支持多值带权重、稀疏、共享embedding权重的DSSM召回实现（tensorflow2）

原创 xulu1352 数据与智能 今天

收录于话题

#推荐算法 3 #推荐系统 3 #TensorFlow 1 #深度学习 2 #召回 1

作者 | xulu1352 目前在一家互联网公司从事推荐算法工作

编辑 | lily

0

前序

关于DSSM模型原理及实现，网上已经有很多质量不错的参考文章了，比如王多鱼的实践DSSM召回（如果对dssm模型原理不熟，建议先阅读这篇文章，再看本文实践部分，本文主要讲实现），总结的非常不错，王多鱼这篇文章DSSM实践是基于浅梦大佬开源deepmatch包实现的，但是在推荐系统实践中如果直接调用别人的模型包会遇到诸多不便，需要在自己业务场景中做finetune；实际生产中，模型所用到的特征往往都是稀疏的，多值变长的，对有些特征我们还想让它们共享embedding，说到这里，我要非常感谢石塔西的这篇文章用TensorFlow实现支持多值、稀疏、共享权重的DeepFM，从这篇文章中，我得到很多启发;本文下面介绍的主要是自己从各位大佬那学习到的知识总结，并无什么创新点，希望对一些刚入坑的童鞋们有所帮助。好了，那我们开始，Talk is cheap, Show me the code.

虽然本文不讲模型原理，但是有两Tricks，还是值得提下，亲测有效，这两tricks在下文实现均有体现。

1 对user及item embedding向量 L2标准化

$$u(x,\theta) \leftarrow u(x,\theta)/||u(x,\theta)||_2, \quad v(x,\theta) \leftarrow v(x,\theta)/||v(x,\theta)||_2$$

Emebdding标准化可以加速模型训练和提升检索效果。

2 增强Softmax效果

通过引入超参数 τ 来增强softmax每个逻辑值的输出：

$$s(x,y) = < u(x,\theta), v(y,\theta) > /\tau$$

微调超参数 τ 可以最大化召回率或精确率。

1

数据预处理

为了逼近真实推荐系统场景的数据处理，这里人为构造部分实际生产数据样例作为演示；

```
data.head()
```

	act	client_id	post_id	client_type	follow_topic_id	all_topic_fav_7	topic_id	read_post_id
0	1	28401	39647119	0	572,92,62,37,35,34,33,32,31,30,29,68,67,65,24,...	502:0.3443,278:0.0868,177:0.0719,1:0.497	135	39588887,39599018,39576294,39553374,39630091
1	1	28401	39645671	0	572,92,62,37,35,34,33,32,31,30,29,68,67,65,24,...	502:0.3443,278:0.0868,177:0.0719,1:0.497	1	39588887,39599018,39576294,39553374,39630091
2	0	28401	39643183	0	572,92,62,37,35,34,33,32,31,30,29,68,67,65,24,...	502:0.3443,278:0.0868,177:0.0719,1:0.497	3	39588887,39599018,39576294,39553374,39630091
3	0	28401	39629847	0	572,92,62,37,35,34,33,32,31,30,29,68,67,65,24,...	502:0.3443,278:0.0868,177:0.0719,1:0.497	4	39588887,39599018,39576294,39553374,39630091
4	1	28401	39613538	0	572,92,62,37,35,34,33,32,31,30,29,68,67,65,24,...	502:0.3443,278:0.0868,177:0.0719,1:0.497	278	39588887,39599018,39576294,39553374,39630091

样本

字段介绍：

act：为label数据 1:正样本，0：负样本

client_id: 用户id

post_id：物料item id 这里称为post_id

client_type:用户客户端类型

follow_topic_id: 用户关注话题分类id

all_topic_fav_7: 用户画像特征，用户最近7天对话题偏爱度刻画，kv键值对形式

topic_id: 物料所属的话题

read_post_id:用户最近阅读的物料id

预训练item embedding 向量

```
ITEM_EMBEDDING
```

```
<tf.Tensor: shape=(112396, 768), dtype=float32, numpy=
array([[ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
         0.       ],
       [ 0.999357,  0.999952,  0.997325, ..., -0.997149, -0.999041,
        -0.149874],
       [ 0.999701,  0.999996,  0.475305, ..., -0.996938, -0.999869,
         0.479093],
       ...,
       [ 0.999113,  0.999849,  0.961474, ..., -0.997135, -0.997072,
         0.113319],
       [ 0.999691,  0.999915,  0.978405, ..., -0.999732, -0.996863,
         0.813737],
       [ 0.999831,  0.999961,  0.999492, ..., -0.99991 , -0.999621,
        -0.260133]], dtype=float32)>
```

这里会有为每个item预训练生成一个embedding向量，存到embedding矩阵中，idx=0行，为一个默认值，当一个item因某些原因未生成其embedding向量，则用默认值0替代。

