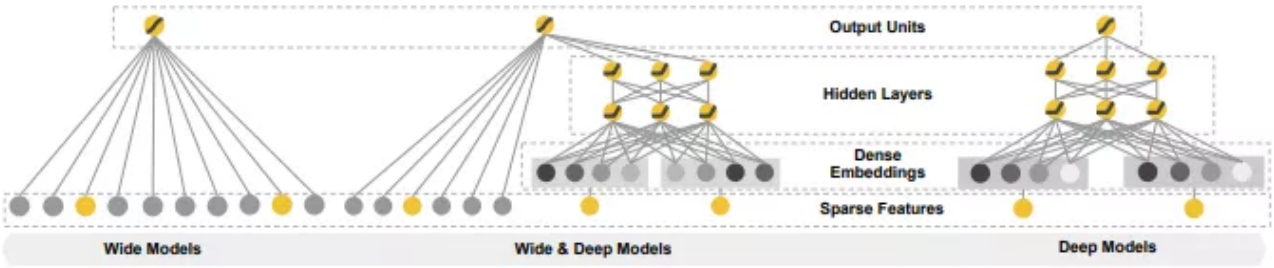
初学者系列：推荐系统Wide & Deep Learning详解

原创 Sha Li 专知 2019-08-24

导读

在信息大爆炸的今天，推荐系统（Recommendation System, RS）用作克服这种信息过载的通用解决方案，旨在从压倒性的在线内容和服务(例如电影、新闻和产品)中找到一组相关项目以满足用户的个人兴趣。用于推荐系统的策略有很多种，本文主要介绍的是在2016年提出的用于Google Play进行APP推荐的Wide & Deep Learning模型。



核心思想 01

Wide & Deep Learning 模型的核心思想是结合广义线性模型的记忆能力（memorization）和深度前馈神经网络模型的泛化能力（generalization）。利用广义线性模型从历史数据中学习特征相关性，利用深度前馈神经网络揭示隐式特征之间的相互作用。在训练过程中同时优化 2 个模型的参数，从而达到整体模型的预测能力最优。

动机 02

逻辑回归的广义线性模型，模型简单并且可解释性强，因此被广泛应用在工业环境中的大规模在线推荐和排名系统中。模型通常使用one-hot编码训练二值化稀疏特征。基于嵌入（Embedding）的模型，例如分解机(FM)或深度神经网络（DNN），通过为每个query和项目特征学习低维密集嵌入向量，进一步推广到以前未见过的query-item特征对。然而，当基础query项目矩阵稀疏且高维时，例如具有特定偏好的用户或具有很少吸引力的小众项目，难以学习查询和项目的有效低维表示。在这种情况下，大多数query-item对之间应该是没有交互的，

但是，密集嵌入（dense embeddings）将导致所有query-item对的非零预测，因此可能过度概括并产生不太相关的推荐。

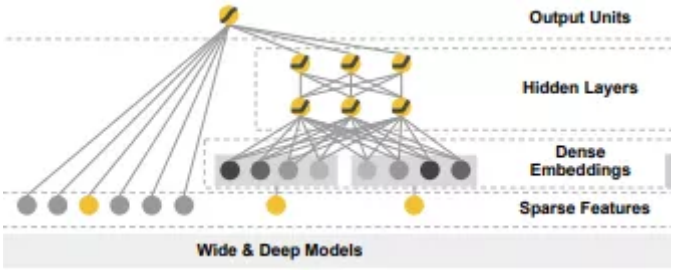
> tips："query"为当用户访问app store的时候生成的许多用户和文本特征。

基于这些问题，提出了Wide & Deep learning模型，联合训练wide线性模型和深度神经网络模型，将memorization和generalization的优点结合起来。使用 wide线性模型（Wide Model）用于记忆，使用深度神经网络模型（Deep Model）用于归纳。

