注意力机制在深度推荐算法中的应用之AFM模型

原创 王多鱼 python科技园 6月27日

1

前言

注意力机制来源于人类最自然的选择性注意的习惯，例如当用户浏览网页或图片时，会选择性的注意页面上特定的区域。基于此现象，在建模的过程中考虑注意力机制，往往会取得不错的收益。

注意力机制已经广泛的应用于深度学习的各个领域，无论是NLP、语音识别还是图像识别，引入注意力机制的模型均取得了不错的效果。从2017年开始，推荐领域也开始尝试引入注意力机制，AFM模型（Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks）就是其中较早的工作之一。

2

AFM模型原理简介

AFM模型可以认为是NFM模型（前文回顾：NFM模型理论与实践）的延伸，在NFM模型中，不同域的特征Embedding向量经过特征交叉后，将各交叉特征向量按照Embedding Size维度进行“加和”，最后输出由多层神经网络组成的输出层。问题的关键在于“加和”池化操作，它相当于是“平等”地对待所有交叉特征，未考虑特征对结果的影响程度，事实上消除了大量有价值的信息。

这里“注意力机制”就登上了推荐系统的历史舞台，基于“不同的交叉特征对结果的影响程度不同”的假设条件。例如：如果应用场景为“预测一位男性用户是否购买鼠标的可能性”，那么“性别=男”和“购买历史包含键盘”这个交叉特征比“性别=男”和“用户年龄=25”这一交叉特征更重要，模型在前一交叉特征上投入了更多的“注意力”。

AFM模型是在“特征交叉层”和“输出层”之间引入注意力网络实现的，注意力网络的作用为每一个交叉特征提供权重。AFM模型结构如图1所示。

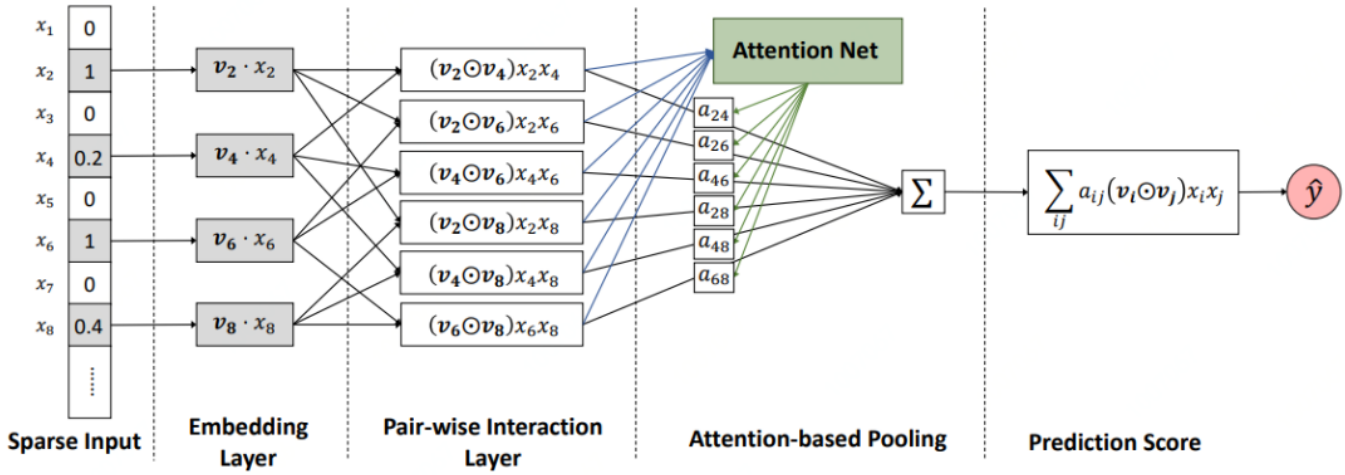


Figure 1: The neural network architecture of our proposed Attentional Factorization Machine model.