定义参数类型

我们将参数归三种类型单值离散型SparseFeat，如topic_id字段；稠密数值类型DenseFeat，如用户访问时间及用户embedding向量等；多值变长离散特征VarLenSparseFeat，如follow_topic_id或者带权重形式all_topic_fav_7；这里延用deepMatch开源包里定义输入变量方式, 需要注意的是，SparseFeat与VarLenSparseFeat类型的特征，如果想共享embedding权重向量，需要指定其与哪个category离散变量特征embedding参数共享，如这里我们想follow_topic_id与all_topic_fav_7里的id embedding与item的topic_id embedding权重共享一套，设置share_embed='topic_id'即可。

```
from collections import namedtuple, OrderedDict
import tensorflow as tf

SparseFeat = namedtuple('SparseFeat', ['name', 'voc_size', 'share_embed','emb
DenseFeat = namedtuple('DenseFeat', ['name', 'pre_embed','reduce_type','dim',
VarLenSparseFeat = namedtuple('VarLenSparseFeat', ['name', 'voc_size', 'share
import tensorflow as tf

SparseFeat = namedtuple('SparseFeat', ['name', 'voc_size', 'share_embed','emb
DenseFeat = namedtuple('DenseFeat', ['name', 'pre_embed','reduce_type','dim',
VarLenSparseFeat = namedtuple('VarLenSparseFeat', ['name', 'voc_size', 'share
```

定义DSSM输入变量参数

除了常见的特征，这里使用用户最近浏览的物料embedding向量的平均作为用户的一个特征即client_embed；我们将follow_topic_id，all_topic_fav7用到的topic_id embedding向量与item的topic_id对应的embedding向量共享，在实际应用中，相近语义的embedding权重共享是很有必要的，大大减少网络训练参数，防止过拟合。

```
feature_columns = [SparseFeat(name="topic_id", voc_size=700, share_embed=None
                   SparseFeat(name='client_type', voc_size=2, share_embed=No
                   VarLenSparseFeat(name="follow_topic_id", voc_size=700, sh
                   VarLenSparseFeat(name="all_topic_fav_7", voc_size=700, sh
                   DenseFeat(name='item_embed',pre_embed='post_id', reduce_t
                   DenseFeat(name='client_embed',pre_embed='read_post_id', r
                  ]

```

```

# 用户特征及贴子特征
user_feature_columns_name = ["follow_topic_id", 'all_topic_fav_7','client_typ
item_feature_columns_name = ["topic_id", 'post_type','item_embed',]
user_feature_columns = [col for col in feature_columns if col.name in user_fe
item_feature_columns = [col for col in feature_columns if col.name in item_fe
```

构造训练tf.dataset数据

首先加载预训练 item embedding向量及离散特征vocabulary

```
def get_item_embed(file_names):
    item_bert_embed = []
    item_id = []
    for file in file_names:
        with open(file, 'r') as f:
            for line in f:
                feature_json = json.loads(line)
                item_bert_embed.append(feature_json['post_id'])
                item_id.append(feature_json['values'])

    item_id2idx = tf.lookup.StaticHashTable(
        tf.lookup.KeyValueTensorInitializer(
            keys=item_id,
            values=range(1, len(item_id)+1),
            key_dtype=tf.string,
            value_dtype=tf.int32),
            default_value=0)
    item_bert_embed = [[0.0]*768] + item_bert_embed
    item_embedding = tf.constant(item_bert_embed, dtype=tf.float32)
    return item_id2idx, item_embedding
# 获取item embedding及其查找关系
ITEM_ID2IDX, ITEM_EMBEDDING = get_item_embed(file_names)

# 定义离散特征集合 ，离散特征vocabulary
DICT_CATEGORICAL = {"topic_id": [str(i) for i in range(0, 700)],
            "client_type": [0,1]
           }
```

