神经网络模型推导、相关 R 包介绍及应用

Neural Network and R Language

姚炜彤　　郑雨薇 (Yuwei Zheng)

版本：*1.0.0*

更新：*May 19, 2019*

本文主要在 Lecture -Neural Network 的基础上对反向传播神经网络机制进行推导，并介绍 R 语言里 nnet, neuralnet, AMORE 四类 package 的特点和相关应用，最后运用 R 自带的数据集 Fishing 在 IIA violation 的情况下用 nnet 实现 Multinomial Probit Model (Both hints and mechanism come from Professor Mao)。 Github Page of Weitong Yao Github Page of Yuwei Zheng

1　神经网络类型和 Two Layer-BP Neural Network 推导

1.1　神经网络类型

作为深度学习的主要算法之一，神经网络有很多方面应用，对于不同的算法要求的神经网络类型也不同。常见的神经网络类型有感知机（单层线性）、前馈型神经网络、卷积神经网络（Convolution，通常用于图像处理）、循环神经网络（数据存在先后顺序关系，常用于语音识别、情感分类、生物 DNA 处理）等等 [1]。常用的编程语言是 Python 及其 Tensorflow，Tensorboard 两个包，其优点在于可以将神经网络的机制可视化和图片处理过程可视化，缺点在于训练速度缓慢，所以通常采用 Tensorflow 的 GPU 版本在 GPU 上进行运算（笔者亲测在 CPU 上的 EPOCH =2000 的 CNN 运行需要 45 分钟左右）。

1.2　Two Layer-BP Neural Network 机制推导 [1][2]

神经网络的三个组成部分为：激活函数，权重，输出。先以正向传播的机制为例：

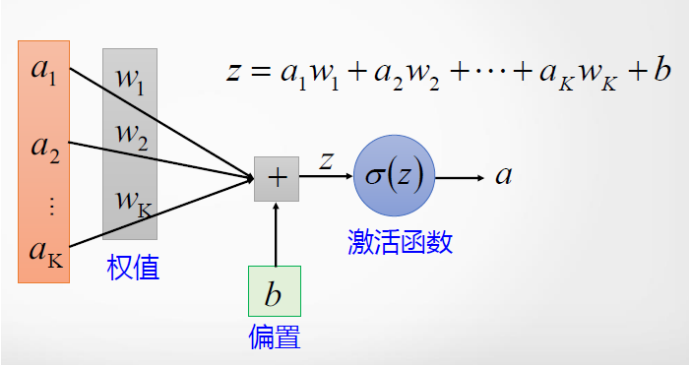


图 1: 来源：《*Python* 与深度学习》厦门大学软件学院邱明

其中每一个节点的输出 a 都将作为下一层的节点的输入，如果用 l 代表层数，j 表示该层的第 j 神经元，w_{ki}^{l+1} 通常是一个矩阵，a_i^l 和 b_k^l 则是向量，Sigmoid 激活函数及其导函数的写法为：

$$z_k^{l+1} = \sum_i^K w_{ki}^{l+1} a_i^l + b_k^{l+1} = \sum_i^K w_{ki}^{l+1} \sigma\left(z_i^l\right) + b_k^{l+1} \tag{1}$$

$$\frac{\partial \sigma\left(z_i^l\right)}{\partial z_i^l} = a_i^l\left(1 - a_i^l\right) \tag{2}$$