> tips：Memorization是从历史数据中学习特征的共性或者相关性；Generalization为相关性的传递，发现历史数据中很少或者没有出现的新的特征组合，来提高推荐物品的多样性。

原理 03

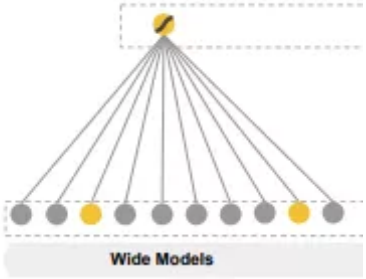
Wide & Deep Learning模型分为Wide model与 Deep model两部分，wide部分学习一些明确的特征依赖关系，deep部分在特征嵌入向量的级联上采用MLP来揭示隐式特征之间的相互作用。



wide 模型

wide 模型是的广义线性模型，如图下图所示。

$$y = W^T x + b$$



其中：

- y是预测值；
- x = [x1，x2，...，xd]是d维特征的向量（包括两部分）；
- w = [w1，w2，...，wd]是模型权重，b是偏差。

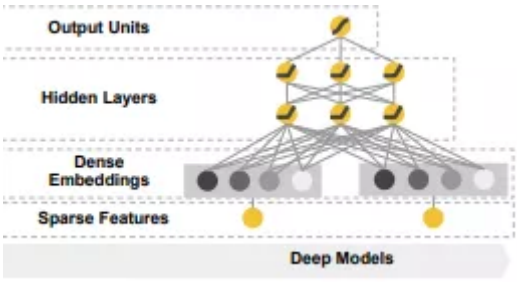
输入x（特征集合）包括**原始输入特征**和**转换特征**。常选择交叉积转换特征（cross-product transformation）（简单的说就是逻辑与运算（AND）），其定义为：

$$\emptyset_k(x) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0,1\}$$

> 交叉积：相当于“逻辑与”运算AND，AND{A,B}当且仅当"A=1且B=1“时，交叉特征才为1。如果第i个特征是第k个变换φk的一部分，则c_{ki}为1，否则为0。

Deep 模型

deep model 是深层前馈神经网络，如下图所示：



对于类别特征，原始输入是特征字符串（例如：“language=en”）每一个稀疏的高维类别特征首先被转换为低维且密集的实值向量，通常被称为嵌入向量（embedding vector）。嵌入向量随机初始化，然后最小化模型训练期间的损失函数。然后将这些低维密集嵌入矢量馈送到前向通道中的神经网络的隐藏层中。每个隐藏层执行以下计算：

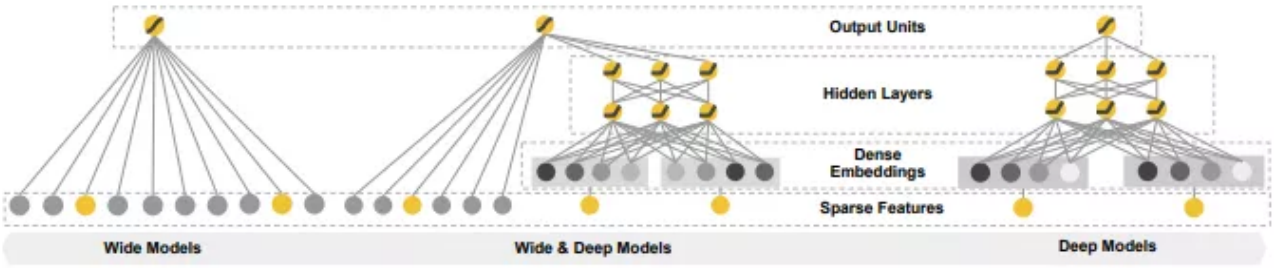
$$a^{(l+1)} = f(W^l a^l + b^l)$$

其中：

- l是层数
- f是激活函数，通常是ReLUs
- a^(l)，b^(l)和W^(l)是第l层的激活，偏差和模型权重。

联合训练

通过将Wide模块和Deep模块的对数加权输出作为预测值，然后送入逻辑损失函数中，用于联合训练。模型的联合训练通过反向传播将输出值的误差梯度通过最小批随机梯度同时传送给Wide和Deep模块。在实验中，作者使用FTRL算法作为wide模块的优化器，使用AdaGrad更新deep模块。