然后，tf.dataset构造

```
DEFAULT_VALUES = [[0],[''],[''],[0.0], [''], [''], [''],['']]
COL_NAME = ['act', 'client_id', 'post_id', 'client_type', 'follow_topic_id'

def _parse_function(example_proto):

    item_feats = tf.io.decode_csv(example_proto, record_defaults=DEFAULT_VAL
    parsed = dict(zip(COL_NAME, item_feats))

    feature_dict = {}
    for feat_col in feature_columns:
        if isinstance(feat_col, VarLenSparseFeat):
            if feat_col.weight_name is not None:
                kvpairs = tf.strings.split([parsed[feat_col.name]], ',').va
                kvpairs = tf.strings.split(kvpairs, ':')
                kvpairs = kvpairs.to_tensor()
                feat_ids, feat_vals = tf.split(kvpairs, num_or_size_splits=
                feat_vals= tf.strings.to_number(feat_vals, out_type=tf.floa
                feature_dict[feat_col.name] = feat_ids
                feature_dict[feat_col.weight_name] = feat_vals
            else:
                feat_ids = tf.strings.split([parsed[feat_col.name]], ',').va
                feat_ids = tf.reshape(feat_ids, shape=[-1])
                feature_dict[feat_col.name] = feat_ids

        elif isinstance(feat_col, SparseFeat):
            feature_dict[feat_col.name] = parsed[feat_col.name]

        elif isinstance(feat_col, DenseFeat):
            if feat_col.pre_embed is None:
                feature_dict[feat_col.name] = parsed[feat_col.name]
            elif feat_col.reduce_type is not None:
                keys = tf.strings.split(parsed[feat_col.pre_embed], ',')
                emb = tf.nn.embedding_lookup(params=ITEM_EMBEDDING, ids=ITEM
                emb = tf.reduce_mean(emb,axis=0) if feat_col.reduce_type ==
                feature_dict[feat_col.name] = emb
            else:
                emb = tf.nn.embedding_lookup(params=ITEM_EMBEDDING, ids=ITEM
                feature_dict[feat_col.name] = emb
```

```
        else:
            raise "unknown feature_columns...."


    label = parsed['act']


    return feature_dict, label


pad_shapes = {}
pad_values = {}

for feat_col in feature_columns:
    if isinstance(feat_col, VarLenSparseFeat):
        max_tokens = feat_col.maxlen
        pad_shapes[feat_col.name] = tf.TensorShape([max_tokens])
        pad_values[feat_col.name] = ''
        if feat_col.weight_name is not None:
            pad_shapes[feat_col.weight_name] = tf.TensorShape([max_tokens])
            pad_values[feat_col.weight_name] = tf.constant(-1, dtype=tf.floa

# no need to pad labels
    elif isinstance(feat_col, SparseFeat):
        if feat_col.dtype == 'string':
            pad_shapes[feat_col.name] = tf.TensorShape([])
            pad_values[feat_col.name] = '9999'
        else:
            pad_shapes[feat_col.name] = tf.TensorShape([])
            pad_values[feat_col.name] = 0.0
    elif isinstance(feat_col, DenseFeat):
        if feat_col.pre_embed is None:
            pad_shapes[feat_col.name] = tf.TensorShape([])
            pad_values[feat_col.name] = 0.0
        else:
            pad_shapes[feat_col.name] = tf.TensorShape([feat_col.dim])
            pad_values[feat_col.name] = 0.0


pad_shapes = (pad_shapes, (tf.TensorShape([])))
```

```
pad_values = (pad_values, (tf.constant(0, dtype=tf.int32)))


filenames= tf.data.Dataset.list_files([
'/recall_user_item_act.csv'
])
dataset = filenames.flat_map(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1))

batch_size = 1024
dataset = dataset.map(_parse_function, num_parallel_calls=60)
dataset = dataset.repeat()
dataset = dataset.shuffle(buffer_size = batch_size*2) # 在缓冲区中随机打乱数据
dataset = dataset.padded_batch(batch_size = batch_size,
                               padded_shapes = pad_shapes,
                              padding_values = pad_values) # 每1024条数据为一
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

# 验证集
filenames_val= tf.data.Dataset.list_files(['/recall_user_item_act_val.csv']
dataset_val = filenames_val.flat_map(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1))

val_batch_size = 1024
dataset_val = dataset_val.map(_parse_function, num_parallel_calls=60)
dataset_val = dataset_val.padded_batch(batch_size = val_batch_size,
                               padded_shapes = pad_shapes,
                              padding_values = pad_values) # 每1024条数据为一
dataset_val = dataset_val.prefetch(buffer_size=tf.data.experimental.AUTOTUNI
```

经过上述逻辑代码预处理后，原始样本csv文件中数据格式已经转化为如下的形式（这里拿batch_size=1 举例），kv形式的特征被拆分为两个Input输入变量一个是category离散ID（如all_topic_fav_7），一个是其对应的weight（如all_topic_fav_7_weight），他们最终被输入到tf.nn.embedding_lookup_sparse(self.embedding,sp_ids=idx, sp_weights=val, combiner='sum') 这个api 对应的sp_ids，sp_weights参数中去。

```
# next(iter(dataset))
({'topic_id': <tf.Tensor: shape=(1,), dtype=string, numpy=array([b'278'], dty
```

```
 'client_type': <tf.Tensor: shape=(1,), dtype=float32, numpy=array([0.], dty
 'follow_topic_id': <tf.Tensor: shape=(1, 20), dtype=string, numpy=
 array([[b'572', b'92', b'62', b'37', b'35', b'34', b'33', b'32', b'31',
         b'30', b'29', b'68', b'67', b'65', b'24', b'20', b'16', b'15',
         b'13', b'12']], dtype=object)>,
 'all_topic_fav_7': <tf.Tensor: shape=(1, 5), dtype=string, numpy=array([[b
 'all_topic_fav_7_weight': <tf.Tensor: shape=(1, 5), dtype=float32, numpy=ar
 'item_embed': <tf.Tensor: shape=(1, 768), dtype=float32, numpy=
 array([[ 0.999586,  0.999861,  0.995566,  0.892292,  0.848516,  0.815888,
         -0.860286, -0.871219,  0.982316, -0.999692,  0.999998,  0.999589,
                                   ......
         -0.943752,  0.999957, -0.990231,  0.999377, -0.997795,  0.999498,
         -0.995729,  0.701236,  0.991473,  0.946505, -0.996337,  0.999991,
          0.991516, -0.997269, -0.993377, -0.9964  , -0.99972 ,  0.880781]],
       dtype=float32)>,
 'client_embed': <tf.Tensor: shape=(1, 768), dtype=float32, numpy=
 array([[ 0.79698   ,  0.7999152 ,  0.78845704,  0.6598178 ,  0.59617054,
          0.5318628 , -0.5754676 , -0.7469004 ,  0.78916025, -0.7958456 ,
                                ......
          0.7989754 , -0.7971929 , -0.0165708 ,  0.7924882 ,  0.73336124,
         -0.794997  ,  0.7999618 ,  0.7634414 , -0.792517  , -0.762231  ,
         -0.7960204 , -0.7998554 ,  0.37363502]], dtype=float32)>},
<tf.Tensor: shape=(1,), dtype=int32, numpy=array([1], dtype=int32)>)
```

2

自定义模型层

```
# 离散多值查找表 转稀疏SparseTensor >> EncodeMultiEmbedding >>tf.nn.embedding_
class SparseVocabLayer(Layer):
    def __init__(self, keys, **kwargs):
        super(SparseVocabLayer, self).__init__(**kwargs)
        vals = tf.range(1, len(keys) + 1)
        vals = tf.constant(vals, dtype=tf.int32)
```

```python
        keys = tf.constant(keys)
        self.table = tf.lookup.StaticHashTable(
            tf.lookup.KeyValueTensorInitializer(keys, vals), 0)

    def call(self, inputs):
        input_idx = tf.where(tf.not_equal(inputs, ''))
        input_sparse = tf.SparseTensor(input_idx, tf.gather_nd(inputs, inpu
        return tf.SparseTensor(indices=input_sparse.indices,
                               values=self.table.lookup(input_sparse.values),
                               dense_shape=input_sparse.dense_shape)

# 自定义Embedding层，初始化时，需要传入预先定义好的embedding矩阵，好处可以共享embe
class EncodeMultiEmbedding(Layer):
    def __init__(self, embedding, has_weight=False, **kwargs):

        super(EncodeMultiEmbedding, self).__init__(**kwargs)
        self.has_weight = has_weight
        self.embedding = embedding


    def build(self, input_shape):
        super(EncodeMultiEmbedding, self).build(input_shape)

    def call(self, inputs):
        if self.has_weight:
            idx, val = inputs
            combiner_embed = tf.nn.embedding_lookup_sparse(self.embedding,s
        else:
            idx = inputs
            combiner_embed = tf.nn.embedding_lookup_sparse(self.embedding,s
        return tf.expand_dims(combiner_embed, 1)

    def get_config(self):
        config = super(EncodeMultiEmbedding, self).get_config()
        config.update({'has_weight': self.has_weight})
        return config

# 稠密权重转稀疏格式输入到tf.nn.embedding_lookup_sparse的sp_weights参数中
class Dense2SparseTensor(Layer):
    def __init__(self):
```

```
        super(Dense2SparseTensor, self).__init__()

    def call(self, dense_tensor):
        weight_idx = tf.where(tf.not_equal(dense_tensor, tf.constant(-1, dty
        weight_sparse = tf.SparseTensor(weight_idx, tf.gather_nd(dense_tenso
        return weight_sparse

    def get_config(self):
        config = super(Dense2SparseTensor, self).get_config()
        return config


# 自定义dnese层含BN， dropout
class CustomDense(Layer):
    def __init__(self, units=32, activation='tanh', dropout_rate =0, use_bn=
        self.units = units
        self.activation = activation
        self.dropout_rate = dropout_rate
        self.use_bn = use_bn
        self.seed = seed
        self.tag_name = tag_name

        super(CustomDense, self).__init__(**kwargs)

    #build方法一般定义Layer需要被训练的参数。
    def build(self, input_shape):
        self.weight = self.add_weight(shape=(input_shape[-1], self.units),
                                 initializer='random_normal',
                                 trainable=True,
                                 name='kernel_' + self.tag_name)
        self.bias = self.add_weight(shape=(self.units,),
                                 initializer='random_normal',
                                 trainable=True,
                                 name='bias_' + self.tag_name)

        if self.use_bn:
            self.bn_layers = tf.keras.layers.BatchNormalization()

        self.dropout_layers = tf.keras.layers.Dropout(self.dropout_rate)
        self.activation_layers = tf.keras.layers.Activation(self.activation
```

```

        super(CustomDense,self).build(input_shape) # 相当于设置self.built = T

    #call方法一般定义正向传播运算逻辑，__call__方法调用了它。
    def call(self, inputs, training = None, **kwargs):
        fc = tf.matmul(inputs, self.weight) + self.bias
        if self.use_bn:
            fc = self.bn_layers(fc)
        out_fc = self.activation_layers(fc)

        return out_fc

    #如果要让自定义的Layer通过Functional API 组合成模型时可以序列化，需要自定义get
    def get_config(self):
        config = super(CustomDense, self).get_config()
        config.update({'units': self.units, 'activation': self.activation,
                       'dropout_rate': self.dropout_rate, 'seed': self.seed
        return config


# cos 相似度计算层
class Similarity(Layer):

    def __init__(self, gamma=1, axis=-1, type_sim='cos', **kwargs):
        self.gamma = gamma
        self.axis = axis
        self.type_sim = type_sim
        super(Similarity, self).__init__(**kwargs)

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

    def call(self, inputs, **kwargs):
        query, candidate = inputs
        if self.type_sim == "cos":
            query_norm = tf.norm(query, axis=self.axis)
            candidate_norm = tf.norm(candidate, axis=self.axis)
        cosine_score = tf.reduce_sum(tf.multiply(query, candidate), -1)
        cosine_score = tf.divide(cosine_score, query_norm * candidate_norm
```

```
        cosine_score = tf.clip_by_value(cosine_score, -1, 1.0) * self.gamma
        return tf.expand_dims(cosine_score, 1)

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

    def get_config(self, ):
        config = {'gamma': self.gamma, 'axis': self.axis, 'type': self.type_
        base_config = super(Similarity, self).get_config()
        return base_config.uptate(config)


# 自定损失函数，加权交叉熵损失
class WeightedBinaryCrossEntropy(tf.keras.losses.Loss):
    """
    Args:
      pos_weight: Scalar to affect the positive labels of the loss function
      weight: Scalar to affect the entirety of the loss function.
      from_logits: Whether to compute loss from logits or the probability.
      reduction: Type of tf.keras.losses.Reduction to apply to loss.
      name: Name of the loss function.
    """

    def __init__(self, pos_weight=1.2, from_logits=False,
                 reduction=tf.keras.losses.Reduction.AUTO,
                 name='weighted_binary_crossentropy'):
        super().__init__(reduction=reduction, name=name)
        self.pos_weight = pos_weight
        self.from_logits = from_logits

    def call(self, y_true, y_pred):
        y_true = tf.cast(y_true, tf.float32)
        ce = tf.losses.binary_crossentropy(
            y_true, y_pred, from_logits=self.from_logits)[:, None]
        ce = ce * (1 - y_true) + self.pos_weight * ce * (y_true)
#         ce =tf.nn.weighted_cross_entropy_with_logits(
#             y_true, y_pred, self.pos_weight, name=None
#         )

        return ce
```

```python

    def get_config(self, ):
        config = {'pos_weight': self.pos_weight, 'from_logits': self.from_lo
        base_config = super(WeightedBinaryCrossEntropy, self).get_config()
        return base_config.uptate(config)
```

3

定义输入及共享层帮助函数

```python
# 定义model输入特征
def build_input_features(features_columns, prefix=''):
    input_features = OrderedDict()
    for feat_col in features_columns:
        if isinstance(feat_col, DenseFeat):
            if feat_col.pre_embed is None:
                input_features[feat_col.name] = Input([1], name=feat_col.nar
            else:
                input_features[feat_col.name] = Input([feat_col.dim], name=
        elif isinstance(feat_col, SparseFeat):
            if feat_col.dtype == 'string':
                input_features[feat_col.name] = Input([None], name=feat_col
            else:
                input_features[feat_col.name] = Input([1], name=feat_col.nar
        elif isinstance(feat_col, VarLenSparseFeat):
            input_features[feat_col.name] = Input([None], name=feat_col.name
            if feat_col.weight_name is not None:
                input_features[feat_col.weight_name] = Input([None], name=fe
        else:
            raise TypeError("Invalid feature column in build_input_features

    return input_features

# 构造自定义embedding层matrix
```

```
def build_embedding_matrix(features_columns):
    embedding_matrix = {}
    for feat_col in features_columns:
        if isinstance(feat_col, SparseFeat) or isinstance(feat_col, VarLenS
            if feat_col.dtype == 'string':
                vocab_name = feat_col.share_embed if feat_col.share_embed e
                vocab_size = feat_col.voc_size
                embed_dim = feat_col.embed_dim
                if vocab_name not in embedding_matrix:
                    embedding_matrix[vocab_name] = tf.Variable(initial_valu

    return embedding_matrix

# 构造自定义 embedding层
def build_embedding_dict(features_columns, embedding_matrix):
    embedding_dict = {}
    for feat_col in features_columns:
        if isinstance(feat_col, SparseFeat):
            if feat_col.dtype == 'string':
                vocab_name = feat_col.share_embed if feat_col.share_embed e
                embedding_dict[feat_col.name] = EncodeMultiEmbedding(embedd
        elif isinstance(feat_col, VarLenSparseFeat):
            vocab_name = feat_col.share_embed if feat_col.share_embed else
            if feat_col.weight_name is not None:
                embedding_dict[feat_col.name] = EncodeMultiEmbedding(embedd
            else:
                embedding_dict[feat_col.name] = EncodeMultiEmbedding(embedd

    return embedding_dict


# dense 与 embedding特征输入
def input_from_feature_columns(features, features_columns, embedding_dict):
    sparse_embedding_list = []
    dense_value_list = []

    for feat_col in features_columns:
        if isinstance(feat_col, SparseFeat) or isinstance(feat_col, VarLenS
            if feat_col.dtype == 'string':
                vocab_name = feat_col.share_embed if feat_col.share_embed e
```

```
                keys = DICT_CATEGORICAL[vocab_name]
                _input_sparse = SparseVocabLayer(keys)(features[feat_col.nar

        if isinstance(feat_col, SparseFeat):
            if feat_col.dtype == 'string':
                _embed = embedding_dict[feat_col.name](_input_sparse)
            else:
                _embed = Embedding(feat_col.voc_size+1, feat_col.embed_dim,
                               embeddings_regularizer=tf.keras.regularizers
            sparse_embedding_list.append(_embed)
        elif isinstance(feat_col, VarLenSparseFeat):
            if feat_col.weight_name is not None:
                _weight_sparse = Dense2SparseTensor()(features[feat_col.wei
                _embed = embedding_dict[feat_col.name]([_input_sparse, _wei

            else:
                _embed = embedding_dict[feat_col.name](_input_sparse)
            sparse_embedding_list.append(_embed)

        elif isinstance(feat_col, DenseFeat):
            dense_value_list.append(features[feat_col.name])

        else:
            raise TypeError("Invalid feature column in input_from_feature_co

    return sparse_embedding_list, dense_value_list


def concat_func(inputs, axis=-1):
    if len(inputs) == 1:
        return inputs[0]
    else:
        return Concatenate(axis=axis)(inputs)

def combined_dnn_input(sparse_embedding_list, dense_value_list):
    if len(sparse_embedding_list) > 0 and len(dense_value_list) > 0:
        sparse_dnn_input = Flatten()(concat_func(sparse_embedding_list))
        dense_dnn_input = Flatten()(concat_func(dense_value_list))
        return concat_func([sparse_dnn_input, dense_dnn_input])
    elif len(sparse_embedding_list) > 0:
```

```
            return Flatten()(concat_func(sparse_embedding_list))
        elif len(dense_value_list) > 0:
            return Flatten()(concat_func(dense_value_list))
        else:
            raise "dnn_feature_columns can not be empty list"
```

4

搭建DSSM模型

```
def DSSM(
    user_feature_columns,
    item_feature_columns,
    user_dnn_hidden_units=(256, 256, 128),
    item_dnn_hidden_units=(256, 256,  128),
    user_dnn_dropout=(0, 0, 0),
    item_dnn_dropout=(0, 0, 0),
    out_dnn_activation='tanh',
    gamma=1.2,
    dnn_use_bn=False,
    seed=1024,
    metric='cos'):

    """
    Instantiates the Deep Structured Semantic Model architecture.
    Args:
        user_feature_columns: A list containing user's features used by the r
        item_feature_columns: A list containing item's features used by the r
        user_dnn_hidden_units: tuple,tuple of positive integer , the layer nu
        item_dnn_hidden_units: tuple,tuple of positive integer, the layer nur
        out_dnn_activation: Activation function to use in deep net
        dnn_use_bn: bool. Whether use BatchNormalization before activation or
        user_dnn_dropout: tuple of float in [0,1), the probability we will dr
        item_dnn_dropout: tuple of float in [0,1), the probability we will dr
```

```
        seed: integer ,to use as random seed.
        gamma: A useful hyperparameter for Similarity layer
        metric: str, "cos" for  cosine
    return: A TF Keras model instance.
    """
    features_columns = user_feature_columns + item_feature_columns
    # 构建 embedding_dict
    embedding_matrix = build_embedding_matrix(features_columns)
    embedding_dict = build_embedding_dict(features_columns, embedding_matrix)

    # user 特征 处理
    user_features = build_input_features(user_feature_columns)
    user_inputs_list = list(user_features.values())
    user_sparse_embedding_list, user_dense_value_list = input_from_feature_co

    user_dnn_input = combined_dnn_input(user_sparse_embedding_list, user_dens

    # item 特征 处理
    item_features = build_input_features(item_feature_columns)
    item_inputs_list = list(item_features.values())
    item_sparse_embedding_list, item_dense_value_list = input_from_feature_co

    item_dnn_input = combined_dnn_input(item_sparse_embedding_list, item_dens


    # user tower
    for i in range(len(user_dnn_hidden_units)):
        if i == len(user_dnn_hidden_units) - 1:
            user_dnn_out = CustomDense(units=user_dnn_hidden_units[i],dropout
                                       use_bn=dnn_use_bn,activation=out_dnn_a
            break
        user_dnn_input = CustomDense(units=user_dnn_hidden_units[i],dropout_r
                                     use_bn=dnn_use_bn,activation='relu', nar


    # item tower
    for i in range(len(item_dnn_hidden_units)):
        if i == len(item_dnn_hidden_units) - 1:
            item_dnn_out = CustomDense(units=item_dnn_hidden_units[i],dropout
                                       use_bn=dnn_use_bn, activation=out_dnn_
```

```
                break
            item_dnn_input = CustomDense(units=item_dnn_hidden_units[i],dropout_r
                                         use_bn=dnn_use_bn,activation='relu', nar


        score = Similarity(type_sim=metric,gamma=gamma)([user_dnn_out, item_dnn_c
        output = tf.keras.layers.Activation("sigmoid", name="dssm_out")(score)
#     score = Multiply()([user_dnn_out, item_dnn_out])
#     output = Dense(1, activation="sigmoid",name="dssm_out")(score)

        model = Model(inputs=user_inputs_list + item_inputs_list, outputs=output)
        model.__setattr__("user_input", user_inputs_list)
        model.__setattr__("item_input", item_inputs_list)
        model.__setattr__("user_embedding", user_dnn_out)
        model.__setattr__("item_embedding", item_dnn_out)

        return model
```