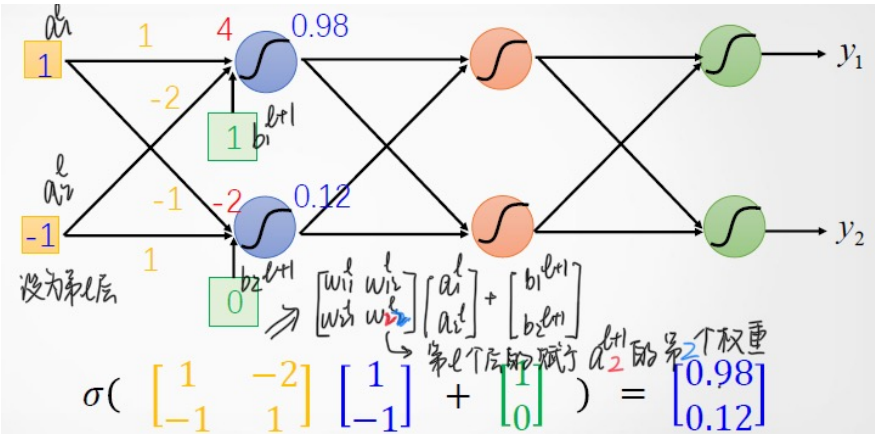


图 2: 来源：《*Python* 与深度学习》厦门大学软件学院邱明

本节介绍两层的全连接反向传播算法神经网络的推导和权重的更新机制，其中激活函数采用 Sigmoid，输出层采用 Softmax。反向传播算法的过程是：（1）向前传导（如上图），得到最后的总的代价函数。（2）计算输出层 L 每一个神经元的误差（求导）。（3）计算输出层之前每一层的第 j 个神经元的偏置 b 和权重 w 误差（求导）。（4）根据（2）（3）修改参数使得 C 变小。从下面的推导可以看出反向传播算法是通过链式法则，不断调整 W 和 b 来实现代价函数 $C = \frac{1}{2}\sum_j \left(y_j - a_i^L\right)^2$ 的最小化。其中 $a_j^l = \sigma\left(z_i^l\right)$ (上一个节点输出 = 下一个节点输入)。

最后一个输出层 L 的第 j 个神经元误差为（当且仅当 $k = j$ 时，$\frac{\partial a_k^L}{\partial z_j^L}$ 才不为 0）：

$$\delta_{\mathrm{j}}^{\mathrm{L}} = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^{\mathrm{L}}} \frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'\left(z_j^L\right) \tag{3}$$

第 l 层的第 j 个神经元误差为（结果可以看到代价函数对 z_j^l 的误差可以转换成 z_j^{l+1} 和 z_j^l 的关系）：

$$\delta_{\mathrm{j}}^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{|\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(z_j^l\right) \tag{4}$$

对第 l 层第 j 个神经元的偏置求导：

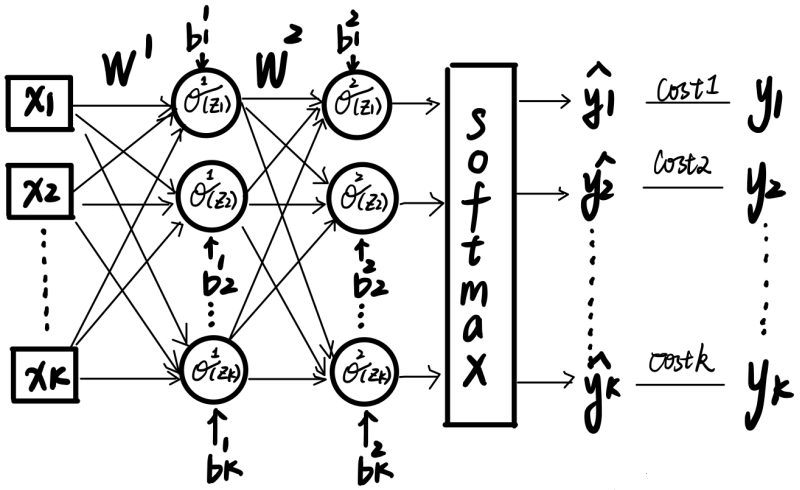
$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} = \delta_{\mathrm{j}}^l \tag{5}$$

对第 l 层第 j 个神经元的权重求导：

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_i \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial \mathrm{w}_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \delta_{\mathrm{j}}^l \tag{6}$$

由上可以看出调整 w 和 b 的参数修改方式：（ϵ 为学习率）

$$w_{kj}^{l} = w_{kj}^{l} - \epsilon a_{k}^{l-1}\delta_{j}^{l} \quad b_{j}^{l} = w_{j}^{l} - \epsilon\delta_{j}^{l} \tag{7}$$



2 R 语言相关包介绍

2.1 nnet

nnet 包的描述是“Software for feed-forward neural networks with a single hidden layer, and for multinomial log-linear models”，即不仅可以提供前馈反向传播神经网络算法, 还可以提供 MNL 拟合，其指令是 multinom() (另一种方法是 Lecture - Classification 使用的 mlogit 包)。神经网络用法如下：

```
nnet(x, y, weights, size, Wts, mask,
     linout = FALSE, entropy = FALSE, softmax = FALSE,
     censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
     maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
     abstol = 1.0e-4, reltol = 1.0e-8, ...)

predict(object, newdata, type = c("raw","class"), ...)
```

