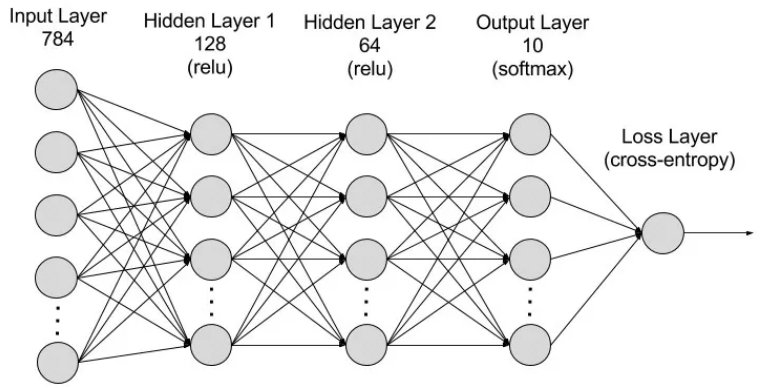
第三课 基础复习与Wide&Deep模型

Max 逆矩统计 2019-08-24

基础复习

我们初学神经网络见到的神经网络结构通常是下面这样的，这是典型的前馈神经网络：

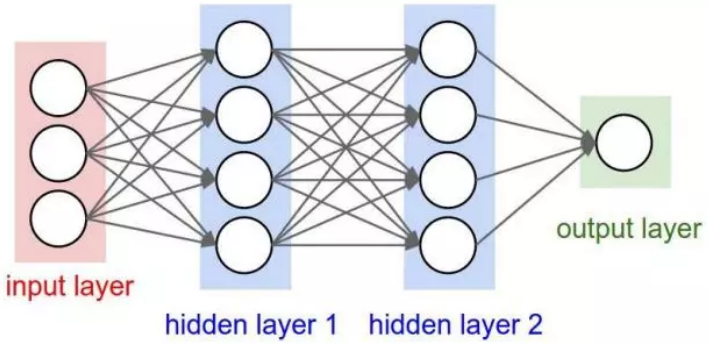


整个框架由输入层（一层）、隐藏层（多层）和输出层（一层）组成。由于隐藏层和输出层每一个结点都与上一层的所有结点相连，所以他们都是全连接层。

可能有很多萌新不太懂神经网络，如果在手机上看理论和公式，只会感到头皮发麻，所以我就简要地介绍一下：

输入层输入数据，这一层的每个结点都对应一个特征；隐藏层的每个结点内包含两个运算：$y=Wx+b$以及$z=\sigma(y)$，$\sigma(.)$称为激活函数，激活函数有很多种可供选择：sigmoid、relu、tanh等。在每个结点的计算完毕后，z将被当做新的x（在文献中，隐藏层传入、传出的x常被记为a）传入下一层的各个结点，以此类推，这个过程成为**向前传播或正向传播。**

