### 反向传播（Backpropagation）核心原理总结

# 一、反向传播的本质与目标

反向传播是神经网络训练中梯度下降的高效实现算法，核心目标是**快速计算高维参数的梯度**。当神经网络参数达百万级时，直接计算梯度效率极低，反向传播通过**链式法则**优化计算流程，将梯度计算复杂度降至与网络前向传播相当。

# 二、数学基础：链式法则的应用

**单变量链式法则**：若 *z*=*h*(*g*(*x*))，则 *dxdz*​=*dydz*​⋅*dxdy*​（*y*=*g*(*x*)）。

**多变量链式法则**：若 *z*=*k*(*x*,*y*)，且 *x*=*g*(*s*)，*y*=*h*(*s*)，则 *dsdz*​=*dxdz*​⋅*dsdx*​+*dydz*​⋅*dsdy*​。

**关键**：梯度可沿变量依赖路径分解为各层偏导的乘积或求和，这是反向传播的理论基础。

# 三、神经网络中的梯度计算流程

## 损失函数与梯度目标：

总损失 *L*=∑*n*​*Cn*，其中 *Cn* 为单样本预测值 *yn* 与真实值 *y*^​*n* 的差异（如交叉熵、均方差）。

目标是计算 ∂*w*∂*L*​=∑*n*​∂*w*∂*Cn*​，即先求单样本梯度再累加。

## 前向传播（Forward Pass）：计算 ∂*w*∂*z*​

*z* 为神经元输入（如 *z*=*x*1​*w*1​+*x*2​*w*2​+*b*），则 ∂*wi*​∂*z*​=*xi*​，即偏导数等于权重连接的输入值。

**过程**：将输入数据代入网络，逐层计算各神经元的输出，为反向传播准备中间值。

## 反向传播（Backward Pass）：计算 ∂*z*∂*C*​

设神经元激活函数为 *σ*(*z*)，输出为 *a*=*σ*(*z*)，则 ∂*z*∂*a*​=*σ*′(*z*)（如 sigmoid 的导数为 *σ*(*z*)(1−*σ*(*z*))）。

*a* 影响后续层的输入 *z*′ 和 *z*′′（如 *z*′=*a*⋅*w*3​+…），则 ∂*a*∂*C*​=∂*a*∂*z*′​⋅∂*z*′∂*C*​+∂*a*∂*z*′′​⋅∂*z*′′∂*C*​=*w*3​⋅∂*z*′∂*C*​+*w*4​⋅∂*z*′′∂*C*​。

**核心公式**：∂*z*∂*C*​=*σ*′(*z*)⋅(*w*3​⋅∂*z*′∂*C*​+*w*4​⋅∂*z*′′∂*C*​)，即当前层梯度由后续层梯度加权求和后乘以激活函数导数。

# 四、反向传播的执行逻辑

**反向计算梯度**：从输出层开始，逆网络结构逐层推导各层神经元的 ∂*z*∂*C*​。

输出层的 ∂*z*∂*C*​ 可直接通过损失函数和激活函数导数计算（如 softmax + 交叉熵时，∂*z*∂*C*​=*y*−*y*^​）。

前层梯度依赖后层梯度，通过权重矩阵传递，形成类似 “反向神经网络” 的计算流程（激活函数替换为导数形式）。

**效率优势**：前向传播与反向传播的计算量相当，均为 *O*(*N*)（*N* 为网络参数数量），避免了直接计算高维梯度的指数级复杂度。

# 五、关键结论

反向传播的本质是**利用链式法则将梯度分解为可逐层计算的偏导项**，通过前向传播存储中间激活值，反向传播时逆向推导梯度，从而高效实现神经网络的参数优化。现代深度学习框架（如 PyTorch）已自动实现该过程，无需手动计算，但理解其原理是掌握神经网络训练的核心。