R 机器学习

第 14 讲　神经网络

张敬信

2022 年 10 月 27 日

哈尔滨商业大学

一. 神经网络简介

神经网络的发展历程以 2006 年为分界点，之前是浅层神经网络（经历 2 盛 2 衰），2006 年，Geoffrey Hinton 首次将深层神经网络命名为深度学习，开启了神经网络的第 3 次复兴之路。

深度学习是机器学习的分支，已发展成独立的热门研究领域。本讲讨论的是浅层神经网络：前向反馈神经网络。深度学习原理基本是一样的，只是层数更多、结构更复杂。

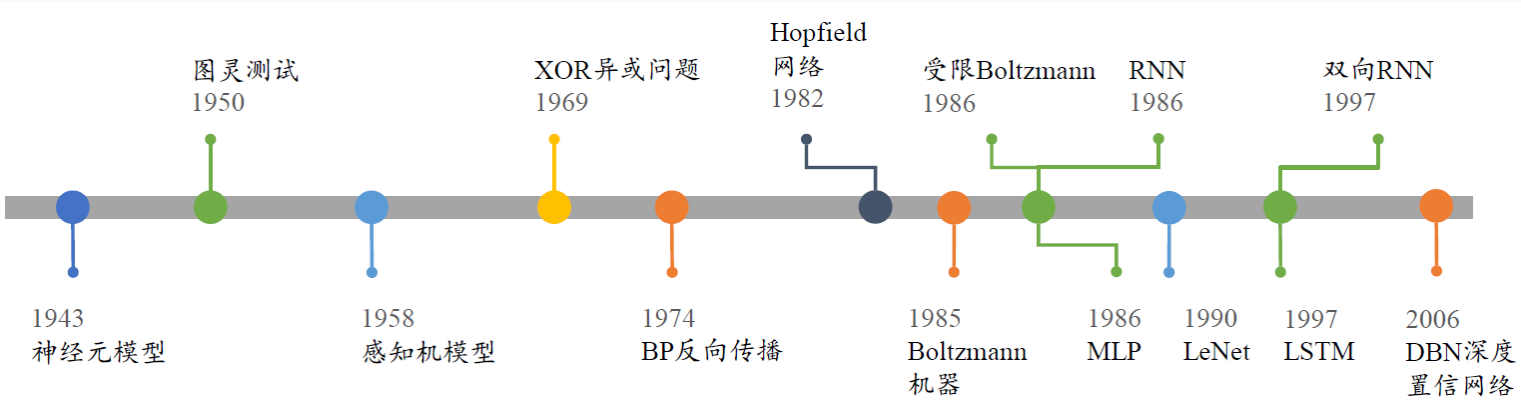
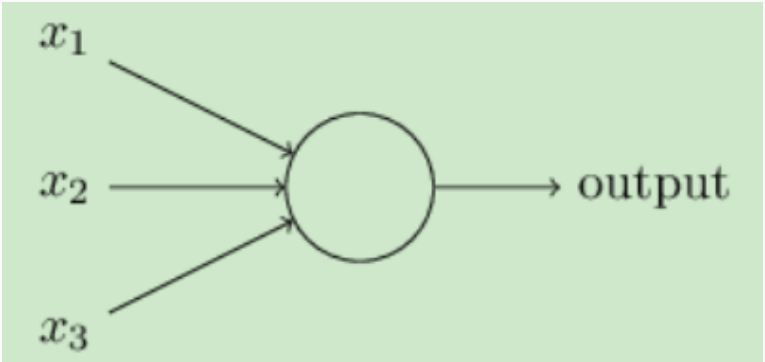


图 1: 浅层神经网络发展路线

感知机的模型，是有若干输入和一个输出的模型：



输出和输入之间学习到一个线性关系，得到中间输出结果：

$$z = \sum_{i=1}^{m} w_i x_i + b$$

接着是一个神经元激活函数：$\mathrm{sign}(z)$, 从而得到想要的输出结果 1 或者 -1.

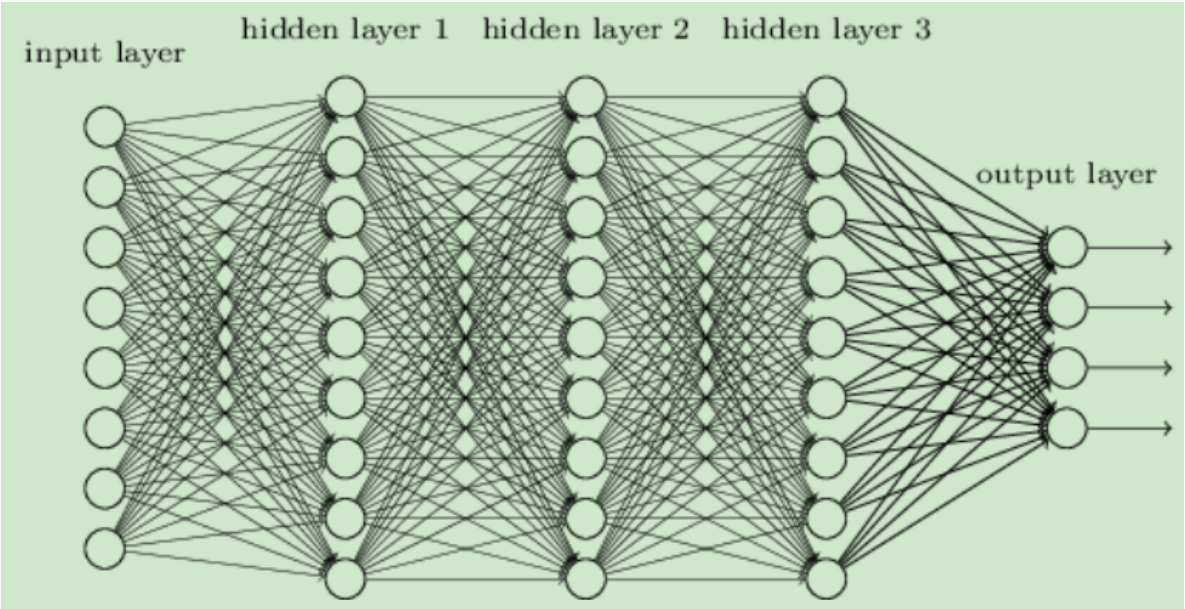
感知机模型只能用于二元分类，且无法学习比较复杂的非线性模型。神经网络模型做了扩展：

- 加入了隐层，隐层可以有多层，增强模型的表达能力；
- 输出层的神经元可以有多个输出，使得模型可以灵活地应用于分类、回归，甚至降维、聚类；
- 激活函数从 $\mathrm{sign}(z)$ 扩展到 Logistic 回归中的 $\mathrm{Sigmoid}$ 函数，后来又出现 $\tanh, \mathrm{softmax}$ 和 ReLU 等，表达能力进一步增强。

