首都师范大学 为学为师 求实求新

深度学习应用与工程实践
5. 神经网络--前向/后向、训练
5. Neural Network, Forward and Backward propagation, training

李冰
Bing Li
Tenure-track Associate Professor
Academy of Multidisciplinary Studies
Capital Normal University

回顾

深度学习应用

计算机视觉 (CV)

- MLP
- CNN
- UNet
- YoLo
- MobileNet

- CNN架构的基本构成模块
- CNN设计原则

自然语

- RNN
- LSTM/GRU
- **Transformer**

生成式模型

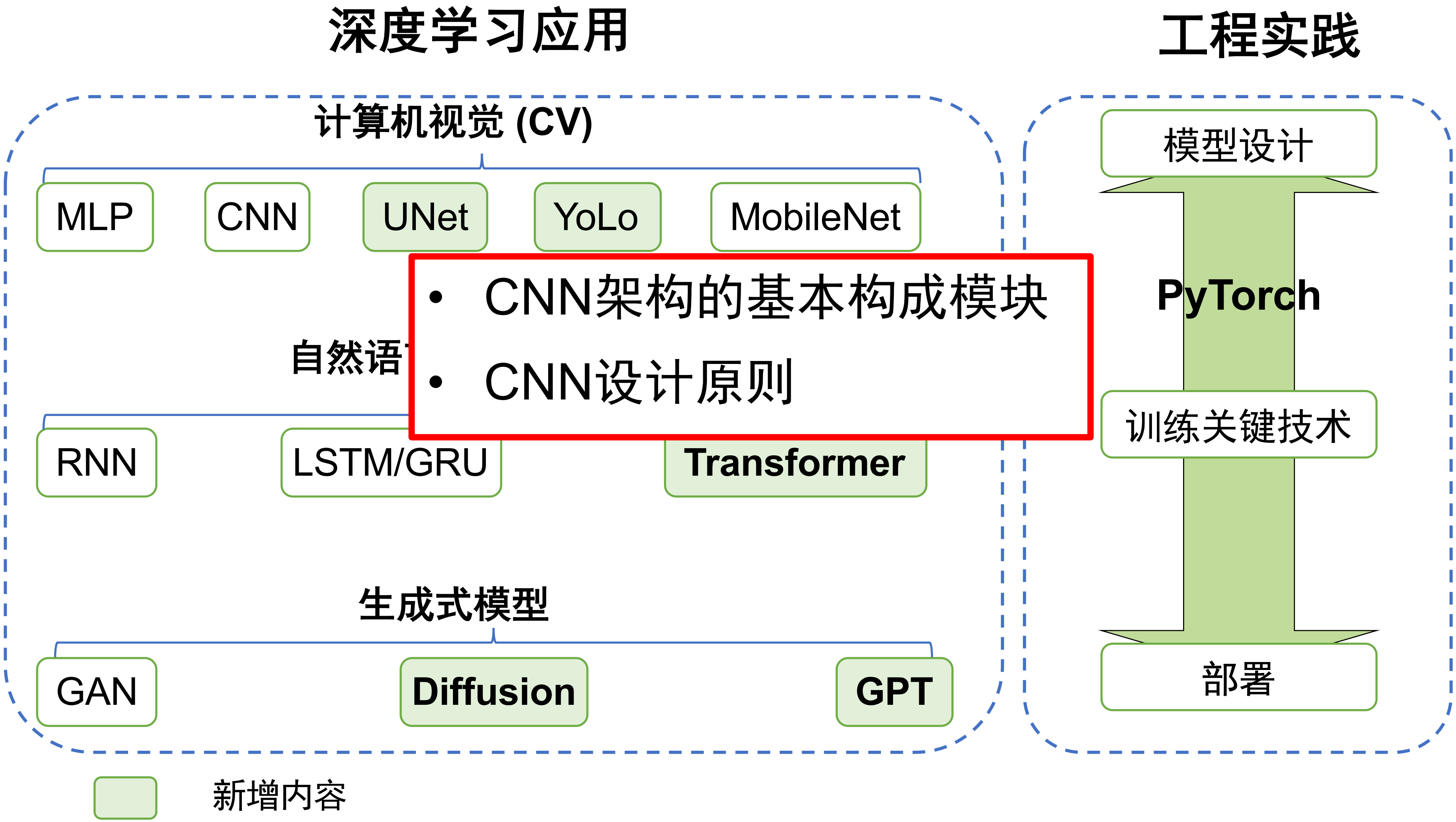
- GAN
- **Diffusion**
- **GPT**

工程实践

- 模型设计
- **PyTorch**
- 训练关键技术
- 部署

新增内容

这节课


深度学习应用
计算机视觉 (CV)
MLP
CNN
UNet
YoLo
MobileNet
自然语
RNN
LSTM/GRU
Transformer
生成式模型
GAN
Diffusion
GPT
工程实践
模型设计
PyTorch
训练关键技术
部署
新增内容
• CNN架构的基本构成模块
• CNN设计原则

Outline: Part 1

- **深度学习目标**
- 线性 & 非线性网络层
  - 线性网络层: 线性神经元
  - 非线性网络层: Logistic (对数比率) 神经元
- 后向传播
  - 后向传播的介绍
  - 非线性网络层的梯度
  - 求导的细节
- 学习 (训练)问题

回顾: 机器学习的目标

- 找到一个适合的拟合函数.
- 监督学习: F*(输入) → 标签
- 非监督学习: F*(输入) → 相似度
- 强化学习: F*(状态, 动作) → 价值

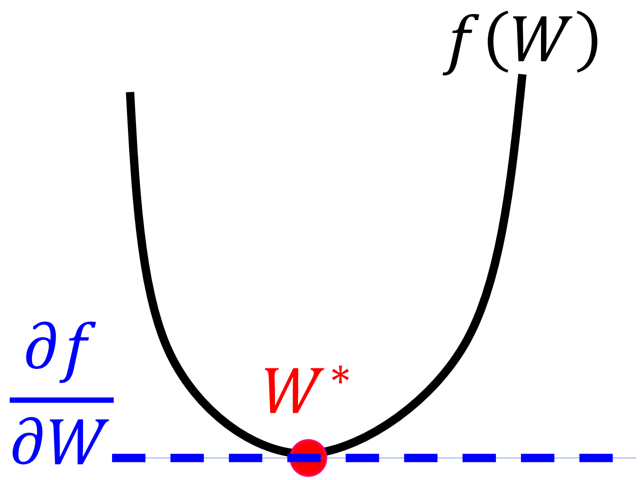
- **损失函数**: 关于模型**权重**W的函数
  - 当前模型 F_W输出与目标值之间的**误差**F^*：$L\big(F_W(x), F^*(x)\big)$
  - 下节课将介绍**常用的损失函数**
- 最小化**误差**的过程叫做训练 **Training**

函数的最优化问题

- 现在考虑这个函数只有一个最小值的情况
- **二次函数的最小值**
  - 一阶导数为零的位置（梯度为零）
- 求 f 对 W 的偏微分，得到 W^*：

$$\frac{\partial f}{\partial W}(\mathrm{W}^*) = 0$$

$$\nabla_w = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \cdots, \frac{\partial}{\partial w_n}\right]^T$$

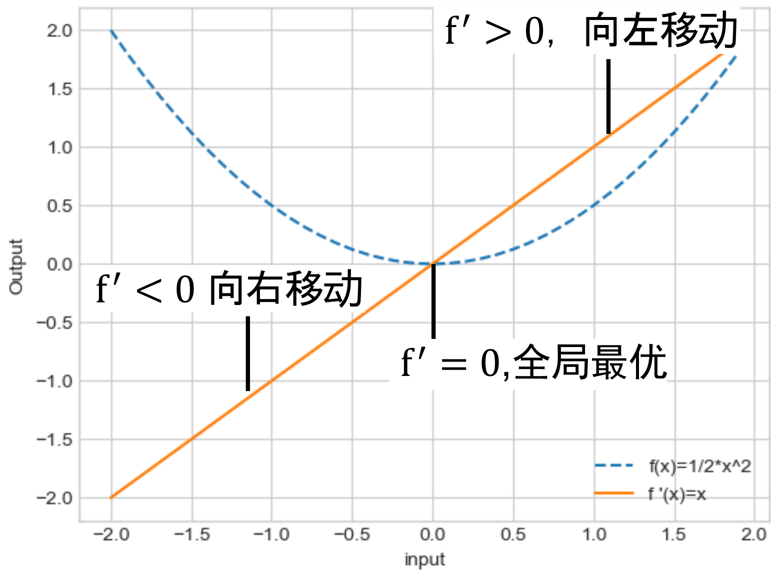


梯度下降法

将x往导数相反的方向移动一小步来减小f(x)

$$f(x) = \frac{1}{2}x^2$$

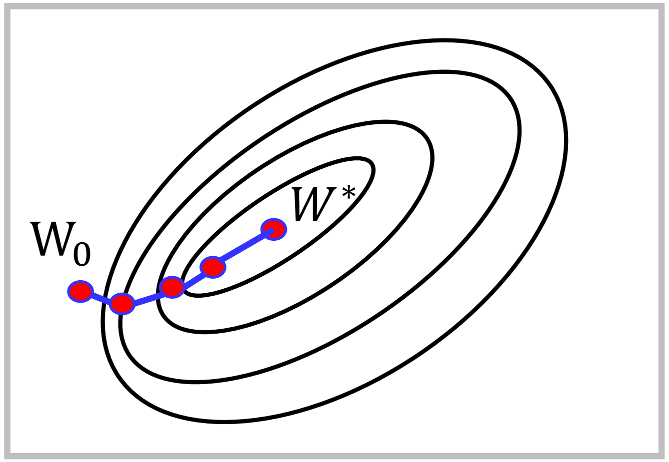
$$f'(x) = x$$



沿着函数下坡的方向直到最小

函数的最优化问题

- 假如函数非常复杂
  - 偏微分方程没有直接的解析解
  - 用偏导
- 用迭代的方法去寻找W^*



$$W^{n+1} = W^n - r\frac{\partial f}{\partial W}(W^n)$$

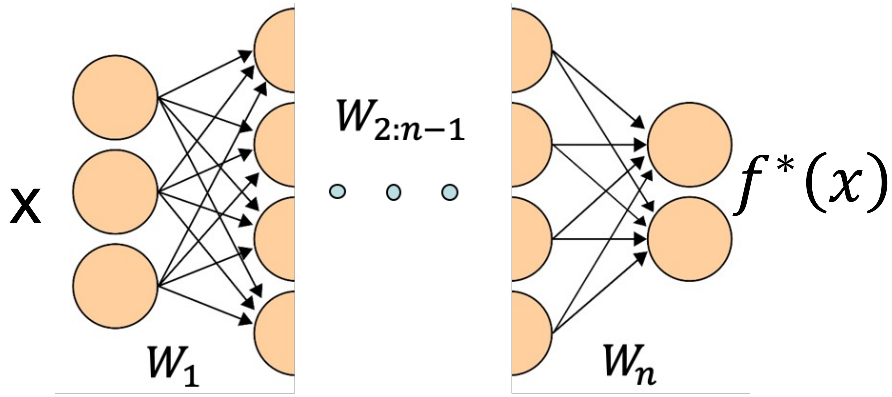
r: 学习率, 定义步长

- 这个过程就是**梯度下降(Gradient Descent)**

“考虑一座在（x1, x2) 点高度是 f(x1, x2) 的山。那么，某一点的梯度方向是在该点坡度最陡的方向. ”