联合训练：两个模型是一起训练所有参数，两个模块只要互相补充对方不足。

对于逻辑回归问题，模型的预测是：

$$P(Y=1 \mid x) = \sigma(W_{wide}^{T}[x, \emptyset(x)] + W_{deep}^{T} a^{(l_f)} + b)$$

其中：

- Y是二进制类别标签
- σ（·）为sigmod函数
- φ（x）是原始特征x的交叉积转换特征
- b是偏置。
- w_wide是所有wide模型权重
- w_deep是最终激活层a_（lf）的权重

实践 04

该模型官方公布的源码中主要包括：census_dataset.py、wide_deep_run_loop.py以及census_main.py三部分。

- census_dataset.py主要处理数据以及生成特征列；

- wide_deep_run_loop.py主要定义训练以及评价；
- census_main.py为主函数，主要定义模型并且进行训练。

官方给定的数据集为'成人'数据集，即人口普查数据预测收入是否超过5万美元/年，数据集中各类属性值有连续值与离散值，属性列表为：

```
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, , , 2174, 0, 40, , <=50K
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, , , 0, 0, 13, , <=5
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners, Not-in-family, , , 0, 0, 40, , <=50K
53, Private, 234721, 11th, 7, Married-civ-spouse, Handlers-cleaners, Husband, , , 0, 0, 40, , <=50K
28, Private, 338409, Bachelors, 13, Married-civ-spouse, Prof-specialty, Wife, , , 0, 0, 40, , <=50K
37, Private, 284582, Masters, 14, Married-civ-spouse, Exec-managerial, Wife, , , 0, 0, 40, , <=50K
49, Private, 160187, 9th, 5, Married-spouse-absent, Other-service, Not-in-family, , , 0, 0, 16, , <=50K
52, Self-emp-not-inc, 209642, HS-grad, 9, Married-civ-spouse, Exec-managerial, Husband, , , 0, 0, 45, , >50K
31, Private, 45781, Masters, 14, Never-married, Prof-specialty, Not-in-family, , , 14084, 0, 50, , >50K
42, Private, 159449, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, , , 5178, 0, 40, , >50K
37, Private, 280464, Some-college, 10, Married-civ-spouse, Exec-managerial, Husband, , , 0, 0, 80, , >50K
30, State-gov, 141297, Bachelors, 13, Married-civ-spouse, Prof-specialty, Husband, , , 0, 0, 40, , >50K
23, Private, 122272, Bachelors, 13, Never-married, Adm-clerical, Own-child, , , 0, 0, 30, , <=50K
```