二. 神经网络算法原理

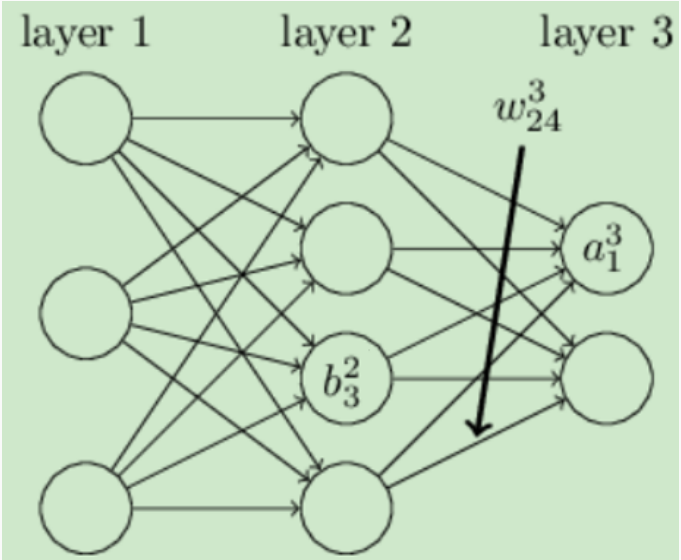
1. 基本结构

神经网络层根据位置可以分为三类：输入层（第 1 层)、隐层（中间层）和输出层（最后 1 层)。



层与层之间是全连接的，即第 i 层的任意一个神经元一定与第 $i+1$ 层的任意一个神经元相连接。整体看起来很复杂，但从小的局部模型来说，还是和感知机一样，即一个线性关系 $z=\sum_i w_i x_i+b$, 加上一个激活函数 $\sigma(z)$.

很多组线性系数 w_i 和偏倚 b 需要表示清楚，以三层神经网络为例：

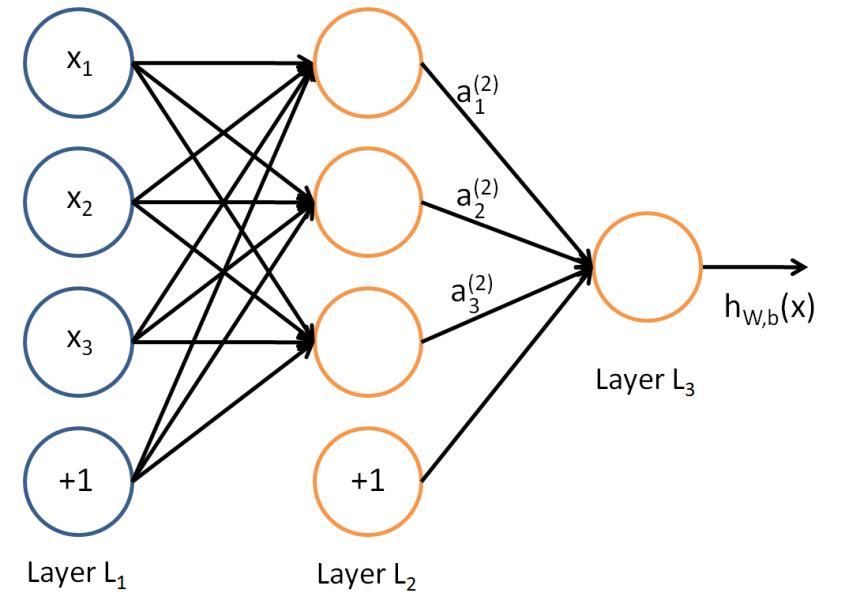


- 第 $l-1$ 层的第 k 个神经元到第 l 层的第 j 个神经元的线性系数记为 w^l_{jk}. （这样反着记，是避免后续做矩阵计算时再做转置）
- 第 $l-1$ 层的第 j 个神经元对应的偏倚记为 b^l_j.

注意，输入层没有 w 参数，输入层没有 b 参数。

2. 前向传播算法

利用与感知机一样的思路，根据上一层的输出计算下一层的输出，即前向传播。



假设第 $l-1$ 层共有 m 个神经元，则对第 l 层的第 j 个神经元的输出 a_j^l, 有

$$a_j^l = \sigma(z_j^l) = \sigma\left(\sum_{k=1}^{m} w_j^l a_k^{l-1} + b_j^l\right)$$

若 $l = 2$, 则 a_k^1 即为输入层的 x_k.

假设第 $l-1$ 层共有 m 个神经元，而第 l 层共有 n 个神经元，则第 l 层的线性系数 w 组成了一个 $n \times m$ 维矩阵 W^l; 第 l 层的偏倚 b 组成了一个 $n \times 1$ 维向量 b^l, 第 $l-1$ 层的输出 a 组成了一个 $m \times 1$ 维向量 a^{l-1}, 第 l 层的未激活前线性输出 z 组成了一个 $n \times 1$ 维向量 z^l, 第 l 层的输出 a 组成了一个 $n \times 1$ 维向量 a^l.

则改用更简洁的矩阵表示，第 l 层的输出为：

$$a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l)$$

总之，前向传播算法就是利用若干个权重系数矩阵 W, 偏倚向量 b 来和输入值向量 x 进行一系列线性运算和激活运算，从输入层开始，根据上面公式一层层的向后计算，一直运算到输出层，得到输出结果为止。

3. 反向传播算法

神经网络模型从输入到输出中间衔接的全部权重系数矩阵 W 和偏倚向量 b 中的元素都是模型参数，与其它机器学习模型一样，需要先初始化一组模型参数，根据输入和前向传播算法计算出预测值，然后损失函数计算损失。接着用梯度下降法进行迭代优化让损失达到极小，得到最优的模型参数，这个过程就是反向传播算法。

第一步，明确损失函数。对样本 (x_k, y_k), 预测值为 $\hat{y}_k$, 以均方误差损失为例：

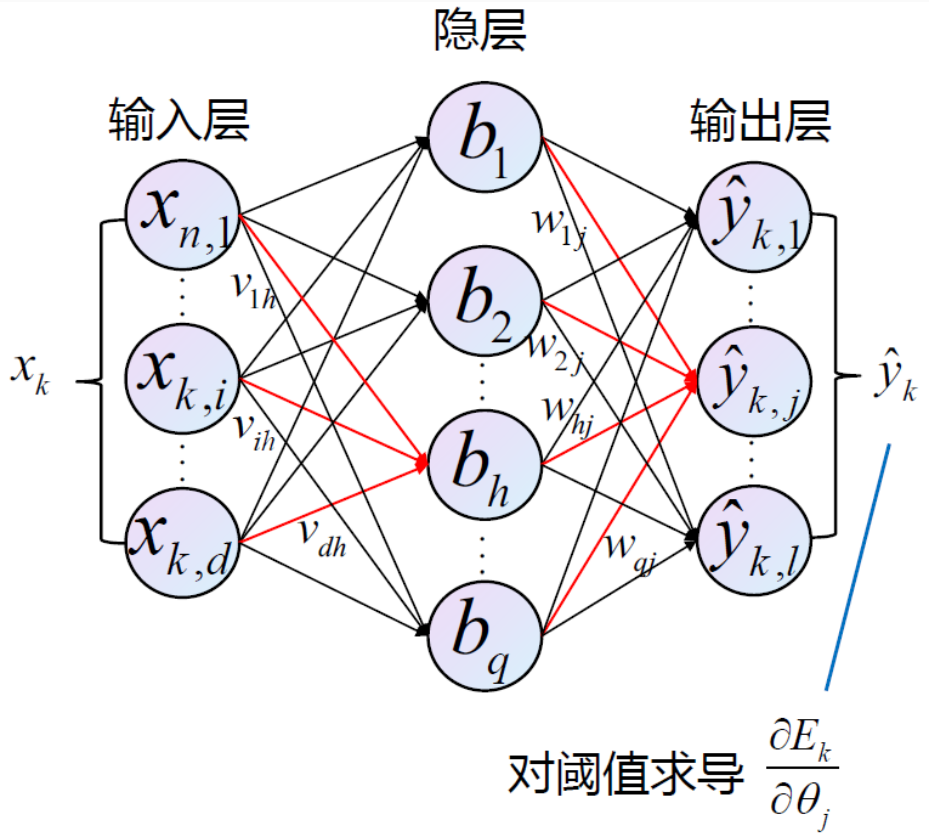
$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