神经网络的例子

- 神经网络有很多层，逐层计算
- 每一层 i ：函数$f_i(W_i,\cdot)$
  - 函数是线性或者非线性
- 整个网络: 所有层联合起来



- 给一个输入，求得输出
  - **前向传播**

    $y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))$
- 计算梯度（训练）
  - **后向传播** $\partial L\big(F(x), F^*(x)\big)/\partial W_i$

Outline: Part 2.1

- 深度学习目标
- 线性 & 非线性网络层
  - **线性网络层: 线性神经元**
  - 非线性网络层: Logistic (对数比率) 神经元
- 后向传播
  - 后向传播的介绍
  - 非线性网络层的梯度
  - 求导的细节
- 学习 (训练)问题

线性神经元

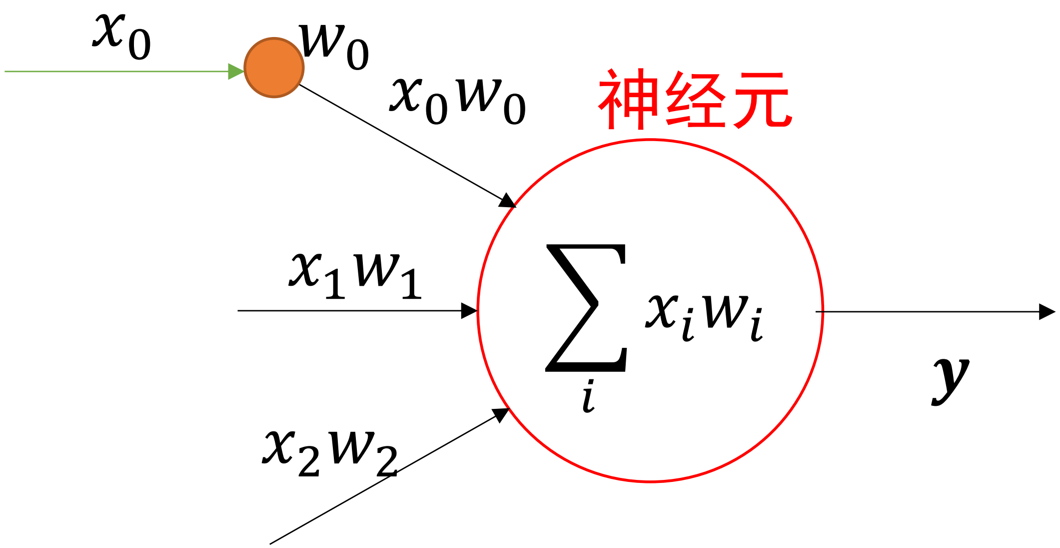
- 线性神经元是输入的加权和。
- **学习的目标是最小化在所有训练样本的误差和.**
  - 误差可以实际输入和目标输入之间的**均方差**.

权重向量

$$y = \sum_i x_i w_i = \mathbf{w}^T \mathbf{x}$$

估计的输出

输入向量



简单的例子

- 线性神经元迭代优化的例子.
- 每天中午在餐馆吃午饭.
  - 午餐有 **肉**,**素菜**, 和 **主食**.
  - 每天拿到几份.
- 收银员只告诉你总价。
  - 几天后, 差不多能猜出来每一小份的价格。
  - 迭代的方法是: 你开始随机猜一个价格，逐渐调整，直到与实际的价格最接近。

怎么解

- 餐费是每份菜的一个线性组合:

$$\textit{价格} = x_{\text{肉}}w_{\text{肉}} + x_{\text{素菜}}w_{\text{素菜}} + x_{\text{主食}}w_{\text{主食}}$$

- 单价是线性神经元中的权重:

$$\mathbf{w} = (w_{\text{肉}}, w_{\text{素菜}}, w_{\text{主食}})$$

- 每次点餐要的份额x是线性神经元中的输入

$$\mathbf{x} = (x_{\text{肉}}, x_{\text{素菜}}, x_{\text{主食}})$$

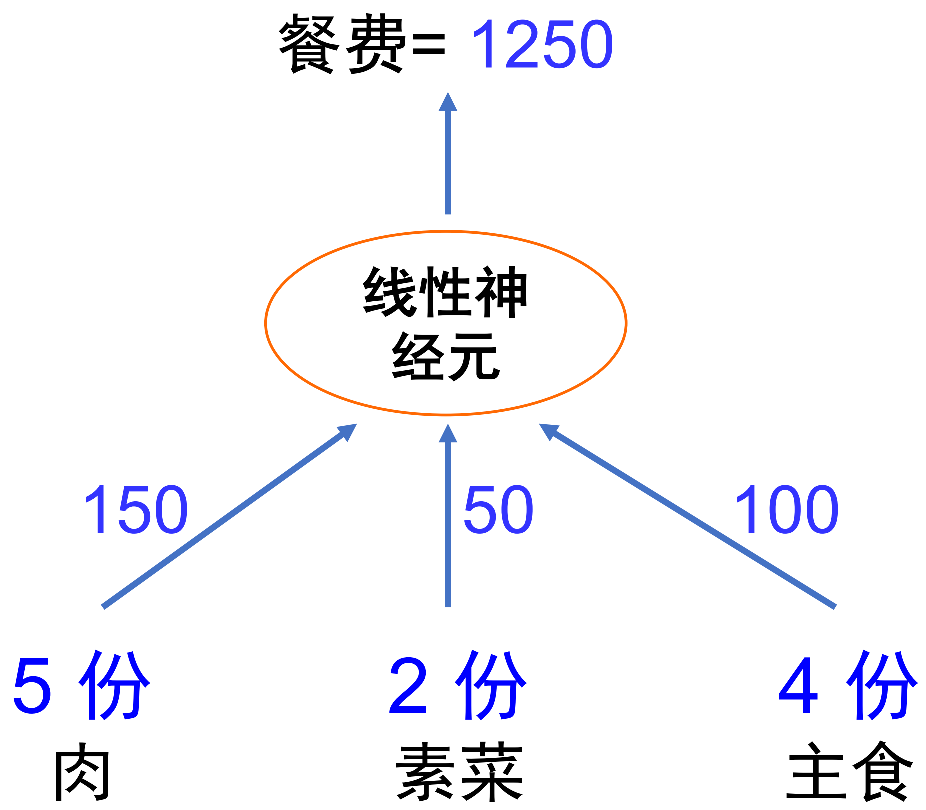
- 我们猜权重，微调权重直到接近收银员收的餐费价格。

收银员用的实际权重值

- 我们有数据集:

- 输入I:
  (# 肉, # 素菜, # 主食)
- 目标值T:
  总餐费

样本数:
  - **I: (5, 2, 4), T: 1250**
  - I: (3, 3, 3), T: 900
  - I: (0, 5, 1), T: 350
  - …



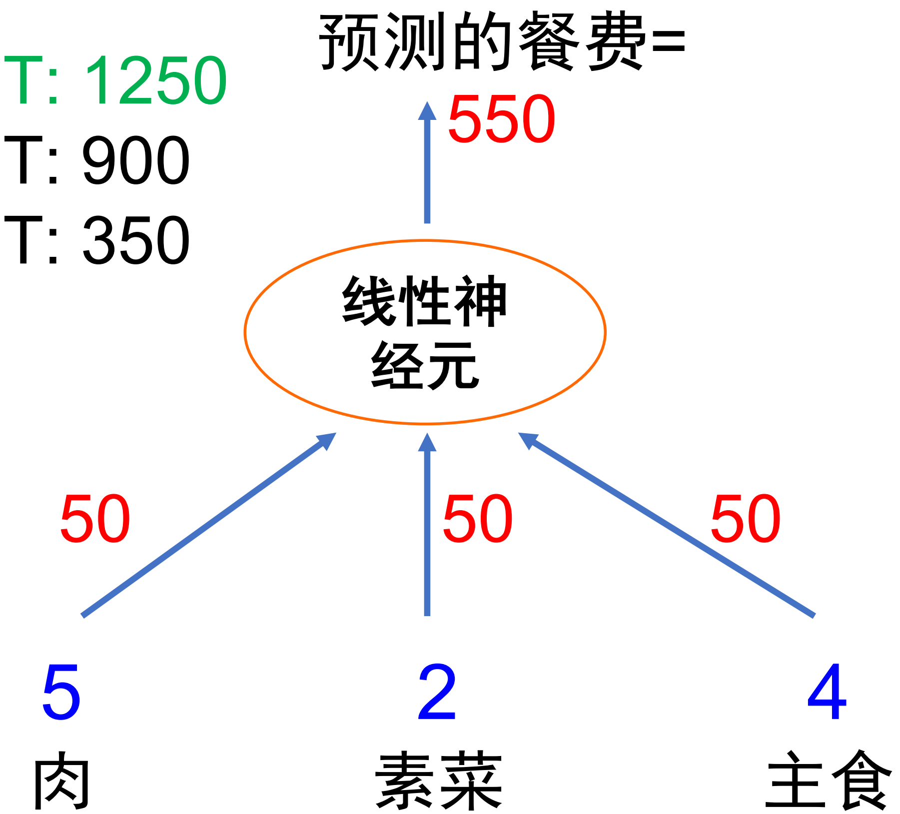
模型最开始是随机初始权重

- 误差

  = 目标 – 预测

  = 1250 – 550 = 700

I: (5, 2, 4), T: 1250
I: (3, 3, 3), T: 900
I: (0, 5, 1), T: 350



- 迭代的学习规则:

