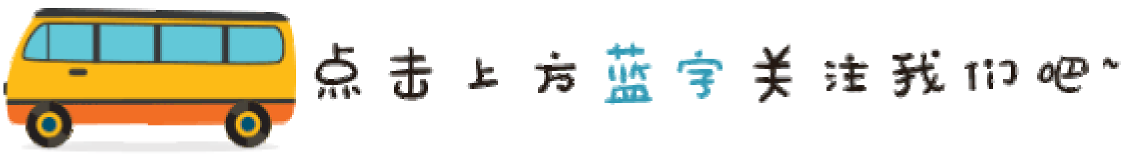
NFM模型理论与实践

王多鱼 python科技园 4月21日



一、理论部分

今天介绍一下NFM模型，NFM模型是FM模型的神经网络化尝试：即将FM的二阶交叉项作为Deep模型的输入，以此加强模型的表达能力。

1. NFM数学表达式

经典的FM模型的数学表达式如公式（1）所示：

$$\hat{y}_{FM}(x)=w_0+\sum_{i=1}^{n} w_i x_i+\sum_{i=1}^{n} \sum_{j=i+1}^{n}<v_i, v_j> x_i x_j$$

公式（1）

在数学形式上，NFM模型的主要思路是用一个表达能力更强的函数替代原FM中二阶隐向量内积部分。NFM的表达式如公式（2）所示：

$$\begin{aligned}
\hat{y}_{NFM}(x)&=w_0+\sum_{i=1}^{n} w_i x_i+deep\left(f(x)\right) \\
&=w_0+\sum_{i=1}^{n} w_i x_i+deep\left(f_{BI}(V_x)\right) \\
&=w_0+\sum_{i=1}^{n} w_i x_i+deep\left(\sum_{i=1}^{n} \sum_{j=i+1}^{n}(x_i v_i) \odot(x_j v_j)\right) \\
&=w_0+\sum_{i=1}^{n} w_i x_i+deep\left(\frac{1}{2}\left[\left(\sum_{x=1}^{n} v_i x_i\right)^2-\sum_{i=1}^{n}(x_i v_i)^2\right]\right)
\end{aligned}$$

公式（2）

2. NFM深度网络部分结构图

NFM模型的深度网络部分结构图如图（1）所示。

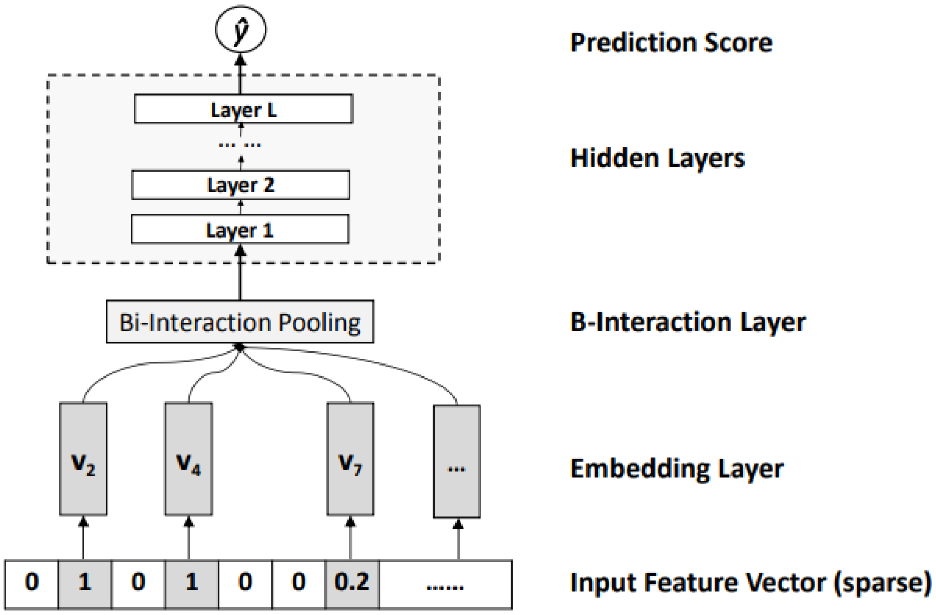


Figure 2: Neural Factorization Machines model (the first-order linear regression part is not shown for clarity).