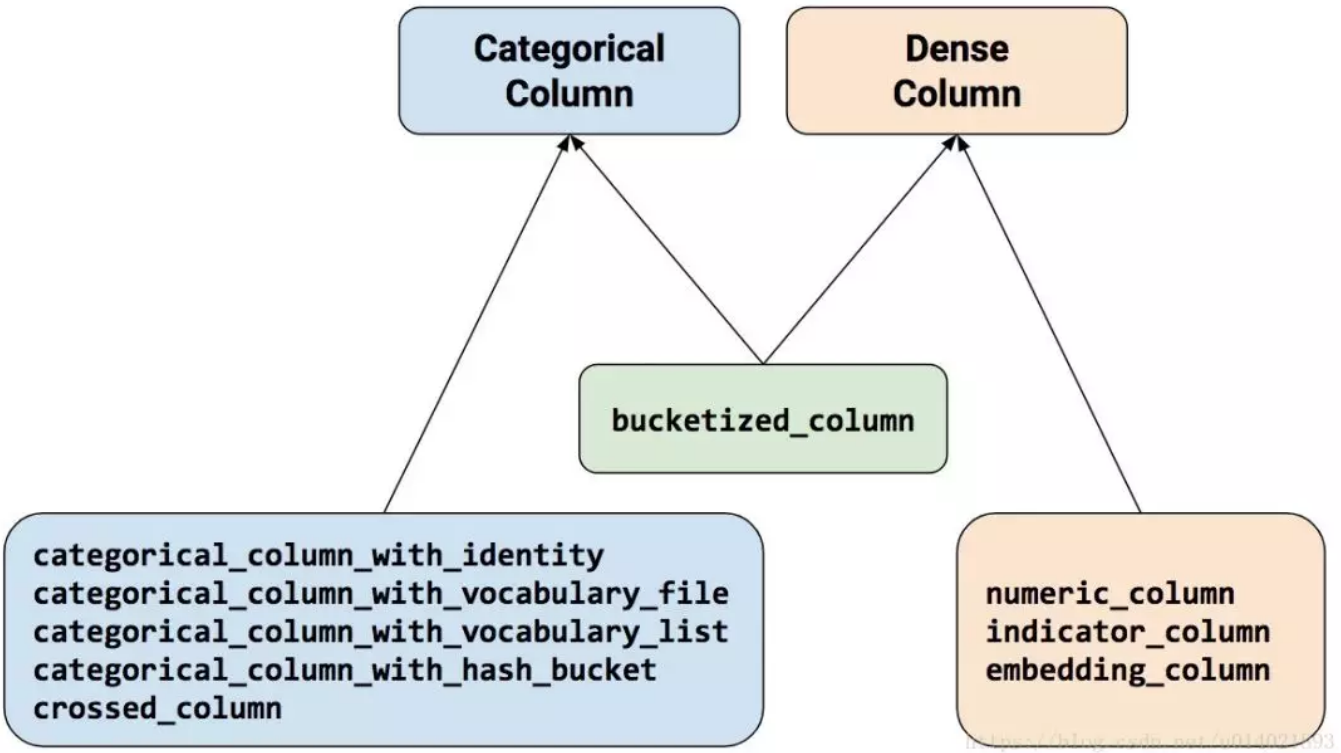
数据处理（census_dataset.py）

- 下载数据集，并对数据进行处理

```
DATA_URL = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult'#数据集地址
TRAINING_FILE = 'adult.data'
TRAINING_URL = '%s/%s' % (DATA_URL, TRAINING_FILE)   #训练数据
EVAL_FILE = 'adult.test'
EVAL_URL = '%s/%s' % (DATA_URL, EVAL_FILE)       #测试数据


_CSV_COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education_num',
    'marital_status', 'occupation', 'relationship', 'race', 'gender',
    'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
    'income_bracket'
]               #数据集中每一列对应特征的名称

_CSV_COLUMN_DEFAULTS = [[0], [''], [0], [''], [0], [''], [''], [''], [''], [''],
                        [0], [0], [0], [''], ['']]#输入数据的默认值，为0的都是连续值

_HASH_BUCKET_SIZE = 1000               #哈希值，在进行特征列转换的时候要用

_NUM_EXAMPLES = {
    'train': 32561,
    'validation': 16281,    #训练集验证集的数量
}

#从URL下载数据，去除首尾空格，替换', '，去除最后一位的'.',将新的数据集放在eval_file中。
def _download_and_clean_file(filename, url):
  """Downloads data from url, and makes changes to match the CSV format."""
  temp_file, _ = urllib.request.urlretrieve(url)#下载指定的URL内容到本地，
  with tf.gfile.Open(temp_file, 'r') as temp_eval_file:  #从temp_file文件中读取数据
    with tf.gfile.Open(filename, 'w') as eval_file:  #从filename中写入数据
      #print(temp_eval_file)
      for line in temp_eval_file:
        line = line.strip()
        line = line.replace(', ', ',')
        if not line or ',' not in line:
          continue
        if line[-1] == '.':
          line = line[:-1]
        line += '\n'
        eval_file.write(line)
        #print(eval_file)
  tf.gfile.Remove(temp_file)
#下载人口普查数据
def download(data_dir):
  """Download census data if it is not already present."""
  tf.gfile.MakeDirs(data_dir)   # 创建一个目录，data_dir为目录名，在这里是/tmp/census_data/

  training_file_path = os.path.join(data_dir, TRAINING_FILE)      #训练集，adult.data
  if not tf.gfile.Exists(training_file_path):                     #判断训练集路径是否存在，不存在则下载数据并进行转换
    _download_and_clean_file(training_file_path, TRAINING_URL)

  eval_file_path = os.path.join(data_dir, EVAL_FILE)
  if not tf.gfile.Exists(eval_file_path):
    _download_and_clean_file(eval_file_path, EVAL_URL)
```

