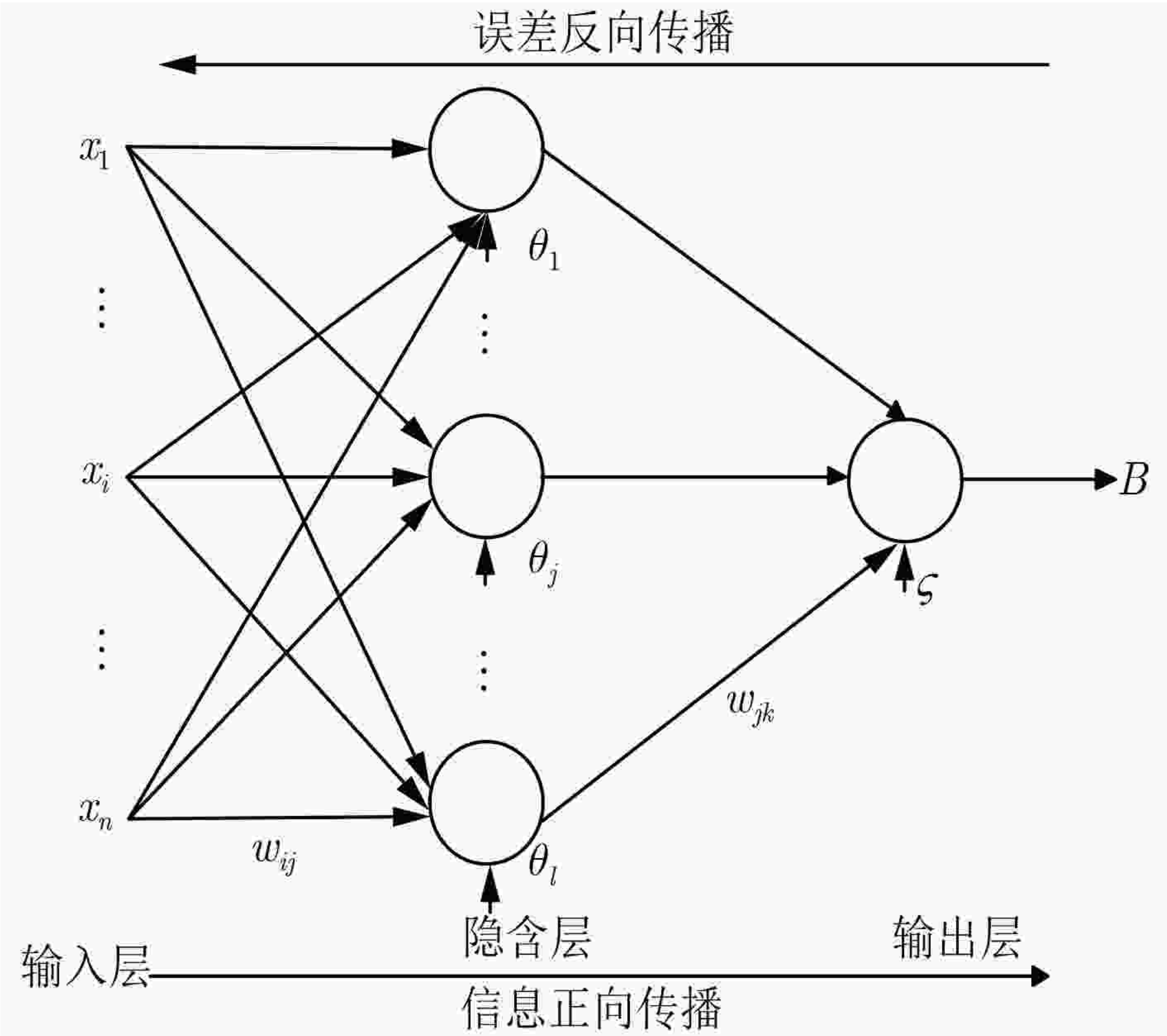
神经网络

摘要

神经网络可以说是目前人工智能算法中应用最为广泛的算法之一，相比Linear regression 、logistics regression、decision tree等机器学习算法等会复杂不少，变化也很多。本文从向量矩阵的角度去理解BP神经网络的数学原理，这个过程会显得更加清晰、简洁。我们已最经典的神经网络三层网络二分类模型为例，逐步推到数学公式，三层网络简单说明如下：