我们想象一下，将波士顿房价数据的各个特征从输入层进入，它们经过多层的各个结点计算，最终汇入到最后一层的单个输出层结点上，可以输出一个合理的数值。



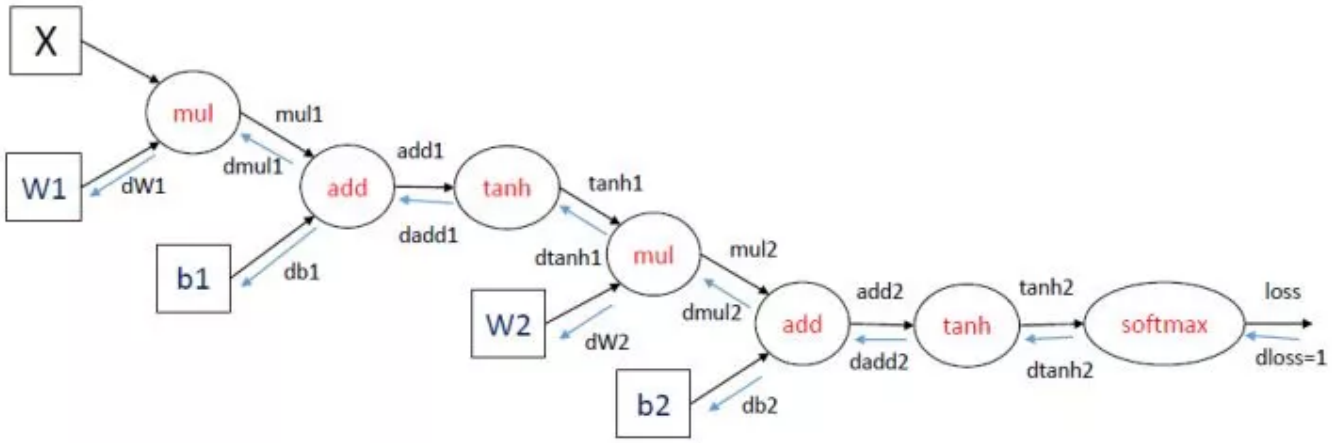
注：多分类问题的输出层Output Layer的结点不止一个，最后一层为softmax，例如第一张图所示。

为了得到理想的拟合值输出，并通过模型作高精度的预测，是们我们构建模型的初衷。对于深度学习模型，我们需要调整的无非是众多结点中运算的参数：W（权重）和b（偏置），提醒一下萌新们，每个结点的W、b都是不一样的😄，激活函数σ(.)是确定的，所以模型理想的输出完全仰仗参数W和b的合理性。参数W、b在刚开始都是随机初始化的，所以第一次正向传播过程计算的出来的值都是有很大偏差的，参数需要在模型训练中不断地调整，以达到最适合的值，深度学习之所以是学习，就是需要模型根据输入的数据自我调整参数（而不是依靠人工和特定的冗长的规则），最后达到理想状态，这就需要反向计算。

因为我们输出的y和真实的Y,是有差别的，在此我们定义损失函数，就是类似于最小二乘法Q函数的东西，它衡量了模型预测值和真实值的差异。我们当然是希望损失函数越小越好，这便是个优化问题了。

接下来会用反向传播算法，简称BP算法（back propagation）来使损失函数达到最小，同时，这个过程会让w和b自我调整。

下图所示是一条完整的路径。箭头指向右边的过程代表梯度的正向传播，mul代表乘法，add代表加法，这两步合为一步即是：y=Wx+b；tanh是激活函数的一种，这一步即为z=σ(y)，这两步运算在一个结点内进行，上面提到过。所以下面这张图演示的其实是X经过了两个结点的运算，输出层为softmax。



向前传播算法（伪代码）：

```
mul1 = X * W1
add1 = mul1 + b1
tanh1 = tanh(add1)

mul2 = tanh1 * W2
add2 = mul2 + b2
tanh2 = tanh(add2)

loss = softmax_loss(tanh2)
```

箭头指向左边的过程代表梯度的反向传播，dloss代表损失函数对参数的求导，一般情况下省略分子，便于代码书写。

反向传播算法（伪代码）：

```
dloss = 1

dtanh2 = softmax_loss_diff(tanh2) * dloss
dadd2 = tanh_diff(add2) * dtanh2
db2 = 1 * dadd2
dmul2 = 1 * dadd2
dW2 = tanh1 * dmul2

dtanh1 = W2 * dmul2
dadd1 = tanh_diff(add1) * dtanh1
```

```
db1 = 1 * dadd1
dmul1 = 1 * dadd1
dW1 = X * dmul1
```