nnet() 里 decay 默认为 0，当不为 0 时表示权重是递减的，防止过拟合；skip 表示是否跳过隐藏层。predict() 里 type = “raw’’ 表述输出的是训练后网络的返回值，type = “class’’ 则返回对应的类。由于 nnet 是 Backpropagation，所以在调整参数方面采用梯度下降法（梯度下降法）。如等式（7）所示，由于 $\frac{\partial C}{\partial w_{jk}^{l}} = a_{k}^{l-1}\delta_{\mathrm{j}}^{l}$，为了使代价函数下降最快，因为代价函数二阶泰勒展开：$C(x) = C(x_k) + \epsilon a_{k}^{l-1}\delta_{\mathrm{j}}^{l} + o(\epsilon)$（$\epsilon$ 即步长，或者学习率），又因为柯西施瓦茨不等式有：$\left|a_{k}^{l-1}\delta_{\mathrm{j}}^{l}\right| \leq \left\|a_{k}^{l-1}\right\| \left\|\delta_{\mathrm{j}}^{l}\right\|$，当且仅当 $a_{k}^{l-1} = -\delta_{\mathrm{j}}^{l}$ 时，C(x) 下降量最大。[3]

2.2 neuralnet

使用反向传播训练神经网络，有权值和没有权值反向跟踪的弹性反向传播。允许自定义激活函数 act.fct 等参数。

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
  stepmax = 1e+05, rep = 1, startweights = NULL,
    learningrate.limit = NULL, learningrate.factor = list(minus = 0.5,
    plus = 1.2), learningrate = NULL, lifesign = "none",
    lifesign.step = 1000, algorithm = "rprop+", err.fct = "sse",
    act.fct = "logistic", linear.output = TRUE, exclude = NULL,
    constant.weights = NULL, likelihood = FALSE)
```

弹性反向传播（RProp 算法）的特点是可以依靠参数梯度的符号，动态调整学习步长（学习率）：当连续误差梯度符号不变时，采用加速策略，加快训练速度；当连续误差梯度符号变化时，采用减速策略，以期稳定收敛。该算法的缺点在于只考虑了单次的步长更新，没有考虑到之前的动态调整情况。[4]

2.3 AMORE

AMORE 包下的 newff() 函数是建立一个标准的前馈神经网络，其参数如下：

```
newff(n.neurons, learning.rate.global, momentum.global, error.criterium, Stao,
  hidden.layer, output.layer, method)
```

newff() 与其他包相比有更强的灵活性，比如可以通过 n.neurons 自定义输入神经元的数量，momentum.global 设置全局的每个神经元的动量，hidden.layer 和 output.layer 分别控制隐藏层和输出层神经元的激活函数包括“purelin”（线性 y=x）,tansig “,"sigmoid", “hardlim” ,"custom"，但是与 neuralnet 包的函数相比，用户自定义激活函数需要从包的环境进行设置，缺乏 neuralnet 包直接调用 function( ) 函数的方便性。

3 MNP 和神经网络结合 (Challenge，Hints from Professor Mao)

本节运用 R 自带的数据集 Fishing 在 IIA violation 的情况下用 nnet 实现 Multinomial Probit Model。首先用 mlogit 包对 Fishing 数据的 mode 进行 MNP 回归，再用 nnet 结合 MNL（假设 residual iid）进行反向传播算法训练，最后设计 MNP 的一般形式的激活函数 actfun( ) 并用原数据集进行测试。结合 MNP 进行多分类的神经网络激活函数设计如下：

对于每一个 mode 的选择的效用为：

$$U_{ij} = \alpha_j + \delta_j \text{ price }_{ij} + \gamma_j \text{ income }_i + e_{ij}, \qquad e_i \sim \mathcal{N}(0, \Sigma) \tag{8}$$

带入回归系数为 (as a benchmark $\widehat{U}_{i,\text{ beach}} = 0$)：

$$\widehat{U}_{i,\text{ pier}} = 2.9661e-01 - 3.7634e-05 \times \text{ income }_{i,\text{ pier}} - 6.9660e-03 \times \text{ price }_i + \epsilon_{i,\text{ pier}}$$

$$\widehat{U}_{i,\text{ boat}} = -1.1658e-01 + 3.9809e-05 \times \text{ income }_{i,\text{ boat}} - 6.9660e-03 \times \text{ price }_i + \epsilon_{i,\text{ boat}}$$

$$\widehat{U}_{i,\text{ charter}} = 4.7498e-01 - 4.6794e-05 \times \text{ income }_{i,\text{ charter}} - 6.9660e-03 \times \text{ price }_{i} + \epsilon_{i,\text{ charter}}$$

经过标准化之后每个拟合方程的 residual 的分布为标准正态分布，所以有：

$$\begin{aligned}
\Pr(y_i = j|x_i) &= \Pr\left(U_{ij} > U_{i\ell}, \forall \ell \neq j|x_i\right) \\
&= \Pr\left(f_j(x_i) + e_{ij} > f_\ell(x_i) + e_{i\ell}, \forall \ell \neq j|x_i\right) \\
&= \Pr\left(e_{ij} - e_{i\ell} > f_j(x_i) - f_\ell(x_i), \forall \ell \neq j|x_i\right) \\
&= \Pr\left(\Delta e_i > \Delta U_j(x_i), \forall \ell \neq j|x_i\right)
\end{aligned} \tag{9}$$

激活函数为 $\int \Pr(y_i = j|x_i)\, d\Delta e_i = 1 - \Phi(\Delta U_j(x_i), \forall \ell \neq j|x_i)$

