推荐系统系列（八）：AFM理论与实践

原创 默存 SOTA Lab 2019-11-27

前言

到目前为止，介绍的模型大多将优化重心放在了特征交叉部分，今天要介绍的AFM（Attentional Factorization Machine）[1]同样如此。该模型于2017年，由浙大与新加坡国立大学合作推出。有意思的是，NFM的作者同样参与了AFM研究，所以在AFM中可以很明显的看到NFM [2]的痕迹。

AFM的贡献在于，将attention机制引入到特征交叉模块。不同于NFM在Bi-Interaction Layer中的处理，AFM在对二阶交叉特征进行sum pooling时，使用attention network计算得到的attention score对交叉特征进行加权求和。因为从直觉上来说，不同的交叉特征有着不同的重要性，所以通过模型显式的学习出每种交叉特征的权重，是有益于模型的性能提升的，并且对于模型的可解释性也有帮助。下面将对AFM进行详细介绍。

分析

1. AFM定义

首先抛出AFM的公式化定义，即公式（1）所示：

$$\hat{y}_{AFM}(X) = w_0 + \sum_{i=1}^{n} w_i x_i + p^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{i,j}(v_i \odot v_j) x_i x_j \qquad (1)$$

接下来，让我们逐步的对公式（1）进行分析。与NFM类似，不妨假设：

$$f(X) = p^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{i,j}(v_i \odot v_j) x_i x_j \qquad (2)$$

那么，公式（1）可表示为：

$$\hat{y}_{AFM}(X) = w_0 + \sum_{i=1}^{n} w_i x_i + f(X) \qquad (3)$$

绝大多数推荐模型都可以用公式（3）来表示，其中求和的三项分别为：全局偏置项、一阶项、高阶项。对于前两项，我们已经很熟悉了，所以我们的重点在于第三项。

前言中提到，AFM的优化重心是在交叉特征部分，即公式中的第三项 $f(X)$ 。论文作者给出了 $f(X)$ 的结构示意图，如下所示：

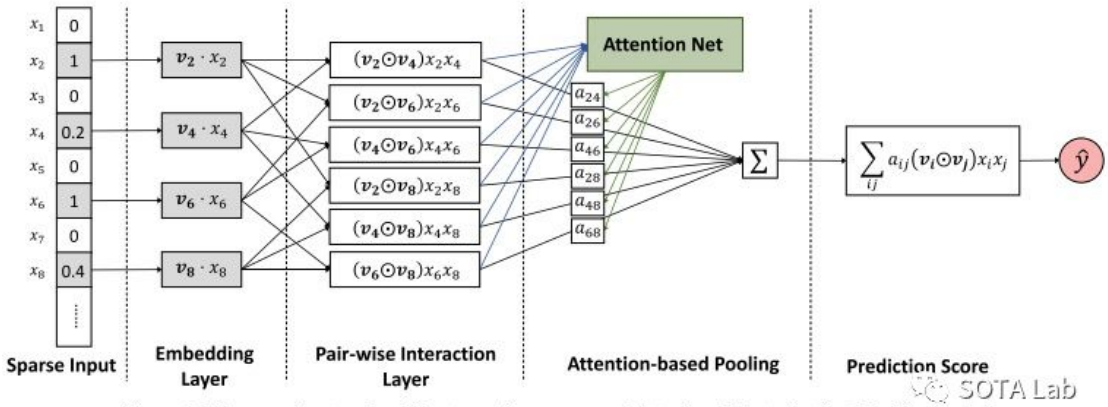

Figure 1: The neural network architecture of our proposed Attentional Factorization Machine model.

从图中可看出，$f(X)$ 共分为5个部分，依次进行拆解。

1.1 Sparse Input

这个部分主要是为了说明输入数据特点。在对输入特征进行二值化、离散化等处理之后，输入数据往往会变成高维稀疏数据。

1.2 Embedding Layer

高维稀疏数据不利于模型的学习。与之前介绍的模型类似，AFM同样使用了Embedding Layer来对高维稀疏特征进行降维，将其表示为低维稠密特征。每个特征对应一个隐向量，将特征值与隐向量进行乘积，得到特征向量作为实际特征表示。