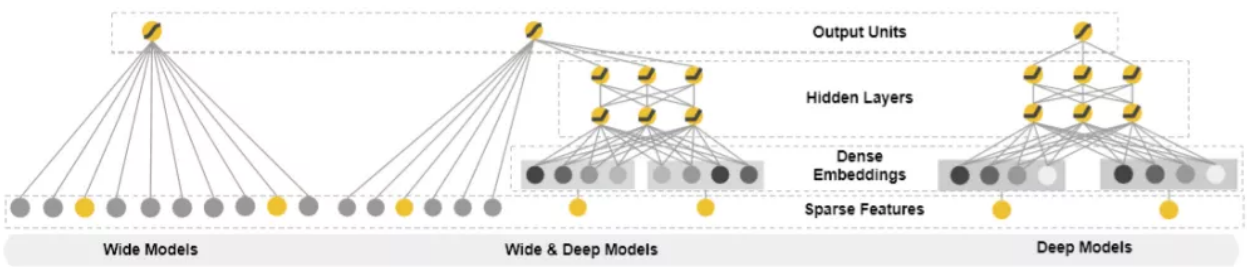
根据以上的算法，我们可以试着逐步将最前面的dW1展开，直到dloss出现。展开后可以发现：dW1其实是包含多个梯度的乘积。

```
dW1 = X * dmul1
    = X * dmul1
    = X * 1 * dadd1
    = X * 1 * tanh_diff(add1) * W2 * dmul2
    = X * 1 * tanh_diff(add1) * W2 * 1 * dadd2
    = X * 1 * tanh_diff(add1) * W2 * 1 * tanh_diff(add2) * dtanh2
    = X * 1 * tanh_diff(add1) * W2 * 1 * tanh_diff(add2) * softmax_loss_di
```

从这个反向传播过程中我们也能看到，**每一步的梯度都继承了前一步的梯度**，话说到这里，你应该对所谓的**梯度反向传播**有了一个大致的认识了吧。

Wide & Deep模型

掌握最初步的神经网络之后，我们再进一步认识一下Wide & Deep模型，它的样子比较奇怪。下图中间的模型是Wide & Deep模型。



看到这里，读者心中可能会产生三个问题：

什么是Wide & Deep模型?

重要参考论文：Google Inc.* **Wide & Deep Learning for Recommender Systems**

Wide&Deep模型由Wide模型和Deep模型这两个模型组成。

Wide模型如上图中的左侧的图所示，实际上，Wide模型就是一个广义线性模型：

$$y = W^T + b$$

其中，特征x=[x_1,x_2,⋯,x_d]是一个d维的向量，W=[W_1,W_2,⋯,W_d]为模型的参数，在y的基础上应用Sigmoid激活函数（二值化0/1）后，即：z=σ(y)，作为最终的输出。可见，Wide model主要采用逻辑回归（LR），LR的特征一般是二值且稀疏的，这里我们采用one-hot编码。

什么是one-hot编码：https://blog.csdn.net/google19890102/article/details/44039761

Deep模型如上图中的右侧的图所示，实际上，Deep模型是一个前馈神经网络。深度神经网络模型通常需要的输入是连续的稠密特征，对于稀疏，高维的类别特征，通常首先将其转换为低维的向量，这个过程也成为embedding（以后会详细介绍）。在训练的时候，首先随机初始化embedding向量，并在模型的训练过程中逐渐修改该向量的值，即将向量作为参数参与模型的训练。

隐藏层的计算方法为：

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

其中，l为层数，a为输入（输出），σ(.)为激活函数，例如relu、tanh和sigmoid等。最后再将两个模型进行连接起来，进行联合训练。联合训练是指同时训练Wide模型和Deep模型，并将两个模型的结果的加和并经过激活函数处理作为最终的预测结果：

$$P(Y=1|x) = \sigma(w_{wide}^T[x, \phi(x)] + w_{deep}^T a^{(l)} + b)$$

其中σ(.)为sigmoid激活函数，ϕ(x)是被挑选的特征的组合(cross-product transformation)，只有当变量同时满足时才会取值为1，否则为0。

$$\phi_k(x) = \prod_{i=1}^{d} x_i^{c_{ki}}, \ c_{ki} \in \{0,1\}$$

模型输入的特征的选取，见CSDN博客：https://blog.csdn.net/yujianmin1990/article/details/78989099

为什么会有这样的模型，其特点和应用场景是什么？

Wide & Deep模型是TensorFlow在2016年发布的一种用于分类和回归的模型，并应用到了Google Play的应用推荐中，没错，谷歌商店的商品推送就是基于Wide & Deep模型为用户推荐可能喜欢的app。

应用在谷歌推荐系统：https://www.jianshu.com/p/dbaf2d9d8c94

推荐系统重点解决memorization（记忆）和generalization（归纳或拓展）这两个问题。例如你喜欢社交软件，它不仅会给你推社交软件（记忆功能）还会给你推送与社交相关但是又属于不同领域的软件，例如唱歌录制的之类的音乐软件（归纳拓展功能），只是具有部分社交功能。

- memorization主要是基于历史数据来学习频繁共同出现的item，也就是根据历史行为数据，推荐相关的item（topical），由Wide模型作为主导。
- generalization主要是基于相关性之间的传递， 探索历史上没有出现的新的特征的组合，也就是实现了推荐的多样性问题（diversity），由Deep模型作为主导。

关于记忆和归纳拓展的详细论述，请移步知乎https://zhuanlan.zhihu.com/p/47293765

Wide & Deep模型如何实现?

下面用加利福利房价数据集简单地实现多输入的Wide & Deep模型。

```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import sklearn
import pandas as pd
import os
import sys
import time
import tensorflow as tf

from tensorflow import keras

print(sys.version_info)
for module in mpl, np, pd, sklearn, tf, keras:
    print(module.__name__, module.__version__)
## --------------------运行结果----------------------
sys.version_info(major=3, minor=6, micro=2, releaselevel='final', serial=0)
```

```
matplotlib 2.0.2
numpy 1.16.2
pandas 0.20.3
sklearn 0.20.0
tensorflow 2.0.0-alpha0
tensorflow.python.keras.api._v2.keras 2.2.4-tf
```

一定要tensorflow 2.0.0-alpha0版本。

```
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()
print(housing.DESCR)  #打印数据集描述
print(housing.data.shape)
print(housing.target.shape)
# ------------------------运行结果------------------------
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None
```

```
This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.
Census Bureau publishes sample data (a block group typically has a populatior
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297

(20640, 8)
(20640,)
```

```
from sklearn.model_selection import train_test_split

x_train_all, x_test, y_train_all, y_test = train_test_split(
    housing.data, housing.target, random_state=7)
x_train, x_valid, y_train, y_valid = train_test_split(
    x_train_all, y_train_all, random_state=11)
print(x_train.shape, y_train.shape)
print(x_valid.shape, y_valid.shape)
print(x_test.shape, y_test.shape)
## ----------------------运行结果----------------------
(11610, 8) (11610,)
(3870, 8) (3870,)
(5160, 8) (5160,)
```

```
from sklearn.preprocessing import StandardScaler
#数据集标准化
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_valid_scaled = scaler.transform(x_valid)
x_test_scaled = scaler.transform(x_test)
```

```
# 构建多输入wide&deep模型
input_wide = keras.layers.Input(shape=[5])  #wide模型输入5列特征
input_deep = keras.layers.Input(shape=[6])  #deep模型输入6列特征
#函数式写法
hidden1 = keras.layers.Dense(30, activation='relu')(input_deep)
hidden2 = keras.layers.Dense(30, activation='relu')(hidden1)
concat = keras.layers.concatenate([input_wide, hidden2])#连接两个模型
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input_wide, input_deep], outputs=[outp
model.compile(loss="mean_squared_error", optimizer="sgd")
callbacks = [keras.callbacks.EarlyStopping(patience=5, min_delta=1e-2)]
model.summary()
## --------------------运行结果----------------------
Model: "model"
__________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connecte
==========================================================================
input_2 (InputLayer)            [(None, 6)]          0
__________________________________________________________________________
dense (Dense)                   (None, 30)           210         input_2[
__________________________________________________________________________
input_1 (InputLayer)            [(None, 5)]          0
__________________________________________________________________________
dense_1 (Dense)                 (None, 30)           930         dense[0]
__________________________________________________________________________
concatenate (Concatenate)       (None, 35)           0           input_1[
```

```
                                                                 dense_1[
__________________________________________________________________________
dense_2 (Dense)                 (None, 1)            36          concaten
==========================================================================
Total params: 1,176
Trainable params: 1,176
Non-trainable params: 0
__________________________________________________________________________
```