```
>
> data("Fishing", package = "mlogit")
> View(Fishing)
> prop.table(table(Fishing$mode))

    beach      pier      boat   charter
0.1133672 0.1505922 0.3536379 0.3824027
>
> library(mlogit)
> library(AER)
>
> #reshape data
> Fish.long <- mlogit.data(Fishing, varying = c(2:9), shape = "wide", choice = "mode")
> head(Fish.long,5)
           mode   income     alt   price  catch chid
1.beach   FALSE 7083.332   beach 157.930 0.0678    1
1.boat    FALSE 7083.332    boat 157.930 0.2601    1
1.charter  TRUE 7083.332 charter 182.930 0.5391    1
1.pier    FALSE 7083.332    pier 157.930 0.0503    1
2.beach   FALSE 1250.000   beach  15.114 0.1049    2
>
> # multinomial probit with one var normalized to zero
> pro_fit <- mlogit(mode~ price|income,Fish.long,reflevel = "beach",probit = TRUE)
> coeftest(pro_fit)

t test of coefficients:

                      Estimate  Std. Error t value  Pr(>|t|)
boat:(intercept)    -1.1658e-01  1.2013e-01 -0.9704  0.332047
charter:(intercept)  4.7498e-01  1.6901e-01  2.8104  0.005031 **
pier:(intercept)     2.9661e-01  1.4304e-01  2.0736  0.038331 *
price               -6.9660e-03  1.4466e-03 -4.8154 1.661e-06 ***
boat:income          3.9809e-05  2.2140e-05  1.7981  0.072417 .
charter:income      -4.6794e-05  2.4837e-05 -1.8840  0.059809 .
pier:income         -3.7634e-05  2.7878e-05 -1.3499  0.177296
boat.charter        -4.0624e-01  1.3961e-01 -2.9098  0.003685 **
boat.pier            6.1337e-01  1.0188e-01  6.0207 2.321e-09 ***
charter.charter      7.6262e-01  2.6189e-01  2.9120  0.003659 **
charter.pier         6.2934e-01  2.2045e-01  2.8548  0.004382 **
pier.pier            1.3423e-01  1.3714e-01  0.9788  0.327890
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

>
> # covariance matrix using "beach" as referenc
> pro_fit$omega$beach
              boat    charter      pier
boat     1.0000000 -0.4062352 0.6133727
charter -0.4062352  0.7466196 0.2307761
pier     0.6133727  0.2307761 0.7903152
>
> library(nnet)
>
> set.seed(5)
> n <-length(Fishing[,1])
> samp <- sample(1:n,n/5)
> traind <- Fishing[-samp,c(2:10)]
> train1 <-as.numeric(Fishing[-samp,1])
> train2 <-Fishing[-samp,1]
> testd <- Fishing[samp,c(2:10)]
> test1 <- as.numeric(Fishing[samp,1])
> test2 <- Fishing[samp,1]
> # label Fishing$mode as 1: beach(benchmark)  2: pier  3: boat  4: charecter
>
>
> fit <- nnet( traind , train1 , maxiter = 1000,
+             linout = TRUE, size = 2, decay = 0.1)
# weights:  23
initial  value 10049.487562
iter  10 value 982.065611
iter  20 value 799.686462
iter  30 value 797.633708
iter  40 value 797.631181
final  value 797.623571
converged
>
>
> # Define an activation func for multinomial probit model
> # Using the mlogit to define the dist of error term
> mode_predict <- predict(fit, testd, type = "raw")
>
> # Correct Rate
> 1-abs(mean(test1 - mode_predict))
[1] 0.9436065
>
> # almost all results in fit_multinomial probit regression are significant
> # 1: beach(benchmark)  2: pier  3: boat  4: charecter
> # uti_ <- [uti_pier, uti_boat, uti_charecter]
> b0 <- c(2.9661e-01, -1.1658e-01, 4.7498e-01)     # intercept
> b1 <- c(-6.9660e-03, -6.9660e-03, -6.9660e-03)  # pirce
```