图（1） NFM模型的深度网络部分结构图

（2.1）输入层、Embedding层、BI层

NFM网络架构的特点非常明显，就是在Embedding层和MLP隐层之间加入特征交叉池化层（Bi-Interaction Pooling Layer）。假设是所有特征域的Embedding集合，特征交叉池化层的具体操作如公式（3）所示。

$$f_{BI}(V_x)=\sum_{i=1}^{n}\sum_{j=i+1}^{n}(x_i v_i)\odot(x_j v_j)$$

（公式 3）

其中，$\odot$ 代表两个向量的元素积操作，即两个长度相同的向量对应维相乘得到元素积向量，示例如下：$[1,2,3]\odot[4,2,1]=[1*4,2*2,3*1]=[4,4,3]$

在进行两两 **K维** Embedding向量的元素积操作后，对交叉特征向量求和，得到池化层的 **K维** 输出向量。再把该向量输入到上层的MLP全连接神经网络中，进行下一步的交叉。

（2.2）MLP全连接层

MLP隐层部分就是常规的nn全连接，结构如下所示：

$$\mathbf{z}_1 = \sigma_1(\mathbf{W}_1 f_{BI}(\mathcal{V}_x) + \mathbf{b}_1),$$
$$\mathbf{z}_2 = \sigma_2(\mathbf{W}_2 \mathbf{z}_1 + \mathbf{b}_2),$$
$$\cdots\cdots$$
$$\mathbf{z}_L = \sigma_L(\mathbf{W}_L \mathbf{z}_{L-1} + \mathbf{b}_L),$$

其中：W_l, b_l分别表示参数矩阵与偏置向量； σ_l 表示激活函数，可以取 sigmoid, tanh, relu等。

（2.3）预测层

最后一层隐藏层加上一个线性变换，作为结果输出，即： $f(x) = h^T z_L$。向量 h 是输出层的权重值。

3. NFM模型的学习过程

由以上分析可知，公式（2）可以写为公式（4）

$$\hat{y}_{NFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + h^T \sigma_L \left(W_L(\ldots\sigma_1(W_1 f_{BI}(V_X) + b_1)\ldots) + b_L \right)$$

公式（4）

采用SGD方式链式法则求解，此处只给出Bi-interaction层的求导，其他的都是常规的nn的求导。

$$\frac{d f_{BI}(V_x)}{d_{v_i}} = (\sum_{j=1}^{n} x_j \vec{v}_j) x_i - x_i^2 \vec{v}_i = \sum_{j=1, j\neq i}^{n} x_i x_j \vec{v}_j$$

公式（5）

实践中一般采用mini-batch的方式，采用Adagrad做优化。

4. 防止过拟合措施

防止模型的过拟合风险，采用以下两种手段提高模型的泛化能力。

（4.1）dropout

将dropout技术用在Bi-interaction层，即：获取$f_{BI}(V_x)$ (**K维**向量)之后，随机drop掉ρ百分比的**latent factors**，这相当于是对于FM的一种新的正则约束形式。同时在隐层也可以应用dropout技术。

(4.2) Batch Normalization

NN中的常规操作：

$$BN(\vec{x}_i) = \gamma \odot (\frac{\vec{x}_i - \mu_B}{\sigma_B}) + \beta$$

其中：μ_B 是该mini-batch的均值，σ_B 是mini-batch的方差，γ 和 β 是可以训练的参数，用来控制normalization的缩放和平移。对于NFM，对Bi-interaction层的输出也进行BN操作。

二、实践环节

1. 首先来看看特征交叉池化层（Bi-Interaction Pooling Layer）的实现

```
# Bi-Interaction Pooling Layer

from  tensorflow.python.keras import  backend as  K
import  tensorflow as  tf

a = [
                [[3., 0., 1., 0.],
                  [1., 2., 3., 4.],
                  [4., 5., 6., 1.]]
        ]

concated_embeds_value = tf.convert_to_tensor(a)
square_of_sum = tf.square(tf.reduce_sum(concated_embeds_value, axis=1, keepdims=True))
sum_of_square = tf.reduce_sum(concated_embeds_value * concated_embeds_value, axis=1, keep

cross_term = 0.5  * (square_of_sum - sum_of_square)
```

```
sess = tf.InteractiveSession()
square_of_sum = sess.run(square_of_sum)
print(square_of_sum)

# [[[ 64.  49.  100.  25.]]]
```

```
sess = tf.InteractiveSession()
sum_of_square = sess.run(sum_of_square)
print(sum_of_square)

# [[[26.  29.  46.  17.]]]


sess = tf.InteractiveSession()
cross_term = sess.run(cross_term)
print(cross_term)

# [[[19.  10.  27.  4.]]]
```