- 输入层，不做处理，只是输入数据
- 隐藏层，单元数为 n_h 我们就以RELU为激活函数，另外一个常用的是sigmoid，我们把激活函数封装成一个类，可以随时替换
- 输出层为一个单元，由于是二分类，我们就用sigmoid拟合对应类1的概率



符号定义

- X 表示输入单个样本向量，Y表示样本集的期望输出（有可能是标量，有可能是向量，统一认为是向量），本文示例由于是二分类，那么Y的每个元素要么是0要么是1，Y是m个元素的向量,但是在运算中为了通用将其转换为$m \times 1$的矩阵。
- m为样本数量,n为样本向量的维度。

- 激活函数用f表示，隐藏层激活函数f_h使用RELU函数:$f_h(x) = max(0, x)$，输出层f_o为sigmoid函数，输出为1类的概率$f_o(x) = \frac{1}{1+exp(-x)}$
- 隐藏层输出为Y_h,输出层的输出为Y_o
- 隐藏层单元的参数为w_h, N_h个隐藏层单元的参数矩阵为W_h,输出层单元的参数为w_o，N_o个输出层单元的参数矩阵为W_o。
- 虽然本文使用二分类即输出单元为1个，但是仍然使用矩阵Yo和W_o表示，从而使得算法是通用的，改成onehot+softmax方式的多分类，也是适用的。
- 从输入到输出的公式：$Y_o = f_o(f_h(X \cdot W_h) \cdot Wo)$
- 注：这里的线性变换$X \cdot W_h$并没有使用偏置，通过在X增加一个全1的列，达到同样的效果。

forward公式推导

$$
\begin{aligned}
x_{hj} &= x_i \cdot w_{hj} \\
y_{hj} &= f_h(x_{hj}) \\
x_o &= y_{hj} \cdot w_o \\
y_o &= f_o(x_o)
\end{aligned}
$$

$$
\begin{aligned}
X_h &= X \cdot W_h \\
Y_h &= f_h(X_h) \\
X_o &= Y_h \cdot W_o \\
Y_o &= f_o(X_o) \\
&= f_o(Y_h \cdot W_o) \\
&= f_o(f_h(X_h) \cdot W_o) \\
&= f_o(f_h(X \cdot W_h) \cdot Wo)
\end{aligned} \tag{1}
$$

其中（1）式即为信息正向传导的向量化公式相应的python代码实现：

```
#forward
Lh = np.dot(X, Wh)
Yh = funcActivation.cal(Lh) #隐藏层输出
Yh = np.insert(Yh, np.shape(Yh)[1], values=allOneCol, axis=1)#add all 1 col
Lo = np.dot(Yh, Wo)
Yo = funcOut.cal(Lo)
```

其中 funcActivation 为隐藏层激活函数的封装，这里使用的Relu，funcOut为输出层激活函数的封装，使用的是sigmoid。

```
class ReLU:
    def cal(self, z):
        return np.clip(z, 0, np.inf)
    def grad(self, x):
        return (x > 0).astype(int)
class Sigmoid:
    def cal(self, z):
        return 1 / (1 + np.exp(-z))
    def grad(self, x):
        z = self.cal(x)
        return z*(1-z)
```

损失函数

最常用的有Mean Square Error均方差损失函数，用于分类的Cross Entropy等。本文以经典的MSE为例。如下（2）式是MSE损失函数的向量化表示。

$$
\begin{aligned}
e &= \frac{1}{2m}\sum_{i=1}^{m}(y_{oi}-y_i)^2 \\
&= \frac{1}{2m}(Y_o-Y)^T\cdot(Y_o-Y) \qquad (2)
\end{aligned}
$$

误差反向传播

我们目的是最小化损失函数，通常使用梯度下降法，逐渐逼近最小极值点。需要逐渐求解参数主要是两个：

W_h 和 W_o ,下面分别计算对应损失函数的偏导数。

$$
\begin{aligned}
\mathrm{de} &= \frac{1}{m}tr((\mathrm{d}Y_o)^T\cdot(Y_o-Y)+(Y_o-Y)^T\cdot\mathrm{d}Y_o) \\
&= \frac{1}{2m}tr((\mathrm{d}Y_o)^T\cdot(Y_o-Y))+\frac{1}{2m}tr((Y_o-Y)^T\cdot\mathrm{d}Y_o) \\
&= \frac{1}{2m}tr(\mathrm{d}Y_o\cdot(Y_o-Y)^T)+\frac{1}{2m}tr((Y_o-Y)^T\cdot\mathrm{d}Y_o) \\
&= \frac{1}{m}tr((Y_o-Y)^T\cdot\mathrm{d}Y_o) \qquad (3) \\
&= \frac{1}{m}tr((Y_o-Y)^T\cdot(f_o^{'}(X_o)\odot\mathrm{d}X_o)) \qquad (4) \\
&= \frac{1}{m}tr(((Y_o-Y)\odot f_o^{'}(X_o))^T\cdot\mathrm{d}X_o) \qquad (5) \\
&= \frac{1}{m}tr(((Y_o-Y)\odot f_o^{'}(X_o))^T\cdot(Y_h\cdot\mathrm{d}W_o+\mathrm{d}Y_h\cdot W_o)) \qquad (6)
\end{aligned}
$$

其中（3）式利用迹$tr(A^T)=tr(A)$ 的性质，所以前后两项一样(4)式到(5)式利用了迹$tr(A^T\cdot(B\odot C))=tr((A\odot B)^T\cdot C)$的性质，其中$\odot$表示矩阵各个元素相乘，对应numpy的multiply。神经网络梯度下降求解参数的时候，是从输出层到隐藏层逆着计算的，所以称之为“反向传播”,因此首先求对W_o的偏导，此时Y_h相当于常数，故(6)式可以的$\mathrm{d}Y_h$ 忽略，继续推导：

$$
\begin{aligned}
\mathrm{de} &= \frac{1}{m}tr(((Y_o-Y)\odot f_o^{'}(X_o))^T\cdot(Y_h\cdot\mathrm{d}W_o)) \\
&= \frac{1}{m}tr((Y_h^T\cdot((Y_o-Y)\odot f_o^{'}(X_o)))^T\cdot\mathrm{d}W_o) \qquad (7) \\
\frac{\partial e}{\partial W_o} &= \frac{1}{m}Y_h^T\cdot((Y_o-Y)\odot f_o^{'}(X_o)) \qquad (8)
\end{aligned}
$$

这里(7)式到(8)式利用了标量对向量或矩阵微分$\mathrm{d}f=tr((\frac{\partial f}{\partial x})^T\mathrm{d}x)$ 的性质。对应python实现：

```
delta = np.multiply(self.lossFunc.grad(Yo, Y) , funcOut.grad(Lo))
dO  = np.dot(Xo.T, delta)
```

同理继续推导$\frac{\partial e}{\partial W_h}$ ：

$$
\begin{aligned}
\mathrm{de} &= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o))^T \cdot (\mathrm{d}Y_h \cdot W_o)) && (9)\\
&= \frac{1}{m} tr(W_o \cdot ((Y_o - Y) \odot f_o'(X_o))^T \cdot \mathrm{d}Y_h)\\
&= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o) \cdot W_o^T)^T \cdot \mathrm{d}Y_h)\\
&= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o) \cdot W_o^T)^T \cdot (f_h'(X_h) \odot \mathrm{d}X_h))\\
&= \frac{1}{m} tr((((Y_o - Y) \odot f_o'(X_o) \cdot W_o^T) \odot f_h'(X_h))^T \cdot \mathrm{d}(X_h)) && (10)\\
&= \frac{1}{m} tr((((Y_o - Y) \odot f_o'(X_o) \cdot W_o^T) \odot f_h'(X_h))^T \cdot \mathrm{d}(X \cdot W_h))\\
&= \frac{1}{m} tr((X^T \cdot (((Y_o - Y) \odot f_o'(X_o) \cdot W_o^T) \odot f_h'(X_h)))^T \cdot \mathrm{d}W_h)\\
\frac{\partial e}{\partial W_h} &= \frac{1}{m} X^T \cdot (((Y_o - Y) \odot f_o'(X_o) \cdot W_o^T) \odot f_h'(X_h)) && (11)
\end{aligned}
$$

