02 神经网络的学习：反向传播与自动微分

机器学习 · 深度学习入门

学习目标

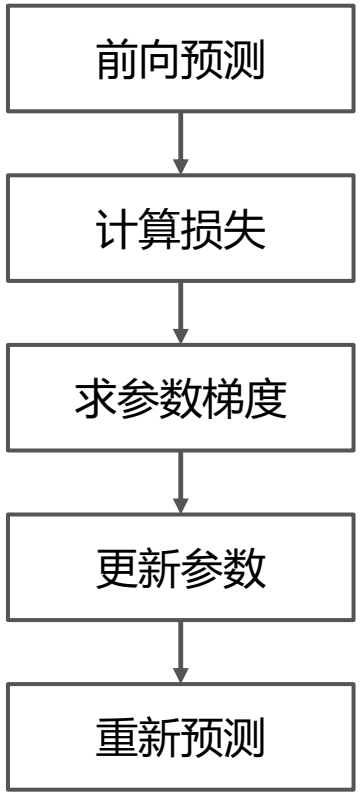
- 解释学习为什么需要参数梯度，以及复杂网络怎样组织求导
- 根据计算图，跟踪局部求导、逆序复用和分支贡献累加
- 跟踪小网络的反向计算、向量推广及一次参数更新
- 将手算对应到自动微分与训练循环，区分记录、缓冲和更新
- 说明有限差分如何用于梯度检查

1 从损失到参数梯度

损失需要参数梯度，计算图组织复合求导。

神经网络如何通过参数调整学习

- 反复调整权重与偏置，寻找训练损失较小的参数
- 通过前向计算得到模型输出，用它与真实标签计算损失
- 保持当前参数不变，计算损失对各参数的局部变化率
- 按梯度更新参数，再计算预测与损失，继续迭代
  - 学习率控制步长；损失评价当前预测，梯度提供调整依据



更新后重新计算损失，继续迭代

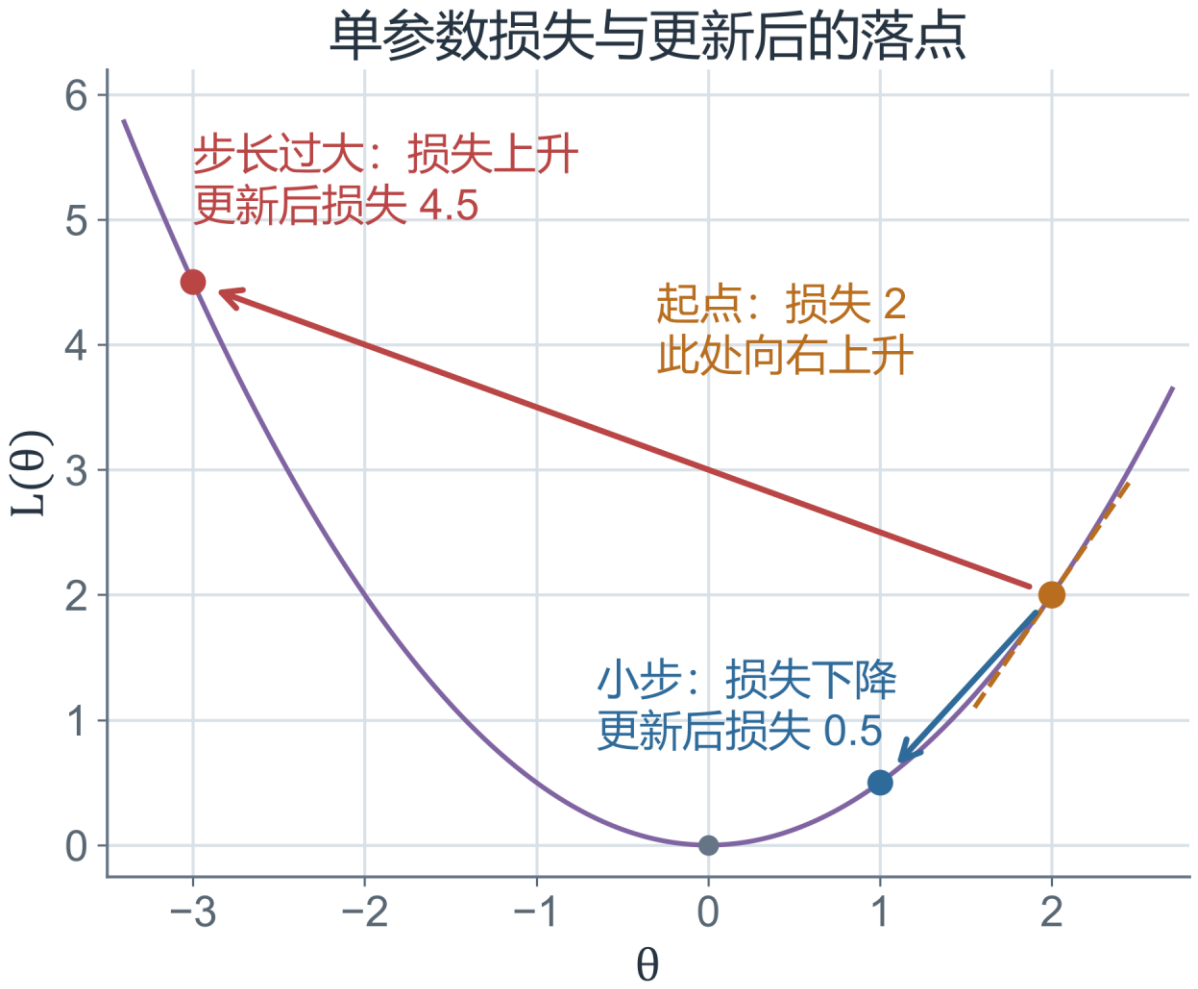
用损失最小化定义学习目标

- 在给定数据、网络结构和损失函数后，调整包含权重与偏置的模型参数 $\boldsymbol{\theta}$
- 训练损失为全部训练样本的平均损失
  - $L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$
  - $f_{\boldsymbol{\theta}}$：模型输出函数；ℓ：单样本损失
  - $\boldsymbol{x}_i, y_i$：第 i 个输入与标签；N：样本数
- 优化目标是使训练损失尽可能小，即 $min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$
  - 迭代不保证全局最优；训练损失小不直接保证新数据上的表现
- 后续先对一个固定样本的损失 ℓ 求导和更新，说明各计算环节；训练集平均损失仍按上式定义

梯度下降的直观解释

- 把损失看作地形高度，寻找损失较小的位置
- 在图示起点，梯度指向局部上升方向；减去梯度，向下降方向迈步
- 学习率控制步长；步长过大，损失仍可能上升
  - 确定了调整方向，还需要计算网络各参数的梯度

图示为单参数损失 $L(\theta) = \theta^2/2$，实际网络损失不一定呈碗状；箭头连接更新前后的点，不表示运动轨迹

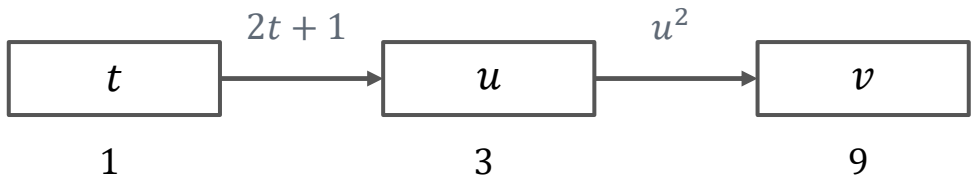


用梯度确定参数的调整方向

- **导数** $f'(t)$ 表示当前点的局部变化率，即曲线的切线斜率
  - 多参数损失只改变 θ_j、固定其他独立参数时，变化率为**偏导数** $\partial L/\partial\theta_j$
  - 当前点可微且变化足够小时，$\Delta L \approx (\partial L/\partial\theta_j)\Delta\theta_j$
- 各参数偏导组成**梯度**；m 为参数个数，分量与参数一一对应
  - $\nabla_{\boldsymbol{\theta}} L = (\partial L/\partial\theta_1, \ldots, \partial L/\partial\theta_m)^{\mathsf{T}}$
  - 向量或矩阵参数的梯度与参数同形
- 在当前参数处求梯度，沿**负梯度方向**同时调整参数
  - $\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}} L$，$\eta > 0$ 为**学习率**，控制步长
  - 偏导为正则减小参数，为负则增大参数；步长足够小时可使损失下降
- 参数只参与局部运算，计算图将组织同一损失对全部参数的求导

计算图表示运算及其依赖

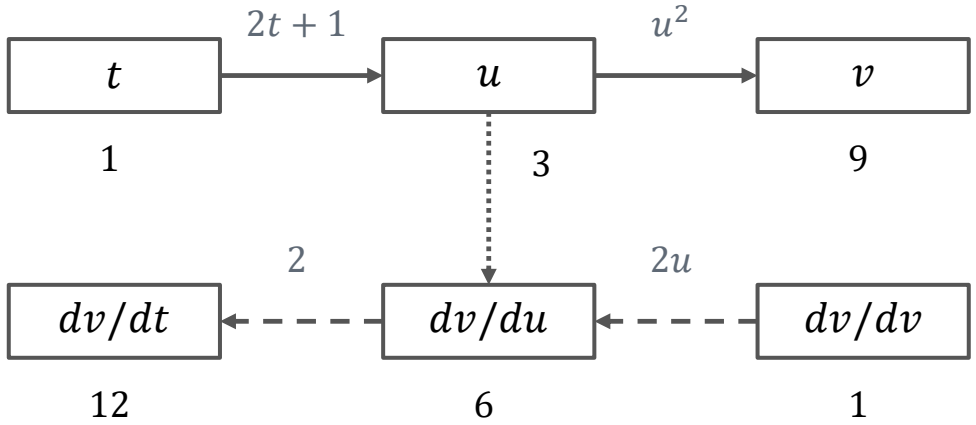
- **计算图**将一次计算拆成简单操作，用有向图表示它们的依赖
- 例：$u = 2t + 1$，$v = u^2$
  - 本图节点表示量，箭头上的表达式说明怎样由上游量计算下游量
  - $t \to u \to v$：先得到 u，才能计算 v
- 在 $t = 1$ 时，沿图前向得到 $u = 3$、$v = 9$
  - 保存当前中间值，后面求局部导数时会用到
- 网络结构图描述层与连接；计算图展开求导需要的运算依赖
  - 后面的反向贡献图另用箭头表示导数贡献；须结合图例区分箭头含义