2. 从实际例子应用一下NFM模型

（2.1）准备数据

titanic数据集 的目标是根据乘客信息预测他们在Titanic号撞击冰山沉没后能否生存。结构化数据一般会使用Pandas中的DataFrame进行预处理。

```
import  numpy as  np
import  pandas as  pd
from  sklearn.model_selection import  train_test_split

df_data = pd.read_csv('data/train.csv')
```

titanic数据集下载地址： https://www.kaggle.com/c/titanic/data

字段说明：

- Survived: 0代表死亡，1代表存活 【y标签】
- Pclass: 乘客所持票类，有三种值(1,2,3) 【类别变量】
- ~~Name: 乘客姓名 【舍去】~~
- Sex: 乘客性别 【类别变量】
- Age: 乘客年龄(有缺失) 【数值特征】
- SibSp: 乘客兄弟姐妹/配偶的个数(整数值) 【数值特征】
- Parch: 乘客父母/孩子的个数(整数值) 【数值特征】
- ~~Ticket: 票号(字符串) 【舍去】~~
- Fare: 乘客所持票的价格(浮点数，0-500不等) 【数值特征】
- Cabin: 乘客所在船舱(有缺失) 【类别变量】
- Embarked: 乘客登船港口:S、C、Q(有缺失) 【类别变量】


```
# 类别变量重新编码
# 数值变量，用0填充缺失值

sparse_feature_list = ["Pclass", "Sex", "Cabin", "Embarked"]
dense_feature_list = ["Age", "SibSp", "Parch", "Fare"]

sparse_feature_reindex_dict = {}
for  i in  sparse_feature_list:
        cur_sparse_feature_list = df_data[i].unique()

        sparse_feature_reindex_dict[i] = dict(zip(cur_sparse_feature_list, \
                range(1,  len(cur_sparse_feature_list)+1)
                                                          )
                                                  )

        df_data[i] = df_data[i].map(sparse_feature_reindex_dict[i])

for  j in  dense_feature_list:
        df_data[j] = df_data[j].fillna(0)


# 分割数据集

data = df_data[sparse_feature_list + dense_feature_list]
label = df_data["Survived"].values

xtrain, xtest, ytrain, ytest = train_test_split(data, label, test_size=0.2, random_state=
```

```
xtrain_data = {"Pclass": np.array(xtrain["Pclass"]), \
                           "Sex": np.array(xtrain["Sex"]), \
                           "Cabin": np.array(xtrain["Cabin"]), \
                           "Embarked": np.array(xtrain["Embarked"]), \
                           "Age": np.array(xtrain["Age"]), \
                           "SibSp": np.array(xtrain["SibSp"]), \
                           "Parch": np.array(xtrain["Parch"]), \
                           "Fare": np.array(xtrain["Fare"])}

xtest_data = {"Pclass": np.array(xtest["Pclass"]), \
                           "Sex": np.array(xtest["Sex"]), \
                           "Cabin": np.array(xtest["Cabin"]), \
                           "Embarked": np.array(xtest["Embarked"]), \
                           "Age": np.array(xtest["Age"]), \
                           "SibSp": np.array(xtest["SibSp"]), \
                           "Parch": np.array(xtest["Parch"]), \
                           "Fare": np.array(xtest["Fare"])}
```

（2.2）构建模型

（2.2.1）加载python模块

```
import tensorflow as tf
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.layers import Input, Embedding, \
        Dot, Flatten, Concatenate, Dense

from tensorflow.keras.models import Model
from tensorflow.python.keras.layers import Layer
from tensorflow.python.keras.initializers import Zeros, glorot_normal
from tensorflow.python.keras.optimizers import Adam
from tensorflow.python.keras.regularizers import l2

from keras.utils import plot_model
```

（2.2.2）定义类别变量的输入层、Embedding层

```
def input_embedding_layer(
        shape=1, \
        name=None, \
        vocabulary_size=1, \
        embedding_dim=1):
```

```
        input_layer = Input(shape=[shape, ], name=name)
        embedding_layer = Embedding(vocabulary_size, embedding_dim)(input_layer)

        return  input_layer, embedding_layer
```

