03 深层神经网络：表示、计算与实现

机器学习 · 深度学习入门

学习目标

- 恢复多层复合表示与一般前向、反向框架
- 跟踪批量形状，解释平均目标与参数梯度
- 将层、参数、损失和循环对应到实现对象
- 读懂完整训练、验证与预测，解释数据与配置作用
- 区分深度、可训练性与泛化表现

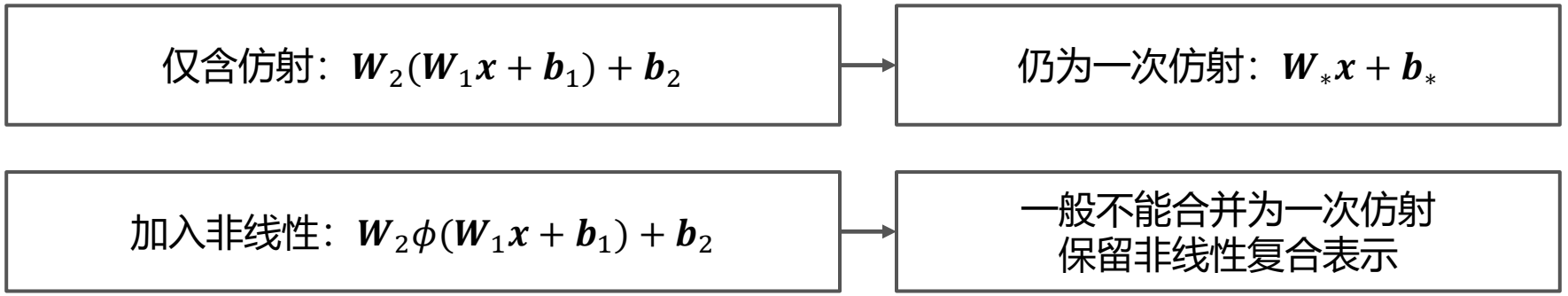
1 多层表示与共同计算

单隐藏层怎样推广为一般多层网络?

从逐步计算到函数复合

- 单隐藏层先形成隐藏表示，再由输出层使用；增加隐藏层，让已形成的表示继续成为下一层加工的对象
- **函数复合**把前一变换的输出作为后一变换的输入；中间结果连接各步计算
  - 先得到 $\boldsymbol{z} = f_1(\boldsymbol{x})$，再计算 $\boldsymbol{o} = f_2(\boldsymbol{z})$；把后一式中的 $\boldsymbol{z}$ 换成 $f_1(\boldsymbol{x})$，得到 $\boldsymbol{o} = f_2(f_1(\boldsymbol{x}))$
  - 计算从内层的 f_1 开始；前一步的输出必须能作为后一步的输入
- 对应单隐藏层：$f_1(\boldsymbol{x}) = \phi(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$，$f_2(\boldsymbol{z}) = \boldsymbol{W}_2\boldsymbol{z} + \boldsymbol{b}_2$
- 多个隐藏层逐级变换表示，而不是各自独立地处理原始输入
  - 对三层模型重新按层编号，可写为 $\boldsymbol{o} = f_3(f_2(f_1(\boldsymbol{x})))$；每一步接收前一步的表示
- 复合说明计算怎样连接；这些变换能否合并为一次仿射，还取决于各步的变换规则

仿射复合与非线性作用

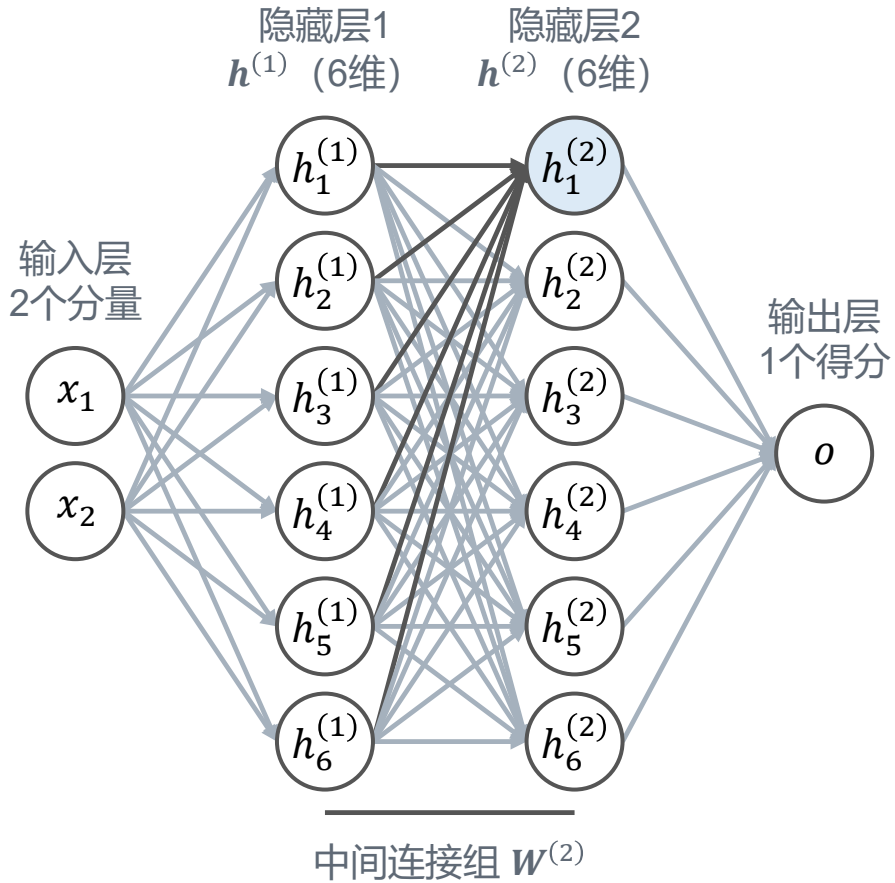


两层变换的局部对比：观察是否可合并

- 展开仿射复合：$W_2(W_1x+b_1)+b_2=(W_2W_1)x+W_2b_1+b_2$
  - 因而可合并为一次仿射：$W_*=W_2W_1$，$b_*=W_2b_1+b_2$
- 非线性激活改变中间表示的变换规则，使复合通常超出单次仿射的表达范围
  - 回看第一讲 ReLU 曲线的转折：正值保留、负值置0；各层继续变换前层的表示
- 表示能力、训练与泛化分别判断；增加深度的实际收益须通过实验检验

网络结构与层表示

- 本讲网络为 **2 → 6 → 6 → 1**：输入2维、两个隐藏层各6个单元、输出1个得分
  - **深度**按带参数层计为3；**宽度**为每层单元数。输入记作 $\boldsymbol{h}^{(0)} = \boldsymbol{x}$，带参数层编号 $l = 1, \ldots, L$，本例 $L = 3$
- 图中每列隐藏单元的输出组成 $\boldsymbol{h}^{(l)}$；相邻两列之间的权重组成 $\boldsymbol{W}^{(l)}$
  - 加深的六条线输入第二隐藏层第1个单元：前层六个输出经加权求和、加偏置与ReLU，形成一个新分量；其余单元同理，合成下一层表示
  - 本讲重点跟踪中间一组连接：$\boldsymbol{h}^{(1)} \xrightarrow{\boldsymbol{W}^{(2)}} \boldsymbol{h}^{(2)}$，$\boldsymbol{W}^{(2)}$ 为 6×6

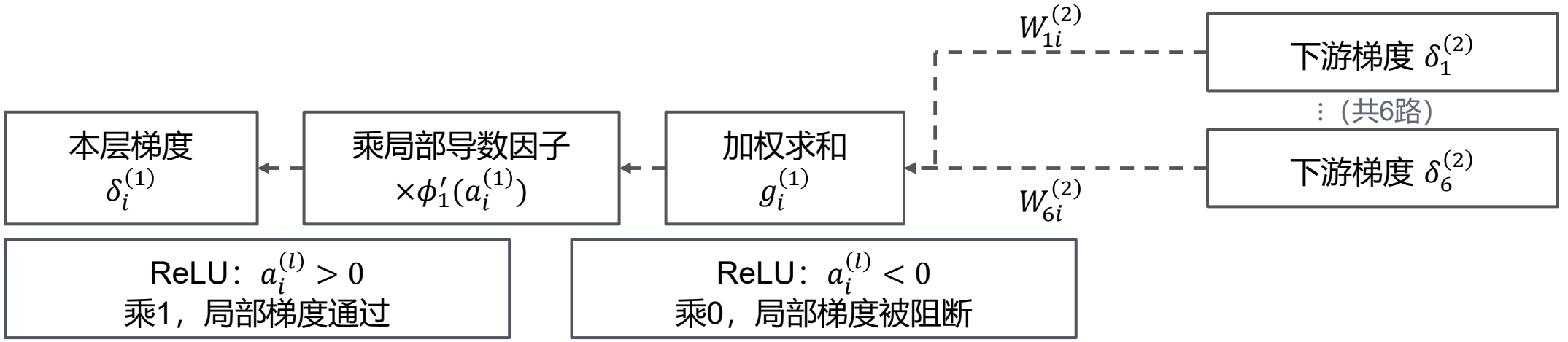


隐藏层使用ReLU；输出直接给得分

一般前向递推

- 所有带参数层先通过仿射运算得到**预激活**：
  - $\boldsymbol{a}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}$，$l = 1, \ldots, L$
  - $\boldsymbol{W}^{(l)}$ 组合前层分量，$\boldsymbol{b}^{(l)}$ 调整各接收节点的加权和
- 隐藏层再逐元素激活：$\boldsymbol{h}^{(l)} = \phi_l(\boldsymbol{a}^{(l)})$，$l < L$
- 本讲输出层直接给 logit：$\boldsymbol{o} = \boldsymbol{a}^{(L)}$；二分类概率为 $p = \sigma(o)$
  - 单输出的唯一分量简写为标量 o；sigmoid 不另计带参数层
- 若前层宽度为 d_{l-1}、当前为 d_l，权重形状为 $d_l \times d_{l-1}$，预激活为 d_l 维
  - 本例第二层 $\boldsymbol{W}^{(2)}$ 为 6×6；同一套层运算依次用于隐藏层1、隐藏层2，再由输出层形成得分

单元的梯度贡献汇总

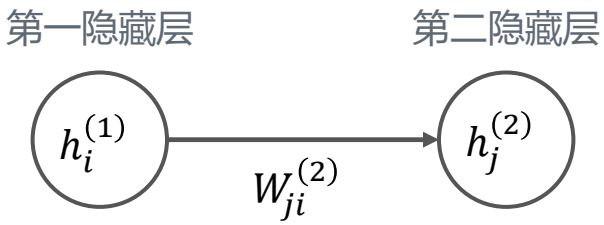


- **预激活梯度** $\boldsymbol{\delta}^{(l)} = \partial\ell/\partial\boldsymbol{a}^{(l)}$ 衡量损失对激活前各分量的敏感度
- 先汇总下游贡献：$\boldsymbol{g}^{(l)} = \partial\ell/\partial\boldsymbol{h}^{(l)} = (\boldsymbol{W}^{(l+1)})^{\top}\boldsymbol{\delta}^{(l+1)}$
  - 固定前层单元 i，$g_i^{(l)} = \sum_{j=1}^{d_{l+1}} W_{ji}^{(l+1)}\,\delta_j^{(l+1)}$：接收单元 j 的贡献沿同一连接传回 i
- 再经过本层激活：$\boldsymbol{\delta}^{(l)} = \boldsymbol{g}^{(l)} \odot \phi_l'(\boldsymbol{a}^{(l)})$，$l < L$
  - 图中固定第一隐藏层的 i，汇总第二隐藏层的六路贡献；$\odot$ 为逐元素相乘

多层反向递推

- 本讲单 logit 配 BCE，从 $\delta^{(L)} = p - y$ 开始逐层传回；$p = \sigma(o)$，y 为标签
  - 本例先得 $\delta^{(3)}$（标量），再得 $\boldsymbol{\delta}^{(2)}$、$\boldsymbol{\delta}^{(1)}$（均为6维）；每个隐藏层重复“汇总贡献、乘局部导数”
- 沿用第2讲反向模式自动微分：按依赖逆序组合局部导数、汇总贡献
  - 本讲将这条规则写成层递推，不为每个隐藏层另设求导方法：
  - $\boldsymbol{g}^{(l)} = (\boldsymbol{W}^{(l+1)})^{\top}\boldsymbol{\delta}^{(l+1)}$，$\boldsymbol{\delta}^{(l)} = \boldsymbol{g}^{(l)} \odot \phi_l'(\boldsymbol{a}^{(l)})$
- 对本例第二隐藏层，下游只有一个输出；第一隐藏层则汇总第二隐藏层的六路贡献
- 逐层复用已求得的下游梯度和前向保存的预激活；反向计算期间保持同一参数状态

单连接参数梯度



来自中间连接组 $\boldsymbol{W}^{(2)}$；只放大 i→j，圆形单元为激活后输出，其余连接和偏置省略

- 中间连接组的一项 $W_{ji}^{(2)}$：从第一隐藏层 i 到第二隐藏层 j
  - 图中只放大这条连接；i 对应权重矩阵的列，j 对应行，接收单元的 $\delta_j^{(2)}$ 是对其预激活 $a_j^{(2)}$ 的梯度
- 一般地，i 为前层输入索引，j 为接收节点索引：$a_j^{(l)} = \sum_i W_{ji}^{(l)} h_i^{(l-1)} + b_j^{(l)}$
- 单连接的局部导数为 $\partial a_j^{(l)} / \partial W_{ji}^{(l)} = h_i^{(l-1)}$，因此
  - $\partial \ell / \partial W_{ji}^{(l)} = \delta_j^{(l)} h_i^{(l-1)}$：复用前向输入与反向敏感度

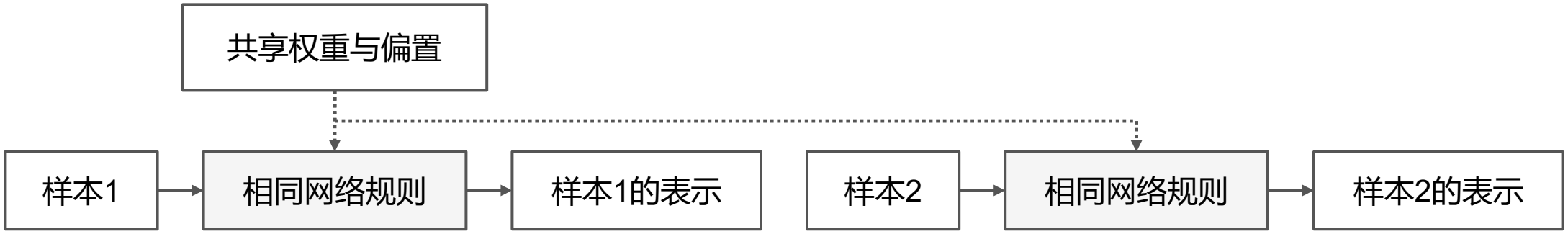
整层参数梯度

- 沿用第2讲的外积关系，将输入换成本层收到的表示
  - $\nabla_{\boldsymbol{W}^{(l)}}\ell = \boldsymbol{\delta}^{(l)}(\boldsymbol{h}^{(l-1)})^{\top}$，$\nabla_{\boldsymbol{b}^{(l)}}\ell = \boldsymbol{\delta}^{(l)}$
  - 第 j 行、第 i 列为 $\left(\nabla_{\boldsymbol{W}^{(l)}}\ell\right)_{ji} = \delta_j^{(l)} h_i^{(l-1)}$；整行对应同一接收单元 j
  - 回到第二层：$\boldsymbol{\delta}^{(2)}(\boldsymbol{h}^{(1)})^{\top}$ 汇集全部36条连接的梯度，与 $\boldsymbol{W}^{(2)}$ 同为 6×6
- 偏置直接相加，$\partial a_j^{(l)}/\partial b_j^{(l)} = 1$，故 $\partial\ell/\partial b_j^{(l)} = \delta_j^{(l)}$
  - 局部核对：输入2、节点梯度 -0.3，得到权重梯度 -0.6、偏置梯度 -0.3
- 每层得到自己的 $\boldsymbol{\delta}^{(l)}$ 后，与该层前向保存的 $\boldsymbol{h}^{(l-1)}$ 配对，就能计算该层参数梯度；同一次反向中不提前更新参数

2 批量计算与实现对象

样本轴、参数共享与模块接口怎样对应？

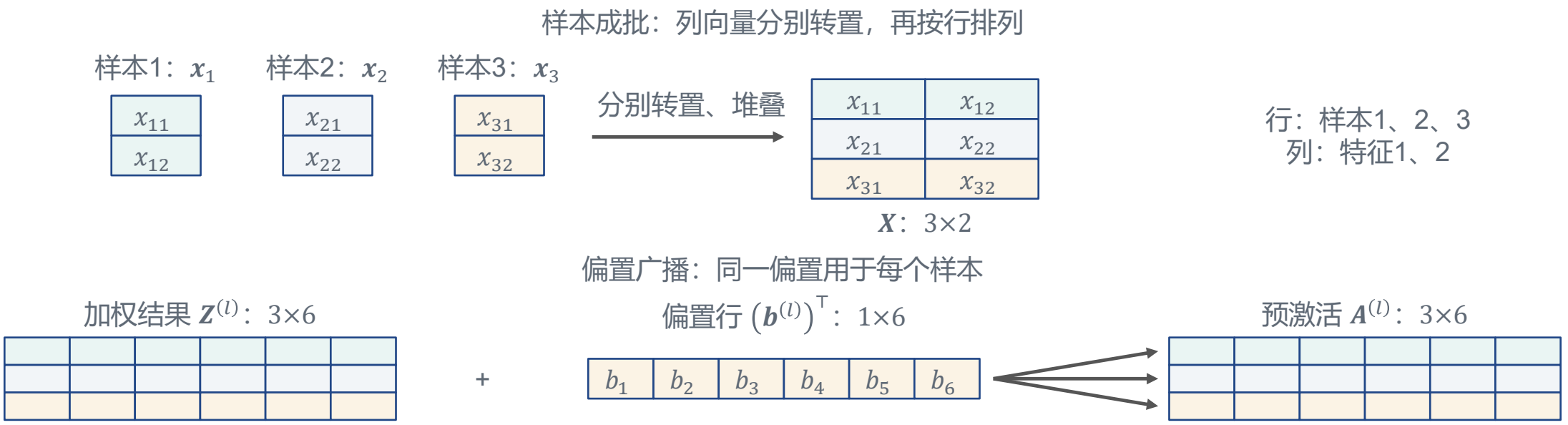
批量参数共享



图中两条路径共用一组参数，各自保留中间激活

- 本讲各样本独立计算，拥有自己的输入与中间激活，所有样本使用**同一组权重和偏置**
- **一个批次**将 B 个样本组织在一起计算，仍使用同一网络规则
  - 接下来将各样本按行排列，再看同一偏置如何作用于每个样本

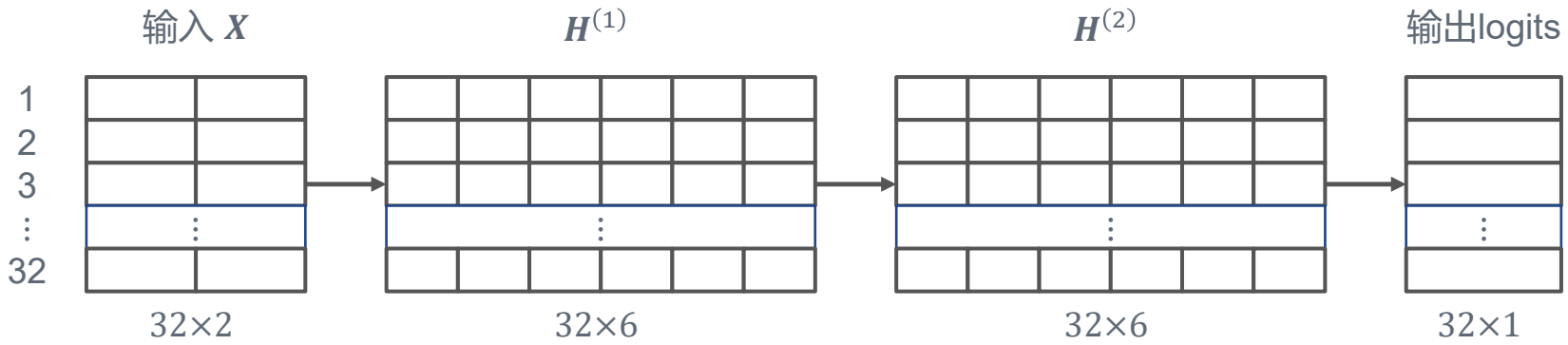
样本成批与偏置广播



图中B=3仅示意排布；下一页实际批量为32。广播展示加法效果，不要求复制B份偏置。

- 将 B 个单样本列向量分别转置，按行组成 $\boldsymbol{X} \in \mathbb{R}^{B\times d_0}$，$\boldsymbol{H}^{(0)} = \boldsymbol{X}$
- 记加权结果 $\boldsymbol{Z}^{(l)} = \boldsymbol{H}^{(l-1)}(\boldsymbol{W}^{(l)})^\top$，则 $\boldsymbol{A}^{(l)} = \boldsymbol{Z}^{(l)} + \mathbf{1}_B(\boldsymbol{b}^{(l)})^\top$
  - **广播**逐行加同一偏置：$A^{(l)}_{rj} = Z^{(l)}_{rj} + b^{(l)}_j$；$\mathbf{1}_B$ 为全一列向量
- 隐藏层仍逐元素激活：$\boldsymbol{H}^{(l)} = \phi_l(\boldsymbol{A}^{(l)})$，$l < L$；输出层的 $\boldsymbol{A}^{(L)}$ 直接给出批量 logits

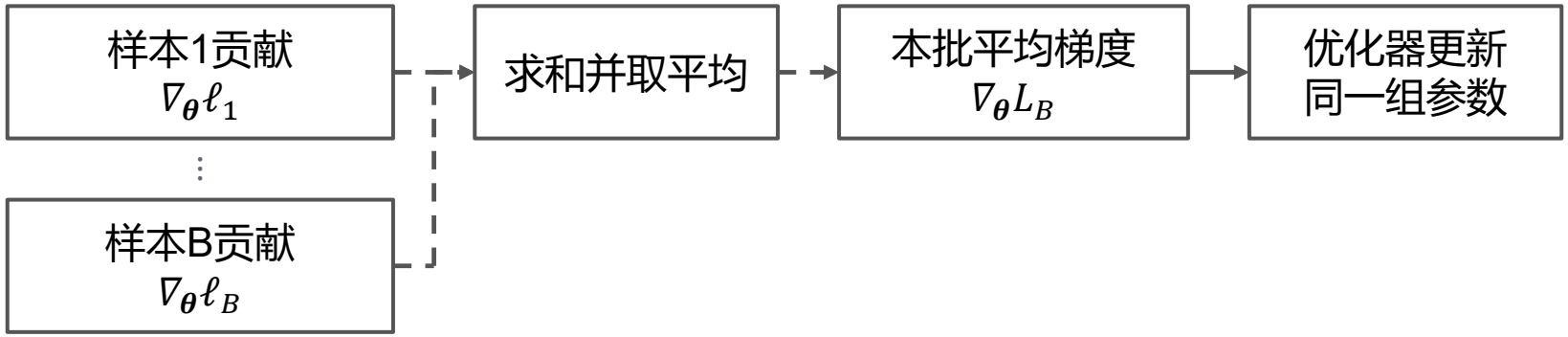
一个批次经过网络的形状



各矩阵同行对应同一样本；B为本批样本数，最后一批可小于设定大小

对象	形状	当前含义
输入 X	32 × 2	32 个二维样本
第一层权重（weight）	6 × 2	输入2维，输出6维
第一隐藏表示	32 × 6	每样本6个分量
第二层 weight	6 × 6	隐藏表示再变换
第二隐藏表示	32 × 6	样本轴保持
输出层 weight	1 × 6	将6维表示变为1个得分
logits 与标签	32 × 1	二分类目标同形

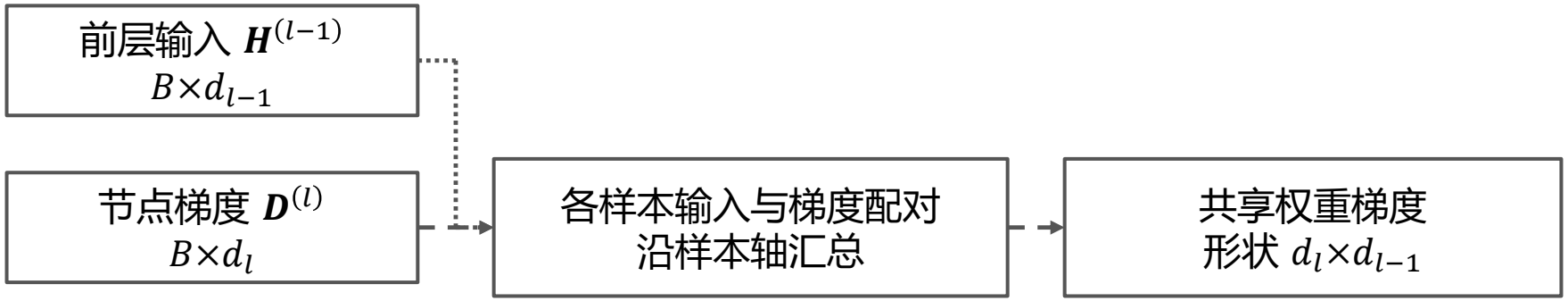
平均损失与平均梯度



所有贡献在同一参数状态下计算，汇总后供本批一次更新使用

- 批量目标：$L_B = \frac{1}{B}\sum_{i=1}^{B}\ell_i$，各样本针对**同一组参数**贡献损失
- 求导可逐项作用于损失之和，因此平均目标的梯度也是平均梯度：$\nabla_{\boldsymbol{\theta}}L_B = \frac{1}{B}\sum_i \nabla_{\boldsymbol{\theta}}\ell_i$
  - $\boldsymbol{\theta}$ 为按固定顺序展开的参数向量；所有贡献在同一参数状态下计算
- 各样本的梯度方向可以不同，平均后共同决定本批的更新依据
- 本例每批汇总后更新一次参数；逐样本求贡献期间保持参数不变

批量反向与梯度汇总



- $\boldsymbol{D}^{(l)} = \partial L_B / \partial \boldsymbol{A}^{(l)}$ 按样本行排列；令 $\boldsymbol{\delta}_r^{(l)}$ 为第 r 个样本自身损失的预激活梯度，则
  - $\boldsymbol{D}_{r,:}^{(l)} = (\boldsymbol{\delta}_r^{(l)})^\top / B$：转置对应行排列，$1/B$ 来自平均目标，后续不再除B
- 权重梯度：$\nabla_{\boldsymbol{W}^{(l)}} L_B = (\boldsymbol{D}^{(l)})^\top \boldsymbol{H}^{(l-1)}$
  - 矩阵乘法沿样本轴汇总各样本的外积贡献；每行已含平均因子
- 偏置梯度：$\nabla_{\boldsymbol{b}^{(l)}} L_B = (\boldsymbol{D}^{(l)})^\top \mathbf{1}_B$，即各行求和
- BCE末端：$\boldsymbol{D}^{(L)} = (\boldsymbol{P} - \boldsymbol{Y})/B$；$\boldsymbol{P}, \boldsymbol{Y}$ 为 $B\times 1$ 的概率与标签矩阵

用 Sequential 组织多层网络

```python
import torch
from torch import nn
model = nn.Sequential(
    nn.Linear(2, 6), nn.ReLU(),
    nn.Linear(6, 6), nn.ReLU(),
    nn.Linear(6, 1))
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(
    model.parameters(), lr=0.1)
```

模块连接

Sequential 按列出的顺序执行模块，将前一模块的输出传给下一模块，实现逐层复合

Linear(2, 6) 把每个样本的2维输入变为6维表示，ReLU 对应逐元素隐藏激活

调用 model(Xb) 执行整个序列；B×2输入依次成为B×6、B×6和B×1的logits

输出与损失

最后一层给logits；BCEWithLogitsLoss合并计算sigmoid与二分类交叉熵，默认对本批损失取平均

参数与优化器

model.parameters() 汇集各 Linear 的权重和偏置；SGD（随机梯度下降）保存这些参数与学习率0.1，供训练循环按本批平均梯度更新；构造优化器时不更新参数

结构、参数状态与基本初始化

项目	本讲对象	作用
结构	$2 \to 6 \to 6 \to 1$	规定计算形式与宽度
参数	三组 weight 与 bias（偏置）	存储当前可训练状态
参数数量	18 + 42 + 7 = 67	三层权重加偏置
初始化	Linear 默认规则，随机种子17，固定初始化随机序列	提供训练起始状态
ReLU	两处逐元素运算	不产生可训练参数

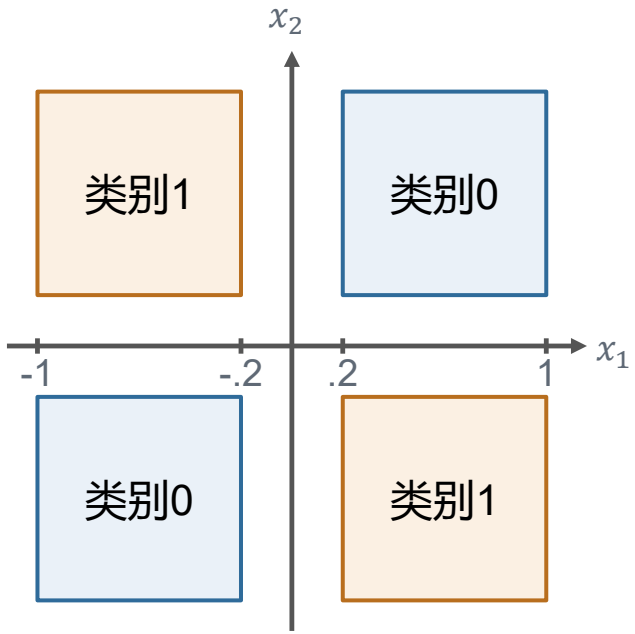
- 每个 Linear 的参数数为 $d_{\text{out}}d_{\text{in}} + d_{\text{out}}$，对应权重元素与偏置元素
- 基本初始化使示例可运行，其尺度与稳定性原理在后续展开

3 完整训练与模型使用

按数据角色组织训练、验证和新输入预测。

训练、验证与测试的分工

集合	本例样本数	作用	是否参数更新
训练	256	每批计算目标与梯度	是
验证	128	观察固定配置的开发表现	否
测试	128	最后独立评价	否



着色区满足 $0.2 \le |x_1|, |x_2| < 1$；白色十字区不采样

- 图中着色区是**采样区域与真实标签规则**，不表示模型已学到的决策边界
  - 两坐标绝对值位于 [0.2,1)；异号标签为1，同号标签为0，坐标轴附近不采样
- 三份数据独立采样；当前配置固定，验证未用于调参

批量读取保持输入与标签配对

```
from torch.utils.data import (
    TensorDataset, DataLoader)
g = torch.Generator().manual_seed(404)
train_set = TensorDataset(Xtr, ytr)
train_loader = DataLoader(
    train_set, batch_size=32,
    shuffle=True,
    generator=g)
```

样本与标签

TensorDataset按同一索引配对Xtr与ytr（256×2与256×1）

批量与顺序

DataLoader每批读取最多32个样本；shuffle按样本对打乱读取顺序，保持输入与标签配对

可追溯条件

固定顺序种子；完整数据生成与训练见随本讲材料提供的 examples/train_demo.py

单批次训练流程

```
model.train()
for Xb, yb in train_loader:
    optimizer.zero_grad(set_to_none=True)
    logits = model(Xb)
    loss = loss_fn(logits, yb)
    loss.backward()
    optimizer.step()
```

梯度清零

backward 默认累加到 grad；zero_grad 清除上一批梯度，使本次更新只使用当前批次的贡献

前向与损失

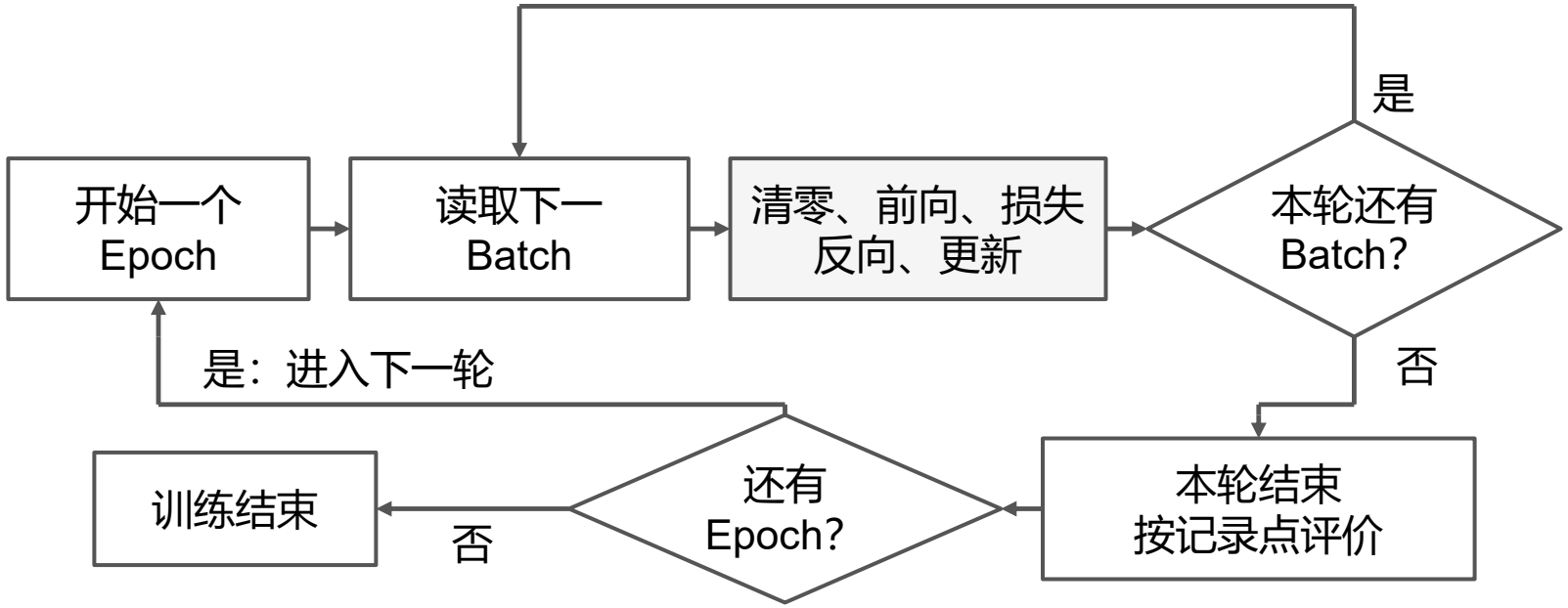
model 计算 logits，loss_fn 对本批样本损失取平均

反向与更新

backward 将梯度写入参数的 grad；step 使用梯度更新参数，下一批使用更新后的参数

前向与反向使用本批更新前的同一参数状态；train 只设置训练模式

Epoch 与 Batch 双层循环



- 一个 **Epoch**（训练轮次）遍历完整训练集；每个Batch更新一次参数
  - 本例256个样本、批量32：每轮8次更新，100轮共800次；图中的两次判断分别控制批次与轮次
- 两种损失记录对应不同的**参数状态**
  - **训练过程记录**：按样本数加权汇总各批平均损失；参数随批次更新
  - **轮末评价**：用同一固定参数重新计算完整训练集与验证集的损失；结果表只列指定轮次

验证模式与梯度记录分别控制

```
model.eval()
with torch.no_grad():
    logits = model(Xva)
    val_loss = loss_fn(logits, yva).item()
    pred = (logits >= 0)
    matches = (pred == (yva >= .5))
    val_acc = matches.float().mean()
```

模块模式

eval切换模块到评价模式，不自动关闭梯度记录；本例只有Linear与ReLU，切换模式不改变前向计算

记录开关

no_grad关闭梯度记录，避免建立反向图；本段不调用 step，参数保持固定

评价量

logit非负等价于sigmoid概率不小于0.5；matches 比较预测类别与真实标签，取平均得到正确率

构造任务上的一次训练记录

epoch	训练损失	验证损失	验证正确率
1	0.6977	0.6980	50.8%
10	0.6900	0.6894	50.8%
25	0.6721	0.6663	79.7%
50	0.5245	0.4804	80.5%
100	0.0612	0.0710	100.0%

- 两种损失均在该 epoch 结束后，用固定当前参数分别评价；单次运行只支持本讲数据与配置
- 第1到10轮验证损失下降，正确率仍为50.8%；损失关注概率得分，正确率只统计阈值分类后的对错

训练后怎样预测新输入

```
model.eval()
with torch.no_grad():
    logits = model(Xnew)
    probabilities = torch.sigmoid(logits)
    predictions = (probabilities >= .5)
```

新输入

Xnew 保持B×2形状，使用固定训练后参数

概率与类别

需要概率时才调用sigmoid；阈值0.5将概率划为类别

标签角色

新输入预测不需要真实标签；计算损失或正确率时需要真实标签

独立测试与结果边界

- 固定100轮模型在128个独立测试样本上评价
  - 测试损失约 0.0767，正确率为 100.0%
- 这一结果支持本次构造分布、样本与配置下的表现
- 没有浅层对照、多次种子重复或真实应用测试
  - 因此不能据此断言深度普遍更好
- 数据划分与实验配置是结果解释的一部分

深层训练需要分别检查的问题

已建立的认识	仍需解决的问题	后续展开
已能计算梯度，本例用SGD更新	更新方向与步长怎样选?	下一讲：优化算法与学习率
梯度逐层经过权重与激活导数	重复变换怎样影响梯度尺度?	后续：训练稳定性与归一化
分别测量训练与独立评价表现	拟合改善是否伴随泛化改善?	正则化与诊断
全连接网络按层变换表示	怎样利用空间或序列关系?	CNN、序列与注意力模型

- 算出梯度不等于已经找到有效更新方式；深度也不必然导致梯度消失或爆炸，仍需结合权重、激活与实际记录判断

本讲总结

- 多层网络用逐层复合组织表示，反向机制沿相同依赖推广
- 批量把样本按行并列，参数共享，平均目标产生平均梯度
- 层与参数、logits与损失、梯度与更新在实现中各有对象
- 训练调整参数，验证与测试固定参数，预测使用新输入
- 深度、训练过程与泛化表现分别判断，并保留实验条件