计算广告 CTR 预估系列(二) - DeepFM 实践

0. 变量说明

为了方便后面的阅读，我们先说明一下各个名称的含义：

1. field_size: 输入 X 在进行 one-hot 之前的特征维度

2. feature_size: 输入 X 在 one-hot 之后的特征维度， 又记作 n

3. embedding_size: one-hot 后的输入，进行嵌入后的维度，又记作 k

4. 代码中 tf 中维度 None 表示任意维度，我们用来表示输入样本的数量这一维度。

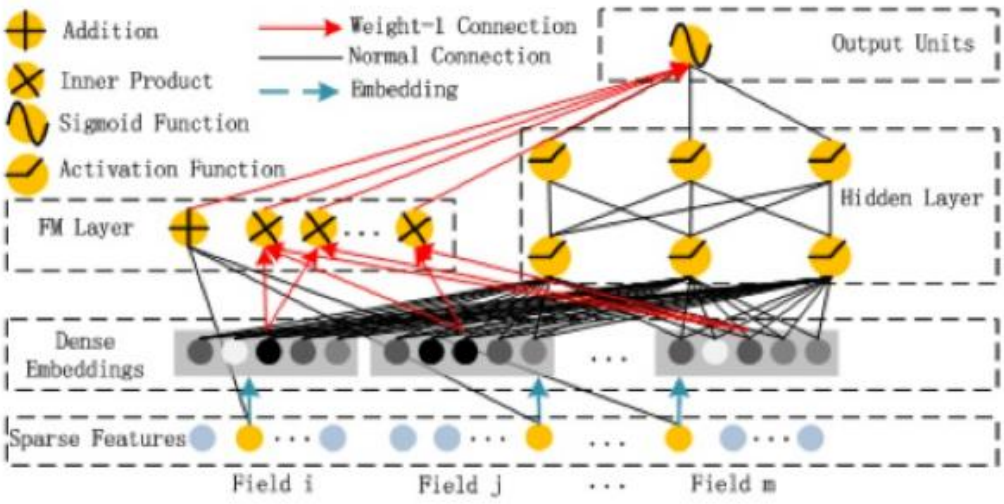
1. 架构图与公式

1.1 架构图

让我们来先回顾一下架构图和公式：

架构图包含两部分：FM Component 和 Deep Component。

其中 FM 部分用于对 1 维特征和 2 维组合特征进行建模；Deep 部分用于对高维组合特征进行建模。



1.2 公式

1.2.1 公式参考

DeepFM 最后输出公式为：

$$\hat{y} = sigmoid(y_{FM} + y_{DNN})$$

其中 FM 部分贡献为：

$$y_{FM} = \langle w, x \rangle + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle V_i, V_j \rangle x_{j_1} \cdot x_{j_2}, \qquad (2)$$

需要注意的点：

1. 这里省去了原始 FM 中的常数项，为了方便。

2. 这里的 x 可以认为是一个样本，d 是 one-hot 之后的总维度，也就是 feature_size. 在原始 FM 论文中对应 n 参数。

Deep 部分贡献为：

这个很好理解，就是神经网络的输出而已，就不再给公式了。

FM 论文中原始公式，方便对比参考：

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

二阶项的化简结果：

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j \\
=&\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i \\
=&\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i x_j - \sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i x_i\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\, x_j\right) - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2 - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)
\end{aligned}
$$

1.2.2 FM Component 维度问题

先说结论：

- FM-1 维：field_size
- FM-2 维组合特征： embedding_size
- 这个和上面的公式有点不一样，但是实际代码实现是这样实现的。

这里比较重要：因为关系到代码里面怎么写，下面分别解释：

先看 FM-1 次项：

首先，我们看到一次项是求和了的。但是在实际代码中，我们并没有对一次项求和。而是输出一个维度为 k 的向量，这里的 k 就是 embedding_size。

为什么要这么做那？我个人的感觉是：提高最后模型的学习效果。

最后神经网络的输出其实可以看做是 logistic regression。它的输入由三部分组成：FM-1 维，FM-二维，Deep 部分。 自然而然，我们不希望 FM-1 维只是一个标量，一个数吧。这显然不利于 LR 模型的学习，那么如果用原始的维度那：公式里是 Wi * Xi，Xi 的维度是 n，n 是 one-hot 之后的维度，这个维度太大了以至于我们才想出了各种办法来解决这个问题，用 n 显然不行。所以，就用 field_size。从逻辑上来说，也是对 one-hot 之前每个特征维度的一种建模。

再看 FM-2 维组合特征部分：

在上节中，我们看到了 FM 二阶项的化简结果。最外层是在 Embeeding_size 上的求和。不做这个求和，而是得到一个 embedding_size 维度的向量，就是送到最后输出单元的 FM 的二阶部分。

原因我想跟上面 FM-1 维的是一样的，求和之后就变成了一个标量，显然不利于后面的学习。

这两部分理解了之后，让我们来看下代码吧！

2. 核心代码拆解

2.1 输入

```
feat_index = tf.placeholder(dtype=tf.int32, shape=[None, config.field_size], name='feat_index')
# [None, field_size]
feat_value = tf.placeholder(dtype=tf.float32, shape=[None, None], name='feat_value')
 # [None, field_size]
label = tf.placeholder(dtype=tf.float16, shape=[None,1], name='label')
```

注意下各个输入变量的维度大小就可以了，没什么特别需要说明的。None 在 tf 里面表示任意维度，此处表示样本数量维度。

2.2 Embedding

```
# Sparse Features -> Dense Embedding
embeddings_origin = tf.nn.embedding_lookup(weights['feature_embedding'], ids=feat_index) # [None, field_size, embedding_size]
```

重点来了，这里是完成 Sparse Features 到 Dense Embedding 的转换。

2.3 FM Component - 1 维特征

1 维特征本来是求和得到一个标量的，为了提高学习效果，我们改为不求和，得到一个 field_size 维度的向量。相当于是进行了一次 k=1 的一次 embedding。

```
y_first_order = tf.nn.embedding_lookup(weights['feature_bias'], ids=feat_index)
# [None, field_size, 1]
```

得到 W 之后，和输入 X 相乘，并缩减维度，得到最终 FM-1 维的输出：

```
w_mul_x = tf.multiply(y_first_order, feat_value_reshape) # [None, field_size, 1] Wi * Xi
y_first_order = tf.reduce_sum(input_tensor=w_mul_x, axis=2) # [None, field_size]
```

2.4 FM Component - 2 维组合特征

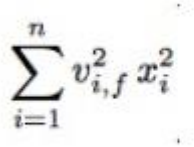
在第一节中，我们给出了 FM-2 维组合特征的化简公式。发现计算的两部分都需要计算 vi * xi。所以我们先把这一部分计算出来：

```
feat_value_reshape = tf.reshape(tensor=feat_value, shape=[-1, config.field_size, 1]) #
-1 * field_size * 1
embeddings = tf.multiply(embeddings_origin, feat_value_reshape)
# [None, field_size, embedding_size] multiply 不是矩阵相乘，而是矩阵对应位置相乘。这里应用了 broadcast 机制。
```

然后分别计算这两部分：

$$\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2$$

```
// sum_square part 先 sum，再 square
summed_features_emb = tf.reduce_sum(input_tensor=embeddings, axis=1) # [None,
embedding_size]
summed_features_emb_square = tf.square(summed_features_emb)
```



// square_sum part

squared_features_emb = tf.square(embeddings)

squared_features_emb_summed = tf.reduce_sum(input_tensor=squared_features_emb, axis=1) # [None, embedding_size]

最后得到最终 FM-2 维组合特征输出结果，维度为 embedding_size:

// second order

y_second_order = 0.5 * tf.subtract(summed_features_emb_square, squared_features_emb_summed)

2.5 Deep Component

网络部分比较简单，只要一层一层的前向传递就可以了。只有一个问题需要说明：网络部分，第一个隐藏层的输入是什么？

我的感觉应该是原始输入嵌入后的结果：

// Deep Component

y_deep = tf.reshape(embeddings_origin, shape=[-1, config.field_size * config.embedding_size]) # [None, field_size * embedding_size]

for i in range(0, len(deep_layers)):

    y_deep = tf.add(tf.matmul(y_deep, weights['layer_%d' % i]), weights['bias_%d' % i])

    y_deep = config.deep_layers_activation(y_deep)

2.6 输出

把前面 FM Component 和 Deep Component 的两部分结合起来，就得到了最后输出单元的输入，经过 sigmoid 函数激活就可以得到最终结果了。

```
// output
concat_input = tf.concat([y_first_order, y_second_order, y_deep], axis=1)
out = tf.add(tf.matmul(concat_input, weights['concat_projection']), weights['concat_bias'])
out = tf.nn.sigmoid(out)
```

3. 完整代码：

这份代码最主要的目的是学习，直接应用于工程的话还需要做一些优化。比如 batch normalization、stack train、batch train 以及代码重构等。

本代码可以帮助你快速实验，实现 DeepFM，掌握其原理。

另外，可直接运行。github 地址 - 欢迎 follow/star/contribute

另外，附上一份整理的 DeepFM 架构图-实现篇 的图。主要是帮助大家理解各个参数的维度、权重 weights 的维度、需要学习的维度、FM Deep 两部分输出的维度。 要想实现 DeepFM，只要把每一部分的维度搞清楚，网络的架构搞清楚就问题不大了。

https://github.com/gutouyu/ML_CIA/blob/master/DeepFM/DeepFM.py

```
import gc
import numpy as np
import pandas as pd

import tensorflow as tf


#####################################
```

```
# 0. Functions
#################################
class Config(object):
"""
用来存储一些配置信息
"""
def __init__(self):
self.feature_dict = None
self.feature_size = None
self.field_size = None
self.embedding_size = 8

self.epochs = 20
self.deep_layers_activation = tf.nn.relu

self.loss = "logloss"
self.l2_reg = 0.1
self.learning_rate = 0.1

def FeatureDictionary(dfTrain=None, dfTest=None, numeric_cols=None,
ignore_cols=None):
"""
目的是给每一个特征维度都进行编号。
1. 对于离散特征，one-hot 之后每一列都是一个新的特征维度。所以，原来的一维度对应的是很多维度，编号也是不同的。
2. 对于连续特征，原来的一维特征依旧是一维特征。

返回一个 feat_dict，用于根据 原特征名称和特征取值 快速查询出 对应的特征编号。

:param dfTrain: 原始训练集
:param dfTest: 原始测试集
```

```
:param numeric_cols: 所有数值型特征

:param ignore_cols: 所有忽略的特征。除了数值型和忽略的，剩下的全部认为是离散型

:return: feat_dict, feat_size

1. feat_size: one-hot 之后总的特征维度。

2. feat_dict 是一个{}， key 是特征 string 的 col_name, value 可能是编号（int），可能也是一个字典。
如果原特征是连续特征： value 就是 int，表示对应的特征编号；

如果原特征是离散特征：value 就是 dict，里面是根据离散特征的 实际取值 查询 该维度的特征编号。 因为离散特征 one-hot 之后，一个取值就是一个维度，
而一个维度就对应一个编号。

"""

assert not (dfTrain is None), "train dataset is not set"

assert not (dfTest is None), "test dataset is not set"


# 编号肯定是要 train test 一起编号的

df = pd.concat([dfTrain, dfTest], axis=0)


# 返回值

feat_dict = {}


# 目前为止的下一个编号

total_cnt = 0


for col in df.columns:

if col in ignore_cols: # 忽略的特征不参与编号

continue


# 连续特征只有一个编号

if col in numeric_cols:

feat_dict[col] = total_cnt

total_cnt += 1
```

```
continue


# 离散特征，有多少个取值就有多少个编号

unique_vals = df[col].unique()

unique_cnt = df[col].nunique()

feat_dict[col] = dict(zip(unique_vals, range(total_cnt, total_cnt +
unique_cnt)))
total_cnt += unique_cnt


feat_size = total_cnt

return feat_dict, feat_size


def parse(feat_dict=None, df=None, has_label=False):

"""

构造 FeatureDict，用于后面 Embedding

:param feat_dict: FeatureDictionary 生成的。用于根据 col 和 value 查询出特征编号的
字典
:param df: 数据输入。可以是 train 也可以是 test,不用拼接

:param has_label: 数据中是否包含 label

:return: Xi, Xv, y

"""

assert not (df is None), "df is not set"


dfi = df.copy()


if has_label:

y = df['target'].values.tolist()

dfi.drop(['id','target'],axis=1, inplace=True)

else:

ids = dfi['id'].values.tolist() # 预测样本的 ids

dfi.drop(['id'],axis=1, inplace=True)
```

```
# dfi 是 Feature index,大小和 dfTrain 相同，但是里面的值都是特征对应的编号。
# dfv 是 Feature value，可以是 binary(0 或 1)，也可以是实值 float，比如 3.14
dfv = dfi.copy()

for col in dfi.columns:
if col in IGNORE_FEATURES: # 用到的全局变量： IGNORE_FEATURES, NUMERIC_FEATURES
dfi.drop([col], axis=1, inplace=True)
dfv.drop([col], axis=1, inplace=True)
continue

if col in NUMERIC_FEATURES: # 连续特征 1 个维度，对应 1 个编号，这个编号是一个定值
dfi[col] = feat_dict[col]
else:
# 离散特征。不同取值对应不同的特征维度，编号也是不同的。
dfi[col] = dfi[col].map(feat_dict[col])
dfv[col] = 1.0

# 取出里面的值
Xi = dfi.values.tolist()
Xv = dfv.values.tolist()

del dfi, dfv
gc.collect()

if has_label:
return Xi, Xv, y
else:
return Xi, Xv, ids
```

```
##################################
# 1. 配置信息
##################################

train_file = "./data/train.csv"
test_file = "./data/test.csv"

IGNORE_FEATURES = [
'id', 'target'
]
CATEGORITAL_FEATURES = [
'feat_cat_1', 'feat_cat_2'
]
NUMERIC_FEATURES = [
'feat_num_1', 'feat_num_2'
]

config = Config()

##################################
# 2. 读取文件
##################################
dfTrain = pd.read_csv(train_file)
dfTest = pd.read_csv(test_file)


##################################
# 3. 准备数据
```

```
###################################

# FeatureDict
config.feature_dict, config.feature_size = FeatureDictionary(dfTrain=dfTrain,
dfTest=dfTest, numeric_cols=NUMERIC_FEATURES, ignore_cols=IGNORE_FEATURES)

# Xi, Xv
Xi_train, Xv_train, y = parse(feat_dict=config.feature_dict, df=dfTrain,
has_label=True)
Xi_test, Xv_test, ids = parse(feat_dict=config.feature_dict, df=dfTest,
has_label=False)
config.field_size = len(Xi_train[0])




###################################
# 4. 建立模型
###################################

# 模型参数
deep_layers = [32,32]
config.embedding_size = 8
config.deep_layers_activation = tf.nn.relu

# BUILD THE WHOLE MODEL
tf.set_random_seed(2018)



# init_weight
weights = dict()
# Sparse Features 到 Dense Embedding 的全连接权重。[其实是 Embedding]
```

```
weights['feature_embedding'] =
tf.Variable(initial_value=tf.random_normal(shape=[config.feature_size,
config.embedding_size],mean=0,stddev=0.1),
name='feature_embedding',

dtype=tf.float32)

# Sparse Featues 到 FM Layer 中 Addition Unit 的全连接。 [其实是 Embedding,嵌入后
维度为 1]
weights['feature_bias'] =
tf.Variable(initial_value=tf.random_uniform(shape=[config.feature_size,
1],minval=0.0,maxval=1.0),
name='feature_bias',

dtype=tf.float32)

# Hidden Layer

num_layer = len(deep_layers)

input_size = config.field_size * config.embedding_size

glorot = np.sqrt(2.0 / (input_size + deep_layers[0])) # glorot_normal: stddev
= sqrt(2/(fan_in + fan_out))
weights['layer_0'] =
tf.Variable(initial_value=tf.random_normal(shape=[input_size,
deep_layers[0]],mean=0,stddev=glorot),
dtype=tf.float32)

weights['bias_0'] = tf.Variable(initial_value=tf.random_normal(shape=[1,
deep_layers[0]],mean=0,stddev=glorot),
dtype=tf.float32)

for i in range(1, num_layer):

glorot = np.sqrt(2.0 / (deep_layers[i - 1] + deep_layers[i]))

# deep_layer[i-1] * deep_layer[i]

weights['layer_%d' % i] =
tf.Variable(initial_value=tf.random_normal(shape=[deep_layers[i - 1],
deep_layers[i]],mean=0,stddev=glorot),
dtype=tf.float32)

# 1 * deep_layer[i]

weights['bias_%d' % i] = tf.Variable(initial_value=tf.random_normal(shape=[1,
deep_layers[i]],mean=0,stddev=glorot),
dtype=tf.float32)

# Output Layer
```

```
deep_size = deep_layers[-1]

fm_size = config.field_size + config.embedding_size

input_size = fm_size + deep_size

glorot = np.sqrt(2.0 / (input_size + 1))

weights['concat_projection'] =
tf.Variable(initial_value=tf.random_normal(shape=[input_size,1],mean=0,stdde
v=glorot),
dtype=tf.float32)

weights['concat_bias'] = tf.Variable(tf.constant(value=0.01),
dtype=tf.float32)



# build_network

feat_index = tf.placeholder(dtype=tf.int32, shape=[None, config.field_size],
name='feat_index') # [None, field_size]
feat_value = tf.placeholder(dtype=tf.float32, shape=[None, None],
name='feat_value') # [None, field_size]
label = tf.placeholder(dtype=tf.float16, shape=[None,1], name='label')


# Sparse Features -> Dense Embedding

embeddings_origin = tf.nn.embedding_lookup(weights['feature_embedding'],
ids=feat_index) # [None, field_size, embedding_size]


feat_value_reshape = tf.reshape(tensor=feat_value, shape=[-1,
config.field_size, 1]) # -1 * field_size * 1


# --------- 一维特征 -----------

y_first_order = tf.nn.embedding_lookup(weights['feature_bias'],
ids=feat_index) # [None, field_size, 1]
w_mul_x = tf.multiply(y_first_order, feat_value_reshape) # [None, field_size,
1] Wi * Xi
y_first_order = tf.reduce_sum(input_tensor=w_mul_x, axis=2) # [None, field_size]


# --------- 二维组合特征 ----------

embeddings = tf.multiply(embeddings_origin, feat_value_reshape) # [None,
field_size, embedding_size] multiply 不是矩阵相乘，而是矩阵对应位置相乘。这里应用
```

了 broadcast 机制。

```
# sum_square part 先 sum，再 square
summed_features_emb = tf.reduce_sum(input_tensor=embeddings, axis=1) # [None, embedding_size]
summed_features_emb_square = tf.square(summed_features_emb)

# square_sum part
squared_features_emb = tf.square(embeddings)
squared_features_emb_summed = tf.reduce_sum(input_tensor=squared_features_emb, axis=1) # [None, embedding_size]

# second order
y_second_order = 0.5 * tf.subtract(summed_features_emb_square, squared_features_emb_summed)



# ----------- Deep Component ------------
y_deep = tf.reshape(embeddings_origin, shape=[-1, config.field_size * config.embedding_size]) # [None, field_size * embedding_size]
for i in range(0, len(deep_layers)):
    y_deep = tf.add(tf.matmul(y_deep, weights['layer_%d' % i]), weights['bias_%d' % i])
    y_deep = config.deep_layers_activation(y_deep)

# ----------- output -----------
concat_input = tf.concat([y_first_order, y_second_order, y_deep], axis=1)
out = tf.add(tf.matmul(concat_input, weights['concat_projection']), weights['concat_bias'])
out = tf.nn.sigmoid(out)

config.loss = "logloss"
config.l2_reg = 0.1
```

```
config.learning_rate = 0.1


# loss
if config.loss == "logloss":
loss = tf.losses.log_loss(label, out)
elif config.loss == "mse":
loss = tf.losses.mean_squared_error(label, out)


# l2
if config.l2_reg > 0:
loss +=
tf.contrib.layers.l2_regularizer(config.l2_reg)(weights['concat_projection']
)
for i in range(len(deep_layers)):
loss += tf.contrib.layers.l2_regularizer(config.l2_reg)(weights['layer_%d' %
i])


# optimizer
optimizer = tf.train.AdamOptimizer(learning_rate=config.learning_rate,
beta1=0.9, beta2=0.999, epsilon=1e-8).minimize(loss)


###################################
# 5. 训练
###################################


# init session
sess = tf.Session(graph=tf.get_default_graph())
sess.run(tf.global_variables_initializer())


# train
feed_dict = {
feat_index: Xi_train,
```

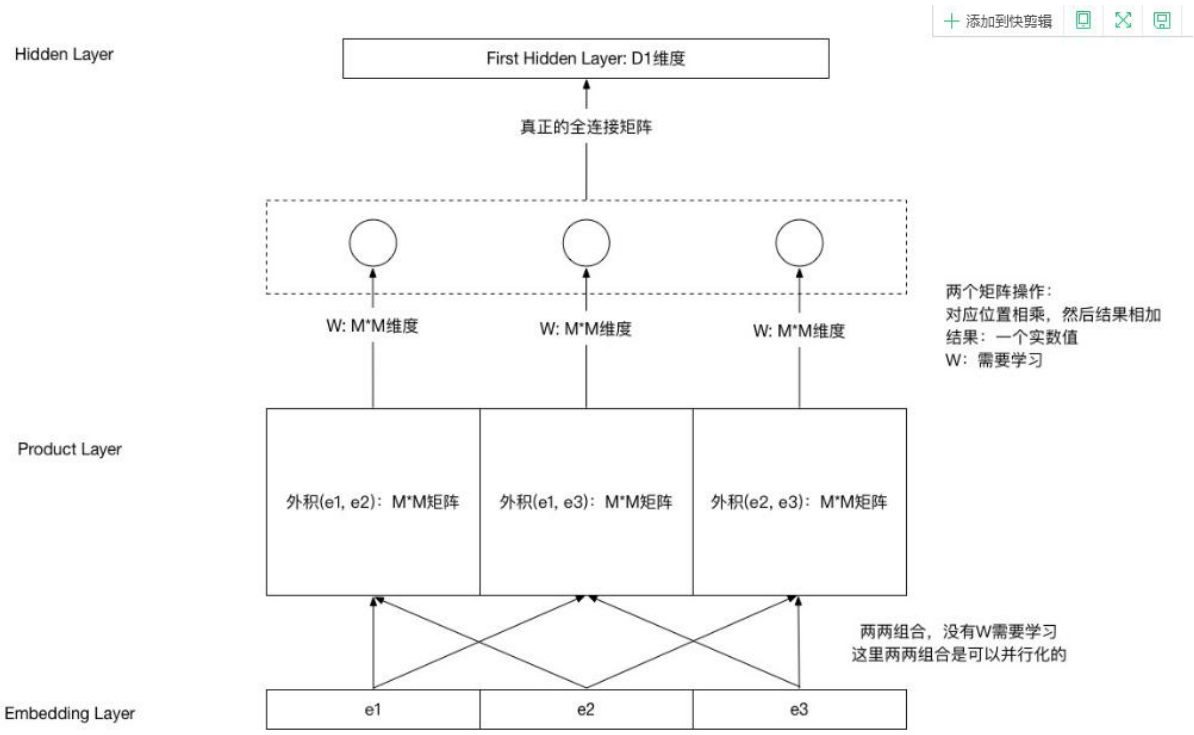
```
feat_value: Xv_train,
label: np.array(y).reshape((-1,1))
}

for epoch in range(config.epochs):
train_loss,opt = sess.run((loss, optimizer), feed_dict=feed_dict)
print("epoch: {0}, train loss: {1:.6f}".format(epoch, train_loss))

###################################
# 6. 预测
###################################
dummy_y = [1] * len(Xi_test)
feed_dict_test = {
feat_index: Xi_test,
feat_value: Xv_test,
label: np.array(dummy_y).reshape((-1,1))
}

prediction = sess.run(out, feed_dict=feed_dict_test)

sub = pd.DataFrame({"id":ids, "pred":np.squeeze(prediction)})
print("prediction:")
print(sub)
```



外积 p 部分架构逻辑说明