(2.2.3）定义 线性层、Bi-Interaction层、DNN层、预测层

```
class  Linear(Layer):

        def  __init__(self, l2_reg=0.0, mode=2, use_bias=True, **kwargs):

                self.l2_reg = l2_reg
                if  mode not  in  [0, 1, 2]:
                        raise  ValueError("mode must be 0, 1 or 2")
                self.mode = mode
                self.use_bias = use_bias
                super(Linear, self).__init__(**kwargs)

        def  build(self, input_shape):
                if  self.use_bias:
                        self.bias = self.add_weight(name='linear_bias',
                                                                                                            shape=(1,),
                                                                                                            initializer=
                                                                                                            trainable=T


                if  self.mode == 1:
                        self.kernel = self.add_weight(
                                'linear_kernel',
                                shape=[int(input_shape[-1]), 1],
                                initializer=tf.keras.initializers.glorot_normal(),
                                regularizer=tf.keras.regularizers.l2(self.l2_reg),
                                trainable=True)

                elif  self.mode == 2  :
                        self.kernel = self.add_weight(
                                'linear_kernel',
                                shape=[int(input_shape[1][-1]), 1],
                                initializer=tf.keras.initializers.glorot_normal(),
                                regularizer=tf.keras.regularizers.l2(self.l2_reg),
                                trainable=True)

                super(Linear, self).build(input_shape)

        def  call(self, inputs, **kwargs):
```

```
        if  self.mode == 0:
                sparse_input = inputs
                linear_logit = reduce_sum(sparse_input, axis=-1, keep_dims=True)
        elif  self.mode == 1:
                dense_input = inputs
                fc = tf.tensordot(dense_input, self.kernel, axes=(-1, 0))
                linear_logit = fc
        else:
                sparse_input, dense_input = inputs
                fc = tf.tensordot(dense_input, self.kernel, axes=(-1, 0))
                linear_logit = tf.reduce_sum(sparse_input, axis=-1, keep_dims=Fal
        if  self.use_bias:
                linear_logit += self.bias

        return  linear_logit

    def  compute_output_shape(self, input_shape):
        return  (None, 1)

    def  compute_mask(self, inputs, mask):
        return  None

    def  get_config(self, ):
        config = {'mode': self.mode, 'l2_reg': self.l2_reg,'use_bias':self.use_bi
        base_config = super(Linear, self).get_config()
        return  dict(list(base_config.items()) + list(config.items()))
```

```
class  BiInteractionPooling(Layer):
    """Bi-Interaction Layer used in Neural FM,compress the
      pairwise element-wise product of features into one single vector.
        Input shape
            - A 3D tensor with shape:``(batch_size,field_size,embedding_size)``.
        Output shape
            - 3D tensor with shape: ``(batch_size,1,embedding_size)``.
        References
            - [He X, Chua T S. Neural factorization machines for sparse predictive an
    """

    def  __init__(self, **kwargs):
        super(BiInteractionPooling, self).__init__(**kwargs)

    def  build(self, input_shape):
        if  len(input_shape) != 3:
                raise  ValueError(
```

```
                    "Unexpected inputs dimensions %d, expect to be 3 dimensior

        super(BiInteractionPooling, self).build(input_shape)

    def call(self, inputs, **kwargs):
        if K.ndim(inputs) != 3:
            raise ValueError(
                "Unexpected inputs dimensions %d, expect to be 3 dimensior

        concated_embeds_value = inputs
        square_of_sum = tf.square(tf.reduce_sum(concated_embeds_value, axis=1, ke
        sum_of_square = tf.reduce_sum(concated_embeds_value * concated_embeds_val
        cross_term = 0.5 * (square_of_sum - sum_of_square)

        return cross_term

    def compute_output_shape(self, input_shape):
        return (None, 1, input_shape[-1])
```

```
class DNN(Layer):
    """The Multi Layer Percetron
      Input shape
        - nD tensor with shape: ``(batch_size, ..., input_dim)``. The most common
      Output shape
        - nD tensor with shape: ``(batch_size, ..., hidden_size[-1])``. For insta
      Arguments
        - **hidden_units**:list of positive integer, the layer number and units i
        - **activation**: Activation function to use.