```
> b2 <- c(-3.7634e-05, 3.9809e-05, -4.6794e-05 )  # income
> coeff <- cbind(b0,b1,b2)
> coeff <- as.matrix(coeff)
>
> uti<- matrix(rep(0),nrow = 946, ncol = 3 ,byrow =T)
> for (i in 1:946){
+   ind <- as.vector(traind[i,])
+   p <- as.vector(ind[,c(2,3,4)])
+   x <- rbind(c(1,1,1),p,as.vector(ind[,9]))
+   uti[i,] <- c(coeff[1,]%*%(x[,1]),coeff[2,]%*%x[,2],coeff[3,]%*%x[,3])
+ }
> uti_<-cbind(rep(0),uti)
>
> pro <- matrix(rep(0),nrow = 946, ncol = 4 ,byrow =T)
> output <-matrix(rep(0),nrow = 946, ncol = 1 ,byrow =T)
> for(i in 1:946){
+   pro[i,] <- 1-pnorm(uti_[i,],0,1)
+   output[i,] <- which(pro[i,]==max(pro[i,]),arr.ind = T)
+ }
>
> 1-abs(mean(train1 - output))
[1] 0.5972516
>
> ######################################################
> ###    Form the generalized activation function
> ######################################################
>
>
> b0 <- c(2.9661e-01, -1.1658e-01, 4.7498e-01)     # intercept
> b1 <- c(-6.9660e-03, -6.9660e-03, -6.9660e-03)  # pirce
> b2 <- c(-3.7634e-05, 3.9809e-05, -4.6794e-05 )  # income
> coeff <- cbind(b0,b1,b2)
> coeff <- as.matrix(coeff)
>
>
> actfun <-function(x){
+   len <- length(x[,1])
+   uti<- matrix(rep(0),nrow = len, ncol = 3 ,byrow =T)
+   for (i in 1:len){
+     ind <- as.vector(x[i,])
+     p <- as.vector(ind[,c(3,4,5)])
+     beta <- rbind(c(1,1,1),p,as.vector(ind[,10]))
+     uti[i,] <- c(coeff[1,]%*%(beta[,1]),coeff[2,]%*%beta[,2],coeff[3,]%*%beta[,3])
+   }
+   uti_<-cbind(rep(0),uti)
+   pro <- matrix(rep(0),nrow = len, ncol = 4 ,byrow =T)
+   output <-matrix(rep(0),nrow = len, ncol = 1 ,byrow =T)
+   for(i in 1:len){
+     pro[i,] <- 1-pnorm(uti_[i,],0,1)
+     output[i,] <- which(pro[i,]==max(pro[i,]),arr.ind = T)
```

```
+   }
+   return(pro)
+ }
> head(actfun(Fishing))
     [,1]      [,2]      [,3]      [,4]
[1,]  0.5 0.8577139 0.8250389 0.8709236
[2,]  0.5 0.4426383 0.5557485 0.4301770
[3,]  0.5 0.8345075 0.5544083 0.5453090
[4,]  0.5 0.4551015 0.6639446 0.5847771
[5,]  0.5 0.7326191 0.5883524 0.5925758
[6,]  0.5 0.8881319 0.5539614 0.5733293
>
> # the 'actfun' can be applied in the neuralnet(...,act.fct = actfun, ...)
> # if 'return(output)' we choose the max likelihood of mode and return class
>
```

4 参考文献

[1]《Python 与深度学习》, 厦门大学软件学院，邱明.

[2] *Michael A. Nielsen. "Neural Network and Deep Learning",Chapter 4*, Determination Press,2015

[3] *"* 最速下降法梯度法 *steepest-descent"* .

[4] 弹性反向传播 *(RProp)* 和均方根反向传播 *(RMSProp)*