第二步，明确参数调整策略。基于梯度下降法，以目标的负梯度方向对参数进行调整：

$$v = v + \Delta v$$

$$\Delta v = -\rho \frac{\partial E_n}{\partial v}$$

第三步，计算输出层阈值 θ_j 的梯度 $\frac{\partial E_k}{\partial \theta_j}$



利用链式法则，可得

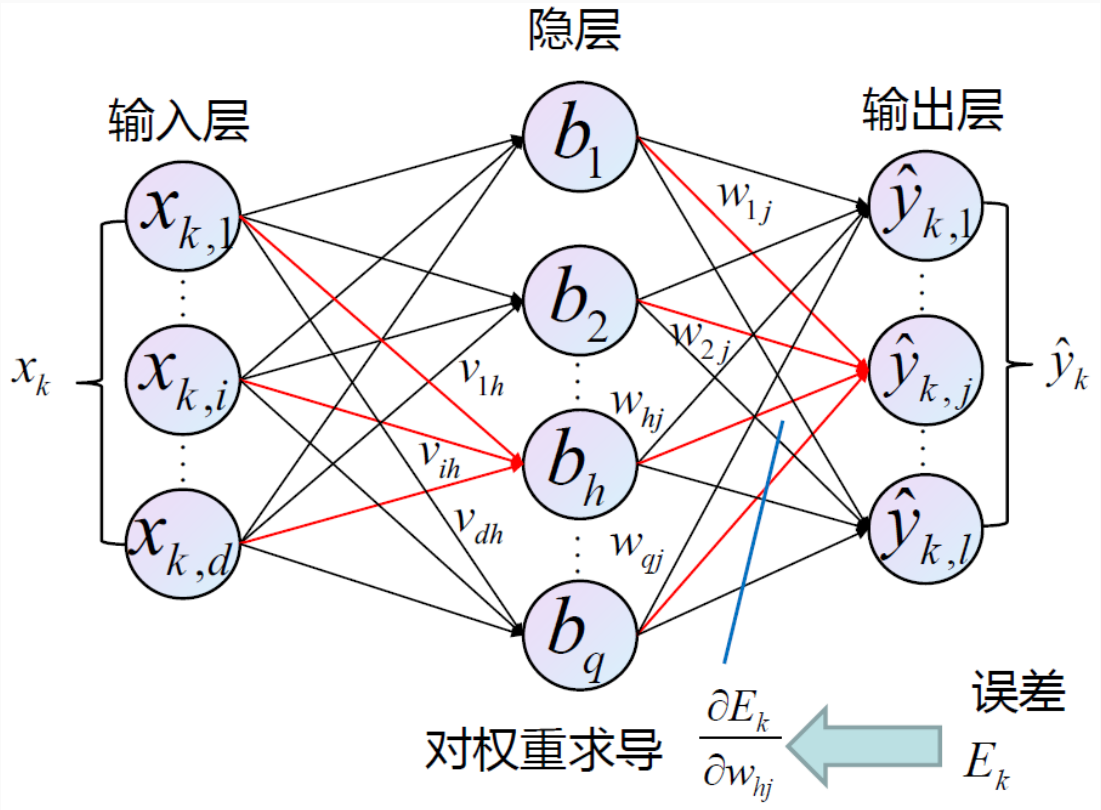
$$\frac{\partial E_k}{\partial \theta_j} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \theta_j}$$

其中，$\frac{\partial E_k}{\partial \hat{y}_j^k} = \hat{y}_j^k - y_j^k$，$\frac{\partial \hat{y}_j^k}{\partial \theta_j} = -\hat{y}_j^k(1 - \hat{y}_j^k)$. 故

$$g_j = \frac{\partial E_k}{\partial \theta_j} = \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$$

更新公式： $\theta_j \leftarrow \theta_j - \eta g_j$.

第四步，计算隐层到输出层连接权重 w_{hj} 的梯度 $\frac{\partial E_k}{\partial w_{hj}}$



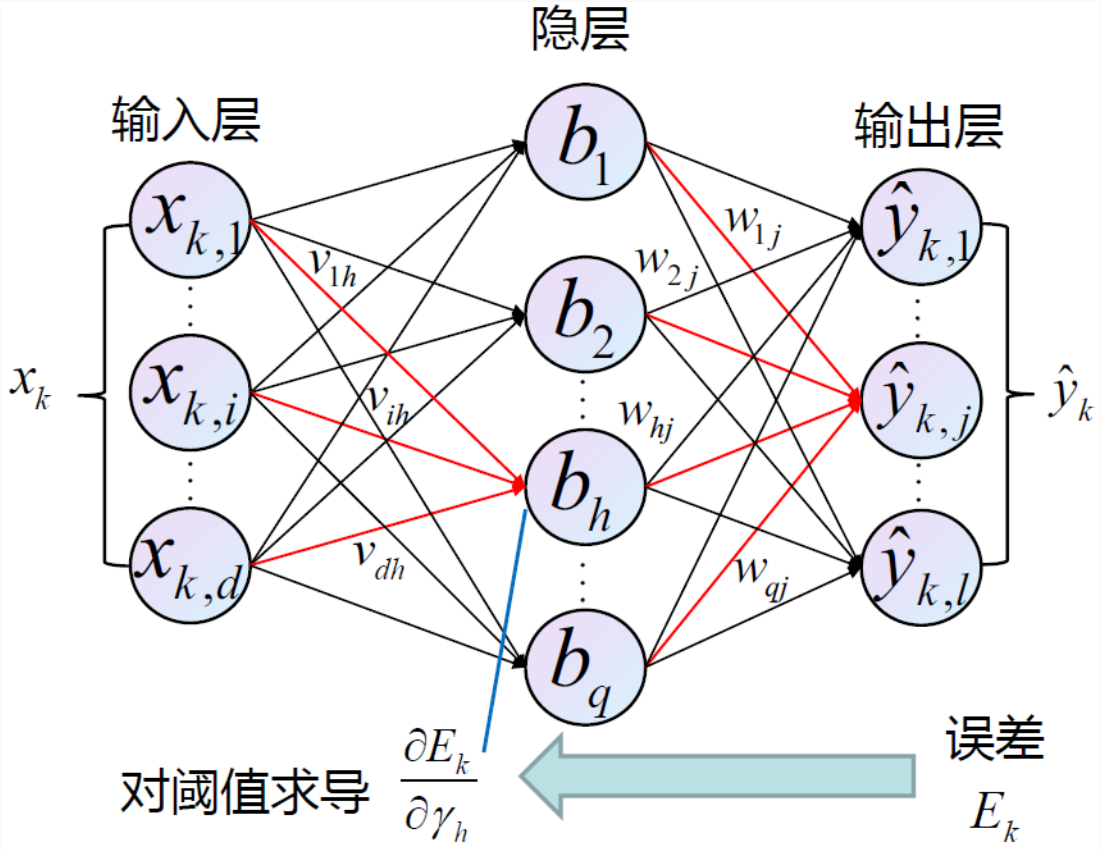
利用链式法则，可得

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$$

其中，$\frac{\partial E_k}{\partial \hat{y}_j^k} = \hat{y}_j^k - y_j^k$, $\frac{\partial \hat{y}_j^k}{\partial \beta_j} = -\hat{y}_j^k(1-\hat{y}_j^k)$, $\frac{\partial \beta_j}{\partial w_{hj}} = b_h$. 从而

$$\frac{\partial E_k}{\partial w_{hj}} = \hat{y}_j^k \cdot (\hat{y}_j^k - y_j^k) \cdot (1-\hat{y}_j^k) \cdot b_h = -g_j b_h$$

第五步，计算隐层阈值 γ_h 的梯度 $\frac{\partial E_k}{\partial \gamma_h}$



利用链式法则，可得

$$\frac{\partial E_k}{\partial \gamma_h} = \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \gamma_h}$$

其中，

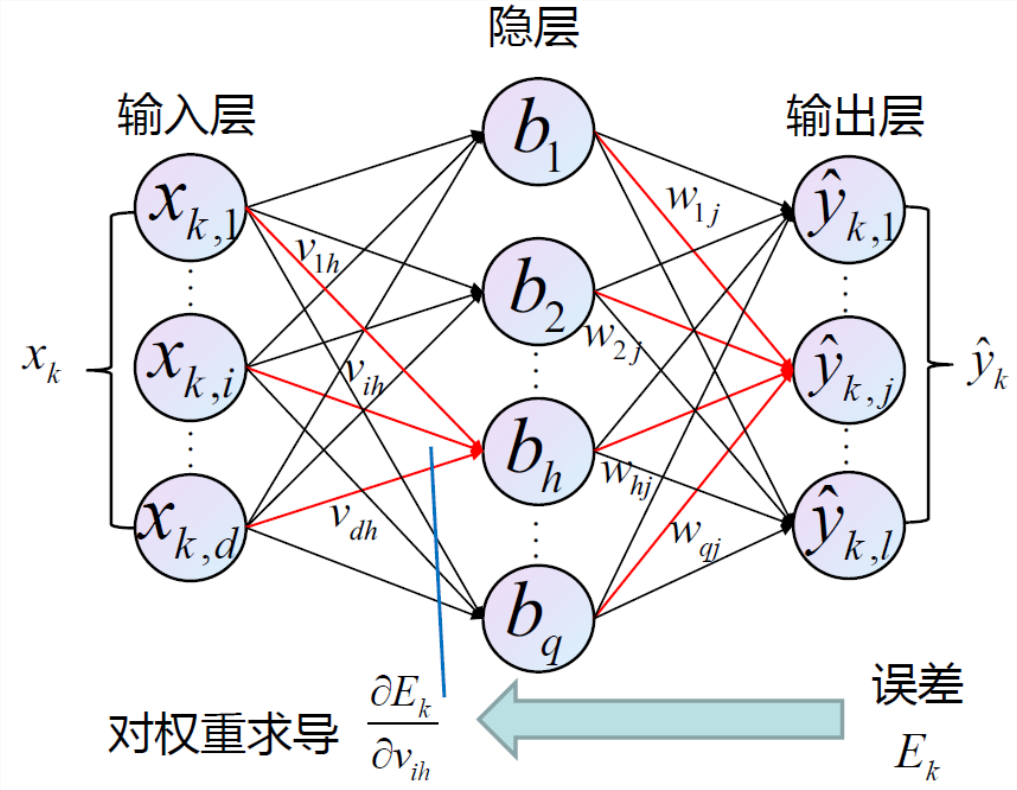
$$\frac{\partial E_k}{\partial b_h} = \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} = -\sum_{j=1}^{l} g_j w_{hj}$$

$$\frac{\partial b_h}{\partial \gamma_h} = \frac{\partial}{\gamma_h} f(\alpha_h - \gamma_h) = -b_h(1-b_h)$$

故 $\frac{\partial E_k}{\partial \gamma_h} = b_h(1-b_h)\sum_{j=1}^{l} w_{hj} g_j$. 令 $e_h = b_h(1-b_h)\sum_{j=1}^{l} w_{hj} g_j$,

更新公式： $\gamma_h \leftarrow \gamma_h - \eta e_h$.

第六步，计算隐层权重 v_{ih} 的梯度 $\frac{\partial E_k}{\partial v_{ih}}$



利用链式法则，可得

$$\frac{\partial E_k}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}}$$

其中，$\frac{\partial E_k}{\partial b_h} = -\sum_{j=1}^{l} g_j w_{hj}$, $\frac{\partial b_h}{\partial \alpha_h} = b_h(1-b_h)$, $\frac{\partial \alpha_h}{\partial v_{ih}} = x_i$, 故

$$\frac{\partial E_k}{\partial v_{ih}} = -b_h(1-b_h)x_i \sum_{j=1}^{l} w_{hj} g_j = -e_h x_i.$$

更新公式： $v_{ih} \leftarrow v_{ih} + \eta e_h x_i.$

结论

由 $\frac{\partial E_k}{\partial \gamma_h} = b_h(1-b_h)\sum_{j=1}^{l} g_j w_{hj}$ 可知，隐层阈值梯度取决于隐层神经元输出、输出层阈值梯度和隐层与输出层的连接权值。

在阈值的调整过程中，当前层的阈值梯度取决于下一层的阈值，这就是 BP 算法的精髓。

由 $\frac{\partial E_k}{\partial w_{hj}} = -g_j b_h$ 可知，当前层的连接权值梯度，取决于当前神经元阈值梯度和上层神经元输出。

只要知道**上一层**神经元的阈值梯度，即可计算**当前层**神经元阈值梯度和连接权值梯度。随后可以计算**输出层**神经元阈值梯度，从而计算出**全网络**的神经元阈值和连接权值梯度。最终达到**训练网络**的目的。

总结 BP 算法流程：

（1）将输入样本提供给输入层神经元

（2）逐层将信号前传至隐层、输出层，产生输出层的结果

（3）计算输出层误差

（4）将误差反向传播至隐藏层神经元

（5）根据隐层神经元对连接权重和阈值进行调整

（6）上述过程循环进行，直至达到某停止条件为止

优点：

- 能够自适应、自主学习。BP 可以根据预设参数更新规则，通过不断调整神经网络中的参数，已达到最符合期望的输出。
- 拥有很强的非线性映射能力。
- 误差的反向传播采用的是成熟的链式法则，推导过程严谨且科学。
- 算法泛化能力很强。

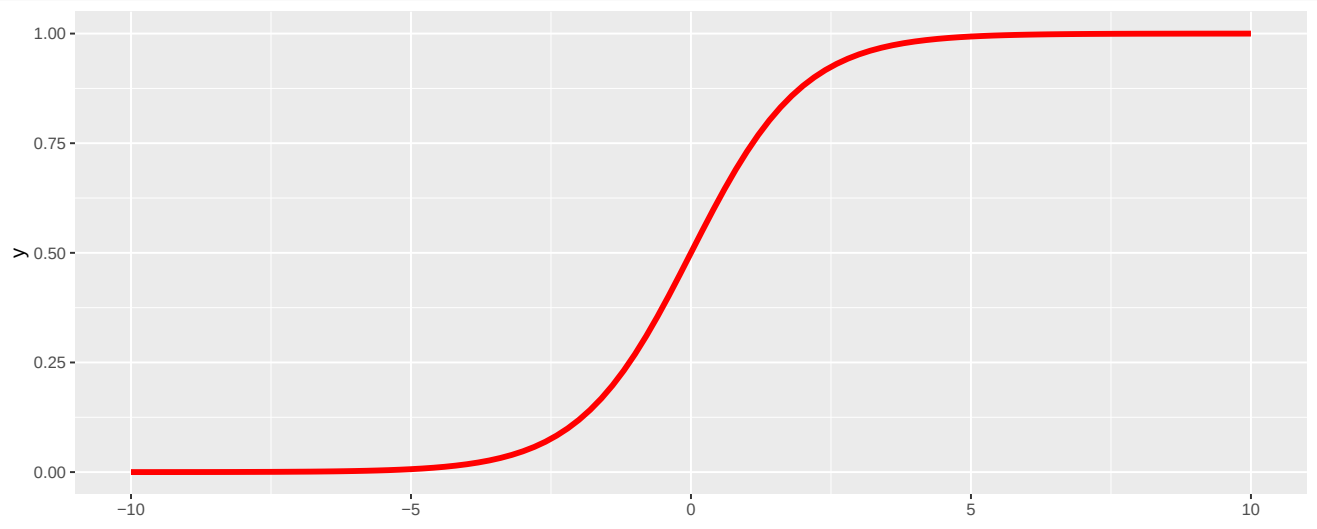
缺点：

- BP 神经网络参数众多，每次迭代需要更新较多数量的阈值和权值，故收敛速度比较慢。
- 网络中隐层含有的节点数目没有明确的准则，需要不断设置节点数字试凑，根据网络误差结果最终确定隐层节点个数
- BP 算法是一种速度较快的梯度下降算法，容易陷入局部极小值的问题。

4. 激活函数

- Sigmoid

$$\text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$$

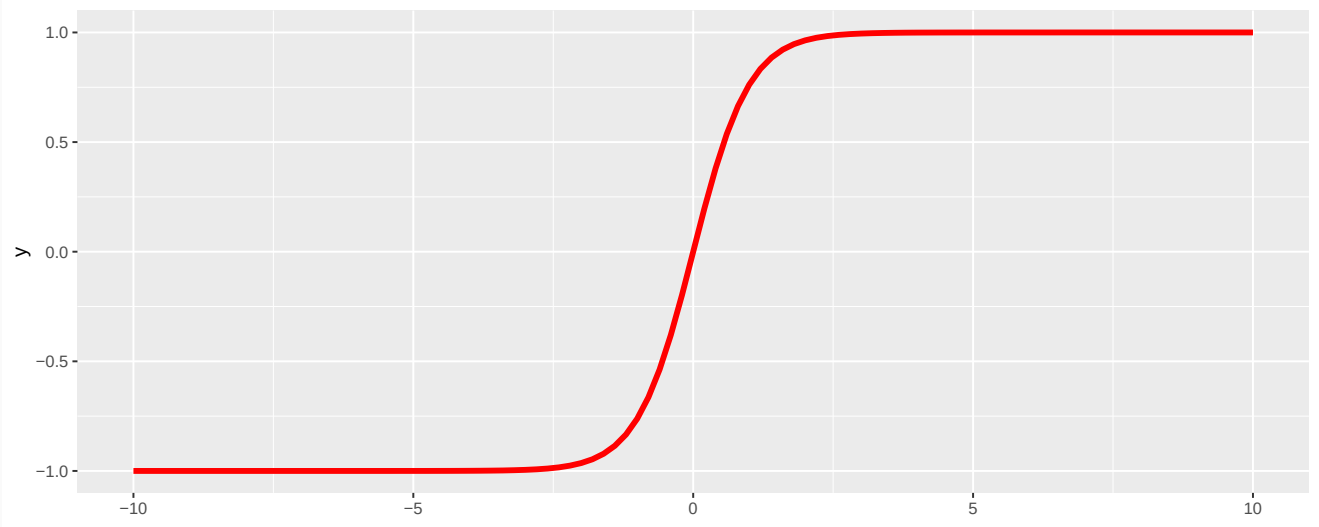


优点：函数处处连续，便于求导；可将函数值的范围压缩至 $[0, 1]$，可用于压缩数据，且幅度不变；便于前向传输。

缺点：在趋向无穷的地方，函数值变化很小，容易出现梯度消失，不利于深层神经的反馈传输；幂函数的梯度计算复杂；收敛速度比较慢。

- tanh

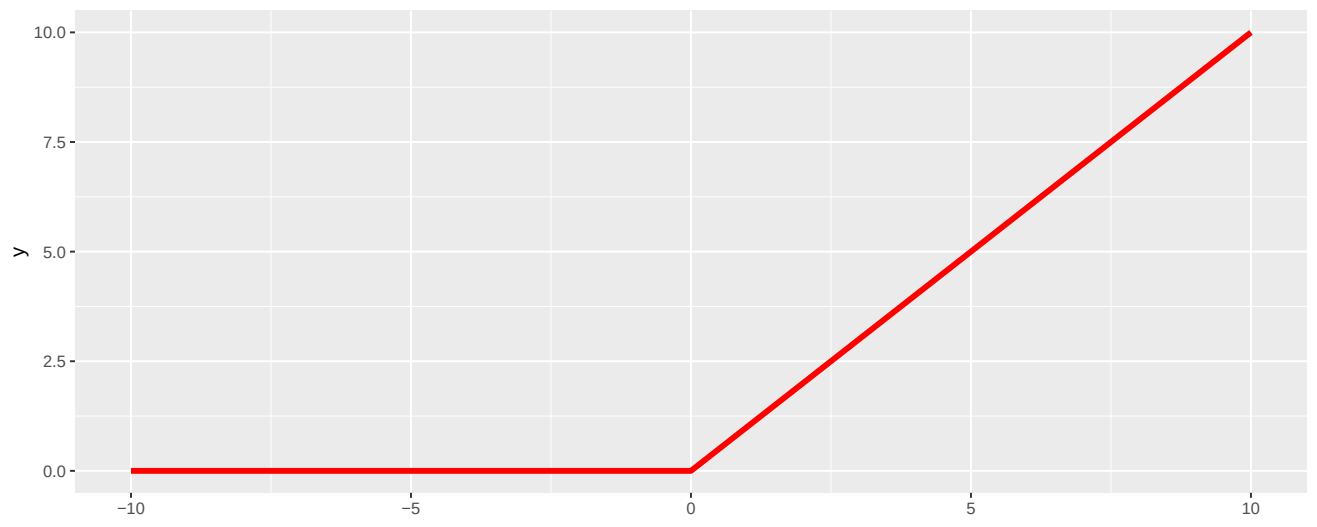
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