- 构建特征列

特征列（feature columns）是用来将采集到的数据进行规范约束，从而得到可以进行tensorflow所支持的类型的特征。tensorflow的特征列构造由tensorflow.feature_column模块来提供，共九种不同的类型。



（图片来源于tf官方文档）

- tf.feature_column.numeric_column：全部由数值型构成的，默认值为float32。
- tf.feature_column.categorical_column_with_hash_bucket：计算输入的哈希值，然后使用模运算符将其置于其中一个 hash_bucket_size 类别。
- tf.feature_column.categorical_column_with_vocabulary_list：根据明确的词汇表将每个字符串映射到一个整数（one—hot类型）。
- tf.feature_column.bucketized_column：据数值范围将其值分为不同的类别（如：年龄<20，经过处理可表示为[1,0,0,0,0]）
- tf.feature_column.crossed_column：将多个特征组合为一个特征，即论文中的交叉积特征转换（cross-product transformation）。
- tf.feature_column.indicator_column：将每个类别视为one-hot矢量中的一个元素，其中匹配类别的值为 1，其余类别为 0
- tf.feature_column.embedding_column：与indicator_column不同 ，经常为0-1之间的小数，而不是只有一个位置（例tf.feature_column.embedding_column（occupation, dimension=3），输出结果为[0.233,0.256,0.487]）

```
#构建特征列
def build_model_columns():
# Continuous variable columns，连续变量的特征
  age = tf.feature_column.numeric_column('age')  #tf.feature_column.numeric_column全部由数值型构建的数值集合
  education_num = tf.feature_column.numeric_column('education_num')
  capital_gain = tf.feature_column.numeric_column('capital_gain')
  capital_loss = tf.feature_column.numeric_column('capital_loss')
  hours_per_week = tf.feature_column.numeric_column('hours_per_week')
#构建标识标签的特征列
  education = tf.feature_column.categorical_column_with_vocabulary_list(             #将词汇表中的字符串表示为one-hot类型
      'education', [
          'Bachelors', 'HS-grad', '11th', 'Masters', '9th', 'Some-college',
          'Assoc-acdm', 'Assoc-voc', '7th-8th', 'Doctorate', 'Prof-school',
          '5th-6th', '10th', '1st-4th', 'Preschool', '12th'])
  marital_status = tf.feature_column.categorical_column_with_vocabulary_list(
      'marital_status', [
          'Married-civ-spouse', 'Divorced', 'Married-spouse-absent',
          'Never-married', 'Separated', 'Married-AF-spouse', 'Widowed'])
  relationship = tf.feature_column.categorical_column_with_vocabulary_list(
      'relationship', [
          'Husband', 'Not-in-family', 'Wife', 'Own-child', 'Unmarried',
          'Other-relative'])
  workclass = tf.feature_column.categorical_column_with_vocabulary_list(
      'workclass', [
          'Self-emp-not-inc', 'Private', 'State-gov', 'Federal-gov',
          'Local-gov', '?', 'Self-emp-inc', 'Without-pay', 'Never-worked'])

  occupation = tf.feature_column.categorical_column_with_hash_bucket(     #类别数量很大，将occupation（职业）的类别转换
                   'occupation', hash_bucket_size=_HASH_BUCKET_SIZE)                #为经过哈希值处理的列
  age_buckets = tf.feature_column.bucketized_column(
      age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])   #将年龄按照boundaries的值划分成几个不同的年龄区间，
                                                                  #例如<18岁的为[1,0,0,0,0,0,0,0,0,0]
 #wide模型的输入（education, marital_status, relationship, workclass, occupation,age_buckets），以及进行交叉积处理之后的结果
  base_columns = [
      education, marital_status, relationship, workclass, occupation,
      age_buckets,
  ]
  crossed_columns = [                                                 #交叉积转换特征列
      tf.feature_column.crossed_column(
          ['education', 'occupation'], hash_bucket_size=_HASH_BUCKET_SIZE),#将教育跟职业组合成一个特征
      tf.feature_column.crossed_column(
          [age_buckets, 'education', 'occupation'],#将年龄，教育，职业组合成一个特征
          hash_bucket_size=_HASH_BUCKET_SIZE),
  ]
  wide_columns = base_columns + crossed_columns
#deep模型的特征
  deep_columns = [
      age,
      education_num,
      capital_gain,
      capital_loss,
      hours_per_week,
      tf.feature_column.indicator_column(workclass),#将每个类别视为one-hot矢量中的一个元素，其中匹配类别的值为 1，其余类别为 0
      tf.feature_column.indicator_column(education),
      tf.feature_column.indicator_column(marital_status),
      tf.feature_column.indicator_column(relationship),
      tf.feature_column.embedding_column(occupation, dimension=8),]#与indicator_column不同 ，经常为0-1之间的小数，
                                                                  #而不是只有一个位置的值为1
  return wide_columns, deep_columns
```