对应的python的实现：

```
delta = np.multiply(self.lossFunc.grad(Yo, Y) , funcOut.grad(Lo))
dH = np.dot(X.T, np.multiply(np.dot(delta, Wo.T[:,:-1]) ,
funcActivation.grad(Lh)))
```

显式使用偏置b的数学推导

以上是使用通过X 增加全1的列的方式，省略偏置b,下面再推导一下使用偏置的方式

$$
\begin{aligned}
x_{hj} &= x_i \cdot w_{hj} + b_{hj}\\
y_{hj} &= f_h(x_{hj})\\
x_o &= y_{hj} \cdot w_o + b_{oj}\\
y_o &= f_o(x_o)
\end{aligned}
$$

$$
\begin{aligned}
X_h &= X \cdot W_h + B_h\\
Y_h &= f_h(X_h)\\
X_o &= Y_h \cdot W_o + B_o\\
Y_o &= f_o(X_o)\\
&= f_o(Y_h \cdot W_o + B_o)\\
&= f_o(f_h(X_h + B_h) \cdot W_o + B_o)\\
&= f_o(f_h(X \cdot W_h + B_h) \cdot Wo + B_o) && (12)
\end{aligned}
$$

$$
\begin{aligned}
de &= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o))^T \cdot \mathrm{d}X_o) && (5)\\
&= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o))^T \cdot \mathrm{d}(Y_h \cdot W_o + B_o))\\
&= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o))^T \cdot (\mathrm{d}(Y_h \cdot W_o) + \mathrm{d}B_o))
\end{aligned}
$$

其中 $\frac{\partial e}{\partial W_o}$ 跟(8)式的结果相同，继续推导 $\frac{\partial e}{\partial B_o}$：

$$
\begin{aligned}
de &= \frac{1}{m} tr(((Y_o - Y) \odot f_o'(X_o))^T \cdot \mathrm{d}B_o)\\
\frac{\partial e}{\partial W_o} &= \frac{1}{m} Y_h^T \cdot ((Y_o - Y) \odot f_o'(X_o)) && (8)\\
\frac{\partial e}{\partial B_o} &= \frac{1}{m} (Y_o - Y) \odot f_o'(X_o)
\end{aligned}
$$

下面推导$\frac{\partial e}{\partial W_o}$ 和 $\frac{\partial e}{\partial B_h}$：

$$
\begin{aligned}
de &= \frac{1}{m}tr((((Y_o - Y)\odot f_o'(X_o)\cdot W_o^T)\odot f_h'(X_h))^T\cdot \mathrm{d}(X_h)) \\
&= \frac{1}{m}tr((((Y_o - Y)\odot f_o'(X_o)\cdot W_o^T)\odot f_h'(X_h))^T\cdot \mathrm{d}(X\cdot W_h + B_h))
\end{aligned} \tag{10}
$$

显然 $\frac{\partial e}{\partial W_h}$ 跟(11)式的结果相同，继续推导 $\frac{\partial e}{\partial B_o}$：

$$
\begin{aligned}
de &= \frac{1}{m}tr((((Y_o - Y)\odot f_o'(X_o)\cdot W_o^T)\odot f_h'(X_h))^T\cdot \mathrm{d}(X_h)) \qquad (10)\\
&= \frac{1}{m}tr((((Y_o - Y)\odot f_o'(X_o)\cdot W_o^T)\odot f_h'(X_h))^T\cdot \mathrm{d}B_h) \\
\frac{\partial e}{\partial W_h} &= \frac{1}{m}X^T\cdot(((Y_o - Y)\odot f_o'(X_o)\cdot W_o^T)\odot f_h'(X_h)) \qquad (11)\\
\frac{\partial e}{\partial B_h} &= \frac{1}{m}((Y_o - Y)\odot f_o'(X_o)\cdot W_o^T)\odot f_h'(X_h)
\end{aligned}
$$

实验效果

使用python库sklearn自带的数据集breast_cancer，共569行数据，30个维度。完整python实现代码:

```
import numpy as np
from sklearn import datasets
from sklearn.datasets import load_breast_cancer
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

dataset = None
DT_FLAG = 1 #1测试预测，2测试回归拟合

if DT_FLAG:
    dataset = load_breast_cancer()
else:
    dataset = datasets.load_boston()

class ReLU:
    def __init__(self):
        return
    def cal(self, z):
        return np.clip(z, 0, np.inf)
    def grad(self, x):
        return (x > 0).astype(int)
class Sigmoid:
    def __init__(self):
        return
    def cal(self, z):
        return 1 / (1 + np.exp(-z))
    def grad(self, x):
        z = self.cal(x)
        return z*(1-z)

class LossMSE:
    def __init__(self):
        return
    def loss(self, y_pred, y):
        return np.sum(np.multiply((y_pred - y) , (y_pred - y))) / 2
/int(y.shape[0])
    def grad(self, y_pred, y):
```

```
        return (y_pred - y)/y.shape[0]
class LossCrossEntropy:
    def loss(self, y_pred, y):
        eps = np.finfo(float).eps
        cross_entropy = -np.sum(y * np.log(y_pred + eps))
        return cross_entropy
    def grad(self, y_pred, y):
        grad = y_pred - y
        return grad

class LinearOut:#如果用于回归，只是线性输出，不做任何转换，是的可以有统一的结构
    def cal(self, z):
        return z
    def grad(self, x):
        self.gradCache = np.ones(np.shape(x))
        return self.gradCache

class NN:
    def __init__(self):
        self.Wh = None #隐藏层参数w
        self.Wo = None #输出层参数w
        self.numHideUnit = 5
        self.funcActivation = ReLU() #隐藏层激活函数
        if DT_FLAG:
            self.lossFunc = LossMSE()
            self.funcOut = Sigmoid()
        else:
            self.lossFunc = LossMSE()
            self.funcOut = LinearOut()
        return
    def fit(self, X, Y):
        scalerX = preprocessing.StandardScaler().fit(X)#StandardScaler
        Y = Y.reshape(-1, 1)
        X = scalerX.transform(X)
        if not DT_FLAG:
            scalerY = preprocessing.StandardScaler().fit(Y)#MinMaxScaler
            Y = scalerY.transform(Y)

        m, n = np.shape(X)

        funcActivation = self.funcActivation
        funcOut = self.funcOut
        # 10 1000 0.01 for breat cancer
        numLayerHide = 5
        maxIterTimes = 2000
        eta  = 0.01
        if DT_FLAG:
            eta  = 10
        else:
            #maxIterTimes = 100000
            eta  = 0.001#eta  = 0.00001
            pass

        outputNodeNum = 1
        Wh = np.random.rand(n, numLayerHide) * 0.01
        Bh = np.random.rand(numLayerHide) * 0.01
        Wo = np.random.rand(numLayerHide, outputNodeNum) * 0.01 #why +1? for
reserver b
```

```python
        Bo = np.random.rand(outputNodeNum) * 0.01

        errLog = []
        dO_old = 0
        dH_old = 0
        for i in range(maxIterTimes):
            #forward
            Lh = np.dot(X, Wh) + Bh
            Yh = funcActivation.cal(Lh) #隐藏层输出

            Lo = np.dot(Yh, Wo) + Bo
            Yo = funcOut.cal(Lo)

            loss = self.lossFunc.loss(Yo, Y)
            errLog.append(loss)
            if loss < 0.001:
                print('fit finish', i)
                break

            delta = np.multiply(self.lossFunc.grad(Yo, Y) , funcOut.grad(Lo))

            dBo  = delta
            dWo  = np.dot(Yh.T, dBo)

            dBh = np.multiply(np.dot(delta, Wo.T) , funcActivation.grad(Lh))
            dWh = np.dot(X.T, dBh)

            Wo = Wo - eta * dWo
            Wh = Wh - eta * dWh
            Bo = Bo - eta * dBo
            Bh = Bh - eta * dBh

        score = 0
        if DT_FLAG == 0:
            score = r2_score(Yo, Y)
        else:
            score = r2_score((Yo > 0.5).astype(int), Y)
        print('score', score)
        return
    if __name__ == '__main__':
        nn = NN()
        print('data', dataset.data.shape)
        nn.fit(dataset.data, dataset.target)
```

总结

- 多层网络