节点表示量；边上标明运算或局部导数

链式法则沿依赖反向计算

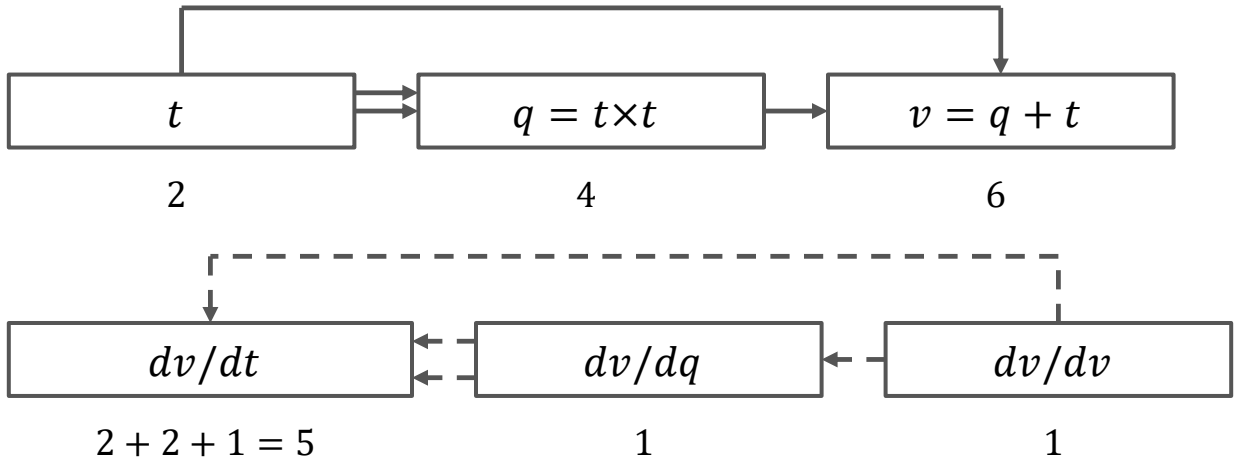
- 在前页的 $t = 1, u = 3$ 处
  - 两个局部导数为 $du/dt = 2$、$dv/du = 2u = 6$
- t 小幅变化 Δt，先引起 $\Delta u = 2\Delta t$
  - 再有 $\Delta v \approx 6\Delta u = 12\Delta t$
- 在当前点可微时，**链式法则**给出
  - $dv/dt = (dv/du)(du/dt) = 6 \times 2 = 12$
- 从 $dv/dv = 1$ 开始反向求导
  - 传给上游的导数 = 下游导数 × 对应局部导数
- 反向传递导数，前向值保持不变



点线：保存前向值，供局部求导调用

共享变量汇总各路径的梯度贡献

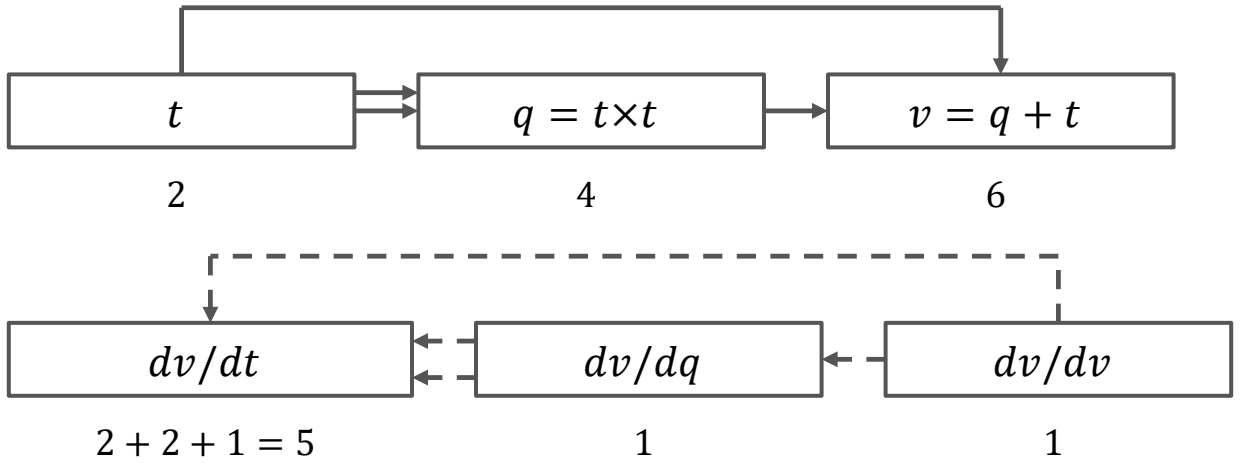
- 同一个 t 同时进入乘法的两个输入和直接相加项
  - $q = t \times t$，$v = q + t$；在 $t = 2$ 处，$q = 4, v = 6$
- 相加操作把收到的导数分别传给 q 和直接输入 t
  - 乘法再向两个输入各传回 $1 \times t = 2$
- 同一变量的所有影响共同形成总变化率：$dv/dt = 2 + 2 + 1 = 5$



同一变量的直接输入；两个输入各乘 $t = 2$；相加：局部导数1；直接路径贡献1

反向传播如何组织一次逆向求导

- **反向传播**从输出的导数种子1开始，按依赖逆序求导
  - 每个运算复用收到的下游导数，乘各输入的局部导数
- 共享变量汇总全部路径贡献后，再向它的上游传递
  - 图中相加与乘法各复用收到的1，同一 t 汇总2、2、1
- 逆序保证下游贡献先就绪；重复局部规则即可求各参数梯度，求导期间参数不变



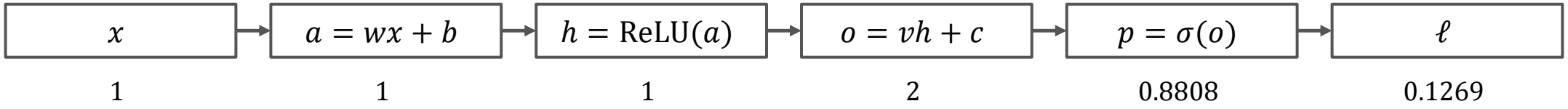
同一变量的直接输入；两个输入各乘 $t = 2$；相加：局部导数1；直接路径贡献1

2 反向梯度与参数更新

跟通小网络求导与一次更新，再推广向量梯度并解释激活门控。

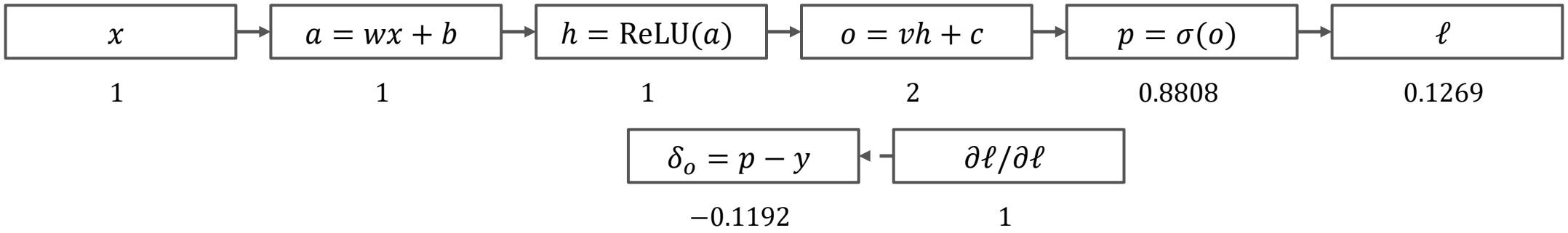
核心小网络的前向状态

- 核心网络的四个参数为 w, b, v, c，当前所有量均为标量
  - $a = wx + b$，$h = ReLU(a)$，$o = vh + c$，$p = \sigma(o)$
- 给定 $x = y = 1$，初始 $(w, b, v, c) = (1,0,2,0)$，前向值见图
  - $\ell = -y\ln p - (1-y)\ln(1-p) \approx 0.1269$
- 后两页固定这些前向值，先求得分梯度，再复用它求参数梯度



损失与输出得分的梯度

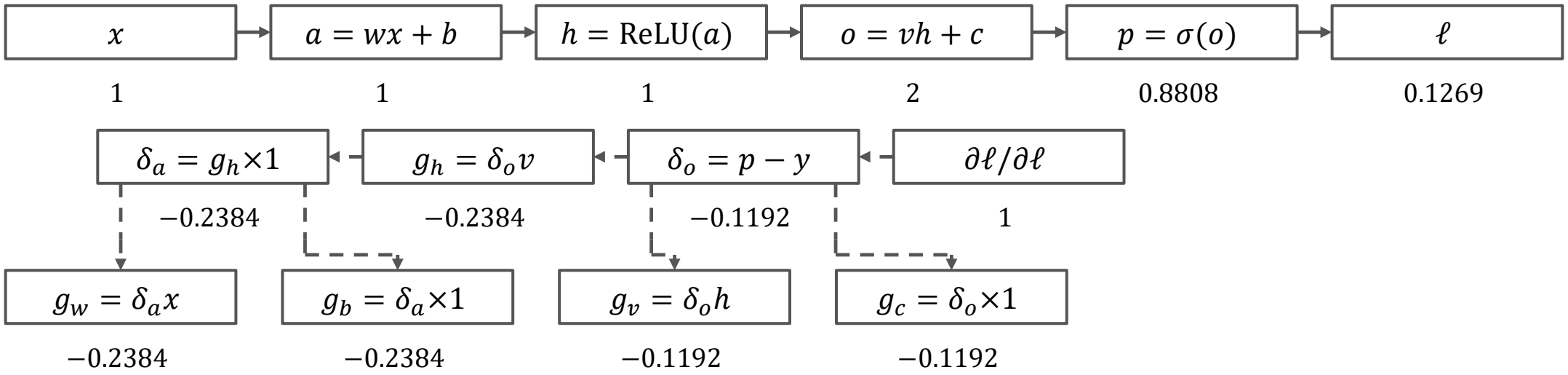
- Sigmoid与二分类交叉熵相接，输出得分梯度为 $\delta_o = \partial \ell / \partial o = p - y$
  - 当前 $\delta_o \approx -0.1192$：得分小幅增加时，损失局部减小
- v, c, h 都通过同一个得分 o 影响损失，可复用这一梯度
- 局部导数核对：$\partial \ell / \partial p = (p - y)/[p(1-p)]$，$dp/do = p(1-p)$
  - 二者相乘得到 $p - y$；这里 $0 < p < 1$，采用当前Sigmoid与交叉熵组合



Sigmoid + 交叉熵

同一下游梯度的复用

- 记 $g_z = \partial \ell / \partial z$；输出层按 $o = vh + c$ 复用 δ_o
  - 对 v 的局部导数是 h，对 c 是1，对 h 是 v
- 隐藏层先经过ReLU：$\delta_a = g_h ReLU'(a)$，当前 $a = 1 > 0$
  - 再按 $a = wx + b$ 传给 w, b，局部导数分别为 $x, 1$
- 图中虚线路径得到各量的损失梯度；四个参数梯度交给下一步更新



局部规则所用值：ReLU导数为1，输入x=1，隐藏值h=1，输出权重v=2；参数梯度；Sigmoid + 交叉熵

一次更新使用同一状态下的全部梯度

- 接续上一单元的单隐藏单元核心例：$\theta = (w, b, v, c)$
  - 更新前 $\theta = (1,0,2,0)$，$\boldsymbol{g} \approx (-0.2384, -0.2384, -0.1192, -0.1192)^{\top}$
  - $\boldsymbol{g}$ 只含四个参数的梯度；g_h, δ_a 是中间量梯度，不参与参数更新
- 按 w, b, v, c 顺序将参数展开为列向量 $\boldsymbol{\theta}$
- 取 $\eta = 0.1$，按 $\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \boldsymbol{g}$ 更新
  - 例如 $w' = 1 - 0.1(-0.2384) \approx 1.0238$
- 四个梯度先在同一参数状态求完，再用于本次更新
  - 不能更新一个参数后，又将新状态求出的梯度混入同一步
- 在当前点可微时，$\Delta \ell \approx \boldsymbol{g}^{\top} \Delta \boldsymbol{\theta}$
  - 取 $\Delta \boldsymbol{\theta} = -\eta \boldsymbol{g}$，则 $\Delta \ell \approx -\eta \|\boldsymbol{g}\|^2$
  - 梯度非零且步长足够小时，损失局部下降；有限步长的结果仍须重算

更新后的损失来自新的前向计算

- 对核心例的同一 $x = 1, y = 1$，四个参数统一更新

量	更新前	更新后
隐藏权重 w	1.0000	1.0238
隐藏偏置 b	0.0000	0.0238
输出权重 v	2.0000	2.0119
输出偏置 c	0.0000	0.0119
输出得分 o	2.0000	2.1198
损失 ℓ	0.1269	0.1134

- 参数更新改变模型；随后重新计算 a', h', o', p', ℓ'
  - 一阶近似用于解释方向，表中损失来自完整重新前向
- 此处只验证当前样本的一次更新，不代表训练集损失或新数据表现必然改善

