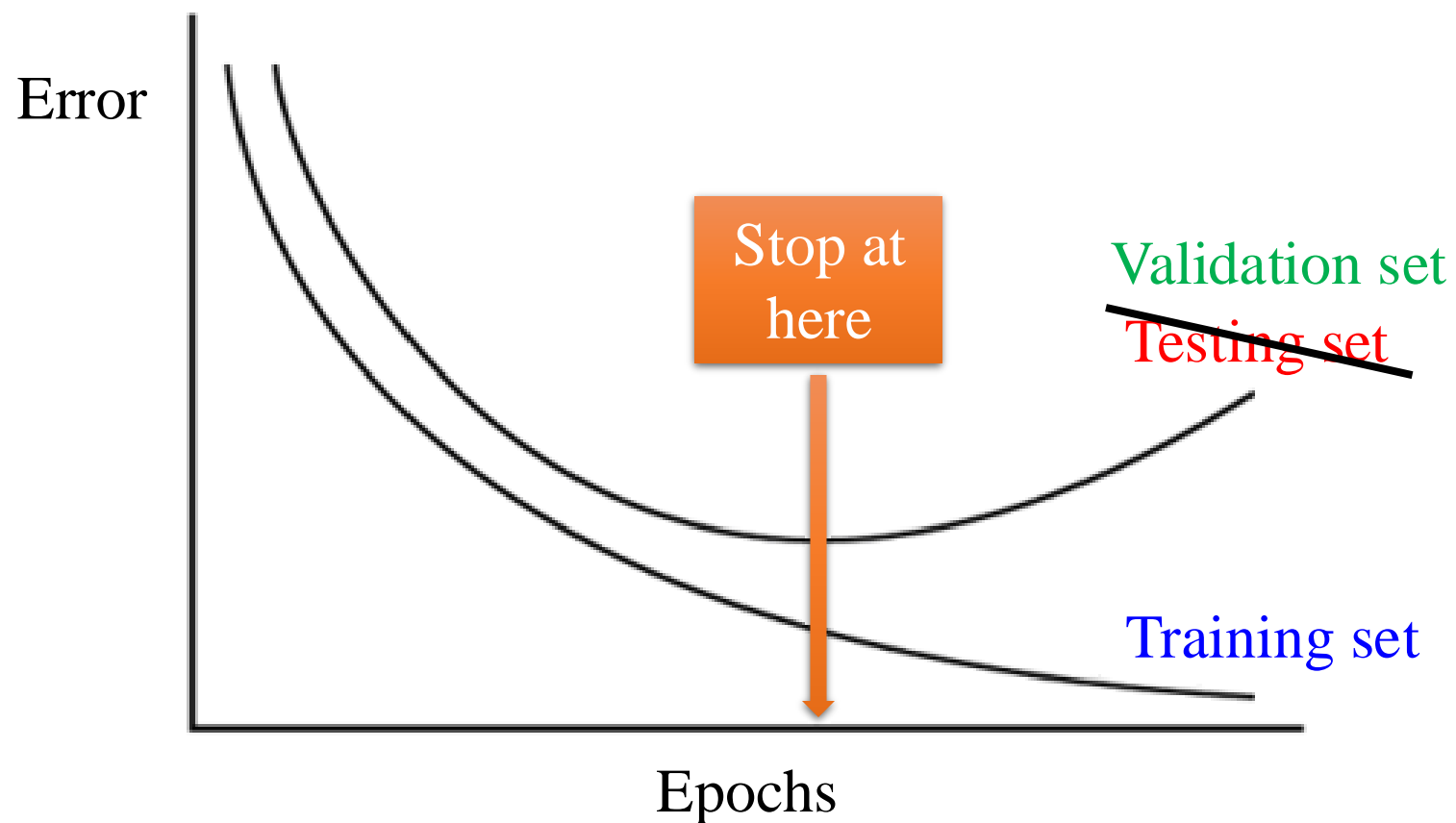
L2正则也称权重衰减 (weight decay)