其输出落在 $[-1, 1]$，可以进行标准化。同时 $\tanh$ 的曲线在较大时变得平坦的幅度没有 sigmoid 那么大，这样求梯度变化值有一些优势。

- reLU

$$\mathrm{reLU}(z) = \max\{0, z\}$$

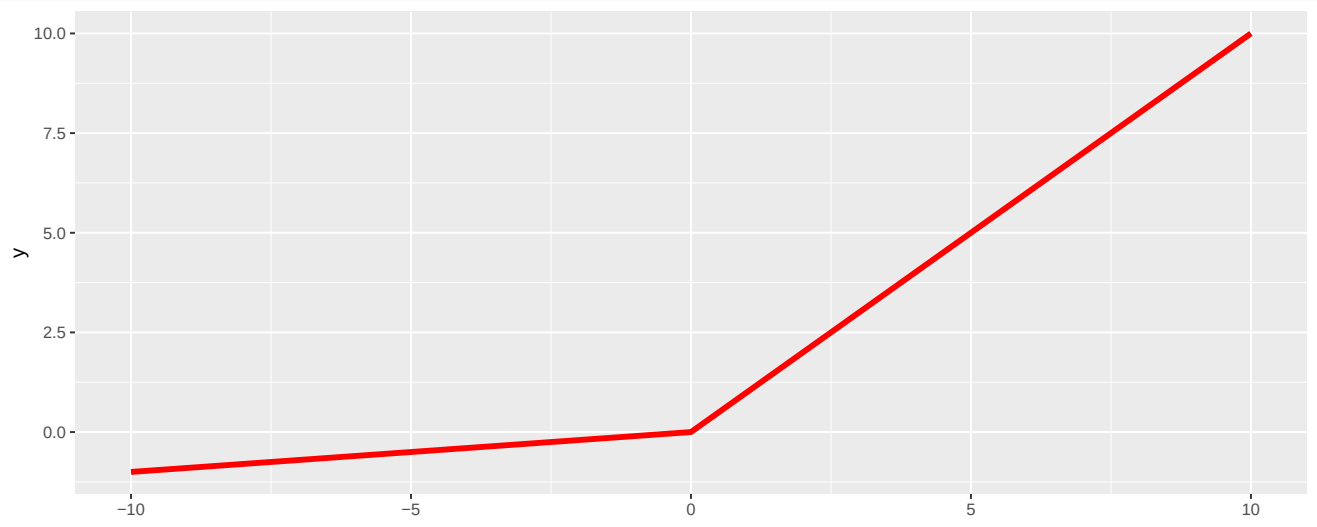


ReLU 在深度学习中大获成功，其好处是梯度连乘的结果也只能取 0（梯度从该位置停止前向传播）或 1（梯度保持值不变进行前向传播），能解决梯度消失问题。

- Leakly ReLU

为了解决神经元”死亡”问题，它与 ReLU 很相似，仅在输入小于 0 部分有差别：ReLU 取值为 0，而 LeakyReLU 取值为负，且有微小的梯度。

$$\mathrm{LeaklyReLU}(z) = \begin{cases} z, & z > 0 \\ \alpha z, & z \leq 0 \end{cases}$$



- softmax

多分类问题专用的激活函数，是二分类 sigmoid 的多分类推广版。若因变量有 K 个类别，则如下的 softmax 变换能将实数值 z_i 转化为属于各个类别的概率值：

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \qquad i = 1, \cdots, K$$

例如，假设输出层为 3 个神经元，而未激活的输出为 $3, 1, -3$, 接 softmax 层转化为属于 3 个类别的概率：

```
z = c(3, 1, -3)
exp(z) / sum(exp(z))
#> [1] 0.87888 0.11894 0.00218
```

- 纯线性函数（恒等变换）

$$f(z) = z$$

注意，对于 BP 神经网络的回归问题，因为因变量一般属于 $(-\infty, \infty)$, 这就需要将激活函数设为纯线性函数，即对输出不做任何变换。即使如此，神经网络模型经过隐层的自然叠加，也已具有了非线性预测的能力。

与 Logistic 回归一样，二分类问题采用交叉熵损失：

$$J(\mathbf{w}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \ln \hat{p}_i + (1 - y^{(i)}) \ln(1 - \hat{p}_i) \right]$$

其中，y_i 为样本真实类别，$\hat{p}_i = \Pr\{y = 1 | X\}$ 为预测为正例的概率。

```
y = c(0, 0, 1, 1)                          # 真实类别
p = c(0.1, 0.2, 0.7, 0.99)                 # 预测为正例的概率
- mean(y * log(p) + (1-y) * log(1-p))      # 交叉熵损失
#> [1] 0.174
```

对应多分类 `softmax` 激活的损失函数是对数似然损失：

$$J(\mathbf{W}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \ln \hat{y}_i$$

其中，y_i 为第 i 个样本虚拟变量表示的真实类别，$\hat{y}_i$ 为第 i 个样本预测属于每个类别的概率向量。

```r
library(tidyverse)
y = sjmisc::to_dummy(tibble(c(0,0,1,1,2,2)), 1)
yh = tribble(~V1, ~V2, ~V3,
             0.6, 0.1, 0.3,
             0.5, 0.2, 0.3,
             0.3, 0.6, 0.1,
             0.1, 0.8, 0.1,
             0.3, 0.3, 0.4,
             0.2, 0.5, 0.3)
res = apply(y * log(yh), 1, sum)
- mean(res)
#> [1] 0.676
```

注 1：神经网络的反向传播需要大量的链式法则求导，上述所有损失函数、激活函数搭配使用时，其求导形式都非常简单，便于计算。

注 2：在深度神经网络反向传播的算法过程中，由于使用了矩阵求导的链式法则，有一大串连乘，若连乘的数值在每层都小于 1 的，则梯度越乘越小，导致**梯度消失**（用 ReLU 激活部分解决），而若连乘的数值在每层都大于 1，则梯度越乘越大，导致**梯度爆炸**（调整初始参数值解决）。

6. 过拟合与正则化

神经网络也有过拟合的问题，解决办法有：

- 类似正则化回归，在构造损失函数时加入 $L1$ 或 $L2$ 正则项；
- 类似集成学习装袋法，对训练数据有放回随机采样，得到多个数据副本训练多个神经网络模型，加权平均或投票法决定最终输出；
- `dropout` 法：类似决策树的剪枝，在用前向传播和反向传播算法训练神经网络模型的分批数据迭代时，分别随机地从全连接神经网络中去掉一部分隐藏层的神经元；
- 增强数据集，例如深度学习中的图像识别，可以将原始图像稍微的平移或者旋转一点点，则得到了一个新的图像。

三. 神经网络回归案例

二手车价格预测，数据包含 1435 个观测，8 个变量，其中 Price 为目标变量，其余为特征，包括 Age（按月车龄）、KM（行驶里程）、Weight（车重）、HP（马力）、CC（发动机尺寸）、MetColor（是否金属色）、Doors（车门数）。

1. 创建任务

```
library(mlr3verse)
dat = read_csv("data/CarPrices.csv")
dat[1:2,]
#> # A tibble: 2 x 8
#>     Age    KM Weight    HP MetColor    CC Doors Price
#>   <dbl> <dbl>  <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl>
#> 1    23 46986   1165    90        1  2000     3 13500
#> 2    23 72937   1165    90        1  2000     3 13750
```