        - **l2_reg**: float between 0 and 1. L2 regularizer strength applied to t
        - **dropout_rate**: float in [0,1). Fraction of the units to dropout.
        - **use_bn**: bool. Whether use BatchNormalization before activation or n
        - **seed**: A Python integer to use as random seed.
    """

    def __init__(self, hidden_units, activation='relu', l2_reg=0, dropout_rate=0, us
        self.hidden_units = hidden_units
        self.activation = activation
        self.dropout_rate = dropout_rate
        self.seed = seed
        self.l2_reg = l2_reg
        self.use_bn = use_bn
        super(DNN, self).__init__(**kwargs)

    def build(self, input_shape):
```

```
        # if len(self.hidden_units) == 0:
        # raise ValueError("hidden_units is empty")
        input_size = input_shape[-1]
        hidden_units = [int(input_size)] + list(self.hidden_units)
        self.kernels = [self.add_weight(name='kernel' + str(i),
                                                              shape=(hidd
                                                              initializer=
                                                              regularizer=
                                                              trainable=Tr
        self.bias = [self.add_weight(name='bias' + str(i),
                                                         shape=(self.hidde
                                                         initializer=Zeros
                                                         trainable=True) f
        if self.use_bn:
            self.bn_layers = [tf.keras.layers.BatchNormalization() for _ in

        self.dropout_layers = [tf.keras.layers.Dropout(self.dropout_rate, seed=se
                                                  range(len(self.hidden_units))

        self.activation_layers = [tf.keras.layers.Activation(self.activation) \
                                                      for _ in range(len(se

        super(DNN, self).build(input_shape)

    def call(self, inputs, training=None, **kwargs):

        deep_input = inputs

        for i in range(len(self.hidden_units)):
            fc = tf.nn.bias_add(tf.tensordot(
                deep_input, self.kernels[i], axes=(-1, 0)), self.bias[i])
            # fc = Dense(self.hidden_size[i], activation=None, \
            # kernel_initializer=glorot_normal(seed=self.seed), \
            # kernel_regularizer=l2(self.l2_reg))(deep_input)
            if self.use_bn:
                fc = self.bn_layers[i](fc, training=training)

            fc = self.activation_layers[i](fc)
            fc = self.dropout_layers[i](fc, training=training)

            deep_input = fc

        return deep_input

    def compute_output_shape(self, input_shape):
```

```
        if  len(self.hidden_units) > 0:
                shape = input_shape[:-1] + (self.hidden_units[-1],)
        else:
                shape = input_shape

        return  tuple(shape)

    def  get_config(self, ):
        config = {'activation': self.activation, 'hidden_units': self.hidden_unit
                      'l2_reg': self.l2_reg, 'use_bn': self.use_bn, 'dropout
        base_config = super(DNN, self).get_config()

        return  dict(list(base_config.items()) + list(config.items()))
```

```
class  PredictionLayer(Layer):
    """
      Arguments
          - **task**: str, ``"binary"`` for binary logloss or ``"regression"`` fo
          - **use_bias**: bool.Whether add bias term or not.
    """

    def  __init__(self, task='binary', use_bias=True, **kwargs):
        if  task not  in  ["binary", "multiclass", "regression"]:
                raise  ValueError("task must be binary, multiclass or regression")
        self.task = task
        self.use_bias = use_bias
        super(PredictionLayer, self).__init__(**kwargs)

    def  build(self, input_shape):
        if  self.use_bias:
                self.global_bias = self.add_weight(
                        shape=(1,), initializer=Zeros(), name="global_bias")

        super(PredictionLayer, self).build(input_shape)

    def  call(self, inputs, **kwargs):
        x = inputs
        if  self.use_bias:
                x = tf.nn.bias_add(x, self.global_bias, data_format='NHWC')
        if  self.task == "binary":
                x = tf.sigmoid(x)

        output = tf.reshape(x, (-1, 1))
```

```
        return  output

    def  compute_output_shape(self,  input_shape):
        return  (None,  1)

    def  get_config(self, ):
        config = {'task': self.task, 'use_bias': self.use_bias}
        base_config = super(PredictionLayer,  self).get_config()
        return  dict(list(base_config.items()) + list(config.items()))
```