Sparse Input：输入稀疏特征；

Embedding Layer：对输入的特征进行Embedding编码；

Pair-wise Interaction Layer：两两特征交叉；

Attention-based Pooling：基于注意力机制的网络层；

Prediction Score：输出预测得分；

2.1 AFM模型的公式表达式

$$\hat{y}_{AFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \mathbf{p}^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j$$

其中：

- 红框中的表达式为线性表达；
- 篮框中的表达式为Attention表达，p 为权重向量；

2.2 AFM模型的Attention部分公式表达式

$$f_{Att}(f_{PI}(\mathcal{E})) = \sum_{(i,j)\in\mathcal{R}_x} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j$$

其中：

- $f_{PI}(\mathcal{E}) = \{(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j\}_{(i,j)\in\mathcal{R}_x}$， $v_i \odot v_j$ 是两个Embedding特征之间的元素积操作；
- a_{ij} 是注意力得分。最简单的方法是使用一个权重参数来表示，但是为了防止特征交叉数据稀疏问题带来的权重参数难以收敛，此处使用单层的全连接网络进行参数学习。

$$a'_{ij} = \mathbf{h}^T ReLU(\mathbf{W}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j + \mathbf{b})$$

$$a_{ij} = \frac{\exp(a'_{ij})}{\sum_{(i,j)\in\mathcal{R}_x} \exp(a'_{ij})}$$

其中要学习模型参数就是特征交叉层到注意力网络全连接层的权重矩阵 $W \in \mathbb{R}^{t\times k}$，偏置向量 $b \in \mathbb{R}^{t\times 1}$，以及全连接层到softmax输出层的权重向量 $h \in \mathbb{R}^{t\times 1}$，$t = f(f-1)/2,\ k = embedding\ dim$。可以看出 a_{ij} 随 $v_i \odot v_j$ 的变化而变化。

3

AFM模型实践

3.1 首先来看一看AFM模型的Attention Layer部分：

$$\mathrm{p}^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{ij}(\mathrm{v}_i \odot \mathrm{v}_j)x_i x_j$$

```
import tensorflow as tf
from tensorflow.keras.layers import Layer
from tensorflow.keras import backend as K
import itertools
from tensorflow.keras.initializers import (Zeros, glorot_normal, glorot_uniform)
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.regularizers import l2

class  AFMLayer(Layer):
        def  __init__(self, attention_factor=4, l2_reg_w=0, dropout_rate=0, seed=1024, **k
                self.attention_factor = attention_factor
                self.l2_reg_w = l2_reg_w
                self.dropout_rate = dropout_rate
                self.seed = seed
                super(AFMLayer, self).__init__(**kwargs)

        def  build(self, input_shape):
                if  not  isinstance(input_shape, list) or  len(input_shape) < 2:
```

```
                raise ValueError('A `AttentionalFM` layer should be called '
                                 'on a list of at least 2 inputs')

            shape_set = set()
            reduced_input_shape = [shape.as_list() for shape in input_shape]
            for i in range(len(input_shape)):
                shape_set.add(tuple(reduced_input_shape[i]))

            if len(shape_set) > 1:
                raise ValueError('A `AttentionalFM` layer requires '
                                 'inputs with same shapes '
                                 'Got different shapes: %s' % (shape_set))

            if len(input_shape[0]) != 3 or input_shape[0][1] != 1:
                raise ValueError('A `AttentionalFM` layer requires '
                                 'inputs of a list with same shape tensor like\
                                 (None, 1, embedding_size)'
                                 'Got different shapes: %s' % (input_shape[0]))

            embedding_size = int(input_shape[0][-1])

            self.attention_W = self.add_weight(shape=(embedding_size,
                               self.attention_factor), initializer=glorot_normal(seed=se
                                                                                 regular
            self.attention_b = self.add_weight(
                shape=(self.attention_factor,), initializer=Zeros(), name="attenti
            self.projection_h = self.add_weight(shape=(self.attention_factor, 1),
                                                                                initi
            self.projection_p = self.add_weight(shape=(
                embedding_size, 1), initializer=glorot_normal(seed=self.seed), name
            self.dropout = tf.keras.layers.Dropout(
                self.dropout_rate, seed=self.seed)

            self.tensordot = tf.keras.layers.Lambda(
                lambda x: tf.tensordot(x[0], x[1], axes=(-1, 0)))

            # Be sure to call this somewhere!
            super(AFMLayer, self).build(input_shape)

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

            if K.ndim(inputs[0]) != 3:
                raise ValueError(
                "Unexpected inputs dimensions %d, expect to be 3 dimensions" % (K.
```

```
            embeds_vec_list = inputs
            row = []
            col = []

            for  r, c in  itertools.combinations(embeds_vec_list, 2):
                    row.append(r)
                    col.append(c)

            p = tf.concat(row, axis=1)
            q = tf.concat(col, axis=1)
            inner_product = p * q

            bi_interaction = inner_product
            attention_temp = tf.nn.relu(tf.nn.bias_add(tf.tensordot(
                    bi_interaction, self.attention_W, axes=(-1, 0)), self.attention_b)

            self.normalized_att_score = tf.nn.softmax(tf.tensordot(
                    attention_temp, self.projection_h, axes=(-1, 0)))
            attention_output = tf.reduce_sum(
                    self.normalized_att_score * bi_interaction, axis=1)

            attention_output = self.dropout(attention_output) # training
            afm_out = self.tensordot([attention_output, self.projection_p])
            return  afm_out

    def  compute_output_shape(self, input_shape):
            if  not  isinstance(input_shape, list):
                    raise ValueError('A `AFMLayer` layer should be called '
                                     'on a list of inputs.')
            return  (None, 1)

    def  get_config(self, ):
            config = {'attention_factor': self.attention_factor,
                      'l2_reg_w': self.l2_reg_w, 'dropout_rate': self.dropout_rate, 'se
            base_config = super(AFMLayer, self).get_config()
            return  dict(list(base_config.items()) + list(config.items()))
```

3.2 AFM模型设计

（1）数据情况

本文使用的数据集是《movielens-1M数据》，数据下载地址：

http://files.grouplens.org/datasets/movielens/ml-1m.zip

将数据集加工成如下格式：

```
2786  2  3  5  2041  1023  1
2786  2  3  5  2041  2697  0
2786  2  3  5  2041  794  0
```

数据说明：

第 1 列	user_id	用户id
第 2 列	gender	用户性别
第 3 列	age	用户年龄
第 4 列	occupation	用户工作
第 5 列	zip	用户邮编
第 6 列	movie_id	用户下一步是否观看的电影
第 7 列	label	1表示正样本，0表示负样本

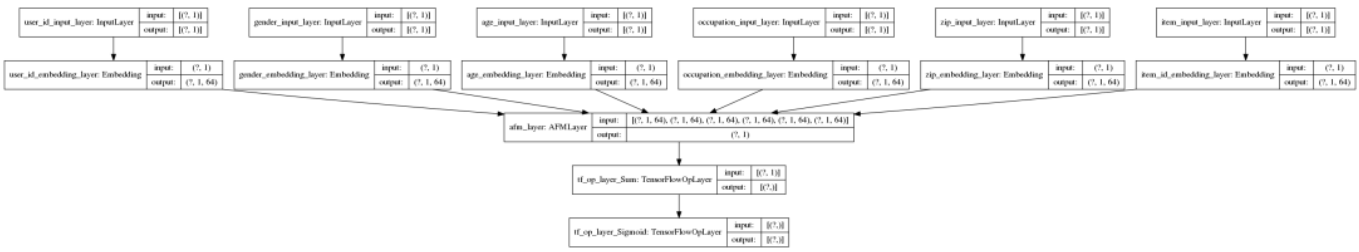
数据加工逻辑见：https://github.com/wziji/deep_ctr/blob/master/AFM/data.py

(2) AFM模型的结构

```
#-*- coding:utf-8 -*-

import tensorflow as tf
from tensorflow.keras.layers import Input, Embedding, concatenate, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from AFMLayer import AFMLayer

def AFM(
        sparse_input_length=1,
        embedding_dim = 64):

        # 1. Input layer
```

```
    user_id_input_layer = Input(shape=(sparse_input_length, ), name="user_id_input_lay
    gender_input_layer = Input(shape=(sparse_input_length, ), name="gender_input_layer
    age_input_layer = Input(shape=(sparse_input_length, ), name="age_input_layer")
    occupation_input_layer = Input(shape=(sparse_input_length, ), name="occupation_inp
    zip_input_layer = Input(shape=(sparse_input_length, ), name="zip_input_layer")
    item_input_layer = Input(shape=(sparse_input_length, ), name="item_input_layer")

    # 2. Embedding layer
    user_id_embedding_layer = Embedding(6040+1, embedding_dim, mask_zero=True, name='u
    gender_embedding_layer = Embedding(2+1, embedding_dim, mask_zero=True, name='gende
    age_embedding_layer = Embedding(7+1, embedding_dim, mask_zero=True, name='age_embe
    occupation_embedding_layer = Embedding(21+1, embedding_dim, mask_zero=True, name='
    zip_embedding_layer = Embedding(3439+1, embedding_dim, mask_zero=True, name='zip_e
    item_id_embedding_layer = Embedding(3706+1, embedding_dim, mask_zero=True, name='i

    sparse_embedding_list = [user_id_embedding_layer, gender_embedding_layer, age_embe
                             occupation_embedding_layer, zip_embedding_layer, item_id_

    # 3. AFM
    attention_factor = int(len(sparse_embedding_list) * (len(sparse_embedding_list)-1)
    afm_out = AFMLayer(attention_factor=attention_factor, l2_reg_w=1e-5, dropout_rate=
            (sparse_embedding_list)

    # Output
    output = tf.nn.sigmoid(tf.reduce_sum(afm_out, axis=-1))
    sparse_input_list = [user_id_input_layer, gender_input_layer, age_input_layer, occ
    model = Model(inputs = sparse_input_list,
                                outputs = output)

    return  model
```

AFM模型结构图如下所示：



（3） 训练AFM模型

```
#-*- coding:utf-8 -*-

import  tensorflow as  tf
from  tensorflow.keras.optimizers import  Adam
from  data_generator import  file_generator
from  AFM_Model import  AFM

# 1. Load data

train_path = "train.txt"
val_path = "test.txt"
batch_size = 1000

n_train = sum([1  for  i in  open(train_path)])
n_val = sum([1  for  i in  open(val_path)])

train_steps = n_train / batch_size
train_steps_ = n_train // batch_size
validation_steps = n_val / batch_size
validation_steps_ = n_val // batch_size

train_generator = file_generator(train_path, batch_size)
val_generator = file_generator(val_path, batch_size)

steps_per_epoch = train_steps_ if  train_steps==train_steps_ else  train_steps_ + 1
validation_steps = validation_steps_ if  validation_steps==validation_steps_ else  validat

print("n_train: ", n_train)
print("n_val: ", n_val)

print("steps_per_epoch: ", steps_per_epoch)
print("validation_steps: ", validation_steps)



# 2. Train model

early_stopping_cb = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, rest
callbacks = [early_stopping_cb]

model = AFM()
print(model.summary())
tf.keras.utils.plot_model(model, to_file='AFM_model.png', show_shapes=True)

model.compile(loss='binary_crossentropy', \
        optimizer=Adam(lr=1e-3), \
```

```
        metrics=['accuracy'])

history = model.fit(train_generator, \
                    epochs=1, \
                    steps_per_epoch = steps_per_epoch, \
                    callbacks = callbacks,
                    validation_data = val_generator, \
                    validation_steps = validation_steps, \
                    shuffle=True)

model.save_weights('AFM_model.h5')
```

本文的代码请见：https://github.com/wziji/deep_ctr/blob/master/AFM

参考：

AFM论文：https://arxiv.org/pdf/1708.04617.pdf

深度学习推进系统，王喆著

https://github.com/shenweichen/DeepCTR

欢迎关注 **“python科技园”** 及 **添加小编** 进群交流。

文章好看点这里