1.3 Pair-wise Interaction Layer

这个部分的主要工作是将特征进行二阶交叉，n 个特征向量进行两两交叉，得到 $n(n-1)/2$ 个二阶交叉特征向量。将Pair-wise Interaction Layer表示为 $f_{PI}(\xi)$ ，有：

$$f_{PI}(\xi) = \{(v_i \odot v_i)x_i x_j\}_{(i,j)\in R_x} \qquad (4)$$

其中，ξ 表示Embedding Layer的输出， R_x 表示所有可能的二阶交叉组合，x_i 表示特征值，v_i 表示维度为 k 的隐向量，运算符 $\odot$ 表示向量对应元素相乘。假设 $m = n(n-1)/2$ ，那么 $f_{PI}(\xi) \in \mathbb{R}^{k\times m}$ 。

细心的同学可能看出来了，Pair-wise Interaction Layer与NFM中的Bi-Interaction Layer在交叉过程完全一样，只是Bi-Interaction Layer在交叉完了之后再进行向量等权求和处理。

与NFM一样，AFM同样可以将FM纳入到模型框架中。如果对 $f_{PI}(\xi)$ 进行如下计算：$\hat{y} = z^T \sum f_{PI}(\xi) + b$ ，其中 $\sum$ 表示向量对应元素求和，$\sum$ 输出为一个向量，z 为参数向量，b

为偏置。假设将 z 固定为全1向量，b 等于0，此时 $\hat{y}$ 与FM的二阶项等价。如果再将全局偏置与一阶项考虑进来，此时模型与FM完全等价。

1.4 Attention-based Pooling

在NFM中，特征向量进行两两交叉之后，直接进行sum pooling，将二阶交叉向量进行等权求和处理。但是直觉上来说，不同的交叉特征应该有着不同的重要性。不重要的交叉特征应该降低其权重，而重要性高的交叉特征应该提高其权重。Attention概念与该思想不谋而合，AFM作者顺势将其引入到模型之中，为每个交叉特征引入重要性权重，最终在对特征向量进行sum pooling时，利用重要性权重对二阶交叉特征进行加权求和。

为了计算特征重要性权重，作者构建了一个Attention Network，其本质是含有一个隐藏层的多层感知机（MLP）。表达式如下：

$$
\begin{aligned}
a'_{i,j} &= h^T Relu(W(v_i \odot v_j)x_i x_j + b) \\
a_{i,j} &= \frac{exp(a'_{i,j})}{\sum_{(i,j)\in R_x} exp(a'_{i,j})}
\end{aligned} \tag{5}
$$

其中，$W \in \mathbb{R}^{t\times k}$，$b \in \mathbb{R}^t$，$h \in \mathbb{R}^t$ 是模型参数。参数 t 是Attention Network隐藏层节点数，在原文中被称之为 $attention factor$。计算得到的 $a_{i,j}$ 即表示对应的二阶交叉特征 $(v_i \odot v_j)x_i x_j$ 的重要性权重。

当得到权重之后，便可以对二阶交叉特征进行加权 sum pooling。可以表示为 $\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_{i,j}(v_i \odot v_j)x_i x_j$，其结果仍然是一个向量，作为下一模型结构的输入。

1.5 Prediction Score

Attention-based Pooling得到二阶交叉信息，在这个部分负责将其表示为一个标量。具体定义为，$\hat{y} = p^T f_{att}(X)$，其中 $f_{att}(X)$ 表示Attention-based Pooling的输出，p 为参数向量。

至此，我们完成了对AFM核心模块 $f(X)$ 的分析。最终AFM的公式定义如下：

$$
\begin{aligned}
\hat{y}_{AFM}(X) &= w_0 + \sum_{i=1}^{n} w_i x_i + f(X) \\
&= w_0 + \sum_{i=1}^{n} w_i x_i + p^T \sum_{i=1}^{n}\sum_{j=i+1}^{n} a_{i,j}(v_i \odot v_j)x_i x_j
\end{aligned} \tag{6}
$$

模型参数集合为 $\{w_0, w_i, v_i, p, h, W, b\}$。

2. 过拟合风险

很明显AFM是对于NFM的改进，但在改进同时引入了更多的参数，增加了模型过拟合风险。作者使用了dropout与L2正则的方式来避免过拟合问题。

对于Pair-wise Interaction Layer部分，作者使用了dropout技术（根据作者开源代码 [3] 分析，实际位置为交叉特征根据重要性权重加权求和之后），而在Attention Network部分使用L2进行正则化。因为在论文中作者使用均方误差作为Loss函数，所以最终的Loss函数表示为：

$$L = \sum_{X \in \tau} (\hat{y}_{AFM}(X) - y(X))^2 + \lambda ||W||^2 \qquad (7)$$

其中，τ 代表全体训练样本。

3. 性能分析

与NFM类似，在对模型进行消融分析时，作者从三个问题出发，设计实验证明模型的优越性。实验数据集为 *Frappe* 与 *MovieLens* 。对比模型分别有：LibFM、HOFM、Wide&Deep、DeepCross。对于所有实验Embedding Size都为256，如未提及， $attention factor$ 均为256。

在实验过程中，作者发现对于AFM、Wide&Deep、DeepCross模型，相对于随机初始化参数，使用预训练的FM参数进行初始化可以提高模型性能。

> For Wide&Deep, DeepCross and AFM, we find that pre-training their feature embeddings with FM leads to a lower RMSE than a random initialization. As such, we report their performance with pre-training.

3.1 问题一

Q：AFM的关键超参数（如，dropout与L2正则化）对于性能的影响如何？

对比三种模型（LibFM、FM、AFM）在不同dropout比例下的表现，其中的FM是将AFM的Attention Network去除得到的等价模型。实验效果如下：

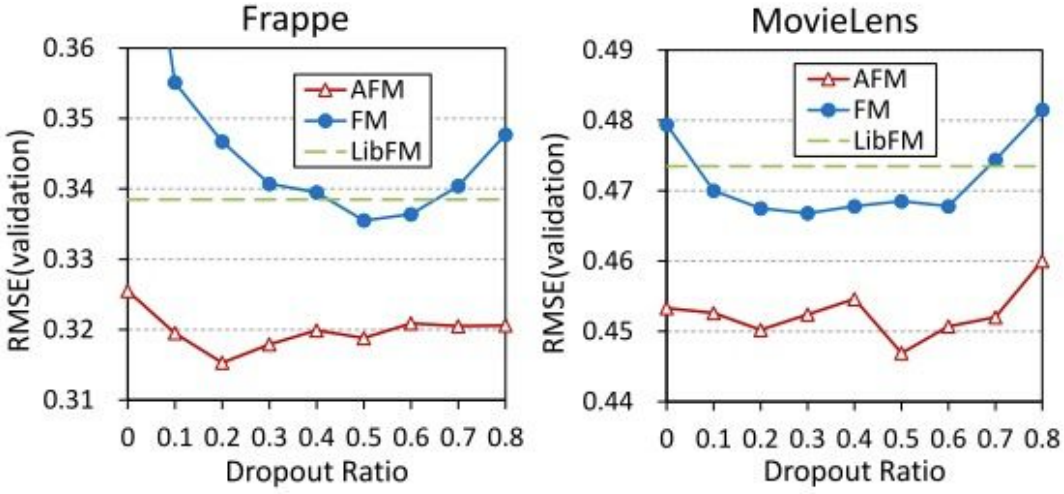

Figure 2: Validation error of AFM and FM *w.r.t.* different dropout ratios on the pair-wise interaction layer

该实验表明了一些结论，

1）在Pair-wise Interaction Layer使用dropout，可以提高模型的表现。

2）对比LibFM与FM，发现FM的效果优于LibFM，可能的原因在于优化器的不同，LibFM使用普通的SGD算法，而FM使用Adagrad进行优化。还有一个原因在于，LibFM使用L2正则来防止过拟合，而该FM模型使用dropout。

3）即使在AFM不使用dropout时，其效果仍大幅优于FM与LibFM，说明Attention Network的有效性。

将AFM设置为最佳的dropout比例，然后我们来看在不同的 λ （L2正则强度）下，AFM的表现如何。对比效果如下：

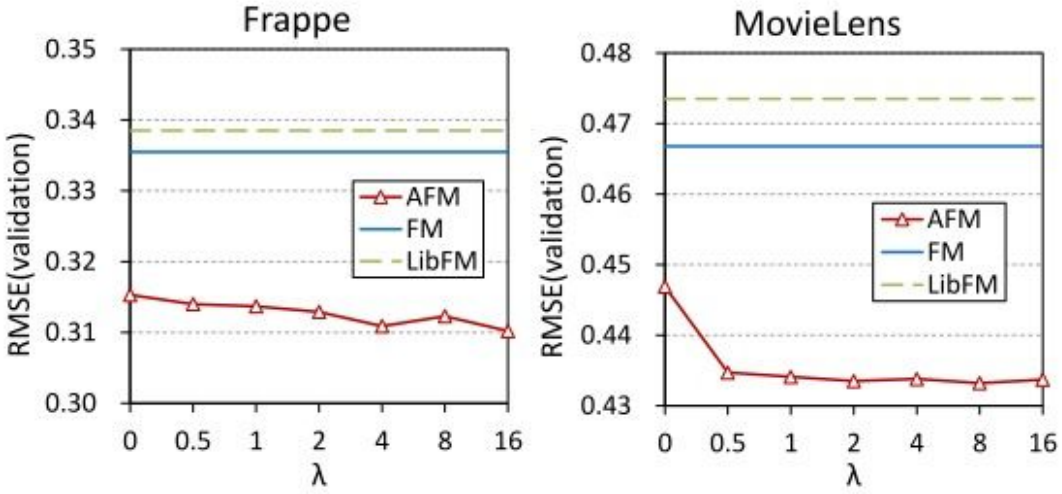

Figure 3: Validation error of AFM *w.r.t.* different regularization strengths on the attention network

可以看出，随着正则化强度的提高，AFM的性能得到进一步提高。这说明仅对Pair-wise Interaction Layer使用dropout是不够的，加上Attention Network的L2正则才能达到最佳效果。

3.2 问题二

Q：Attention Network能够有效地学习交叉特征的重要性吗？

在回答这个问题之前，我们应该选择一个合适的 $attention factor$ 。下图展示了在不同的 $attention factor$ 下模型的表现，每次试验都将 dropout 与 λ 设置到最佳。

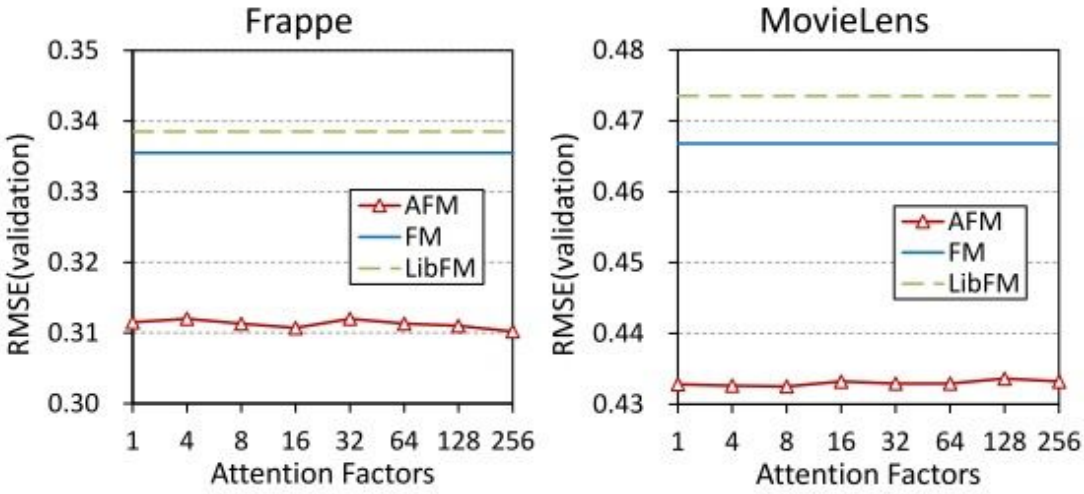

Figure 4: Validation error of AFM *w.r.t.* different attention factors

从图上可以看出，随着 $attention factor$ 的改变，AFM的表现相当稳定。上文中提到，Attention Network是一个含有一层隐藏层的感知机。所以当 $attention factor$ 等于1时，Attention Network的隐藏层只有一个节点。尽管条件设定很严格，AFM仍然能有突出的表现，这说明了Attention Network的合理性，不同的交叉特征根据重要性权重进行加权求和，能够有效提高模型表现。

此外，对比AFM与FM在训练过程中的收敛速度，结果如下。在两个数据集上，AFM都有着不错的表现。

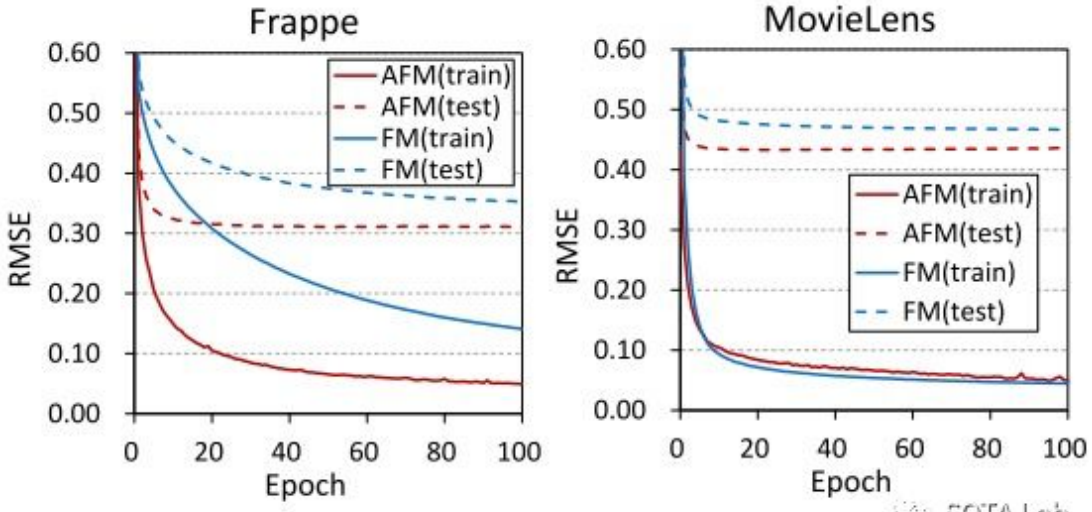

Figure 5: Training and test error of each epoch

为了能够更深入的分析Attention Network的优势，作者设计实验。首先将Attention Network参数固定，将所有的 $a_{i,j}$ 设置为 $1/|R_x|$，模拟FM训练Embedding向量，直到模型收敛。然后再将Embedding Layer进行固定，放开Attention Network进行训练，直到模型再次收敛。结果显示，第二次收敛模型性能提高了3%。这充分说明了Attention Network的有效性。

在实验过程中抽取三条样本进行观察，在FM阶段与放开Attention Network之后的阶段，Item-Tag的交叉特征都是最重要的，但是在FM阶段模型是将所有交叉特征等权求和的（权重均为

0.33）。而在加入Attention Network之后，模型给Item-Tag分配了更高的权重，三条样本分别分配了0.38、0.42、0.37权重。这说明模型确实能够自适应的调整不同交叉特征的权重，提高模型表现。

Table 1: The attention_score*interaction_score of each feature interaction of three test examples on MovieLens.

#	Model	User-Item	User-Tag	Item-Tag	$\hat{y}$
1	FM	0.33*-1.81	0.33*-2.65	0.33*4.55	0.03
	FM+A	0.34*-1.81	0.27*-2.65	0.38*4.55	0.39
2	FM	0.33*-1.62	0.33*-1.00	0.33*3.32	0.24
	FM+A	0.38*-1.62	0.20*-1.00	0.42*3.32	0.56
3	FM	0.33*-1.40	0.33*-1.26	0.33*4.68	0.67
	FM+A	0.33*-1.40	0.29*-1.26	0.37*4.68	0.89

3.3 问题三

Q：在稀疏数据集上，AFM与最好的模型比较是否有提高呢？

对比几种模型，结果如下。AFM参数少，但是表现最佳。

Table 2: Test error and number of parameters of different methods on embedding size 256. M denotes "million".

	Frappe		MovieLens	
Method	Param#	RMSE	Param#	RMSE
LibFM	1.38M	0.3385	23.24M	0.4735
HOFM	2.76M	0.3331	46.40M	0.4636
Wide&Deep	4.66M	0.3246	24.69M	0.4512
DeepCross	8.93M	0.3548	25.42M	0.5130
AFM	**1.45M**	**0.3102**	**23.26M**	**0.4325**

实践

仍然使用 $MovieLens100K$ 作为数据集，核心代码如下。

参数含义：

`vec_dim` ：代表embedding vector维度。

`field_lens` ：list结构，其中每个元素代表对应Field有多少取值。例如gender有两个取值，那么其对应的元素为2。

`attention_factor` ：与论文中含义一致。

`lr` ：学习率。

`lamda` ：L2正则化强度。

```
class AFM(object):
    def __init__(self, vec_dim=None, field_lens=None, attention_factor=None, lr=None, dropout_rate=
        self.vec_dim = vec_dim
        self.field_lens = field_lens
        self.field_num = len(field_lens)
        self.attention_factor = attention_factor
        self.lr = lr
        self.dropout_rate = dropout_rate
        self.lamda = float(lamda)

        self.l2_reg = tf.contrib.layers.l2_regularizer(self.lamda)

        self._build_graph()

    def _build_graph(self):
        self.add_input()
        self.inference()

    def add_input(self):
        self.x = [tf.placeholder(tf.float32, name='input_x_%d'%i) for i in range(self.field_num)]
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')
        self.is_train = tf.placeholder(tf.bool)

    def inference(self):
        with tf.variable_scope('linear_part'):
            w0 = tf.get_variable(name='bias', shape=[1], dtype=tf.float32)
            linear_w = [tf.get_variable(name='linear_w_%d'%i, shape=[self.field_lens[i]], dtype=tf.
            linear_part = w0 + tf.reduce_sum(
                tf.concat([tf.reduce_sum(tf.multiply(self.x[i], linear_w[i]), axis=1, keep_dims=Tru
                axis=1, keep_dims=True) # (batch, 1)
        with tf.variable_scope('emb_part'):
            emb = [tf.get_variable(name='emb_%d'%i, shape=[self.field_lens[i], self.vec_dim], dtype
            emb_layer = tf.stack([tf.matmul(self.x[i], emb[i]) for i in range(self.field_num)], axi

        with tf.variable_scope('pair_wise_interaction_part'):
            pi_embedding = []
            for i in range(self.field_num):
```

```
            for j in range(i+1, self.field_num):
                pi_embedding.append(tf.multiply(emb_layer[:,i,:], emb_layer[:,j,:])) # [(batch,
        pi_embedding = tf.stack(pi_embedding, axis=1) # (batch, F*(F-1)/2, K)
        cross_num = self.field_num * (self.field_num - 1) / 2

    with tf.variable_scope('attention_network'):
        # (K, t)
        att_w = tf.get_variable(name='attention_w', shape=[self.vec_dim, self.attention_factor]
        att_b = tf.get_variable(name='attention_b', shape=[self.attention_factor], dtype=tf.flo
        att_h = tf.get_variable(name='attention_h', shape=[self.attention_factor, 1], dtype=tf.
        # wx+b
        attention = tf.matmul(tf.reshape(pi_embedding, shape=(-1, self.vec_dim)), att_w)+att_b
        # relu(wx+b)
        attention = tf.nn.relu(attention)
        # h^T(relu(wx+b))
        attention = tf.reshape(tf.matmul(attention, att_h), shape=(-1, cross_num)) # (batch, F*
        # softmax
        attention_score = tf.nn.softmax(attention) # (batch, F*(F-1)/2)
        attention_score = tf.reshape(attention_score, shape=(-1, cross_num, 1)) # (batch, F*(F-

    with tf.variable_scope('prediction_score'):
        weight_sum = tf.multiply(pi_embedding, attention_score) # (batch, F*(F-1)/2, K)
        weight_sum = tf.reduce_sum(weight_sum, axis=1) # (batch, K)
        weight_sum = tf.layers.dropout(weight_sum, rate=self.dropout_rate, training=self.is_tra
        p = tf.get_variable(name='p', shape=[self.vec_dim, 1], dtype=tf.float32)
        pred_score = tf.matmul(weight_sum, p) # (batch, 1)

    self.y_logits = linear_part + pred_score
    self.y_hat = tf.nn.sigmoid(self.y_logits)
    self.pred_label = tf.cast(self.y_hat > 0.5, tf.int32)
    self.loss = -tf.reduce_mean(self.y*tf.log(self.y_hat+1e-8) + (1-self.y)*tf.log(1-self.y_hat
    reg_variables = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
    if len(reg_variables) > 0:
        self.loss += tf.add_n(reg_variables)
    self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)
```

reference

[1] Xiao, Jun, et al. "Attentional factorization machines: Learning the weight of feature interactions via attention networks." *arXiv preprint arXiv:1708.04617* (2017).

[2] He, Xiangnan, and Tat-Seng Chua. "Neural factorization machines for sparse predictive analytics." *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2017.

[3] https://github.com/hexiangnan/attentional_factorization_machine

专注知识分享，欢迎关注 SOTA Lab~

如果这篇文章对你有帮助，可以点击下方“在看”，推荐给更多小伙伴✌

阅读原文