5

训练及保存模型

训练模型

```
model= DSSM(
        user_feature_columns,
        item_feature_columns,
        user_dnn_hidden_units=(256, 256, 128),
        item_dnn_hidden_units=(256, 256, 128),
        user_dnn_dropout=(0, 0, 0),
        item_dnn_dropout=(0, 0, 0),
        out_dnn_activation='tanh',
        gamma=1,
        dnn_use_bn=False,
        seed=1024,
```

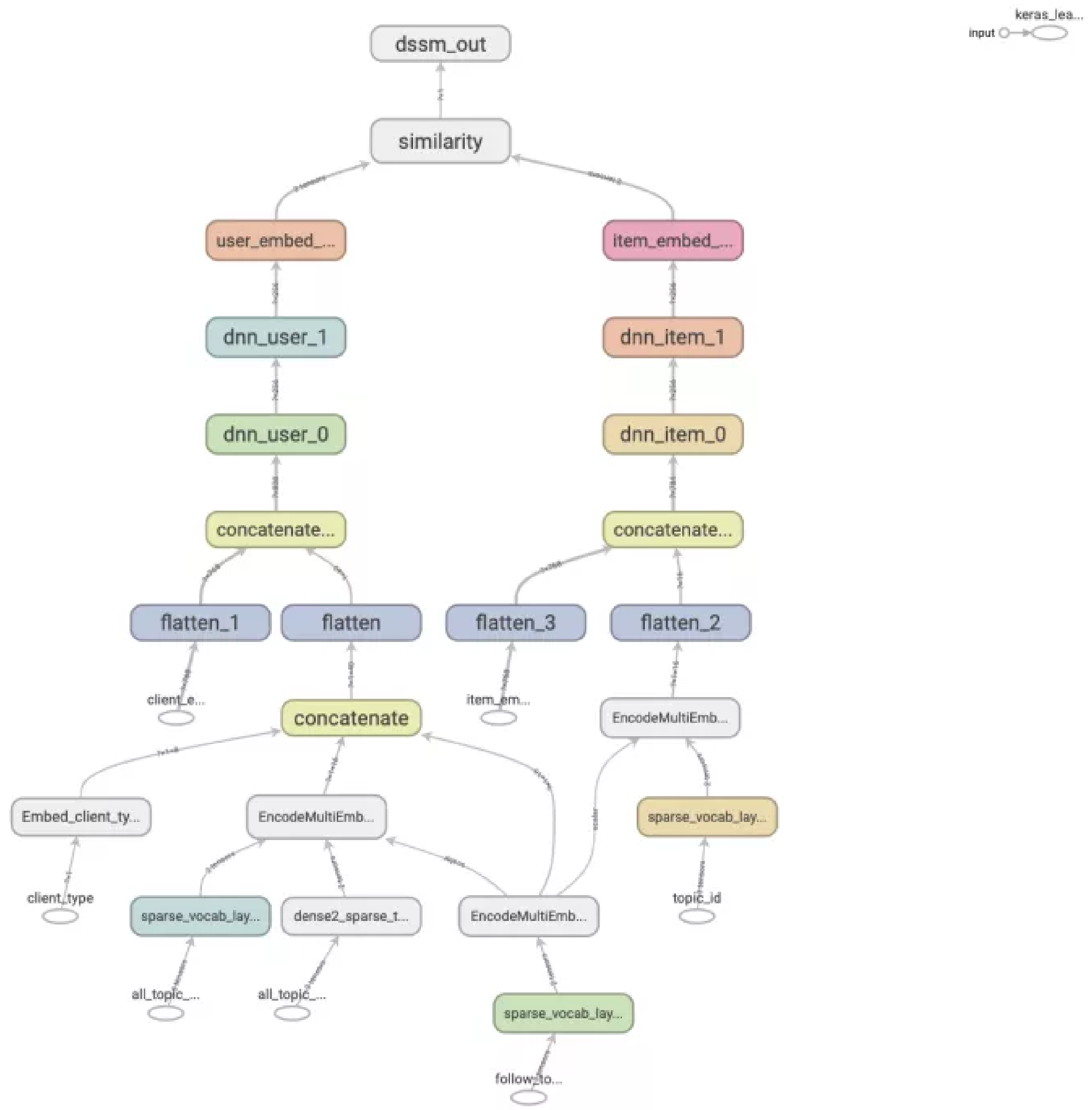
```
    metric='cos')

model.compile(optimizer='adagrad',
              loss={"dssm_out": WeightedBinaryCrossEntropy(),
                   },
              loss_weights=[1.0,],
              metrics={"dssm_out": [tf.keras.metrics.AUC(name='auc')]}
             )


log_dir = '/mywork/tensorboardshare/logs/' + datetime.datetime.now().strftime
tbCallBack = TensorBoard(log_dir=log_dir,  # log 目录
                 histogram_freq=0,  # 按照何等频率（epoch）来计算直方图，0为不计算
                 write_graph=True,  # 是否存储网络结构图
                 write_images=True,# 是否可视化参数
                 update_freq='epoch',
                 embeddings_freq=0,
                 embeddings_layer_names=None,
                 embeddings_metadata=None,
                        profile_batch = 40)

#
#
total_train_sample =  115930
total_test_sample =   1181
train_steps_per_epoch=np.floor(total_train_sample/batch_size).astype(np.int32
test_steps_per_epoch = np.ceil(total_test_sample/val_batch_size).astype(np.in
history_loss = model.fit(dataset, epochs=1,
          steps_per_epoch=train_steps_per_epoch,
          validation_data=dataset_val, validation_steps=test_steps_per_epoch,
          verbose=1,callbacks=[tbCallBack])
```