向量形式表达同一局部求导关系

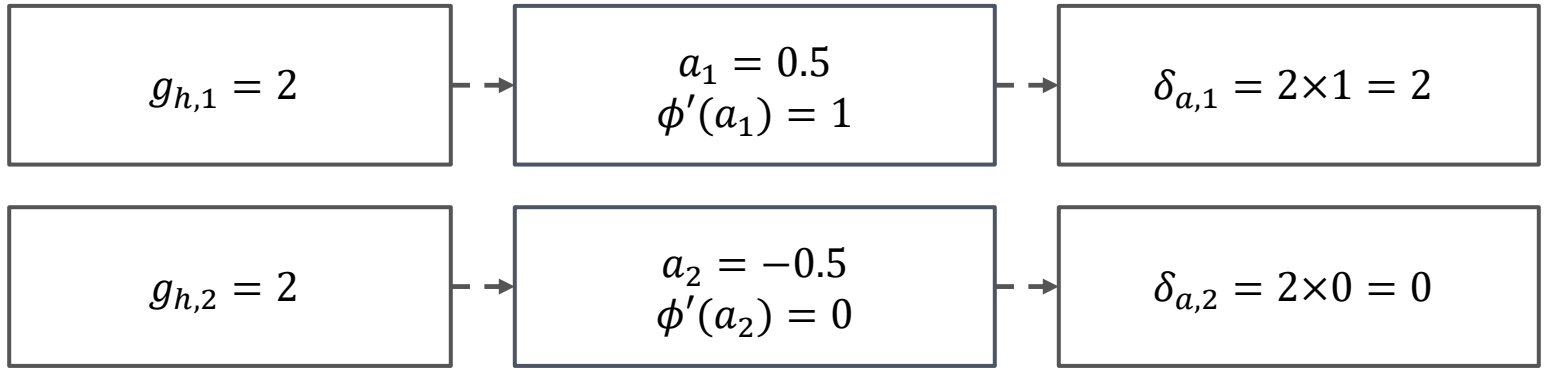
- 将标量参数 w, b, v, c 分别推广为 $\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, b_2$，单输出得分仍为标量
  - $\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$，$\boldsymbol{h} = \phi(\boldsymbol{a})$，$o = \boldsymbol{W}_2 \boldsymbol{h} + b_2$
- 输入有 d 个分量，隐藏层有 m 个单元；$\boldsymbol{W}_1 \in \mathbb{R}^{m \times d}$
  - 行 j 对应接收输入的隐藏单元，列 r 对应输入分量：$a_j = \sum_r W_{1,jr} x_r + b_{1,j}$
  - 记 $\delta_{a,j} = \partial \ell / \partial a_j$，则对应权重梯度为 $\partial \ell / \partial W_{1,jr} = \delta_{a,j} x_r$
  - 逐项配对得到**外积** $\nabla_{\boldsymbol{W}_1} \ell = \boldsymbol{\delta}_a \boldsymbol{x}^\top$，$\nabla_{\boldsymbol{b}_1} \ell = \boldsymbol{\delta}_a$；梯度与参数同形

	x_1	x_2
$\delta_{a,1}$	$\partial \ell / \partial W_{1,11}$ $\delta_{a,1} x_1$	$\partial \ell / \partial W_{1,12}$ $\delta_{a,1} x_2$
$\delta_{a,2}$	$\partial \ell / \partial W_{1,21}$ $\delta_{a,2} x_1$	$\partial \ell / \partial W_{1,22}$ $\delta_{a,2} x_2$

行：m个隐藏单元的梯度；列：d个输入分量；图示m=d=2，各交叉格为对应权重梯度

ReLU的局部导数控制梯度通过

- 向量回传逐分量执行同一规则：$\boldsymbol{\delta}_a = \boldsymbol{g}_h \odot \phi'(\boldsymbol{a})$
  - $\boldsymbol{g}_h = \nabla_{\boldsymbol{h}}\ell$ 为输出层传回的梯度；$\odot$ 表示逐元素乘积，两侧梯度均为 m 维向量
- 控制收到的梯度相同：两个位置均取 $g_{h,j} = 2$，只比较ReLU所在区间
  - 正区局部导数为1，梯度通过；负区局部导数为0，梯度被阻断
- 在 $a = 0$ 处ReLU不可导；PyTorch在该点采用导数为0的约定



相同的下游梯度；由当前预激活决定；正区通过，负区阻断

零梯度可以来自激活门控或零输入

- 延续门控对照：$\boldsymbol{\delta}_a = (2,0)^\top$，取输入 $\boldsymbol{x} = (1,0)^\top$
  - 外积 $\nabla_{\boldsymbol{W}_1}\ell = \boldsymbol{\delta}_a \boldsymbol{x}^\top = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$
- 第二行全零来自激活门控；第一行第二项为零来自输入 $x_2 = 0$
  - 同为零，发生位置和原因不同；这些结论只描述当前样本与状态
- 输出层仍沿用标量局部规则，$\boldsymbol{W}_2 \in \mathbb{R}^{1\times m}$
  - $\boldsymbol{g}_h = \boldsymbol{W}_2^\top \delta_o$，$\nabla_{\boldsymbol{W}_2}\ell = \delta_o \boldsymbol{h}^\top$，$\partial\ell/\partial b_2 = \delta_o$

	$x_1 = 1$	$x_2 = 0$	
$\delta_{a,1} = 2$	$2\times 1 = 2$	$2\times 0 = 0$	零输入使第二列归零
$\delta_{a,2} = 0$	$0\times 1 = 0$	$0\times 0 = 0$	门控使第二行归零

3 自动微分与学习循环

程序组织同一求导过程，循环区分缓冲、计算与更新。

自动微分为基本运算配备局部规则

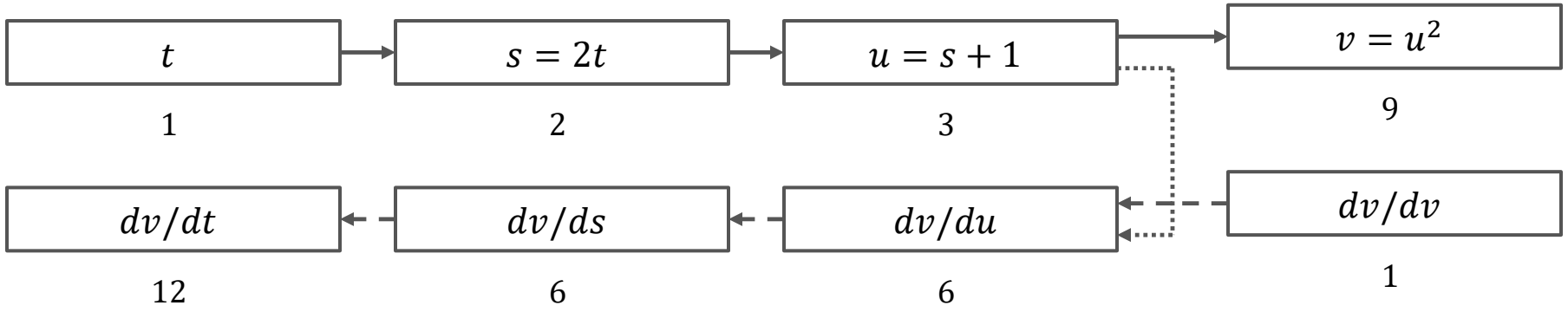
- **自动微分**把程序中的基本运算与局部求导规则配对，按链式法则组合导数
- 熟悉的 $u = 2t + 1, v = u^2$ 可拆为三个基本运算；每种运算的规则可重复调用

运算记录	输入依赖	反向规则所需信息
$s = 2t$	t	乘常数2，局部导数为2
$u = s + 1$	s	加常数1，局部导数为1
$v = u^2$	u	保存当前 u，局部导数为 $2u$

- 系统组合已有局部规则，不必为每个新组合手写整体导数
- 自动微分不采用有限扰动估计，没有有限差分截断误差，仍有浮点误差

程序如何记录依赖并调用反向规则

- 在PyTorch中，前向执行建立本次运算依赖，并保留反向所需数值
  - 平方运算需要当前 $u = 3$；乘常数与加常数按各自规则处理
- **反向模式自动微分**从标量输出的导数种子1开始，逆序调用规则
  - 每次把下游导数乘局部导数，写回对应输入；共享输入的贡献相加
- 神经网络的反向传播采用这套计算组织；参数更新由优化器另行执行



乘常数规则：×2；加常数规则：×1；平方规则：×$2u$；保存 $u = 3$

实线：前向依赖；点线：保存值供规则调用；虚线：逆序写回输入梯度

核心小网络在代码中对应同一组量

```python
import torch
import torch.nn.functional as F
dtype = torch.float64
x = torch.tensor(1., dtype=dtype)
y = torch.tensor(1., dtype=dtype)
w = torch.tensor(
    1., dtype=dtype, requires_grad=True)
b = torch.tensor(
    0., dtype=dtype, requires_grad=True)
v = torch.tensor(
    2., dtype=dtype, requires_grad=True)
c = torch.tensor(
    0., dtype=dtype, requires_grad=True)
```

样本与参数

重新建立更新前的 (w,b,v,c)=(1,0,2,0)，与前述手算示例采用相同参数

x,y 分别是固定输入和标签；四个参数是直接创建、而非由已有张量运算得到的标量**叶张量**

梯度与精度设置

requires_grad 声明参数需要梯度；不表示参数会自动更新

float64 用于数值核对，不是所有训练都必须采用的设置

backward 沿当前运算依赖计算梯度

```
a = w * x + b
h = torch.relu(a)
o = v * h + c
loss = F.binary_cross_entropy_with_logits(
    o, y)
loss.backward()
```

前向计算

三行前向分别对应预激活、隐藏表示和输出得分

binary_cross_entropy_with_logits 接收得分 o 和标签 y

内部以稳定形式实现 Sigmoid 配二分类交叉熵，不先重复调用 Sigmoid

backward 使用本次前向的依赖与局部导数

四个叶参数的 grad 保存梯度；它们的参数值仍保持更新前状态

手算中的 g_h,δ_a 是中间梯度

默认不必把它们保存在中间张量的 grad 中，仍可用于反向计算

用相同状态核对手算与程序梯度

- 核对条件：同一核心网络、$x = y = 1$、更新前参数 $(1, 0, 2, 0)$

参数	手算梯度	程序 grad 输出
w	−0.238406	−0.238406
b	−0.238406	−0.238406
v	−0.119203	−0.119203
c	−0.119203	−0.119203

- 程序列读取四个参数的 grad；本次 CPU float64 结果与手算一致
- 结果不一致时，先核对样本、参数状态、损失定义与梯度缓冲
- 表中数值已舍入；实际核对使用未舍入值与数值容差

图内贡献求和与梯度缓冲累积

- 一次图内求导：$t \times t + t$ 的三条贡献在同一 t 处相加
  - 得到的是当前输出对这个变量的完整导数
- 多次 backward：每次得到的叶参数梯度可继续加到已有 grad
  - 即使重新前向形成新图，已有 grad 仍可能保留

对比	图内路径贡献	跨调用梯度缓冲
相加对象	同一输出经不同路径的导数贡献	不同反向调用写入的结果
本讲处理	按链式法则必须汇总	独立更新步开始时清零

- 独立更新步开始时清零；多个小批量有意累积时，一个周期内保留grad
  - 同一图反向后中间保存量通常释放；再次计算时重新前向
- 清空grad不改变参数；累积周期结束后再更新参数并清零

一次学习迭代的职责

```
params = [w, b, v, c]
optimizer = torch.optim.SGD(params, lr=0.1)
optimizer.zero_grad(set_to_none=True)
a = w * x + b
o = v * torch.relu(a) + c
loss = F.binary_cross_entropy_with_logits(o,y)
loss.backward()
optimizer.step()
```

清零与前向

清空旧梯度，再用当前参数前向计算损失

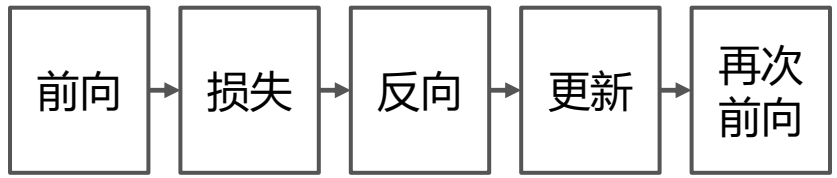
梯度与更新

backward 计算梯度，step 用梯度更新参数

循环接口

前两行初始化参数列表与优化器；每轮重复从 zero_grad 到 step 的步骤

下一轮使用更新后参数重新前向；第三讲扩展批量与验证



梯度记录与梯度缓冲的职责

- 张量是否需要梯度由 requires_grad 声明；本例叶参数的 grad 保存反向结果
- 本模型在 **no_grad** 内执行前向或手动更新时，相应运算不记录到反向求导图
  - 预测只需前向结果，可在 no_grad 中执行；手动更新也可在其中进行
- 本例调用 **zero_grad(set_to_none=True)**，将叶参数的 grad 设为 None
  - 下一次 backward 写入当前梯度；no_grad 与重新前向均不会清空旧 grad

反向计算的代价与有限差分检查

- 反向计算复用当前前向的中间值及依赖，需要相应的存储与图遍历
- **中央有限差分**根据两次扰动后的损失估计单个偏导
  - $\partial \ell / \partial \theta_j \approx [\ell(\boldsymbol{\theta} + \varepsilon \boldsymbol{e}_j) - \ell(\boldsymbol{\theta} - \varepsilon \boldsymbol{e}_j)]/(2\varepsilon)$
  - $\boldsymbol{e}_j$ 仅在第 j 个分量为 1，其余为 0；$\varepsilon > 0$ 为扰动尺度，其他参数固定
- 对核心例更新前的隐藏权重 w，取 $\varepsilon = 10^{-6}$，检查值约为 -0.2384，与手算相符
- 检查全部 m 个参数时，逐参数中央差分通常需约 $2m$ 次损失计算，还受扰动尺度影响
  - 因此有限差分用于小规模核对；训练采用复用中间结果的反向模式自动微分

本讲总结

- 学习需要参数梯度，复杂网络需要把整体求导组织为可复用的局部计算
- 链式法则连接局部导数，共享变量将各路径的贡献相加
- 反向传播从损失出发逆序计算，使不同参数复用已有的下游梯度
  - 向量外积与激活门控仍遵循相同局部关系
- 自动微分按计算依赖组合基本运算的局部求导规则，计算当前点的导数值
  - 反向模式从标量输出逆序传播并汇总贡献；框架据此组织网络求导
  - 梯度记录、缓冲累积和参数更新各有职责；有限差分提供独立数值核对
- 更新参数后重新前向，得到新状态下的损失与梯度，形成学习循环