- 定义Estimator的输入函数

```
#定义Estimator的输入函数
def input_fn(data_file, num_epochs, shuffle, batch_size):
  assert tf.gfile.Exists(data_file), (
      '%s not found. Please make sure you have run census_dataset.py and '
      'set the --data_dir argument to the correct path.' % data_file)#判断数据地址是否正确
#读取csv文件，制作数据集
  def parse_csv(value):
    tf.logging.info('Parsing {}'.format(data_file)) #输出tensorflow的日志信息，便于追踪模型训练时，获得fit操作后的信息
    columns = tf.decode_csv(value, record_defaults=_CSV_COLUMN_DEFAULTS)#将CSV记录转换为张量。每一列映射到一个张量
    features = dict(zip(_CSV_COLUMNS, columns))  #features={'age':18 ,}
    labels = features.pop('income_bracket')#除列表中的income_bracket，并且返回该元素的值。
    classes = tf.equal(labels, '>50K')  #判断label是否>50K
    return features, classes

  dataset = tf.data.TextLineDataset(data_file)#生成一个dataset，dataset中的每一个元素就对应了文件中的一行

  if shuffle:
    dataset = dataset.shuffle(buffer_size=_NUM_EXAMPLES['train'])#打乱元素顺序

  dataset = dataset.map(parse_csv, num_parallel_calls=5)#map接收一个函数，Dataset中的每个元素都会被当作这个函数的输入，并将函数返
回值作为新的Dataset

  dataset = dataset.repeat(num_epochs)#repeat的功能就是将整个序列重复多次，原来的数据时一个epoch的 现在就是所有epoch的
  dataset = dataset.batch(batch_size)#划分batch
  return dataset
```

主函数（census_main.py）

- 参数设置

包括数据地址、模型地址 、batch_size等，

```
#自定义的参数设置
def define_census_flags():
  wide_deep_run_loop.define_wide_deep_flags()
  flags.adopt_module_key_flags(wide_deep_run_loop)
  #自定义参数设置
  flags_core.set_defaults(data_dir='/tmp/census_data',  #数据地址
                          model_dir='/tmp/census_model',#模型地址
                          train_epochs=40,
                          epochs_between_evals=2,
                          inter_op_parallelism_threads=0,
                          intra_op_parallelism_threads=0,
                          batch_size=40)
```

- 定义模型

为给定的模型类型构建适当的估算器，主要包括单独使用wide模型时使用线性分类器（LinearClassifier）、单独使用deep模型时使用DNN分类器（DNNClassifier）以及wide&wide模型（DNNLinearCombinedClassifier）。

```
#为给定的模型类型构建适当的估算器，对于不同类型的模型分别使用不同的estimator
def build_estimator(model_dir, model_type, model_column_fn, inter_op, intra_op):
  """Build an estimator appropriate for the given model type."""
  wide_columns, deep_columns = model_column_fn()
  hidden_units = [100, 75, 50, 25]
  run_config = tf.estimator.RunConfig().replace(
      session_config=tf.ConfigProto(device_count={'GPU': 0},
                                    inter_op_parallelism_threads=inter_op,
                                    intra_op_parallelism_threads=intra_op))
#单独wide模型的线性分类器
  if model_type == 'wide':
    return tf.estimator.LinearClassifier(
        model_dir=model_dir,
        feature_columns=wide_columns,
        config=run_config)
#单独deep模型的DNN分类器
  elif model_type == 'deep':
    return tf.estimator.DNNClassifier(
        model_dir=model_dir,
        feature_columns=deep_columns,
        hidden_units=hidden_units,
        config=run_config)
#wide&deep 模型的联合分类器
  else:
    return tf.estimator.DNNLinearCombinedClassifier(
        model_dir=model_dir,
        linear_feature_columns=wide_columns,
        dnn_feature_columns=deep_columns,
        dnn_hidden_units=hidden_units,
        config=run_config)
```

> | tips：关于DNNLinearCombinedClassifier的详细信息，可以点击**DNNLinearCombinedClassifier**查看

- 开始训练

在训练部分，主要调掉用了wide_deep_run_loop.py里的run_loop（）函数，输入特征列进行训练，并且得到评价指标。

```
#构造所有必需的函数并调用run_loop，训练部分。
def run_census(flags_obj):
  """Construct all necessary functions and call run_loop.

  Args:
    flags_obj: Object containing user specified flags.
  """
#获得的训练数据集、验证数据集
  if flags_obj.download_if_missing:
    census_dataset.download(flags_obj.data_dir)      #如果没有数据 调用download（）下载数据

  train_file = os.path.join(flags_obj.data_dir, census_dataset.TRAINING_FILE)#训练数据地址/tmp/census_data/adult.data
  test_file = os.path.join(flags_obj.data_dir, census_dataset.EVAL_FILE)

  def train_input_fn():
    return census_dataset.input_fn(
        train_file, flags_obj.epochs_between_evals, True, flags_obj.batch_size)#调用input_fn（）得到dataset

  def eval_input_fn():
    return census_dataset.input_fn(test_file, 1, False, flags_obj.batch_size)

  tensors_to_log = {
      'average_loss': '{loss_prefix}head/truediv',
      'loss': '{loss_prefix}head/weighted_loss/Sum'
  }
#使用模型进行训练并，计算评价指标
  wide_deep_run_loop.run_loop(                       #调用run_loop（），得到模型对不用模块输入不同的特征进行训练，得到准确率
      name="Census Income", train_input_fn=train_input_fn,
      eval_input_fn=eval_input_fn,
      model_column_fn=census_dataset.build_model_columns,
      build_estimator_fn=build_estimator,
      flags_obj=flags_obj,
      tensors_to_log=tensors_to_log,
      early_stop=True)


def main(_):
  with logger.benchmark_context(flags.FLAGS):
    run_census(flags.FLAGS)  #flags.FLAGSFLAGS是一个对象，保存了解析后的命令行参数


if __name__ == '__main__':
  tf.logging.set_verbosity(tf.logging.INFO)
  define_census_flags()
  absl_app.run(main)#解析命令行参数，调用main函数
```