  $\Delta w_i = \varepsilon\, x_i\,(t - y)$

  - (t-y) = 700
  - $X_{肉} = 5$
  - $X_{素菜} = 2$
  - $X_{主食} = 4$

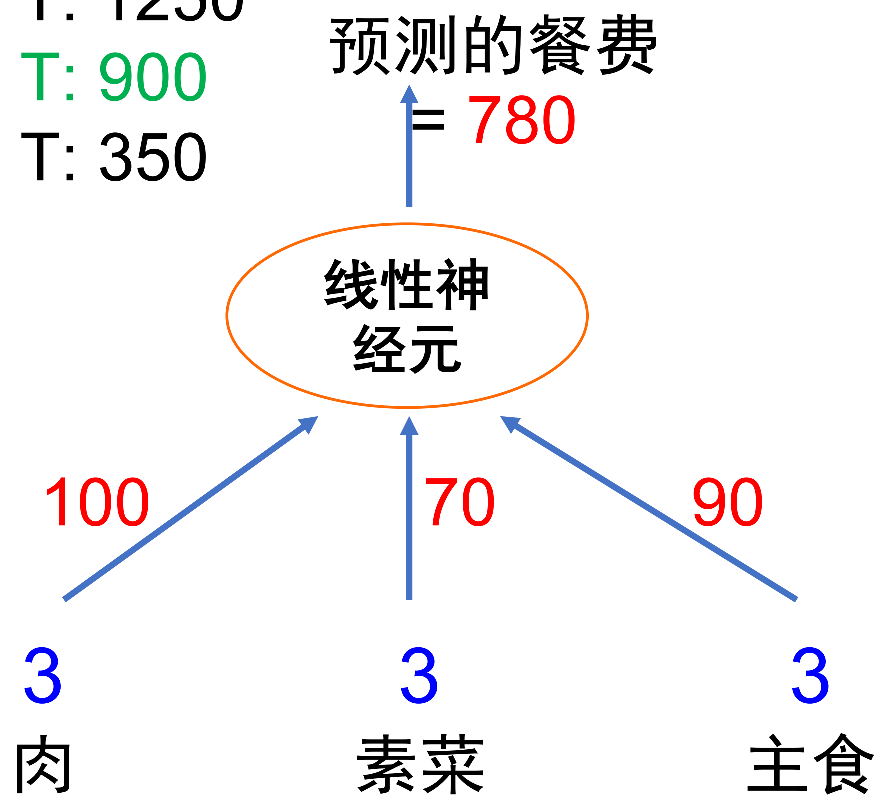
- 学习率 ε =1/70,
- 权重的修改量为:

  $\Delta w_{肉} = +50$, $\Delta w_{素菜} = +20$, $\Delta w_{主食} = +40$.

模型最开始是随机初始权重

I: (5, 2, 4), T: 1250
I: (3, 3, 3), T: 900
I: (0, 5, 1), T: 350

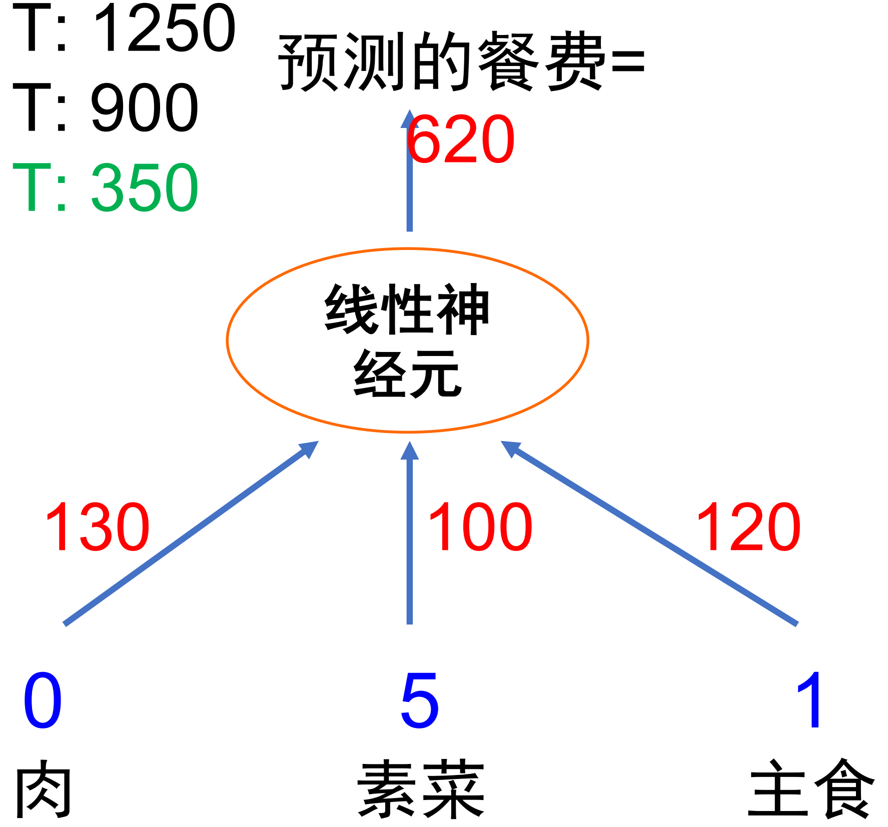
- 误差
  = 目标 – 预测
  = 900 – 780 = 120
- 迭代学习规则:
  $\Delta w_i = \varepsilon\, x_i\, (t - y)$
  - (t-y) = 120
  - $X_{肉} = 3$
  - $X_{素菜} = 3$
  - $X_{主食} = 3$
- 学习率 ε 是1/12,



- 权重的修改量为:
  $\Delta w_{肉} = +30$, $\Delta w_{素菜} = +30$, $\Delta w_{主食} = +30$.

模型最开始是随机初始权重

I: (5, 2, 4), T: 1250
I: (3, 3, 3), T: 900
I: (0, 5, 1), T: 350



- 误差

  = 目标 – 预测

  = 350 – 620 = -270

- 迭代学习规则:

  $\Delta w_i = \varepsilon x_i (t - y)$

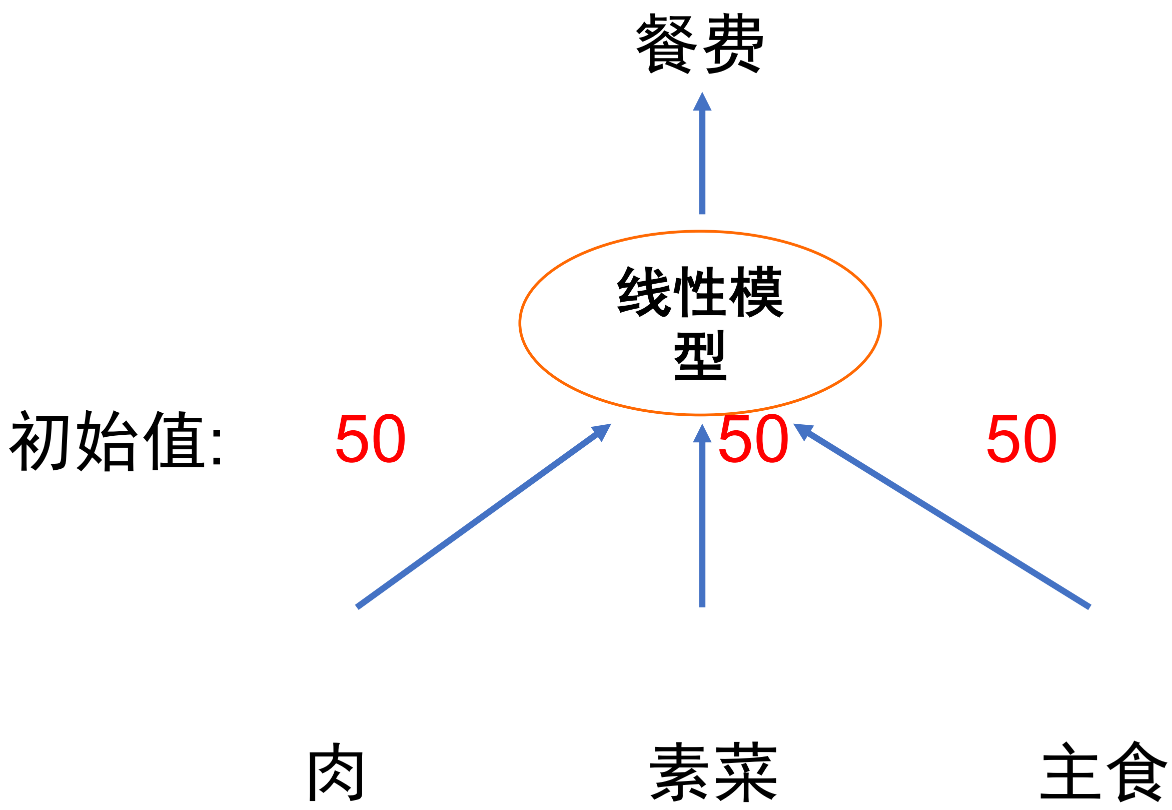
  - (t-y) = -270
  - $x_{肉}$= 0
  - $x_{素菜}$= 2
  - $x_{主食}$= 1

- 学习率 ε 是 2/27

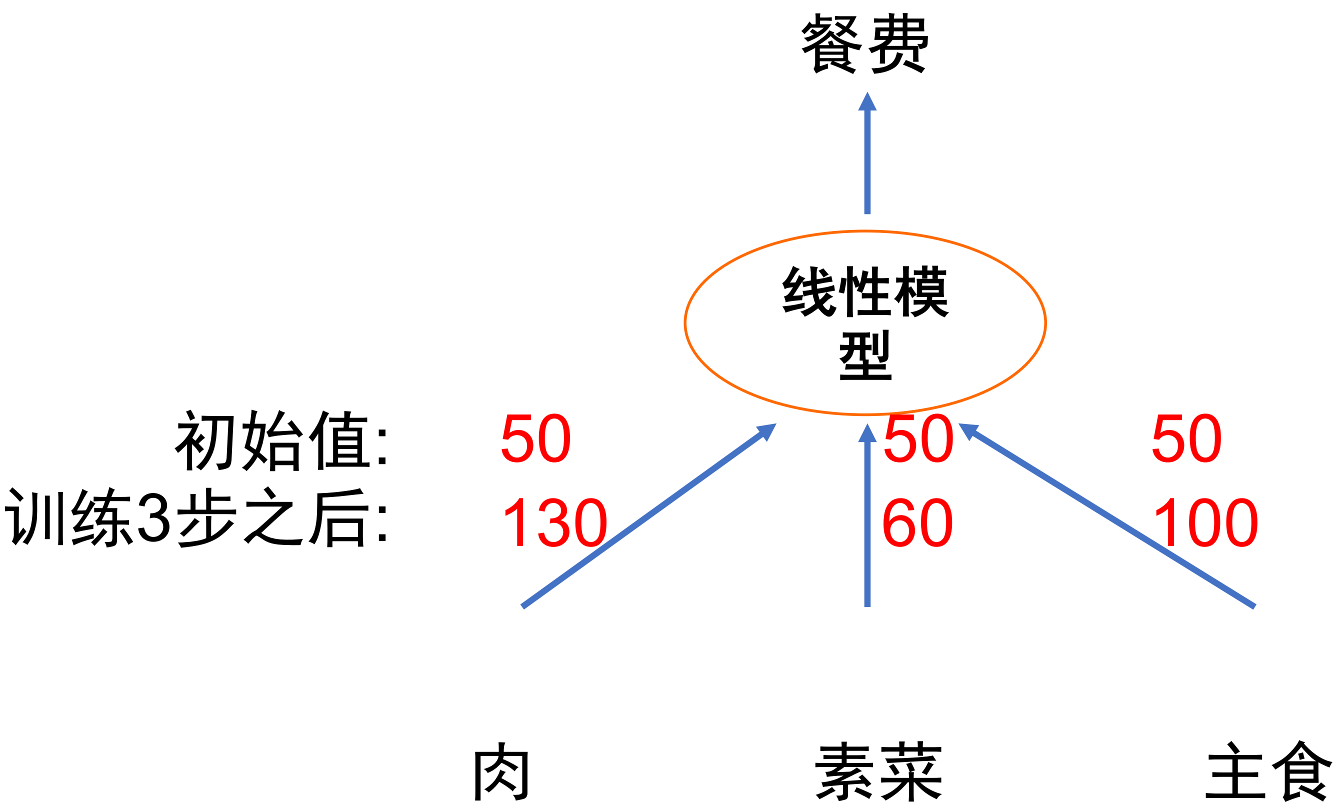
- 权重的修改量为:

  $\Delta w_{肉}$ = +0, $\Delta w_{素菜}$ = -40, $\Delta w_{主食}$ =-20

模型最开始是随机初始权重



模型最开始是随机初始权重



模型最开始是随机初始权重

餐费

线性模型

	肉	素菜	主食
初始值:	50	50	50
训练3步之后:	130	60	100
真实值:	150	50	100

批量样本的训练

- **误差**是所有样本的误差之和

$$E = \frac{1}{2} \sum_{n \in training} (t^{(n)} - y^{(n)})^2$$

- 求误差在权重上的的偏微分，即权重的梯度

- 更新：根据所有样本的误差之和在权重上的梯度值，进行更新。

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon x_i^{(n)} (t^{(n)} - y^{(n)})$$

批量样本的训练

- 误差是所有样本的误差之和

$$E = \frac{1}{2}\sum_{n \in training}(t^{(n)} - y^{(n)})^2$$

- 求误差在权重上的的偏微分，即权重的梯度

链式法则

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_n \frac{\partial y^{(n)}}{\partial w_n}\frac{\partial E^{(n)}}{\partial y^{(n)}}$$

$y^n = \sum w_i x_i^n$

$$= -\sum_n x_i^{(n)}(t^{(n)} - y^{(n)})$$

- 更新：根据所有样本的误差之和在权重上的梯度值，进行更新。

$$\Delta w_i = -\varepsilon\frac{\partial E}{\partial w_i} = \sum_n \varepsilon x_i^{(n)}(t^{(n)} - y^{(n)})$$

批量样本的训练

- 误差是所有样本的误差之和

$$E=\frac{1}{2}\sum_{n\in training}(t^{(n)}-y^{(n)})^2$$

- 求误差在权重上的的偏微分，即权重的梯度

链式法则

$$\frac{\partial E}{\partial w_i}=\frac{1}{2}\sum_n \boxed{\frac{\partial y^{(n)}}{\partial w_n}}\boxed{\frac{\partial E^{(n)}}{\partial y^{(n)}}}$$

$y^n=\sum w_i x_i^n$

$$=-\boxed{\sum_n}\boxed{x_i^{(n)}}\boxed{(t^{(n)}-y^{(n)})}$$

- 更新：根据所有样本的误差之和在权重上的梯度值，进行更新。

$$\Delta w_i=-\varepsilon\frac{\partial E}{\partial w_i}=\sum_n \varepsilon x_i^{(n)}(t^{(n)}-y^{(n)})$$

链式法则

- 复合函数求导
  - 设y=g(x), z = f(g(x))=f(y),链式法则是：

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

- 扩展到向量形式，x是m维向量($\in R^m$)，y是n维向量($\in R^n$)，g是从R^m 到 R^{n} 的映射，f是从R^{n} 到R的映射

$$\frac{\partial z}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \frac{\partial z}{\partial \mathbf{y}}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \cdots, \frac{\partial y_1}{\partial x_m} \\ \vdots, \vdots, \ddots, \vdots \\ \frac{\partial y_n}{\partial x_1}, \frac{\partial y_n}{\partial x_2}, \cdots, \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

Outline: Part 2.2

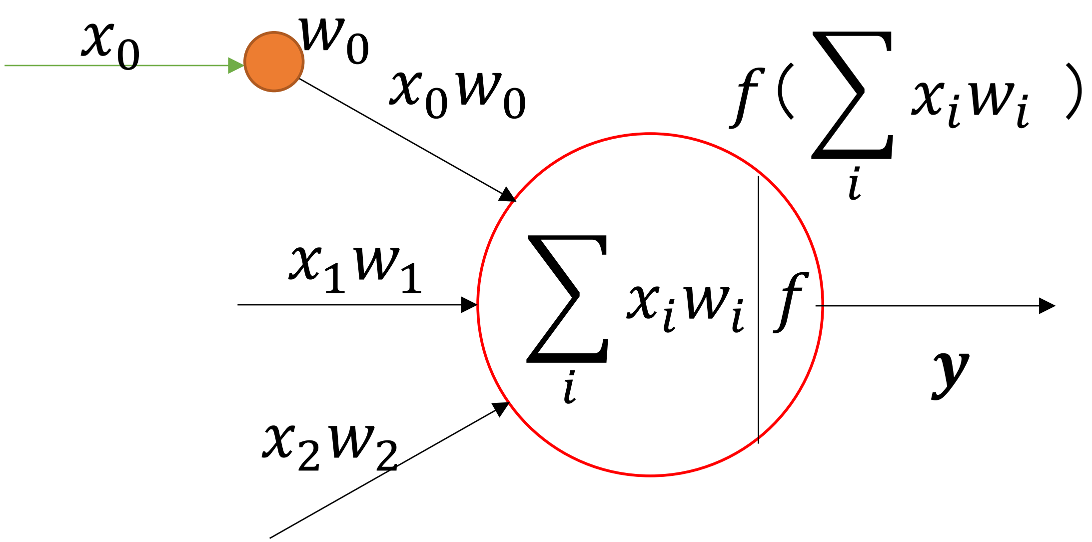
- 深度学习目标
- 线性 & 非线性网络层
  - 线性网络层: 线性神经元
  - **非线性网络层: Logistic (对数几率) 神经元**
- 后向传播
  - 后向传播的介绍
  - 非线性网络层的梯度
  - 求导的细节
- 学习 (训练)问题

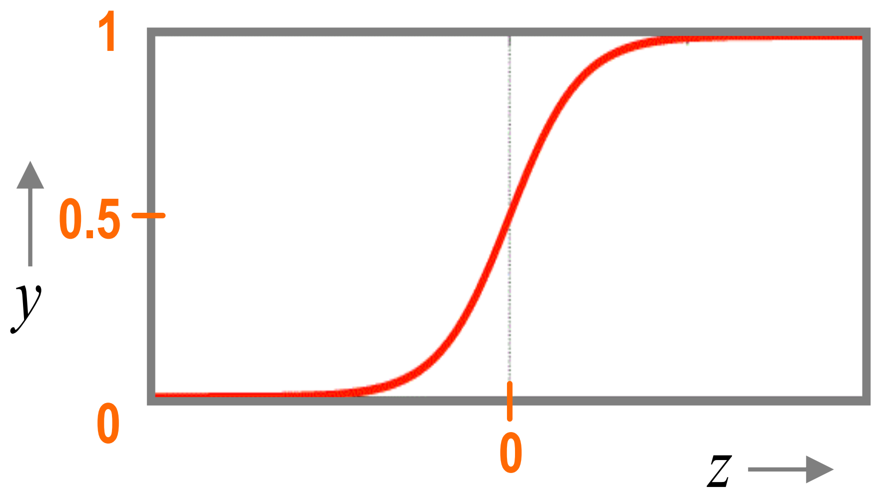
对数几率神经元 (非线性层)

- 平滑有边界的函数.
  - 导数容易求
  - 这意味着学习简单

假设函数 $z = b + \sum_i x_i w_i$

Sigmoid $y = \dfrac{1}{1+e^{-z}}$





对数几率神经元的求导

- z对权重w求偏微分

$$z = b + \sum_i x_i w_i$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- y对z求偏微分:

$$y = \frac{1}{1 + e^{-z}}$$

$$\frac{dy}{dz} = y(1 - y)$$

对数几率神经元的求导

- y对z求偏微分

$$y = \frac{1}{1+e^{-z}} = \left(1+e^{-z}\right)^{-1}$$

$$\frac{dy}{dz} = \frac{-1\left(-e^{-z}\right)}{\left(1+e^{-z}\right)^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y)$$

因为 $$\frac{e^{-z}}{1+e^{-z}} = \frac{\left(1+e^{-z}\right)-1}{1+e^{-z}} = \frac{\left(1+e^{-z}\right)}{1+e^{-z}} - \frac{1}{1+e^{-z}} = 1-y$$

Outline: Part 3.1

- 深度学习目标
- 线性 & 非线性网络层
  - 线性网络层: 线性神经元
  - 非线性网络层: Logistic (对数比率) 神经元
- **后向传播**
  - **后向传播的介绍**
  - 非线性网络层的梯度
  - 求导的细节
- 学习 (训练)问题