(2.2.4) 定义NFM模型结构

```
def  nfm(sparse_feature_list, \
            sparse_feature_reindex_dict, \
            dense_feature_list, \
            dnn_hidden_units=(128,  128), \
            l2_reg_embedding=1e-5, \
            l2_reg_linear=1e-5, \
            l2_reg_dnn=0, \
            init_std=0.0001, \
            seed=1024, \
            bi_dropout=0.2,
            dnn_dropout=0.2, \
            dnn_activation='relu', \
            task='binary'):

    sparse_input_layer_list = []
    sparse_embedding_layer_list = []

    dense_input_layer_list = []


    # 1.  Input & Embedding sparse features
    for  i  in  sparse_feature_list:
        shape = 1
        name = i
        vocabulary_size = len(sparse_feature_reindex_dict[i]) + 1
        embedding_dim = 64

        cur_sparse_feaure_input_layer,  cur_sparse_feaure_embedding_layer = \
            input_embedding_layer(
                shape = shape, \
                name = name, \
                vocabulary_size = vocabulary_size, \
```

```
                    embedding_dim = embedding_dim)

        sparse_input_layer_list.append(cur_sparse_feaure_input_layer)
        sparse_embedding_layer_list.append(cur_sparse_feaure_embedding_layer)


    # 2. Input dense features
    for j in dense_feature_list:
        dense_input_layer_list.append(Input(shape=(1, ), name=j))


    # === linear ===
    sparse_linear_input = Concatenate(axis=-1)(sparse_embedding_layer_list)
    dense_linear_input = Concatenate(axis=-1)(dense_input_layer_list)
    linear_logit = Linear()([sparse_linear_input, dense_linear_input])


    # === nfm cross ===
    nfm_input = Concatenate(axis=1)(sparse_embedding_layer_list)
    bi_out = BiInteractionPooling()(nfm_input)

    if bi_dropout:
        bi_out = tf.keras.layers.Dropout(bi_dropout)(bi_out, training=None)

    bi_out = Flatten()(bi_out)


    dnn_input = Concatenate(axis=-1)([bi_out, dense_linear_input])
    dnn_output = DNN(dnn_hidden_units, dnn_activation, l2_reg_dnn, dnn_dropout, False
    dnn_logit = tf.keras.layers.Dense(1, use_bias=False, activation=None)(dnn_output)


    # === finally dense ===
    out = PredictionLayer(task)(tf.keras.layers.add([linear_logit, dnn_logit]))
    nfm_model = Model(inputs = sparse_input_layer_list + dense_input_layer_list, outp

    return nfm_model
```