run_loop（）函数的定义如下所示：

```
#定义循环训练
def run_loop(name, train_input_fn, eval_input_fn, model_column_fn,
             build_estimator_fn, flags_obj, tensors_to_log, early_stop=False):
 #model_column_fn=census_dataset.build_model_columns
  model_helpers.apply_clean(flags.FLAGS)  #apply_clean是官方例程里面提供的用来清理现存model的方法
  #调用build_estimator_fn，加载模型
  model = build_estimator_fn(
      model_dir=flags_obj.model_dir, model_type=flags_obj.model_type,
      model_column_fn=model_column_fn,
      inter_op=flags_obj.inter_op_parallelism_threads,
      intra_op=flags_obj.intra_op_parallelism_threads)

  run_params = {
      'batch_size': flags_obj.batch_size,
      'train_epochs': flags_obj.train_epochs,
      'model_type': flags_obj.model_type,
  }

  benchmark_logger = logger.get_benchmark_logger()
  benchmark_logger.log_run_info('wide_deep', name, run_params,
                                test_id=flags_obj.benchmark_test_id)

  loss_prefix = LOSS_PREFIX.get(flags_obj.model_type, '')
  tensors_to_log = {k: v.format(loss_prefix=loss_prefix)
                    for k, v in tensors_to_log.items()}
  train_hooks = hooks_helper.get_train_hooks(
      flags_obj.hooks, model_dir=flags_obj.model_dir,
      batch_size=flags_obj.batch_size, tensors_to_log=tensors_to_log)
#开始训练

  for n in range(flags_obj.train_epochs // flags_obj.epochs_between_evals):

    model.train(input_fn=train_input_fn, hooks=train_hooks)#训练

    results = model.evaluate(input_fn=eval_input_fn)#评价函数

    # Display evaluation metrics
    tf.logging.info('Results at epoch %d / %d',
                    (n + 1) * flags_obj.epochs_between_evals,
                    flags_obj.train_epochs)
    tf.logging.info('-' * 60)

    for key in sorted(results):
      tf.logging.info('%s: %s' % (key, results[key]))

    benchmark_logger.log_evaluation_result(results)

    if early_stop and model_helpers.past_stop_threshold(
        flags_obj.stop_threshold, results['accuracy']):
      break

  # Export the model
  if flags_obj.export_dir is not None:
    export_model(model, flags_obj.model_type, flags_obj.export_dir,
                 model_column_fn)
```

训练结果

一共40个epoch，下图为第40个epoch的训练结果。

```
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:121] Results at epoch 40 / 40
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:122] ------------------------------------------------
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] accuracy: 0.85473865
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] accuracy_baseline: 0.76377374
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] auc: 0.9068305
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] auc_precision_recall: 0.7700139
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] average_loss: 0.31818268
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] global_step: 32580
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] label/mean: 0.23622628
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] loss: 12.696892
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] precision: 0.73440963
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] prediction/mean: 0.24778935
I0806 16:47:23.528290 11044 wide_deep_run_loop.py:125] recall: 0.60322416
```

代码地址：

https://github.com/tensorflow/models/tree/master/official/wide_deep

论文地址：https://arxiv.org/pdf/1606.07792.pdf

-END-

专·知

专知，专业可信的人工智能知识分发，让认知协作更快更好！欢迎登录www.zhuanzhi.ai，注册登录专知，获取更多AI知识资料！

欢迎微信扫一扫加入**专知人工智能知识星球群**，获取**最新AI专业干货知识教程视频资料和与专家交流咨询**！