后向传播

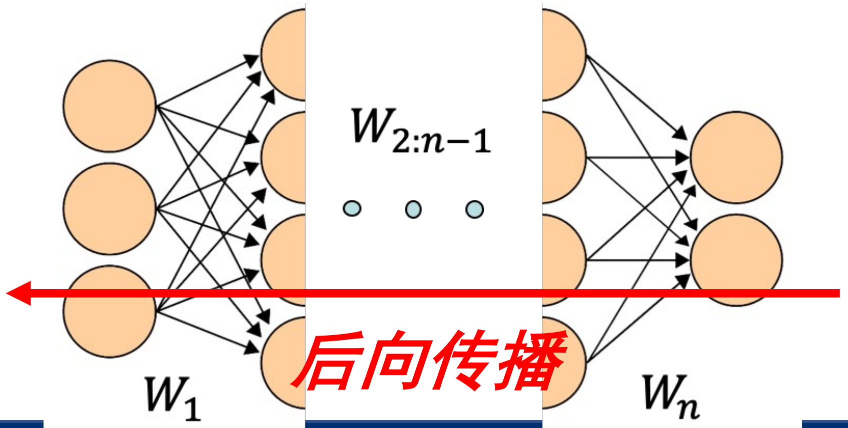
- 前向传播

W_1 ：第一层的权重

$$y = F(x) := f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))$$

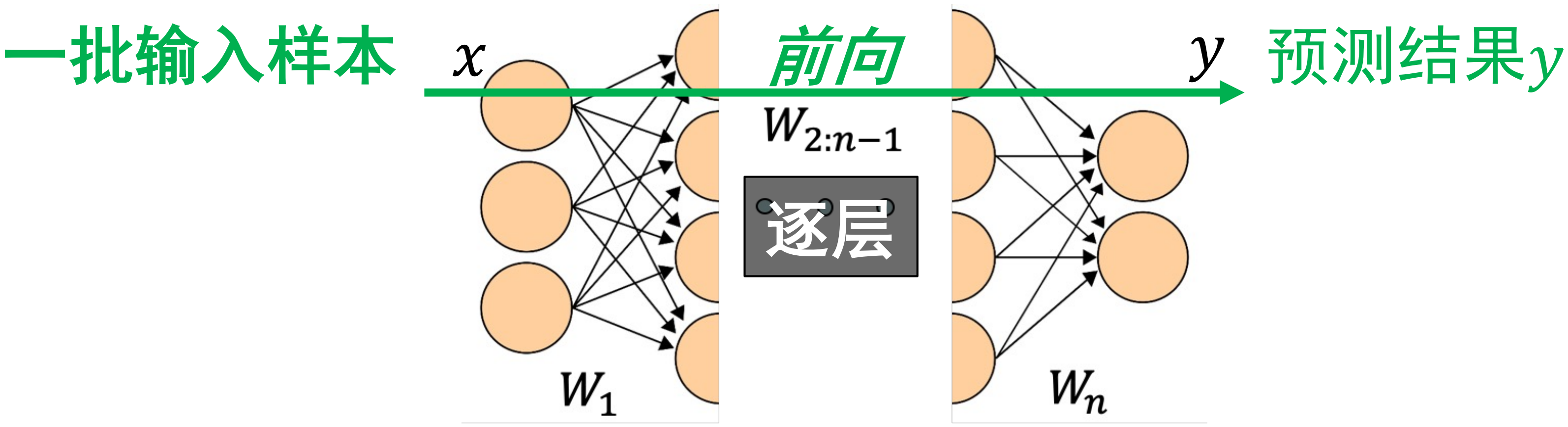
f_1 ： 第一层的输出 x ： 输入

- 后向传播
  - 基于链式法则 $\frac{\partial Loss}{\partial W_i} = \frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_i}{\partial W_i}$



训练迭代过程

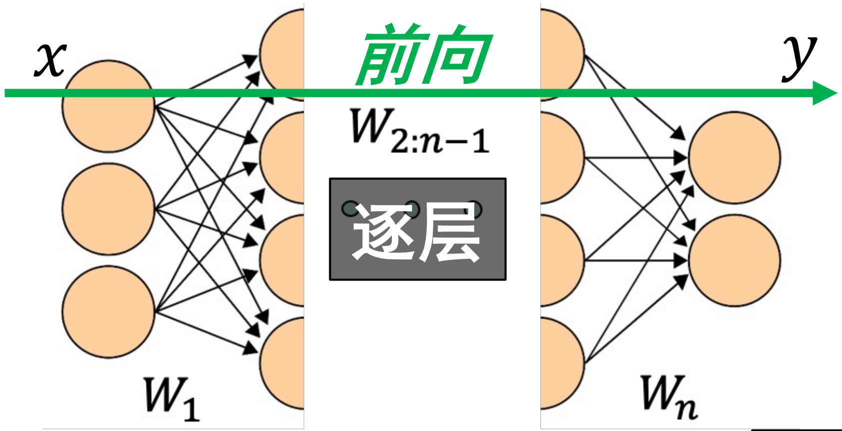
$$y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))$$



训练迭代过程

$$\boxed{y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))}$$

一批输入样本



预测结果y

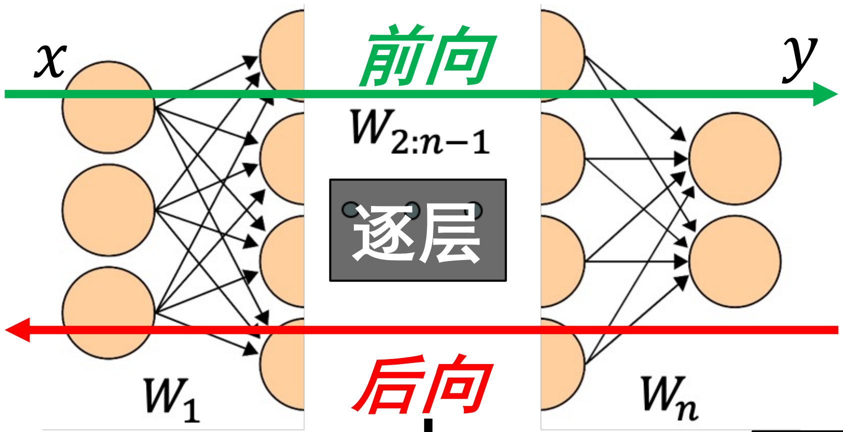
与实际值y^*的误差

$$\boxed{Loss = L(y, y^*)}$$

训练迭代过程

$$y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{...}f_1(W_1, x)))$$

一批输入样本



预测结果y

与实际值y^*的误差

$$Loss = L(y, y^*)$$

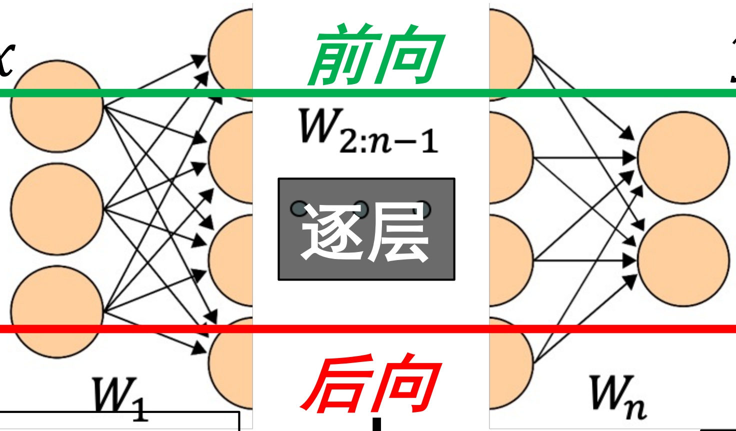
对 i_{th} 层的参数

$$\frac{\partial Loss}{\partial W_i} = \frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_i}{\partial W_i}$$

训练迭代过程

$$\boxed{y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))}$$

一批输入样本 x



y 预测结果y

与实际值y^*的误差

更新参数

$$\boxed{Loss = L(y, y^*)}$$

$$\boxed{W^{n+1} = W^n - r\frac{\partial Loss}{\partial W}(W^n)}$$

W^n: 当前权重值

W^{n+1}: 新权重值

对 i_{th} 层的参数

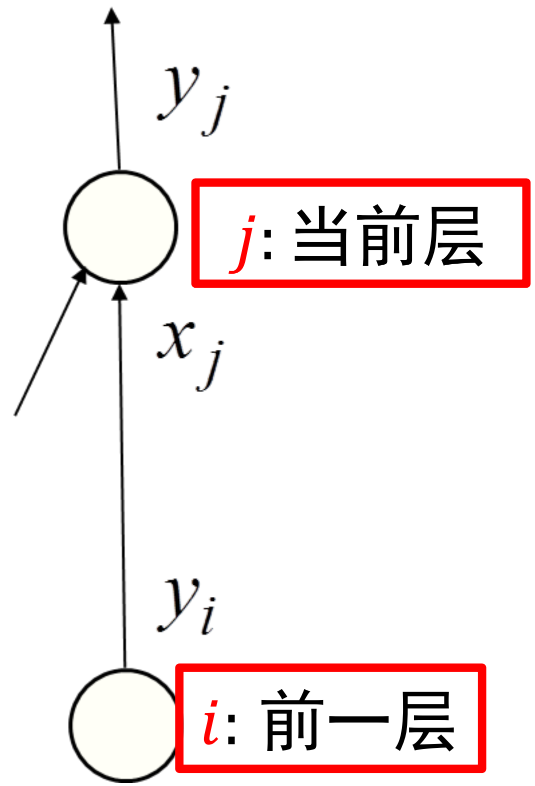
$$\boxed{\frac{\partial Loss}{\partial W_i} = \frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_i}{\partial W_i}}$$

Outline: Part 3.2

- 深度学习目标
- 线性 & 非线性网络层
  - 线性网络层: 线性神经元
  - 非线性网络层: Logistic (对数比率) 神经元
- 后向传播
  - 后向传播的介绍
  - **求导的细节**
- 学习 (训练)问题

表示上的不同

- 有多个层的网络，我们用显式索引表示网络是第几层.
  - y 某层的输入
  - x 某层的输入
  - 索引下标表示一个单元在哪一层



后向传播算法

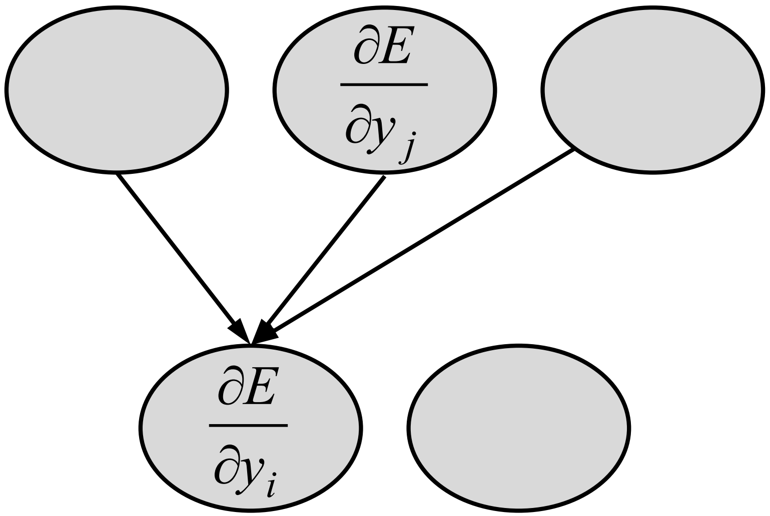
对一个样本做后向传播算法分析

$$E=\frac{1}{2}\sum_{j\in output}\left(t_j-y_j\right)^2$$

- 首先根据误差，计算误差对输出预测结果的梯度.

$$\frac{\partial E}{\partial y_j}=-\left(t_j-y_j\right)$$

- 根据上次误差的对输出的梯度计算当前层对输出的梯度.



回顾：非线性神经元

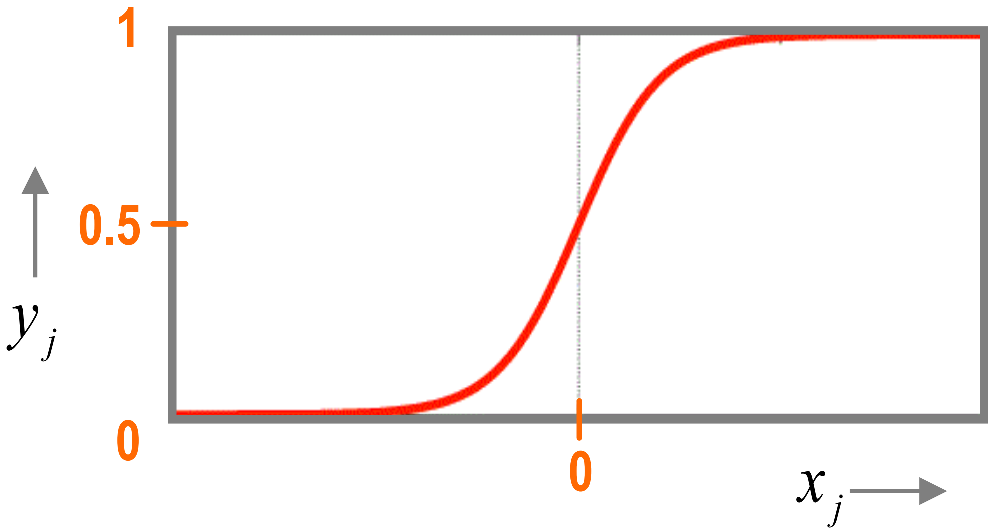
- 对于后向传播, 需要神经元的导数容易求得.
  - 通常，用logistic函数
  - 输入和权重得到一个光滑的输出结果。

前一层y_i

$$x_j = b_j + \sum_i y_i w_{ij}$$

当前层y_j

$$y_j = \frac{1}{1+e^{-x_j}}$$



$$\frac{\partial x_j}{\partial w_{ij}} = y_i \qquad \frac{\partial x_j}{\partial y_i} = w_{ij}$$

$$\frac{dy_j}{dx_j} = y_j\,(1 - y_j)$$

非线性神经元

- 用链式法则求得logistic 单元权重的梯度。
- 同时需要求得输出相对于每个权重的梯度:

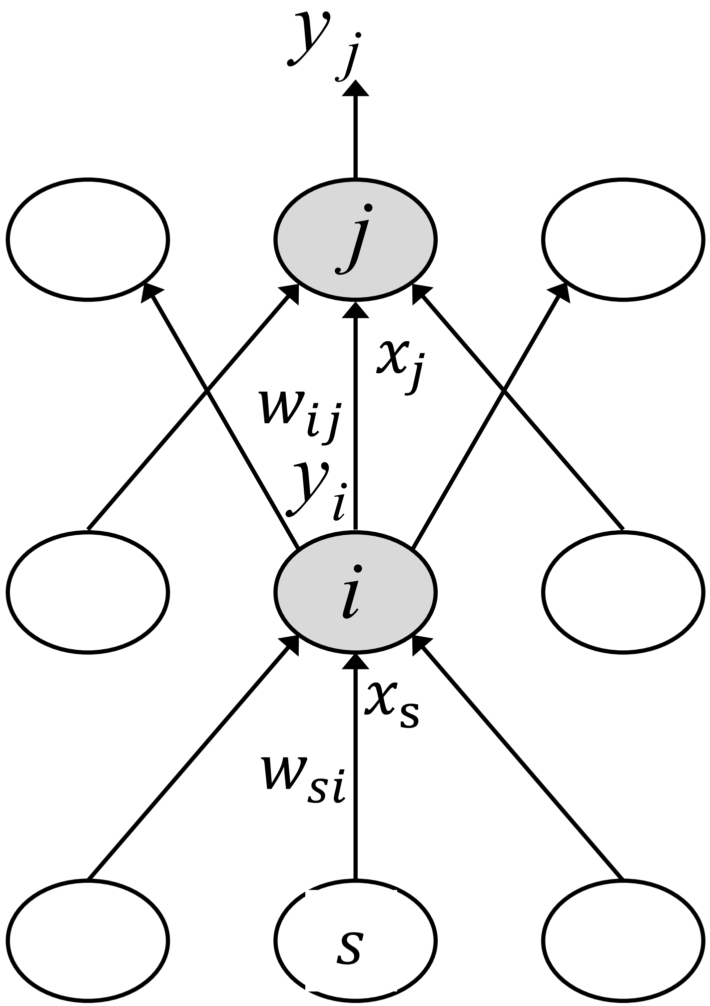
$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i y(1-y) \qquad E = \frac{1}{2}\sum_{n \in training}(t^{(n)} - y^{(n)})^2$$

梯度下降

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^{(n)}}{\partial w_n}\frac{\partial E}{\partial y^{(n)}} = -\boxed{\sum_n}\,\boxed{x_i^{(n)}}\,\boxed{y^{(n)}(1-y^{(n)})}\,\boxed{(t^{(n)} - y^{(n)})}$$

额外项= 对数几率函数的斜率

后向传播算法



对于第j层的输入为x_j, 输出为y_j

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}} \frac{\partial E}{\partial x_j} = y_i \frac{\partial E}{\partial x_j}$$

$$\frac{\partial E}{\partial x_j} = \frac{\partial y_j}{\partial x_j} \frac{\partial E}{\partial y_j} = y_j \left(1 - y_j\right) \frac{\partial E}{\partial y_j}$$

$$E = \frac{1}{2} \sum_{j \in output} \left(t_j - y_j\right)^2$$

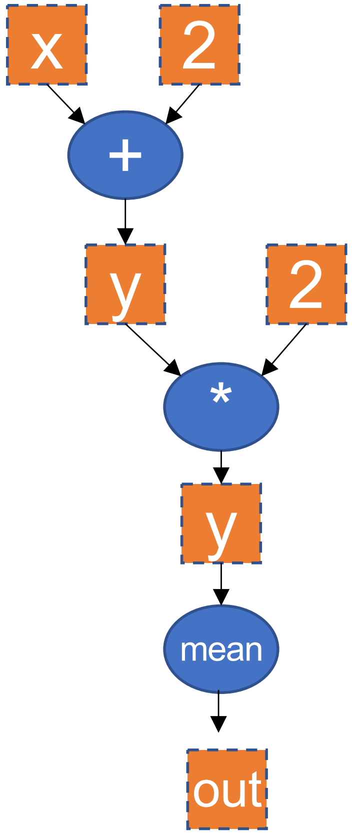
$$\frac{\partial E}{\partial y_j} = -\left(t_j - y_j\right)$$

Pytorch中的反向传播

- 计算过程用计算图表示，计算图是前向传播和反向传播的结构基础
- 张量是构建计算图的基础

```
x = torch.ones(2,2, requires_grad = True)
y = x+2
z = y*y*2
out = z.mean()
print(z, out)
out.backward()
print(x.grad)
```

- .backward() 可自动求出标量对所有变量的梯度，并将梯度值存在各个变量tensor节点中，
- .grad即可读取梯度

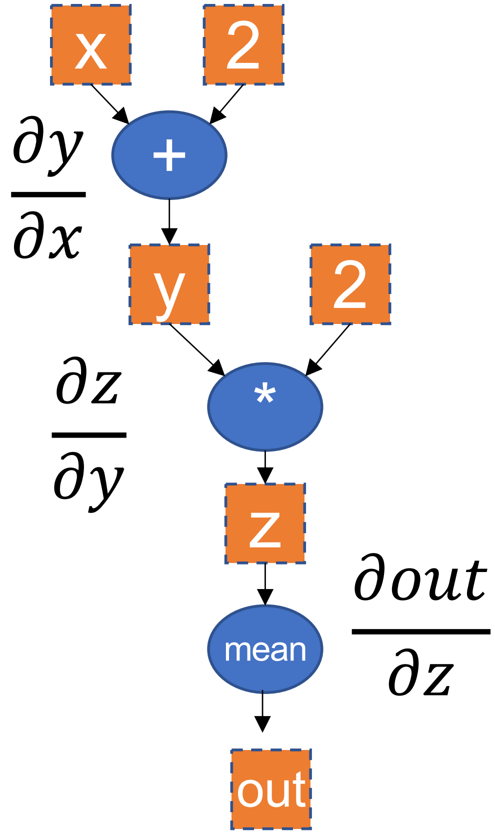


Pytorch中的反向传播

- 计算过程用计算图表示，计算图是前向传播和反向传播的结构基础
- 张量是构建计算图的基础

```
x = torch.ones(2,2, requires_grad = True)
y = x+2
z = y*y*2
out = z.mean()
print(z, out)
out.backward()
print(x.grad)
```

- 自动求微分Autograd
- 基于链式求导法则和雅克比矩阵，以及图结构，构建出包含梯度计算方法的反向传播计算图



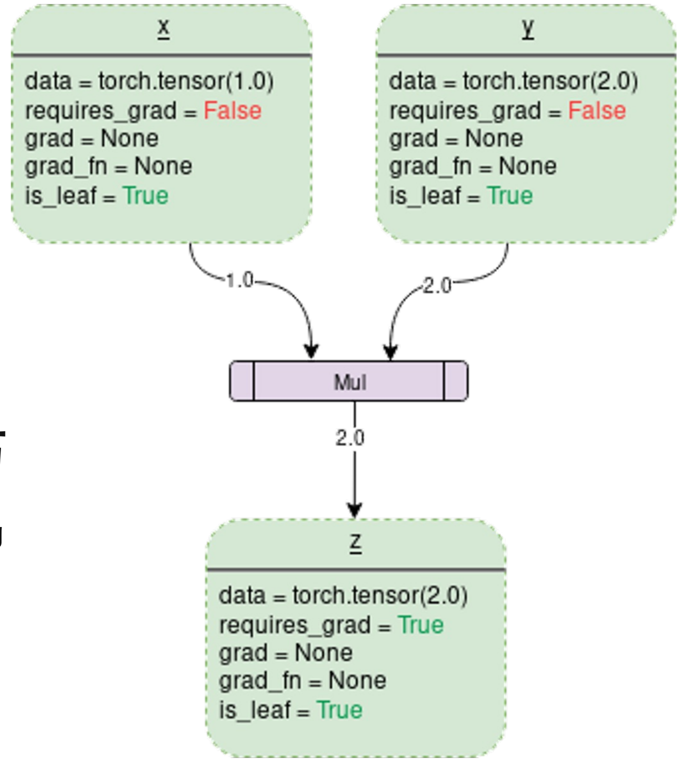
Pytorch中的反向传播

A graph is created on the fly


```
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
W_h = torch.randn(20, 20)
W_x = torch.randn(20, 10)
```

Pytorch中的反向传播

- Tensor张量
  - data:数据
  - requires_grad:判断该tensor是否需要被跟踪，用以计算梯度，默认为False
  - **grad:**初始为None；requires_grad = False时为None，否则当某out节点调用out.backward()时，生成新tensor节点，存放计算后的**∂out/∂x**梯度值，梯度值不会自动清空(下次调用out.backward()时可累积)
  - **grad_fn:**反向传播时，用来计算梯度的函数；
  - **is_leaf:**表明该tensor 在计算图中是否为叶子节点



完整的学习过程

- 反向传播算法
  - 在单个训练样本上针对每个权重w计算误差导数

$$\frac{\partial E}{\partial w}$$

- 一个完整的学习过程，我们仍然需要实现以下的目标：
  - 优化问题：我们如何得到权重对每一组样本都有效？
  - 泛化问题：我们如何确保所学习的权重对新的样本有效？

Outline: Part 4

- 深度学习目标
- 线性 & 非线性网络层
  - 线性网络层: 线性神经元
  - 非线性网络层: Logistic (对数比率) 神经元
- 后向传播
  - 后向传播的介绍
  - 非线性网络层的梯度
  - 求导的细节
- **学习 (训练)问题**

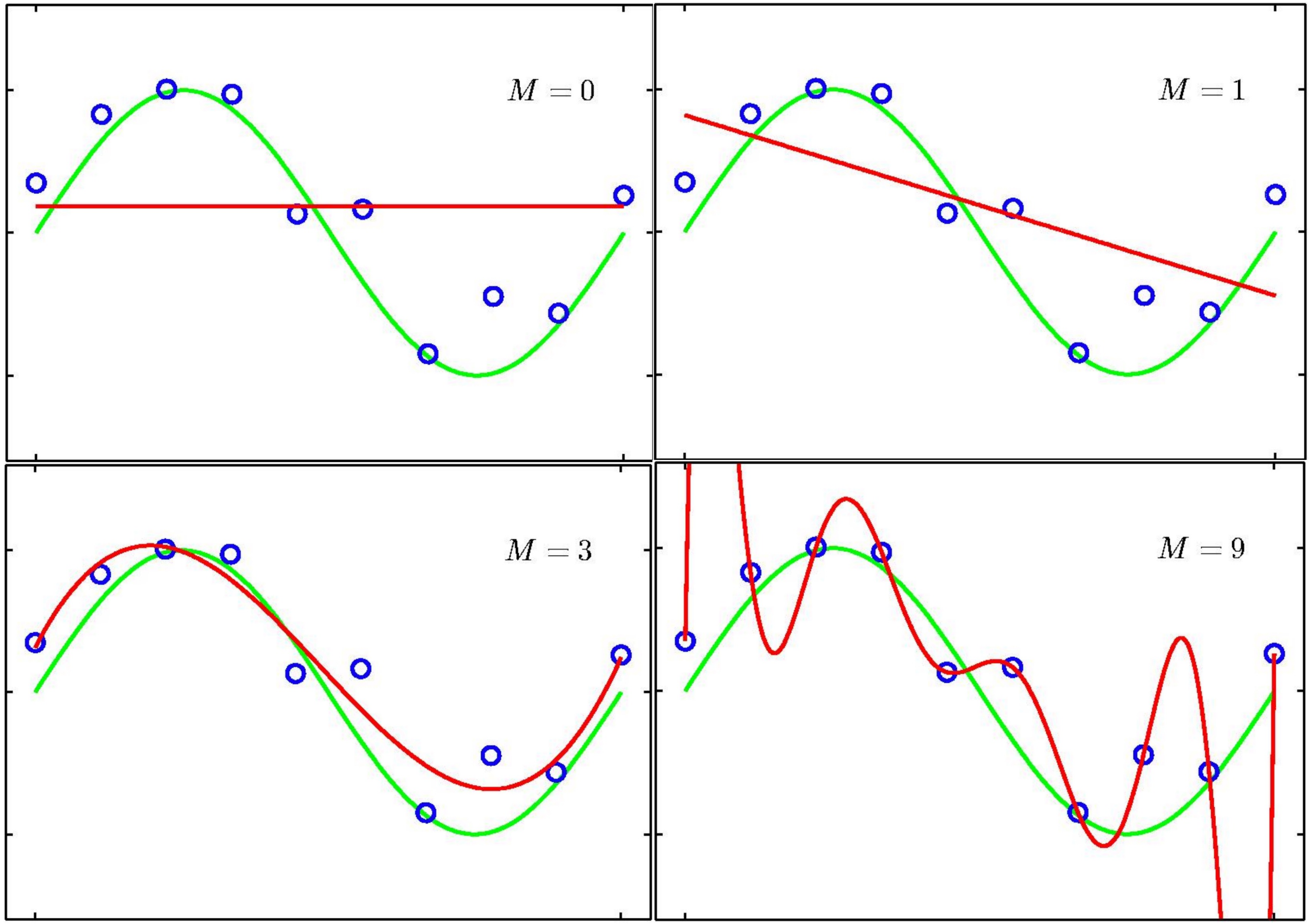
过拟合

功能过于强大的模型的缺点：

- 训练数据包含很多从输入到输出的映射规则，但也存在以下两种噪声.
  - 目标值可能不可靠
  - 采样错误， 训练集中有少数量数据恰巧存在规律而被学习到.
- 当模型在学习时，它无法确定哪些规律是真实的，哪些是由采样误差引起的.
  - 所以它可能学到了噪声里的规则.
  - 如果模型非常强大，那它会非常的拟合采样错误. 这造成灾难性后果.

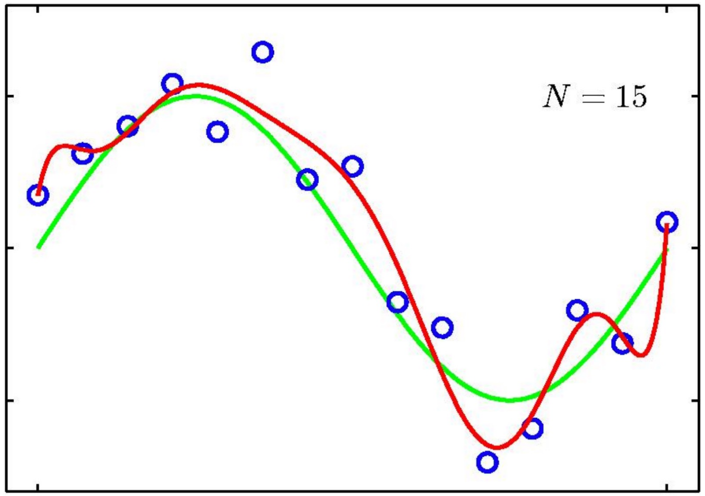
过拟合：哪个模型更好

- 绿色 曲线是正确的方程.
- 蓝色是数据点，X和绿色曲线是一致的，但是 有噪声



减少过拟合的方法

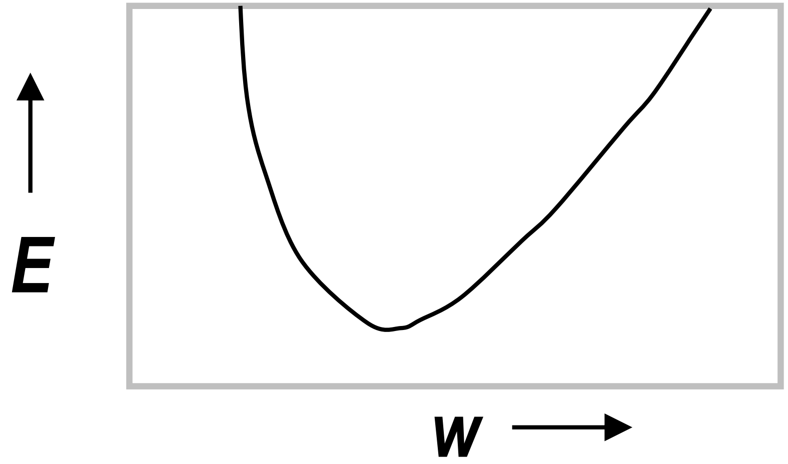
- 很多方法.
  - L-norm 正则化
  - 权重衰减
  - 用验证集
    - 提早终止学习
  - 模型平均
  - Dropout
  - 生成式预训练

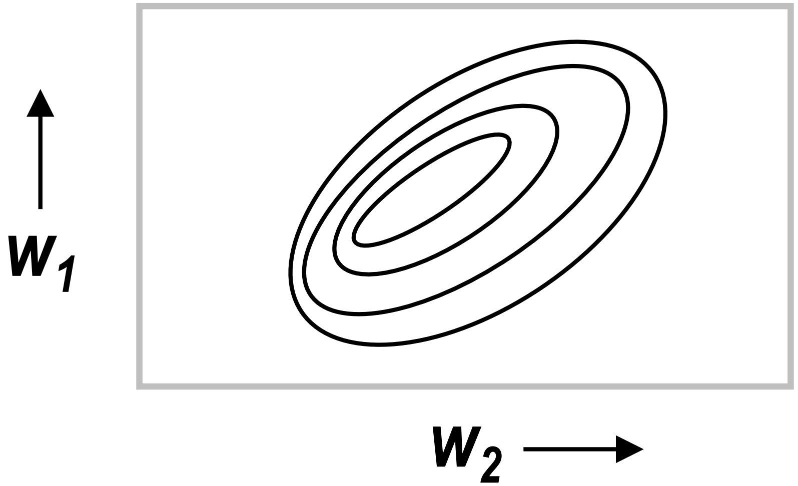


- 上述方法会在之后的过程中介绍.