Wide模型输入取数据集前5个特征，Deep模型输入取数据集后6个特征。有人可能会疑惑，数据集总共才8个特征，为什么还能重叠？是的，这个没关系的，不会影响模型的运行。

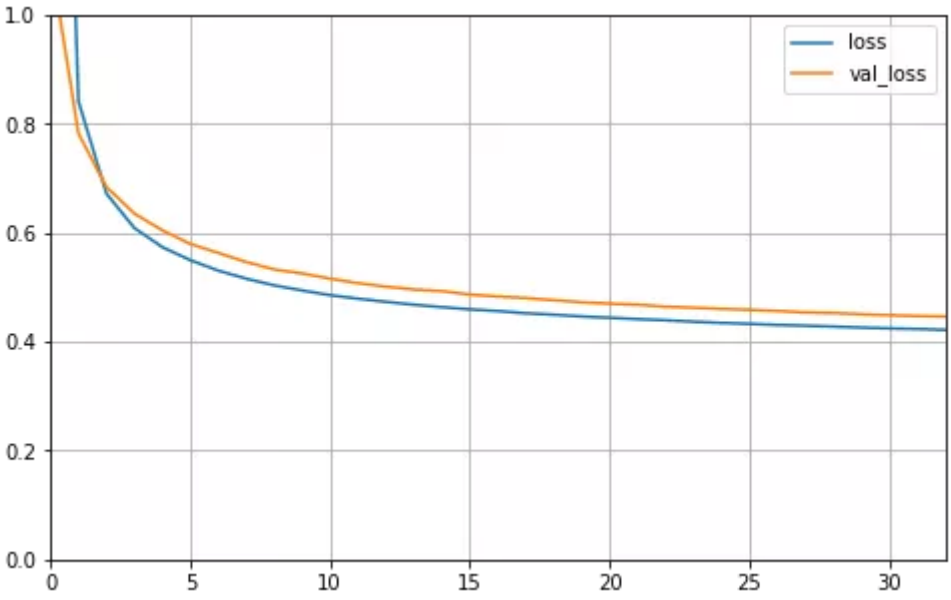
```
x_train_scaled_wide = x_train_scaled[:, :5]
x_train_scaled_deep = x_train_scaled[:, 2:]
x_valid_scaled_wide = x_valid_scaled[:, :5]
x_valid_scaled_deep = x_valid_scaled[:, 2:]
x_test_scaled_wide = x_test_scaled[:, :5]
x_test_scaled_deep = x_test_scaled[:, 2:]

history = model.fit([x_train_scaled_wide, x_train_scaled_deep],
                    y_train,
                    validation_data = (
                        [x_valid_scaled_wide, x_valid_scaled_deep],
                        y_valid),
                    epochs = 100,
                    callbacks = callbacks)
```

训练过程就不打印了，各位在自己电脑上试运行。

```
def plot_learning_curves(history):
    pd.DataFrame(history.history).plot(figsize=(8, 5))
    plt.grid(True)
    plt.gca().set_ylim(0, 1)
    plt.show()
```

```
plot_learning_curves(history)
```



可以看到，损失函数在训练过程中不断下降，降到0.4，在测试集上测试一下，结果也是0.4左右。当然，这个结果并不是最优的

```
model.evaluate([x_test_scaled_wide, x_test_scaled_deep], y_test)
#------------------------运行结果------------------------
0.43763476656388867
```

因为只是demo，目的仅仅是想让大家清楚如何实现Wide&Deep模型，所以并没有调参优化（用CPU调参太费时间了TAT，求众筹GPU...）。