（2.2.5）应用NFM模型

```
nfm_model = nfm(sparse_feature_list, \
                sparse_feature_reindex_dict, \
                dense_feature_list)
```

（2.2.6）打印NFM模型 summary

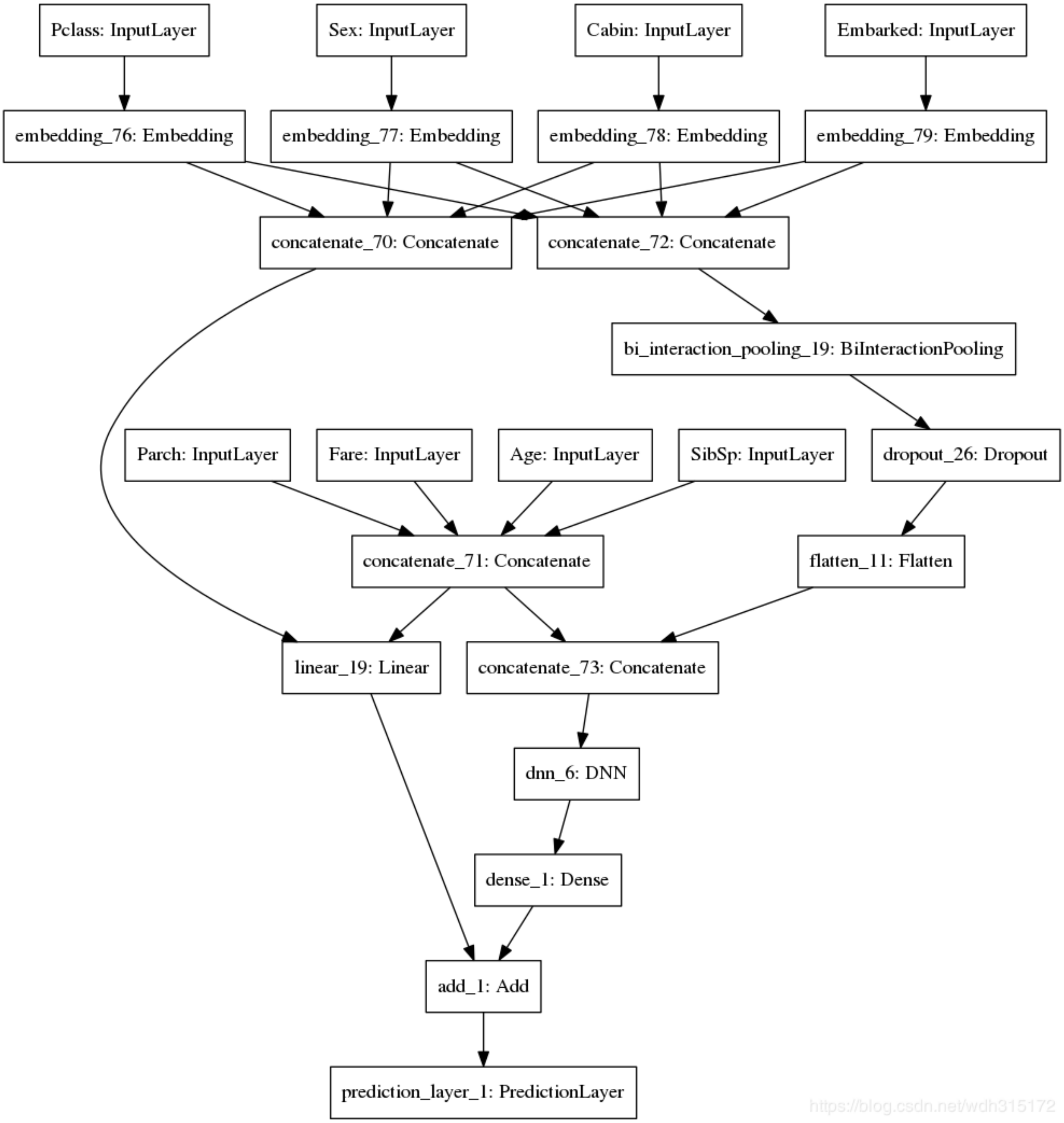
```
print(nfm_model.summary())
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Pclass (InputLayer)             (None, 1)            0
__________________________________________________________________________________________________
Sex (InputLayer)                (None, 1)            0
__________________________________________________________________________________________________
Cabin (InputLayer)              (None, 1)            0
__________________________________________________________________________________________________
Embarked (InputLayer)           (None, 1)            0
__________________________________________________________________________________________________
embedding_76 (Embedding)        (None, 1, 64)        256         Pclass[0][0]
__________________________________________________________________________________________________
embedding_77 (Embedding)        (None, 1, 64)        192         Sex[0][0]
__________________________________________________________________________________________________
embedding_78 (Embedding)        (None, 1, 64)        9536        Cabin[0][0]
__________________________________________________________________________________________________
embedding_79 (Embedding)        (None, 1, 64)        320         Embarked[0][0]
__________________________________________________________________________________________________
concatenate_72 (Concatenate)    (None, 4, 64)        0           embedding_76[0][0]
                                                                 embedding_77[0][0]
                                                                 embedding_78[0][0]
                                                                 embedding_79[0][0]
__________________________________________________________________________________________________
bi_interaction_pooling_19 (BiIn (None, 1, 64)        0           concatenate_72[0][0]
__________________________________________________________________________________________________
Age (InputLayer)                (None, 1)            0
__________________________________________________________________________________________________
SibSp (InputLayer)              (None, 1)            0
__________________________________________________________________________________________________
Parch (InputLayer)              (None, 1)            0
__________________________________________________________________________________________________
Fare (InputLayer)               (None, 1)            0
__________________________________________________________________________________________________
dropout_26 (Dropout)            (None, 1, 64)        0           bi_interaction_pooling_19[0][0]
__________________________________________________________________________________________________
```

```
________________________________________________________________________________________________
concatenate_71 (Concatenate)    (None, 4)           0           Age[0][0]
                                                                SibSp[0][0]
                                                                Parch[0][0]
                                                                Fare[0][0]
________________________________________________________________________________________________
flatten_11 (Flatten)            (None, 64)          0           dropout_26[0][0]
________________________________________________________________________________________________
concatenate_73 (Concatenate)    (None, 68)          0           flatten_11[0][0]
                                                                concatenate_71[0][0]
________________________________________________________________________________________________
concatenate_70 (Concatenate)    (None, 1, 256)      0           embedding_76[0][0]
                                                                embedding_77[0][0]
                                                                embedding_78[0][0]
                                                                embedding_79[0][0]
________________________________________________________________________________________________
dnn_6 (DNN)                     (None, 128)         25344       concatenate_73[0][0]
________________________________________________________________________________________________
linear_19 (Linear)              (None, 1)           5           concatenate_70[0][0]
                                                                concatenate_71[0][0]
________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)           128         dnn_6[0][0]
________________________________________________________________________________________________
add_1 (Add)                     (None, 1)           0           linear_19[0][0]
                                                                dense_1[0][0]
________________________________________________________________________________________________
prediction_layer_1 (PredictionL (None, 1)           1           add_1[0][0]
================================================================================================
Total params: 35,782
Trainable params: 35,782
Non-trainable params: 0
________________________________________________________________________________________________
```