误差超平面是拓展的权重空间

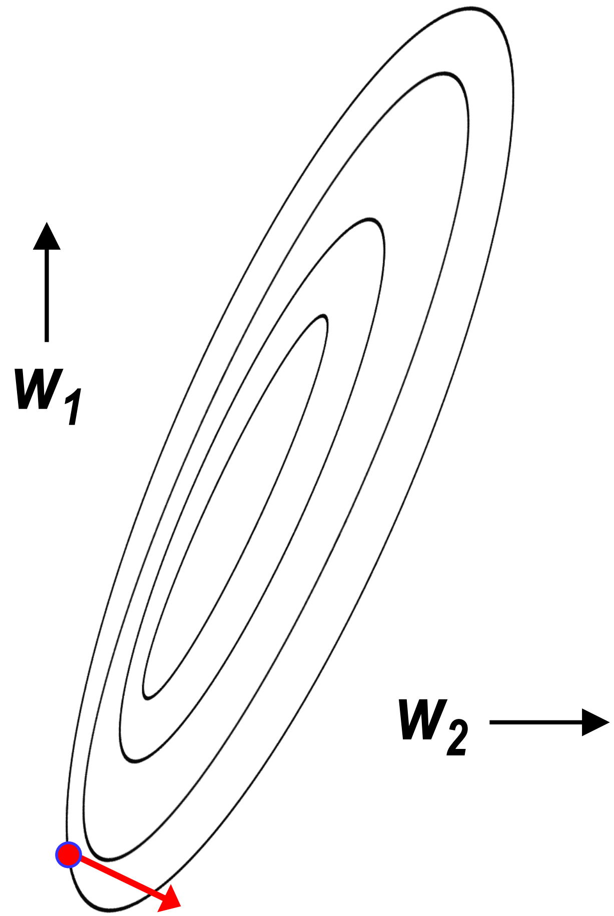
- 误差超平面在一个水平轴是w，纵轴是误差的空间.
  - 线性神经元，平方误差，误差平面是个二次碗.
  - 垂直横截面是抛物线(如右图)。
  - 水平横截面是椭圆（轮廓线）
- 多层非线性网络的误差超平面相当复杂





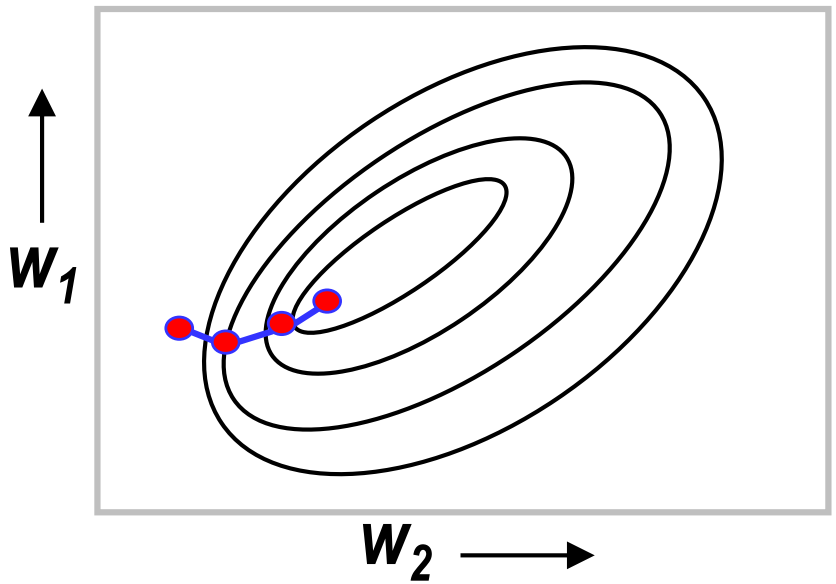
用权重梯度来优化模型的问题

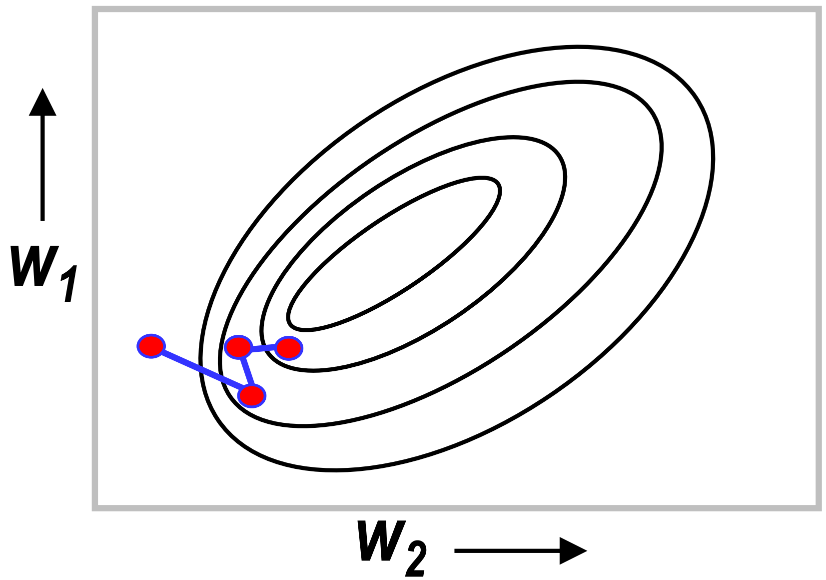
- 多久更新一次权重
  - 在线: 每个训练样本.
  - 全批量（全数据集）.
  - 小批量更新: 训练集一小部分样本.
- 更新量是多少？
  - 用一个固定的学习率?
  - 调整全局学习率?
  - 不同层的权重采用不同的学习率?
- 不用最陡峭的梯度?



批量的选择

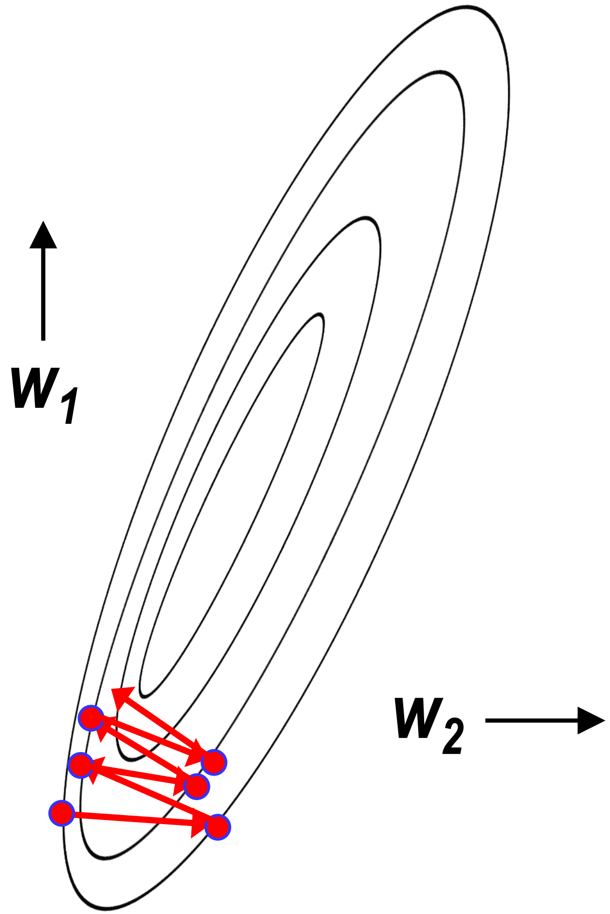
- 小批量或在线学习（one-sample batch）会沿最陡下降方向呈现之字形更新：
- 全批量在误差面的最陡峭处进行下降.
  - 垂直等高线的方向下降.





学习速度慢的原因

- 如果椭圆非常扁平，则最陡的下降方向可能会偏离最小的方向!
  - 红色渐变向量在椭圆的短轴上具有较大的分量，而在椭圆的长轴上具有较小的分量。
  - 这不是我们想要的。



在这节课，我们学习了

- 深度学习是怎么工作的
  - 一个简单的线性例子
- 线性和非线性神经元
- 如何训练神经网络
  - 后向传播怎么工作
- 训练的一些问题
  - 过拟合
  - 权重梯度的一些问题

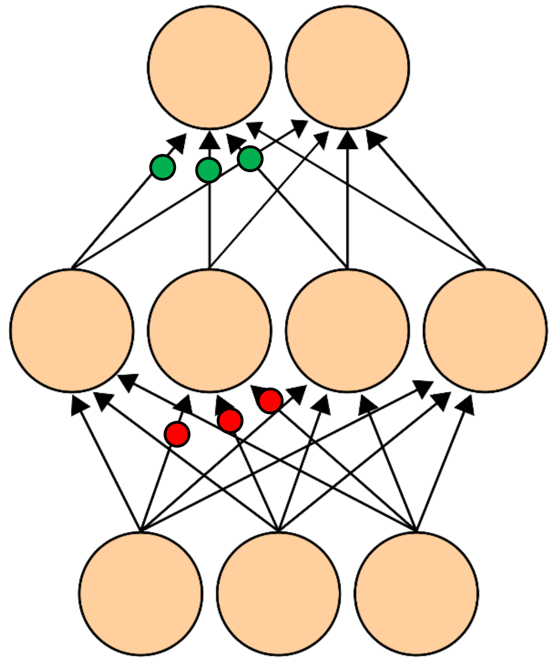
Backup Slides

为什么用后向传播

- 网络如果没有隐藏层，能力非常有限
  - 没有隐藏层的话，网络无法很好的模拟输入—输出映射关系。
  - 多个线性隐藏层的帮助不大，因为模型仍然只能模拟线性关系。
  - 固定的非线性输出帮助也不大。
- 有效的网络，有多个自适应的非线性隐藏层。
  - 我们如何训练这种网络？
  - 有效途径是：调整所有层的权重（不仅是最后一层）；
  - 对于隐藏层来说，学习权重也就需要学习特征值。
  - 隐藏层的行为是未知的

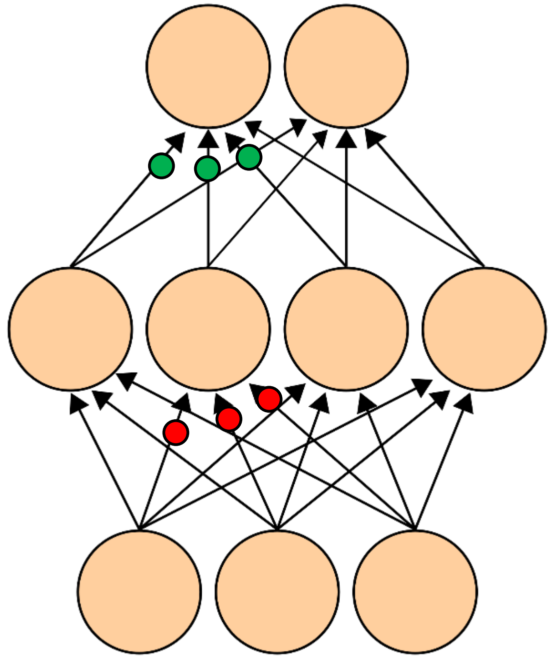
学习隐藏层

- 没有隐藏层的话，网络无法很好的模拟输入—输出映射关系。
- 感知机的隐藏层的特征值是通过手动设置，但太难
  - 这基于深刻的分析和观察，因此手动很难的去选择到合适的特征值.
  - 反复的试错调整得到好的特征值.
- 自动化特征值设计



通过调整权重完成学习

- 想法与进化的想法类似
- 随机调整权重，观察是否会提高性能，如果性能提高，保留这个改变。
  - 这是一种强化学习的方式.
  - 非常低效. 需要对一组有代表性的训练集进行多次前传，以改变一个权重。 反向传播要好得多
  - 在学习快要结束时，过大的权重调整值会使情况变得更糟，因为权重需要具有正确的相对值。



学习隐藏到输出的权重简单（绿色）。
学习输入到隐藏的权重难（红色）。

Outline: Part 3.2

- 深度学习目标
- 线性 & 非线性网络层
  - 线性网络层: 线性神经元
  - 非线性网络层: Logistic (对数比率) 神经元
- 后向传播
  - 后向传播的介绍
  - 求导的细节
- 学习(训练)问题