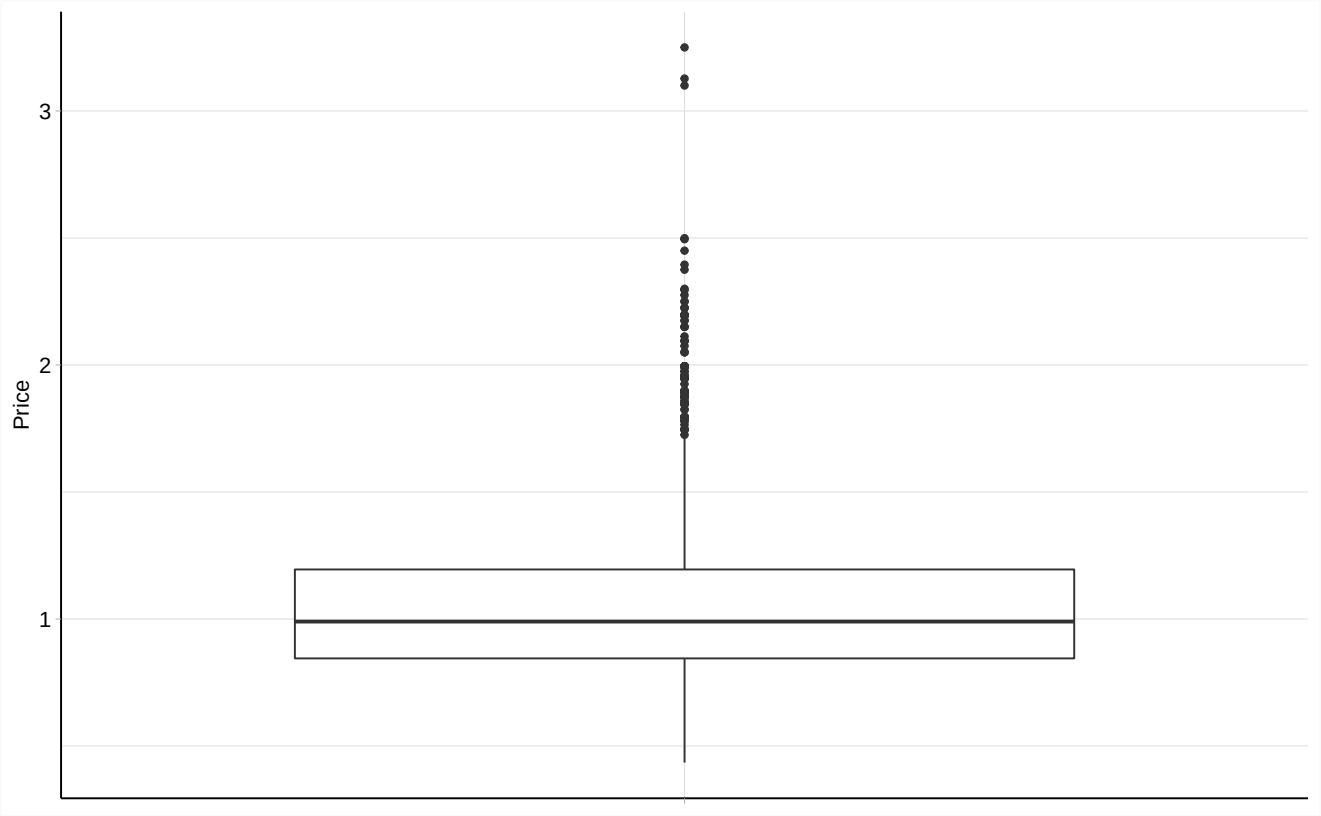
数据预处理

- MetColor 转化成因子，其它特征分正向、负向归一化到 $[-1, 1]$
- Price 除以 10000, 转化成万美元

```
rescale = function(x, type = "pos", a = 0, b = 1) {
  rng = range(x, na.rm = TRUE)
  switch (type,
    "pos" = (b - a) * (x - rng[1]) / (rng[2] - rng[1]) + a,
    "neg" = (b - a) * (rng[2] - x) / (rng[2] - rng[1]) + a)
}
dat = dat %>%
  mutate(MetColor = as.factor(MetColor),
         across(1:2, rescale, type = "neg", a = -1),
         across(c(3:4,6:7), rescale, a = -1),
         Price = Price / 10000)
```

```r
task = as_task_regr(dat, target = "Price")
task
#> <TaskRegr:dat> (1435 x 8)
#> * Target: Price
#> * Properties: -
#> * Features (7):
#>   - dbl (6): Age, CC, Doors, HP, KM, Weight
#>   - fct (1): MetColor
```

```
autoplot(task)
```



```
nnet_reg = lrn("regr.nnet")   # 需要 nnet 包
nnet_reg
#> <LearnerRegrNnet:regr.nnet>
#> * Model: -
#> * Parameters: size=3
#> * Packages: mlr3, mlr3learners, nnet
#> * Predict Types:  [response]
#> * Feature Types: numeric, factor, ordered
#> * Properties: weights
```

3. 划分训练集和测试集

- 做留出（Holdout）重抽样，80% 作为训练集，其余 20% 作为测试集
- 为了保持训练集、测试集的因变量数据具有相似的分布，采用分层抽样方法
- 用 partition() 函数对任务做划分，默认按因变量分层，取出训练集索引和测试集索引

```
set.seed(123)
split = partition(task, ratio = 0.8)
# 默认 stratify = TRUE
```

```
nnet_reg$param_set   # 查看所有超参数及默认值
#> <ParamSet>
#>             id    class lower upper nlevels        default
#>  1:       Hess ParamLgl    NA    NA       2          FALSE
#>  2:    MaxNWts ParamInt     1   Inf     Inf           1000
#>  3:        Wts ParamUty    NA    NA     Inf <NoDefault[3]>
#>  4:     abstol ParamDbl  -Inf   Inf     Inf          1e-04
#>  5:   censored ParamLgl    NA    NA       2          FALSE
#>  6: contrasts ParamUty    NA    NA     Inf
#>  7:      decay ParamDbl  -Inf   Inf     Inf              0
#>  8:       mask ParamUty    NA    NA     Inf <NoDefault[3]>
#>  9:      maxit ParamInt     1   Inf     Inf            100
#> 10: na.action ParamUty    NA    NA     Inf <NoDefault[3]>
#> 11:       rang ParamDbl  -Inf   Inf     Inf            0.7
#> 12:     reltol ParamDbl  -Inf   Inf     Inf          1e-08
```

- 对模型中超参数：隐层神经元个数 `size` 做调参
- 使用自动调参器，需要设置学习器、重抽样方法、模型评估指标、搜索空间、终止条件、搜索方法

```
search_space = ps(size = p_int(lower = 2, upper = 30))

at = auto_tuner(
  learner = nnet_reg,
  resampling = rsmp("cv", folds = 3),
  measure = msr("regr.rsq"),
  search_space = search_space,
  method = "random_search",
  term_evals = 20)
```

- 在训练集上启动调参过程

```
lgr::get_logger("mlr3")$set_threshold("warn")
lgr::get_logger("bbotk")$set_threshold("warn")
set.seed(12)
at$train(task, row_ids = split$train)
#> # weights:  172
#> initial  value 4878.780262
#> iter  10 value 12.160967
#> iter  20 value 9.504070
#> iter  30 value 8.457319
#> iter  40 value 7.745437
#> iter  50 value 7.466033
#> iter  60 value 7.253821
#> iter  70 value 7.106630
#> iter  80 value 6.984700
#> iter  90 value 6.813887
#> iter 100 value 6.648236
```

- 查看最优超参数

```
at$tuning_result    # 调参结果
#>    size learner_param_vals  x_domain regr.rsq
#> 1:    5          <list[1]> <list[1]>    0.893
```