□ 不同正则项权重对于分类性能的影响



<http://playground.tensorflow.org/>

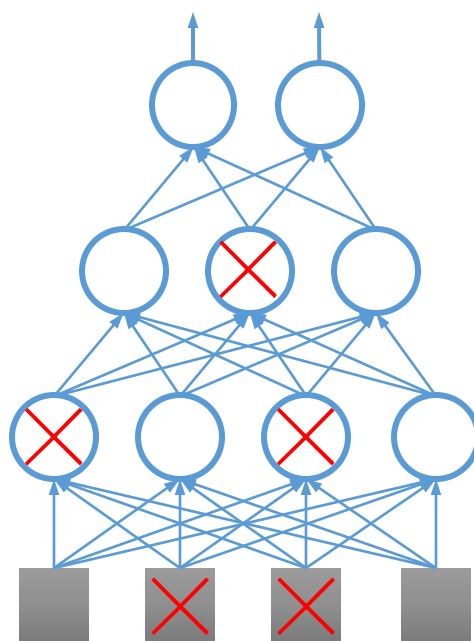
提前终止 (Early Stop)



- 验证集上准确率下降（损失上升）的时候停止训练
- 训练很长时间，保存在验证集上最优的模型

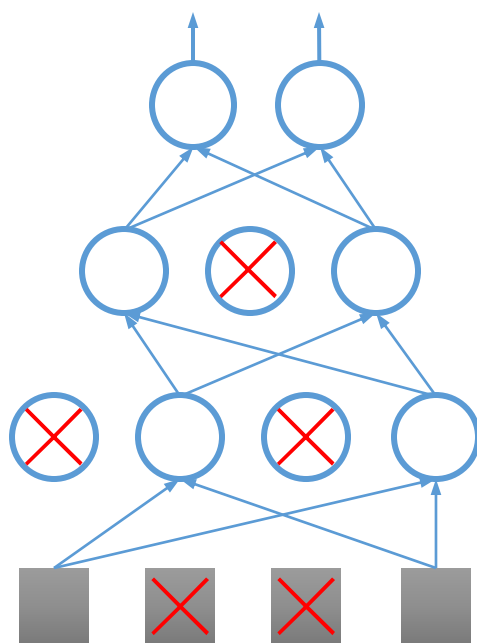
Dropout

□在训练时，取得一个batch后，以概率 p 设置一些神经元为0



Dropout

□在训练时，取得一个batch后，以概率 p 设置一些神经元为0



- 网络结构发生了变化
- 用新网络在batch上计算梯度
- 在新网络上进行参数更新

每次取新batch时，都需要重新随机对神经元置为0

Dropout—训练时的实现

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

输入单元被包括概率为0.8，隐藏单元为0.5

Dropout

□测试时，没有dropout，即所有的神经员都处于激活状态

□缩放激活函数的输出，使得每个神经元测试时输出等于训练时的期望输出

权重比例推断规则

如果训练时dropout 的概率为 p ，那么所有的权重要乘以 $1 - p$

Dropout—训练和测试时实现

该概率为保留概率

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

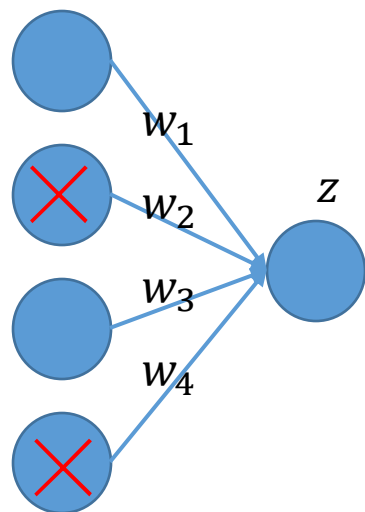
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```


Dropout—直观解释

训练时dropout

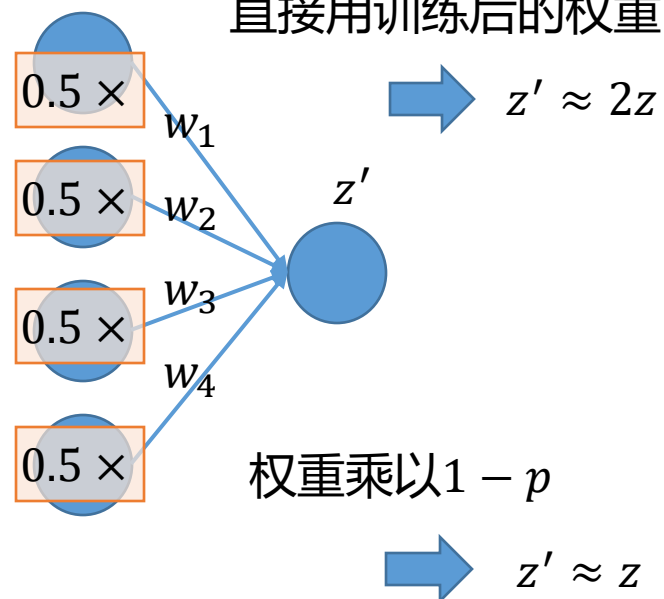
dropout 概率为0.5



测试时dropout

No dropout

直接用训练后的权重



Dropout动机

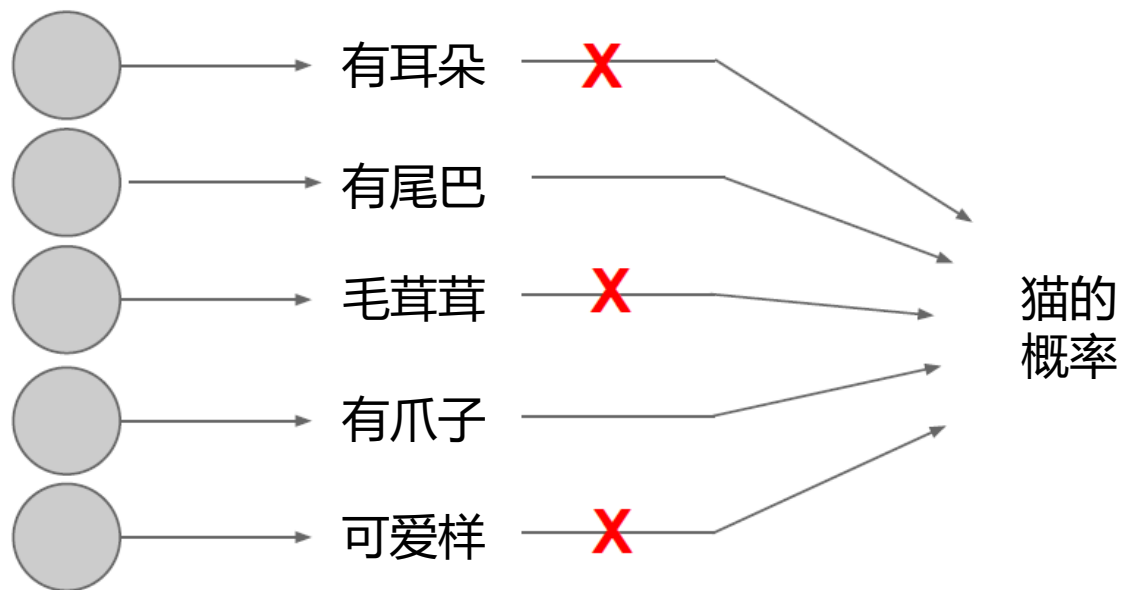
1

强制网络有冗余特征表示

2

防止特征的co-adaptation

co-adaptation: feature detectors只有在一些其它特定的feature detectors存在时才能发挥作用的情况



□ 人脸识别情形下的动机：

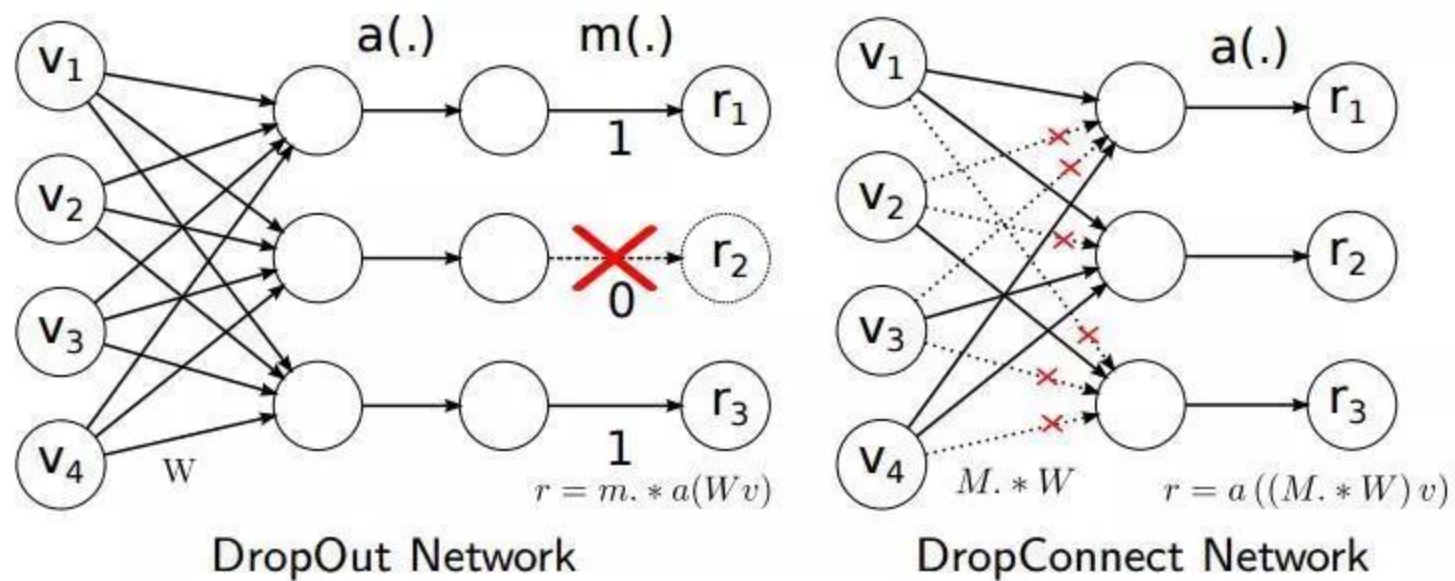
1

模型学得通过鼻检测脸的隐藏单元 h_i ，丢失 h_i 对应于擦除图像中有鼻子的信息

2

模型必须学习另一种 h_i ，要么是鼻子存在的冗余编码，要么是像嘴这样的脸部的另一特征

DropConnect



DropConnect: 将节点中的每个与其相连的输入权值以 $1-p$ 的概率变成0

Dropout: 随机的将隐层节点的输出变成0

目录

01

梯度下降和随机梯度下降

02

反向传播算法

03

深度学习的三个步骤和快速入门

04

随机梯度下降可能存在的问题

05

神经网络训练优化要点与技巧

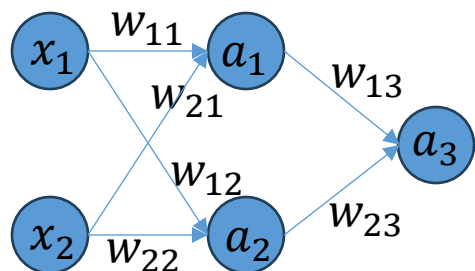
06

参数初始化

参数初始化

□参数不能全部初始化为0！为什么？

- 对称权重问题：前馈神经网络中同一层权重初始化为相同值。
- 前向传播中得到的每一层中的不同神经元激活值相同。
- 反向传播中得到的同一层的各个权重参数的梯度值相同。
- 使得隐层神经元没有区分性，神经网络无法有效学习数据中的特征。



$$a_1 = f(w_{11}x_1 + w_{21}x_2 + b_1) = f(0)$$

$$a_2 = f(w_{12}x_1 + w_{22}x_2 + b_2) = f(0)$$

$$a_3 = \text{sigmoid}(w_{13}a_1 + w_{23}a_2 + b_3) = \text{sigmoid}(0)$$

$$\text{第一次更新: } a_1 = a_2 \Rightarrow \frac{\partial L}{\partial w_{13}} = \frac{\partial L}{\partial w_{23}} \Rightarrow w_{13} - \eta \frac{\partial L}{\partial w_{13}} = w_{23} - \eta \frac{\partial L}{\partial w_{23}}$$

$$\text{第一次更新: } \frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial w_{22}} = 0$$

随后更新：由于对称性，保持 $a_1 = a_2 \Rightarrow$ 神经元输出没有区分性

□初始化方法

- 预训练初始化
- 随机初始化
- 固定值初始化（例如：偏置（Bias）通常用 0 来初始化）

随机初始化：基于固定方差的初始化

□ Gaussian分布初始化

- 参数从一个固定均值（比如0）和固定方差（比如0.01）的Gaussian分布进行随机初始化。

$$W = \sigma * \text{np.random.randn}(\text{fan_in}, \text{fan_out})$$

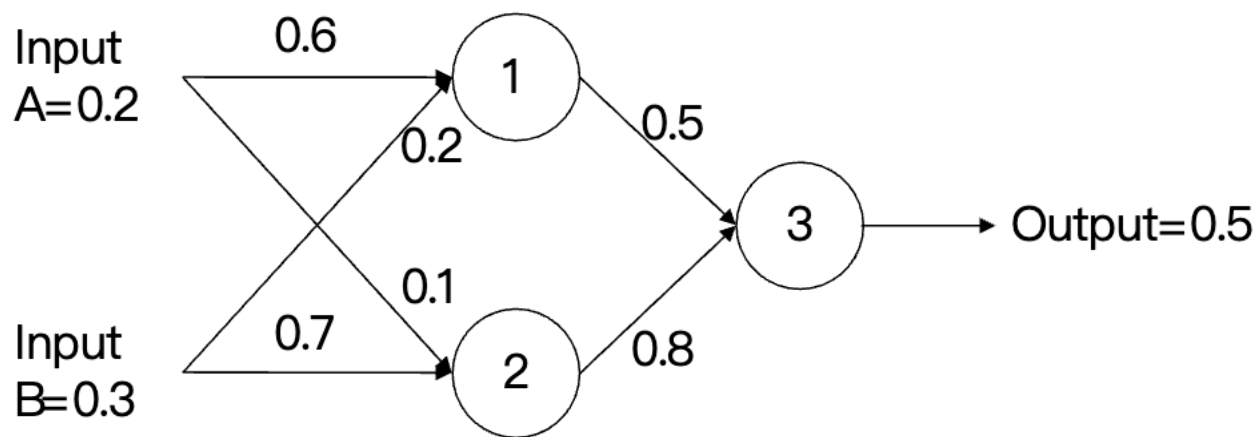
□ 均匀分布初始化

- 参数可以在区间 $[-r, r]$ 内采用均匀分布进行初始化

$$W = 2 * r * \text{np.random.rand}(\text{fan_in}, \text{fan_out}) - r$$

作业习题

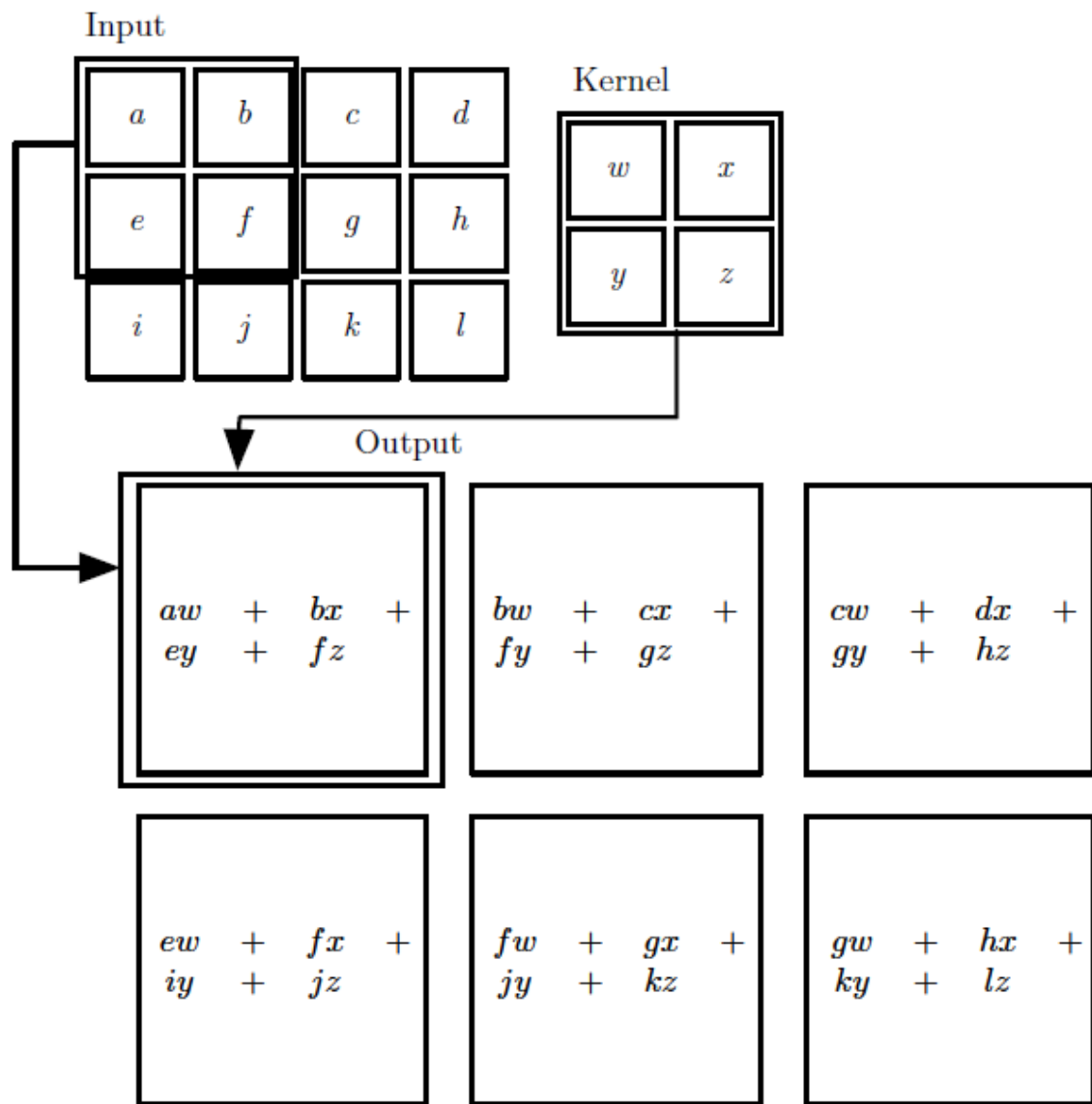
(1) 在如下神经网络，假设激活函数为ReLU，用平方损失 $\frac{1}{2}(y - \hat{y})^2$ 计算误差，请用BP算法更新一次所有参数（学习率为1），给出更新后的参数值，并计算给定输入值 $x=(0.2,0.3)$ 时初始时和更新后的输出值，检查参数更新是否降低了平方损失值



(2) 计算 $\sigma(x) = \frac{1}{1+\exp(-x)}$ 的一阶和二阶导数、 $\log \text{softmax}(\mathbf{x})_{[i]} = \log \frac{\exp(x_i)}{\sum_{j=1}^C \exp(x_j)}$ 的梯度

(3) 假设有一个前馈神经网络，包含L层（不包括输入层），每一层的输入和输出维度都是n，即每层有n个输入和n个神经元。激活函数为Sigmoid，损失函数为均方误差。在前向传播中，每一层主要进行哪些运算？大约需要多少次乘法和加法操作？整个网络的前向传播，总共大约需要多少次乘法操作？（请用n和L表示）；在反向传播中，每一层需要计算误差项并更新权重，主要涉及哪些计算？每层大约需要多少次乘法操作？整个网络的反向传播，总共大约需要多少次乘法操作？

作业习题



(4) 如果用如下卷积替换前馈网络中的线性变换，试计算误差项 $\frac{\partial L}{\partial z_{ij}^{(l)}}$ 的递推式

$$z_{ij}^{(l)} = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} w_{uv} a_{i+u, j+v}^{(l-1)}$$

$$a_{ij}^{(l)} = f(z_{ij}^{(l)})$$

(5) 如果有池化层，这一层的误差如何反向传播？