模型结构summary



保存模型

```
# 用户塔 item塔定义
user_embedding_model = Model(inputs=model.user_input, outputs=model.user_embed
item_embedding_model = Model(inputs=model.item_input, outputs=model.item_embed
# 保存
tf.keras.models.save_model(user_embedding_model,"/Recall/DSSM/models/dssmUser/
tf.keras.models.save_model(item_embedding_model,"/Recall/DSSM/models/dssmItem/
```

获取user embedding 及item embedding

```
user_query = {'all_topic_fav_7': np.array([['294', '88', '60', '1']]),
  'all_topic_fav_7_weight':np.array([[ 0.0897,  0.2464,  0.0928,  0.5711,]]),
  'follow_topic_id': np.array([['75', '73', '74', '92', '62', '37', '35', '34
  'client_type': np.array([0.]),
    'client_embed': np.array([[-9.936600e-02,  2.752400e-01, -4.314620e-01,
          -5.263000e-02, -4.490300e-01, -3.641180e-01, -3.545410e-01,
          -2.315470e-01,  4.641480e-01,  3.965120e-01, -1.670170e-01,
          -5.480000e-03, -1.646790e-01,  2.522832e+00, -2.946590e-01,
                                    ......
          -1.151946e+00, -4.008270e-01,  1.521650e-01, -3.524520e-01,
           4.836160e-01, -1.190920e-01,  5.792700e-02, -6.148070e-01,
          -7.182930e-01, -1.351920e-01,  2.048980e-01, -1.259220e-01]])}

item_query = {
  'topic_id': np.array(['1']),
  'item_embed': np.array([[-9.936600e-02,  2.752400e-01, -4.314620e-01,  3.39
          -5.263000e-02, -4.490300e-01, -3.641180e-01, -3.545410e-01,
          -2.315470e-01,  4.641480e-01,  3.965120e-01, -1.670170e-01,
                                    ......
          -1.151946e+00, -4.008270e-01,  1.521650e-01, -3.524520e-01,
           4.836160e-01, -1.190920e-01,  5.792700e-02, -6.148070e-01,
          -7.182930e-01, -1.351920e-01,  2.048980e-01, -1.259220e-01]]),
}

user_embs = user_embedding_model.predict(user_query)
item_embs = item_embedding_model.predict(item_query)

# 结果:
# user_embs:
# array([[ 0.80766946,  0.13907856, -0.37779272,  0.53268254, -0.3095821 ,
#          0.2213103 , -0.24618168, -0.7127088 ,  0.4502724 ,  0.4282374 ,
#         -0.36033005,  0.43310016, -0.29158285,  0.8743557 ,  0.5113318 ,
#          0.26994514, -0.35604447,  0.33559784, -0.28052363,  0.38596702,
#          0.5038488 , -0.32811972, -0.5471834 , -0.07594685,  0.7006799 ,
#         -0.24201767,  0.31005877, -0.06173763, -0.28473467,  0.61975694,
                                  ......
#         -0.714099  , -0.5384026 ,  0.38787717, -0.4263588 ,  0.30690318,
#          0.24047776, -0.01420124,  0.15475503,  0.77783686, -0.43002903,
#          0.52561694,  0.37806144,  0.18955356, -0.37184635,  0.5181224 ,
#         -0.18585253,  0.05573007, -0.38589332, -0.7673693 , -0.25266737,
```

```
#          0.51427466,  0.47647673,  0.47982445]], dtype=float32)
# item_embs:
# array([[-6.9417924e-01, -3.9942840e-01,  7.2445291e-01, -5.8977932e-01,
#         -5.8792406e-01,  5.3883100e-01, -7.8469634e-01,  6.8996024e-01,
#         -7.6087400e-02, -4.4855604e-01,  8.4910756e-01, -4.7288817e-01,
#         -9.0812451e-01, -4.0452164e-01,  8.8695991e-01, -7.9177713e-01,
                                       ......
#         -9.7515762e-01, -5.2411711e-01,  9.2708725e-01, -1.3903661e-01,
#          7.8691095e-01, -8.0726832e-01, -7.3851186e-01,  2.7774110e-01,
#         -4.1870885e-02,  4.7335419e-01,  3.4424815e-01, -5.8394599e-01]],
#       dtype=float32)
```

6

线上Serving

我们这里向量召回检索框架用的是Milvus，用户的UE是线上实时获取的，item的embedding是异步获取存到Milvus平台上。

参考文献

王多鱼：实践DSSM召回模型

石塔西：用TensorFlow实现支持多值、稀疏、共享权重的DeepFM

https://github.com/shenweichen/