5. 训练模型

- 用调出的最优参数更新学习器的参数集，然后训练模型

```
nnet_reg$param_set$values =
  at$tuning_result$learner_param_vals[[1]]
nnet_reg$train(task, row_ids = split$train)
#> # weights:  46
#> initial  value 1204.050529
#> iter  10 value 57.392714
#> iter  20 value 25.432528
#> iter  30 value 16.037903
#> iter  40 value 15.080412
#> iter  50 value 14.824141
#> iter  60 value 14.760277
#> iter  70 value 14.735181
#> iter  80 value 14.698750
#> iter  90 value 14.633069
```

```
predictions = nnet_reg$predict(task,
                               row_ids = split$test)
predictions
#> <PredictionRegr> for 288 observations:
#>     row_ids truth response
#>           1 1.350    1.636
#>          31 1.295    1.562
#>         186 0.695    0.153
#> ---
#>         357 1.499    1.392
#>         110 3.250    2.860
#>         141 2.395    2.508
```

```r
msr("regr.rmse")$score(predictions)  # 均方根误差
#> [1] 0.132
msr("regr.rsq")$score(predictions)   # R 方
#> [1] 0.876
```

7. 预测新数据

```
newdata = dat[1:5,-8]
nnet_reg$predict_newdata(newdata)
#> <PredictionRegr> for 5 observations:
#>  row_ids truth response
#>        1    NA     1.64
#>        2    NA     1.56
#>        3    NA     1.63
#>        4    NA     1.54
#>        5    NA     1.52
```

注意，真正的新数据，需要做同样的归一化等预处理。

以 iris 数据为例，这是多分类问题。

1. 选取任务

```
task = tsk("iris")
task
#> <TaskClassif:iris> (150 x 5): Iris Flowers
#> * Target: Species
#> * Properties: multiclass
#> * Features (4):
#>   - dbl (4): Petal.Length, Petal.Width, Sepal.Length, Sepa
```

```
nnet_classif = lrn("classif.nnet")   # 需要 nnet 包
nnet_classif
#> <LearnerClassifNnet:classif.nnet>
#> * Model: -
#> * Parameters: size=3
#> * Packages: mlr3, mlr3learners, nnet
#> * Predict Types:  response, [prob]
#> * Feature Types: numeric, factor, ordered
#> * Properties: multiclass, twoclass, weights
```

3. 划分训练集和测试集

- 做留出（Holdout）重抽样，80% 作为训练集，其余 20% 作为测试集
- 为了保持训练集、测试集的因变量数据具有相似的分布，采用分层抽样方法
- 用 partition() 函数对任务做划分，默认按因变量分层，取出训练集索引和测试集索引

```
set.seed(123)
split = partition(task, ratio = 0.8)
# 默认 stratify = TRUE
```

```
nnet_classif$param_set   # 查看所有超参数及默认值
#> <ParamSet>
#>             id    class lower upper nlevels        default
#>  1:       Hess ParamLgl    NA    NA       2          FALSE
#>  2:    MaxNWts ParamInt     1   Inf     Inf           1000
#>  3:        Wts ParamUty    NA    NA     Inf <NoDefault[3]>
#>  4:     abstol ParamDbl  -Inf   Inf     Inf          1e-04
#>  5:   censored ParamLgl    NA    NA       2          FALSE
#>  6: contrasts ParamUty    NA    NA     Inf
#>  7:      decay ParamDbl  -Inf   Inf     Inf              0
#>  8:       mask ParamUty    NA    NA     Inf <NoDefault[3]>
#>  9:      maxit ParamInt     1   Inf     Inf            100
#> 10: na.action ParamUty    NA    NA     Inf <NoDefault[3]>
#> 11:       rang ParamDbl  -Inf   Inf     Inf            0.7
#> 12:     reltol ParamDbl  -Inf   Inf     Inf          1e-08
```

- 对模型中超参数：隐层神经元个数 size 做调参
- 使用自动调参器，需要设置学习器、重抽样方法、模型评估指标、搜索空间、终止条件、搜索方法

```
search_space = ps(size = p_int(lower = 2, upper = 20))

at = auto_tuner(
  learner = nnet_classif,
  resampling = rsmp("cv", folds = 3),
  measure = msr("classif.acc"),
  search_space = search_space,
  method = "random_search",
  term_evals = 20)
```

- 在训练集上启动调参过程

```
lgr::get_logger("mlr3")$set_threshold("warn")
lgr::get_logger("bbotk")$set_threshold("warn")
set.seed(12)
at$train(task, row_ids = split$train)
#> # weights:  91
#> initial  value 95.691428
#> iter  10 value 7.594173
#> iter  20 value 7.279926
#> iter  30 value 4.455860
#> iter  40 value 4.322333
#> iter  50 value 4.319984
#> iter  60 value 4.314784
#> iter  70 value 4.314483
#> iter  80 value 4.314411
#> final  value 4.314407
#> converged
```

- 查看最优超参数

```
at$tuning_result    # 调参结果
#>    size learner_param_vals  x_domain classif.acc
#> 1:   11          <list[1]> <list[1]>        0.95
```

5. 训练模型

- 用调出的最优参数更新学习器的参数集，然后训练模型

```
nnet_classif$param_set$values =
  at$tuning_result$learner_param_vals[[1]]
nnet_classif$train(task, row_ids = split$train)
#> # weights:  91
#> initial  value 166.825101
#> iter  10 value 49.922461
#> iter  20 value 3.385058
#> iter  30 value 0.288802
#> iter  40 value 0.003393
#> iter  50 value 0.001060
#> final  value 0.000072
#> converged
```

6. 模型预测及评估

```
predictions = nnet_classif$predict(task,
                                   row_ids = split$test)
predictions
#> <PredictionClassif> for 30 observations:
#>     row_ids     truth   response prob.setosa prob.versicol
#>           1    setosa     setosa    1.00e+00        1.81e-
#>           2    setosa     setosa    1.00e+00        1.81e-
#>           6    setosa     setosa    1.00e+00        1.81e-
#> ---
#>         134 virginica versicolor    2.73e-46        1.00e+
#>         138 virginica  virginica    7.07e-47        2.82e-
#>         147 virginica  virginica    7.07e-47        2.82e-
```

```
predictions$confusion                    # 混淆矩阵
#>              truth
#> response      setosa versicolor virginica
#>   setosa          10          0         0
#>   versicolor       0         10         2
#>   virginica        0          0         8
msr("classif.acc")$score(predictions)  # 准确率
#> [1] 0.933
```

7. 预测新数据

```
newdata = task$head(5)[,-1]
nnet_classif$predict_newdata(newdata)
#> <PredictionClassif> for 5 observations:
#>  row_ids truth response prob.setosa prob.versicolor prob.v
#>        1  <NA>   setosa           1        1.81e-06      7
#>        2  <NA>   setosa           1        1.81e-06      7
#>        3  <NA>   setosa           1        1.81e-06      7
#>        4  <NA>   setosa           1        1.81e-06      7
#>        5  <NA>   setosa           1        1.81e-06      7
```

主要参考文献

[1] mlr3book. 2022. https://mlr3book.mlr-org.com/

[2] 刘建平. 深度神经网络 (DNN) 系列. 博客园, 2017.

[3] 龙良曲. Tensorflow 深度学习：深入理解人工智能算法设计. 清华大学出版社, 2020.

[4] 黄海广. 机器学习课件全集 v1. 温州大学. 2021.