(2.2.7）输出NFM模型结构图

```
plot_model(nfm_model, to_file='nfm_model.png')
```



（2.2.8）编译 NFM 模型，训练模型

```
nfm_model.compile(loss='binary_crossentropy', \
              optimizer=Adam(lr=1e-3), \
              metrics=['accuracy'])
```

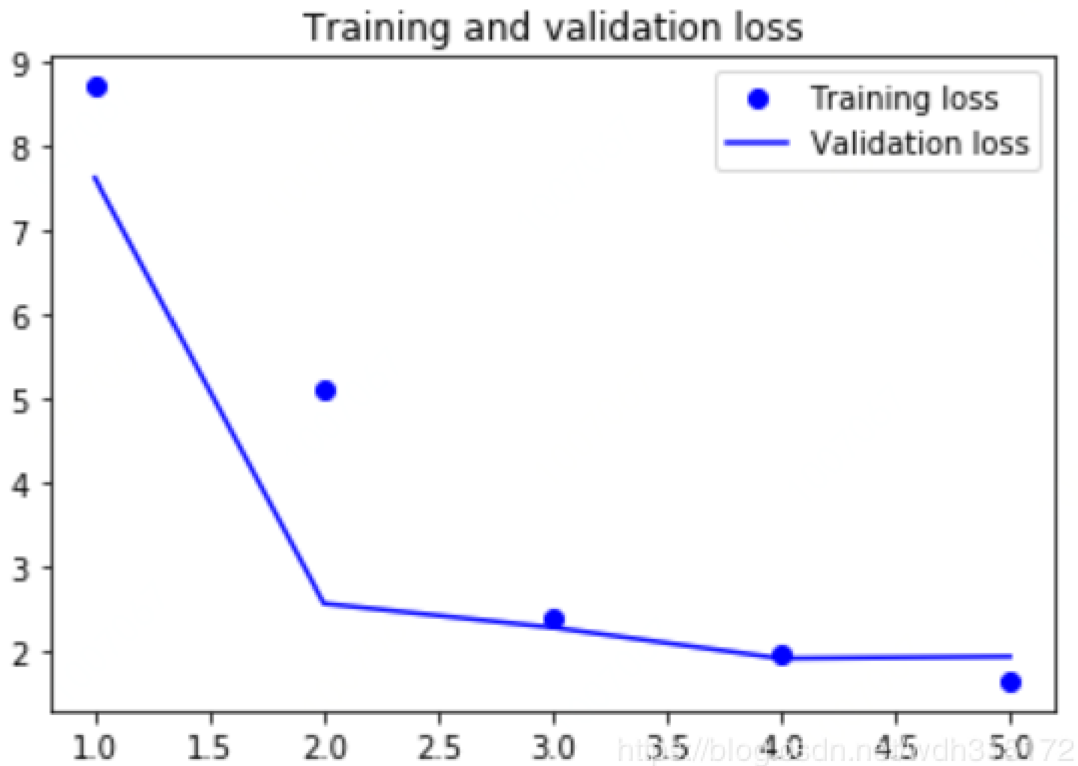
```
history = nfm_model.fit(xtrain_data, ytrain, epochs=5, batch_size=32, validation_data=(x
```

```
Train on 712 samples, validate on 179 samples
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
712/712 [==============================] - 0s 536us/sample - loss: 8.7202 - acc: 0.3933 - val_loss: 7.6303 - val_acc: 0.4525
Epoch 2/5
712/712 [==============================] - 0s 143us/sample - loss: 5.1172 - acc: 0.5126 - val_loss: 2.5739 - val_acc: 0.6145
Epoch 3/5
712/712 [==============================] - 0s 138us/sample - loss: 2.4104 - acc: 0.6320 - val_loss: 2.2888 - val_acc: 0.6201
Epoch 4/5
712/712 [==============================] - 0s 136us/sample - loss: 1.9834 - acc: 0.6980 - val_loss: 1.9198 - val_acc: 0.6592
Epoch 5/5
712/712 [==============================] - 0s 141us/sample - loss: 1.6550 - acc: 0.7416 - val_loss: 1.9423 - val_acc: 0.6480
```

(2.2.9）绘制损失函数图

```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
print(plt.show())
```



参考: