DeepTables

发布 *0.1.1*

2020 年 04 月 14 日

Home:

CHAPTER 1

DeepTables: 表格式数据深度学习工具包

DeepTables(DT) 是一个易用的工具包，在结构化数据上释放深度学习的洪荒之力。

CHAPTER 2

Overview

MLP（也称为全连接神经网络）在学习分布表示方面表现出低效。感知器层的“添加”操作在探索乘法特征交互方面表现不佳。在大多数情况下，手工特征工程是必要的，这项工作需要广泛的领域知识和非常繁琐。如何在神经网络中有效地学习特征间的相互作用成为最重要的问题。

近年来，已有许多模型被提出用于 CTR 预测，并继续优于现有的最新方法。众所周知的例子有 FM、DeepFM、Wide&Deep、DCN、PNN 等，这些模型在合理利用表格式数据的情况下也能提供良好的性能。

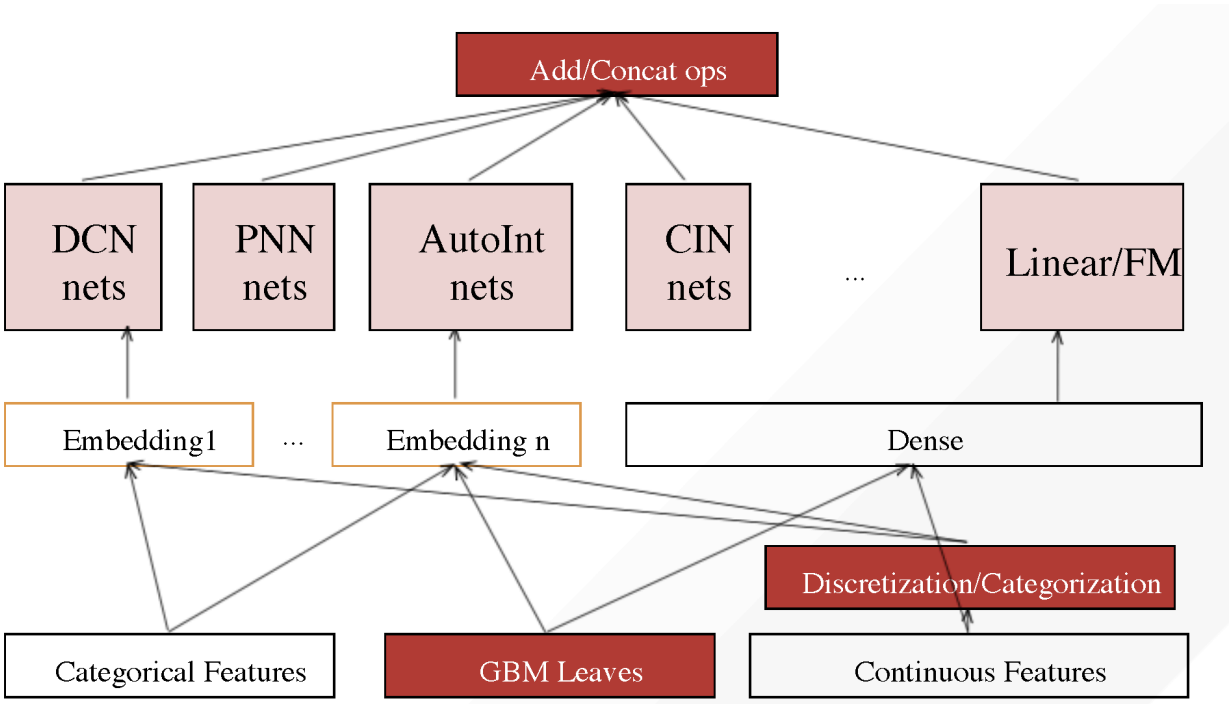
DT 致力于利用最新的研究成果为用户提供一个关于表格数据的端到端工具包。

设计 DT 时考虑到了这些关键目标:

- 使用方便，非专家也可以使用。
- 默认配置下提供良好的性能。
- 结构灵活，易于用户扩展。

DT 构建神经网络遵循以下步骤:

1. 提取到类别型特征送到 Embedding 层。
2. 连续型特征送到 Dense 层或者离散化类别化后送到 embedding 层处理。
3. Embedding/Dense 层输出到不同的网络组件中。
4. 不同的网络组件通过 Add/Concat 操作融合后作为模型的输出。


Add/Concat ops
DCN nets
PNN nets
AutoInt nets
CIN nets
...
Linear/FM
Embedding1
...
Embedding n
Dense
Discretization/Categorization
Categorical Features
GBM Leaves
Continuous Features

CHAPTER 3

DeepTables 的优势

- 特征预处理或者加工简单。
  - 不管是数据科学家还是没有建模能力的业务人员都可以使用。
  - 比传统高度依赖特征工程的机器学习简单易用。
- 默认配置下具有良好的性能
  - 包含了一系列近几年的最优秀的网络组件。
- 非常容易上手。
  - 5 行代码就可以训练任何表格数据集。
- 开放架构设计。
  - 支持插件扩展。

CHAPTER 4

Example

```
from deeptables.models.deeptable import DeepTable, ModelConfig
from deeptables.models.deepnets import DeepFM

dt = DeepTable(ModelConfig(nets=DeepFM))
dt.fit(X, y)
preds = dt.predict(X_test)
```

4.1 Quick-Start

4.1.1 安装引导

环境依赖

Python 3: DT 需要 Python 3.6 3.7。

Tensorflow >= 2.0.0: DT 基于 TensorFlow。

安装 DeepTables

```
pip install deeptables
```

GPU 安装 (可选): 如果你有 gpu，并且想用它们加速训练，可以用下面命令安装：

```
pip install deeptables[gpu]
```

验证安装：

```
python -c "from deeptables.utils.quicktest import test; test()"
```

启动 Notebook Docker 容器

你可以通过 Docker 快速尝试 DeepTables:

1. 拉取一个 DeepTables 镜像。
2. 启动容器。

拉取最新镜像:

```
docker pull datacanvas/deeptables-example
```

用这个命令启动 Docker 容器：

```
docker run -it -p 8830:8888 -e NotebookToken="your-token"  datacanvas/deeptables-example
```

The value "your-token" is a user specified string for the notebook and can be empty.

notebook 服务运行在 https://host-ip-address:8830?token=your-token 启动浏览器访问这个 URL，你能看到 Notebook 的页面：

4.1.2 5 行代码开启 DT 之旅

适用范围

DT 能用来解决基于表格式数据的分类、回归预测问题。

简单例子

DT 通过极其简单的接口支持这些任务，而不需要处理数据清理和特性工程。你甚至不指定任务类型，DT 将自动推断。

```
from deeptables.models.deeptable import DeepTable, ModelConfig
from deeptables.models.deepnets import DeepFM

dt = DeepTable(ModelConfig(nets=DeepFM))
dt.fit(X, y)
preds = dt.predict(X_test)
```

4.1.3 数据集

DT 有几个用于演示或测试的内置数据集，涵盖了二进制分类、多类分类和回归任务。所有数据集都是通过 deeptables.datasets.dsutils 访问的。

Adult

Associated Tasks: **Binary Classification**

根据人口普查数据预测收入是否超过 5 万美元/年，也被成为”Census Income”数据集。

```
from deeptables.datasets import dsutils
df = dsutils.load_adult()
```

See: http://archive.ics.uci.edu/ml/datasets/Adult

Glass Identification

Associated Tasks: **Multi-class Classification**

来自美国法医科学服务局;6 种玻璃; 根据氧化物含量定义 (i.e. Na, Fe, K, etc)

```
from deeptables.datasets import dsutils
df = dsutils.load_glass_uci()
```

See: http://archive.ics.uci.edu/ml/datasets/Glass+Identification

Boston house-prices

Associated Tasks: **Regression**

```
from deeptables.datasets import dsutils
df = dsutils.load_boston()
```

See: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html

Examples

See: Examples

4.2 Exapmles

4.2.1 Binary Classification

This example demonstrate how to use WideDeep nets to solve a binary classification prediction problem.

```
from deeptables.models.deeptable import DeepTable, ModelConfig
from deeptables.models.deepnets import WideDeep
from deeptables.datasets import dsutils
from sklearn.model_selection import train_test_split

#Adult Data Set from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/
↪datasets/Adult
df_train = dsutils.load_adult()
y = df_train.pop(14)
X = df_train

#`auto_discrete` is used to decide wether to discretize continous varibles automatically.
conf = ModelConfig(nets=WideDeep, metrics=['AUC','accuracy'], auto_discrete=True)
dt = DeepTable(config=conf)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model, history = dt.fit(X_train, y_train, epochs=100)

score = dt.evaluate(X_test, y_test)

preds = dt.predict(X_test)
```

4.2.2 Multiclass Classification

This simple example demonstrate how to use a DNN(MLP) nets to solve a multiclass task on MNIST dataset.

```
from deeptables.models import deeptable
from tensorflow import keras

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(60000, 784).astype('float32') / 255
x_test = x_test.reshape(10000, 784).astype('float32') / 255
```

(下页继续)

(续上页)

```
conf = deeptable.ModelConfig(nets=['dnn_nets'], optimizer=keras.optimizers.RMSprop())
dt = deeptable.DeepTable(config=conf)

model, history = dt.fit(x_train, y_train, epochs=10)

score = dt.evaluate(x_test, y_test, batch_size=512, verbose=0)

preds = dt.predict(x_test)
```

4.2.3 Regression

This example shows how to use DT to predicting Boston housing price.

```
from deeptables.models.deeptable import DeepTable, ModelConfig
from deeptables.datasets import dsutils
from sklearn.model_selection import train_test_split

df_train = dsutils.load_boston()
y = df_train.pop('target')
X = df_train

conf = ModelConfig(
    metrics=['RootMeanSquaredError'],
    nets=['dnn_nets'],
    dnn_params={
        'dnn_units': ((256, 0.3, True), (256, 0.3, True)),
        'dnn_activation': 'relu',
    },
    earlystopping_patience=5,
)

dt = DeepTable(config=conf)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model, history = dt.fit(X_train, y_train, epochs=100)

score = dt.evaluate(X_test, y_test)
```

4.3 ModelConfig

ModelConfig is the most important parameter in DT. It is used to set how to clean and preprocess the data automatically, and how to assemble various network components to building a neural nets for prediction tasks, as well as the setting of hyper-parameters of nets, etc. If you do not change any settings in ModelConfig, DT will work in most cases as well. However, you can get a better performance by tuning the parameters in ModelConfig.

We describe in detail below.

4.3.1 Simple use case for ModelConfig

```
from deeptables.models.deeptable import DeepTable, ModelConfig
from deeptables.models.deepnets import DeepFM

conf = ModelConfig(
    nets=DeepFM, # same as `nets=['linear','dnn_nets','fm_nets']`
    categorical_columns='auto', # or categorical_columns=['x1', 'x2', 'x3', ...]
    metrics=['AUC', 'accuracy'], # can be `metrics=['RootMeanSquaredError']` for␣
↪regression task
    auto_categorize=True,
    auto_discrete=False,
    embeddings_output_dim=20,
    embedding_dropout=0.3,
    )
dt = DeepTable(config=conf)
dt.fit(X, y)
```

4.3.2 Parameters

nets

list of str or custom function, (default=`['dnn_nets']`)

You can use multiple components to compose neural network joint training to perform prediction tasks.

The value of nets can be any combination of component name, preset model and custom function.

components:

- 'dnn_nets'
- 'linear'

- 'cin_nets'
- 'fm_nets'
- 'afm_nets'
- 'opnn_nets'
- 'ipnn_nets'
- 'pnn_nets' ,
- 'cross_nets'
- 'cross_dnn_nets'
- 'dcn_nets' ,
- 'autoint_nets'
- 'fg_nets'
- 'fgcnn_cin_nets'
- 'fgcnn_fm_nets'
- 'fgcnn_ipnn_nets'
- 'fgcnn_dnn_nets'
- 'fibi_nets'
- 'fibi_dnn_nets'

preset models: (in package deeptables.models.deepnets)

- DeepFM
- xDeepFM
- DCN
- PNN
- WideDeep
- AutoInt
- AFM
- FGCNN
- FibiNet

custom function:

```
def custom_net(embeddings, flatten_emb_layer, dense_layer, concat_emb_dense, config,␣
↪model_desc):
    out = layers.Dense(10)(flatten_emb_layer)
    return out
```

examples:

```
from deeptables.models.deeptable import ModelConfig, DeepTable
from deeptables.models import deepnets
from tensorflow.keras import layers

#preset model
conf = ModelConfig(nets=deepnets.DeepFM)

#list of str(name of component)
conf = ModelConfig(nets=['linear','dnn_nets','cin_nets','cross_nets'])

#mixed preset model and names
conf = ModelConfig(nets=deepnets.WideDeep+['cin_nets'])

#mixed names and custom function
def custom_net(embeddings, flatten_emb_layer, dense_layer, concat_emb_dense, config,␣
↪model_desc):
    out = layers.Dense(10)(flatten_emb_layer)
    return out
conf = ModelConfig(nets=['linear', custom_net])
```

categorical_columns

list of strings or ‘auto’, optional, (default=`'auto'`)

Only categorical features will be passed into embedding layer, and most of the components in DT are specially designed for the embedding outputs for feature extraction. Reasonable selection of categorical features is critical to model performance.

If **list of strings**, interpreted as column names.

If `'auto'`, get the categorical columns automatically. `object`, `bool` and `category` columns will be selected by default, and [auto_categorize] will no longer take effect.

If not necessary, we strongly recommend use default value `'auto'`.

exclude_columns

list of strings, (default=[])

pos_label

str or int, (default=`None`)

The label of positive class, used only when task is binary.

metrics

list of strings or callable object, (default=`['accuracy']`)

List of metrics to be evaluated by the model during training and testing. Typically you will use `metrics=['accuracy']` or `metrics=['AUC']`. Every metric should be a built-in evaluation metric in tf.keras.metrics or a callable object like `r2(y_true, y_pred):...`.

See also: https://tensorflow.google.cn/versions/r2.0/api_docs/python/tf/keras/metrics

auto_categorize

bool, (default=`False`)

Whether to automatically categorize eligible continuous features.

- `True`:
- `False`:

cat_exponent

float, (default=`0.5`), between 0 and 1

Only usable when auto_categrization = `True`.

Columns with `(number of unique values < number of samples ** cat_exponent)` will be treated as categorical feature.

cat_remain_numeric

bool, (default=`True`)

Only usable when auto_categrization = `True`.

Whether continuous features transformed into categorical retain numerical features.

- `True`:

- False:

auto_encode_label

bool, (default=True)

Whether to automatically perform label encoding on categorical features.

auto_imputation

bool, (default=True)

Whether to automatically perform imputation on all features.

auto_discrete

bool, (default=False)

Whether to discretize all continuous features into categorical features.

fixed_embedding_dim

bool, (default=True)

Whether the embeddings output of all categorical features uses the same 'output_dim'. It should be noted that some components require that the output_dim of embeddings must be the same, including **FM**, **AFM**, **CIN**, **MultiheadAttention**, **SENET**, **InnerProduct**, etc.

If False and embedding_output_dim=0, then the output_dim of embeddings will be calculated using the following formula:

```
min(4 * int(pow(voc_size, 0.25)), 20)
#voc_size is the number of unique values of each feature.
```

embeddings_output_dim

int, (default=4)

embeddings_initializer

str or object, (default='uniform')

Initializer for the embeddings matrix.

embeddings_regularizer

str or object, (default=`None`)

Regularizer function applied to the `embeddings` matrix.

dense_dropout

float, (default=`0`) between 0 and 1

Fraction of the dense input units to drop.

embedding_dropout

float, (default=`0.3`) between 0 and 1

Fraction of the embedding input units to drop.

stacking_op

str, (default=`'add'`)

- `'add'`
- `'concat'`

output_use_bias

bool, (default=`True`)

optimizer

str(name of optimizer) or optimizer instance or ‘auto’, (default=`'auto'`)

See `tf.keras.optimizers`.

- ‘auto’ : Automatically select optimizer based on task type.

loss

str(name of objective function) or objective function or `tf.losses.Loss` instance or ‘auto’, (default=’auto’)

See `tf.losses`.

- ‘auto’ : Automatically select objective function based on task type.

home_dir

str, (default=`None`)

The home directory for saving model-related files. Each time running `fit(...)` or `fit_cross_validation(...)`, a subdirectory with a time-stamp will be created in this directory.

monitor_metric

str, (default=`None`)

earlystopping_patience

int, (default=`1`)

gpu_usage_strategy

str, (default=`'memory_growth'`)

- `'memory_growth'`
- `'None'`

distribute_strategy:

tensorflow.python.distribute.distribute_lib.Strategy, (default=`None`)

dnn_params

dictionary Only usable when ‘dnn_nets’ or a component using ‘dnn’ like ‘pnn_nets’ ,’ dcn_nets’ included in [nets].

```
{
    'dnn_units': ((128, 0, False), (64, 0, False)),
    'dnn_activation': 'relu'}
)
```

autoint_params

dictionary Only usable when ‘autoint_nets’ included in [nets].

```
{
    'num_attention': 3,
    'num_heads': 1,
    'dropout_rate': 0,
    'use_residual': True
}
```

fgcnn_params

dictionary Only usable when ‘fgcnn_nets’ or a component using ‘fgcnn’ included in [nets].

```
{
    'fg_filters': (14, 16),
    'fg_widths': (7, 7),
    'fg_pool_widths': (2, 2),
    'fg_new_feat_filters': (2, 2),
}
```

fibinet_params

dictionary Only usable when ‘fibi_nets’ included in [nets].

```
{
    'senet_pooling_op': 'mean',
    'senet_reduction_ratio': 3,
    'bilinear_type': 'field_interaction',
}
```

cross_params

dictionary Only usable when ‘cross_nets’ included in [nets].

```
{
    'num_cross_layer': 4,
}
```

pnn_params

dictionary Only usable when ‘pnn_nets’ or ‘opnn_nets’ included in [nets].

```
{
    'outer_product_kernel_type': 'mat',
}
```

afm_params

dictionary Only usable when ‘afm_nets’ included in [nets].

```
{
    'attention_factor': 4,
    'dropout_rate': 0
}
```

cin_params

dictionary Only usable when ‘cin_nets’ included in [nets].

```
{
    'cross_layer_size': (128, 128),
    'activation': 'relu',
    'use_residual': False,
    'use_bias': False,
    'direct': False,
    'reduce_D': False,
}
```

4.4 Models

In recent years, a lot of neural nets have been proposed to CTR prediction and continue to outperform existing state-of-the-art approaches. Well-known examples include FM, DeepFM, Wide&Deep, DCN, PNN, etc. DT provides most of these models and will continue to introduce the latest research findings in the future.

4.4.1 Wide&Deep

Cheng, Heng-Tze, et al. “Wide & deep learning for recommender systems.” Proceedings of the 1st workshop on deep learning for recommender systems. 2016.

Retrieve from: https://dl.acm.org/doi/abs/10.1145/2988450.2988454

Wide & Deep learning—jointly trained wide linear models and deep neural networks—to combine the benefits of memorization and generalization for recommender systems. We productionized and evaluated the system on Google Play, a commercial mobile app store with over one billion active users and over one million apps. Online experiment results show that Wide & Deep significantly increased app acquisitions compared with wide-only and deep-only models.

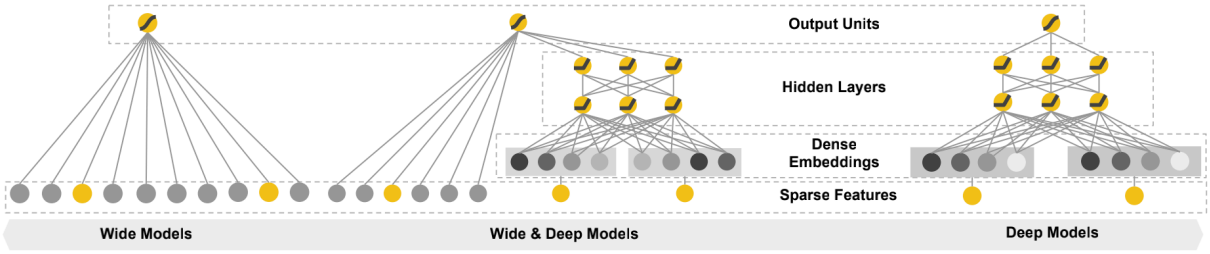


Figure 1: The spectrum of Wide & Deep models.

4.4.2 DCN(Deep & Cross Network)

Wang, Ruoxi, et al. "Deep & cross network for ad click predictions." Proceedings of the ADKDD' 17. 2017. 1-7.

Retrived from: https://dl.acm.org/doi/abs/10.1145/3124749.3124754

Deep & Cross Network (DCN) keeps the benefits of a DNN model, and beyond that, it introduces a novel cross network that is more efficient in learning certain bounded-degree feature interactions. In particular, DCN explicitly applies feature crossing at each layer, requires no manual feature engineering, and adds negligible extra complexity to the DNN model.

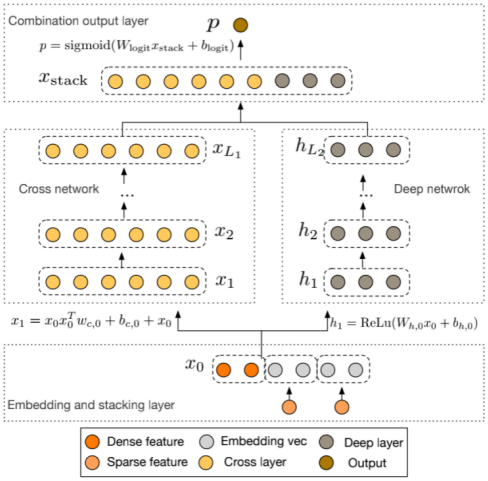


Figure 1: The Deep & Cross Network

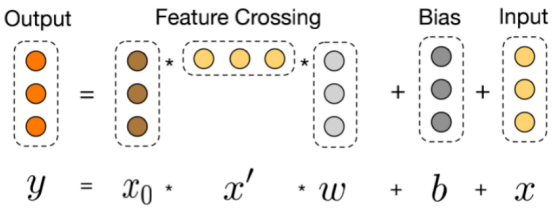


Figure 2: Visualization of a cross layer.

4.4.3 PNN

Qu, Yanru, et al. "Product-based neural networks for user response prediction."2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016.

Retrived from: https://ieeexplore.ieee.org/abstract/document/7837964/

> Product-based Neural Networks (PNN) with an embedding layer to learn a distributed representation of the categorical data, a product layer to capture interactive patterns between inter-field categories, and further fully connected layers to explore high-order feature interactions.

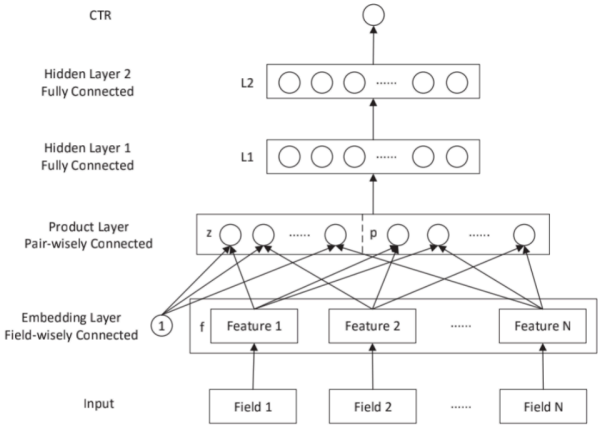


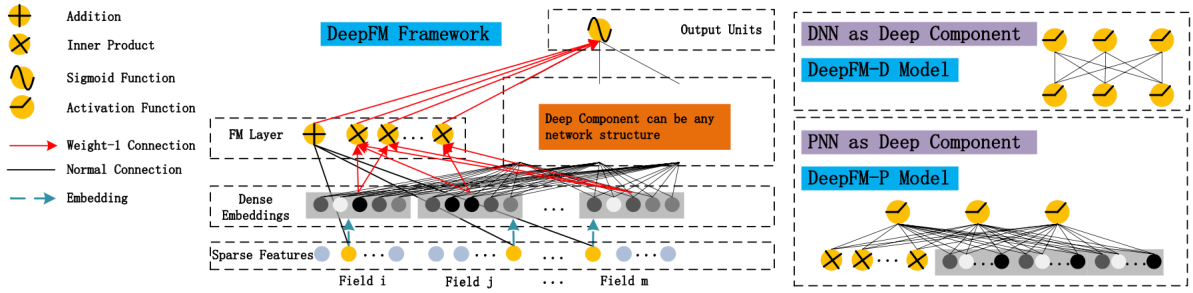
Fig. 1: Product-based Neural Network Architecture.

4.4.4 DeepFM

Guo, Huifeng, et al. "Deepfm: An end-to-end wide & deep learning framework for CTR prediction." arXiv preprint arXiv:1804.04950 (2018).

Retrieve from: https://arxiv.org/abs/1804.04950

> DeepFM, combines the power of factorization machines for recommendation and deep learning for feature learning in a new neural network architecture. Compared to the latest Wide & Deep model from Google, DeepFM has a shared raw feature input to both its "wide" and "deep" components, with no need of feature engineering besides raw features. DeepFM, as a general learning framework, can incorporate various network architectures in its deep component.



4.4.5 xDeepFM

Lian, Jianxun, et al. "xdeepfm: Combining explicit and implicit feature interactions for recommender systems." Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018.

Retrieve from: https://dl.acm.org/doi/abs/10.1145/3219819.3220023

> A novel Compressed Interaction Network (CIN), which aims to generate feature interactions in an explicit fashion and at the vector-wise level. We show that the CIN share some functionalities with convolutional neural networks (CNNs) and recurrent neural networks (RNNs). We further combine a CIN and a classical DNN into one unified model, and named this new model eXtreme Deep Factorization Machine (xDeepFM).

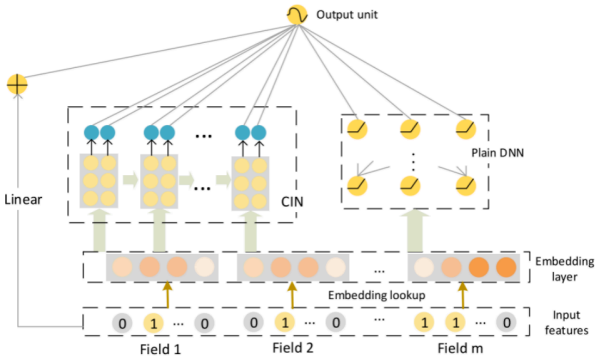


Figure 5: The architecture of xDeepFM.

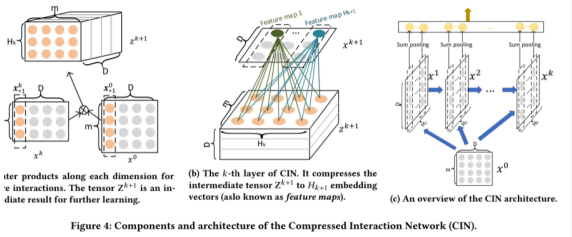


Figure 4: Components and architecture of the Compressed Interaction Network (CIN).

4.4.6 AFM

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

Retrieve from: https://arxiv.org/abs/1708.04617

Attentional Factorization Machine (AFM), which learns the importance of each feature interaction from data via a neural attention network. Extensive experiments on two real-world datasets demonstrate the effectiveness of AFM. Empirically, it is shown on regression task AFM betters FM with a 8.6% relative improvement, and consistently outperforms the state-of-the-art deep learning methods Wide&Deep and DeepCross with a much simpler structure and fewer model parameters.

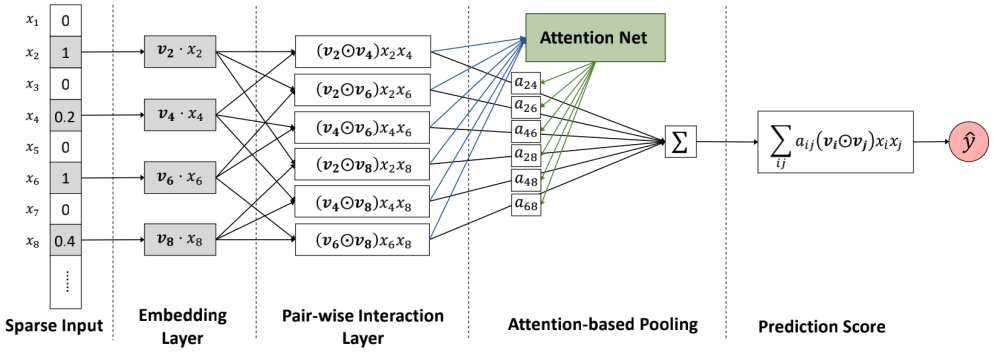


Figure 1: The neural network architecture of our proposed Attentional Factorization Machine model.

4.4.7 AutoInt

Song, Weiping, et al. "Autoint: Automatic feature interaction learning via self-attentive neural networks." Proceedings of the 28th ACM International Conference on Information and Knowledge Management. 2019.

Retrieve from: https://dl.acm.org/doi/abs/10.1145/3357384.3357925

AutoInt can be applied to both numerical and categorical input features. Specifically, we map both the numerical and categorical features into the same low-dimensional space. Afterwards, a multihead self-attentive neural network with residual connections is proposed to explicitly model the feature interactions in the lowdimensional space. With different layers of the multi-head selfattentive neural networks, different orders of feature combinations of input features can be modeled. The whole model can be efficiently fit on large-scale raw data in an end-to-end fashion.

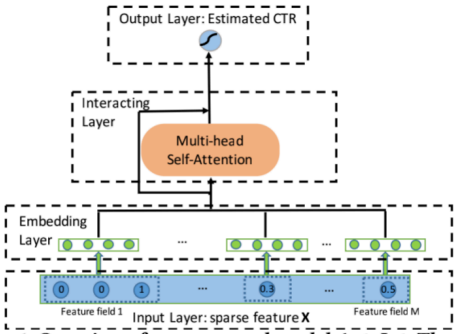


Figure 1: Overview of our proposed model AutoInt. The details of embedding layer and interacting layer are illustrated in Figure 2 and Figure 3 respectively.

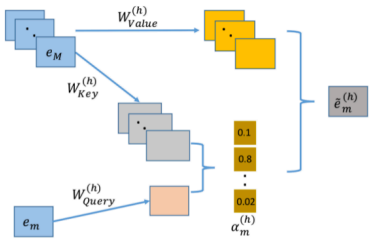


Figure 3: The architecture of interacting layer. Combinatorial features are conditioned on attention weights, i.e., $\alpha_m^{(h)}$.

4.4.8 FiBiNet

Huang, Tongwen, Zhiqi Zhang, and Junlin Zhang. "FiBiNET: combining feature importance and bilinear feature interaction for click-through rate prediction." Proceedings of the 13th ACM Conference on Recommender Systems. 2019.

Retrieve from: https://dl.acm.org/doi/abs/10.1145/3298689.3347043

> FiBiNET as an abbreviation for Feature Importance and Bilinear feature Interaction NETwork is proposed to dynamically learn the feature importance and fine-grained feature interactions. On the one hand, the FiBiNET can dynamically learn the importance of features via the Squeeze-Excitation network (SENET) mechanism; on the other hand, it is able to effectively learn the feature interactions via bilinear function.

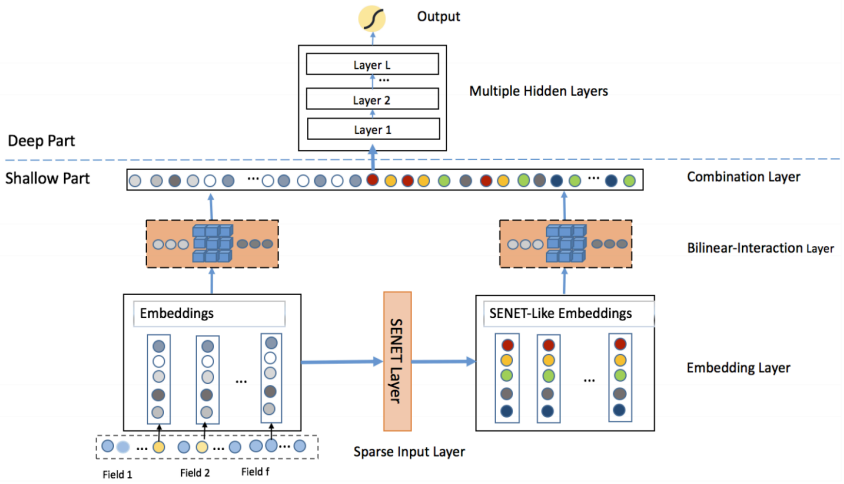


Figure 1: The architecture of our proposed FiBiNET

4.4.9 FGCNN

Liu, Bin, et al. "Feature generation by convolutional neural network for click-through rate prediction." The World Wide Web Conference. 2019.

Retrieve from: https://dl.acm.org/doi/abs/10.1145/3308558.3313497

> Feature Generation by Convolutional Neural Network (FGCNN) model with two components: Feature Generation and Deep Classifier. Feature Generation leverages the strength of CNN to generate local patterns and recombine them to generate new features. Deep Classifier adopts the structure of IPNN to learn interactions from the augmented feature space. Experimental results on three large-scale datasets show that FGCNN significantly outperforms nine state-of-the-art models. Moreover, when applying some state-of-the-art models as Deep Classifier, better performance is always achieved, showing the great compatibility of our FGCNN model. This work explores a novel direction for CTR predictions: it is quite useful to reduce the learning difficulties of DNN by automatically identifying important features.

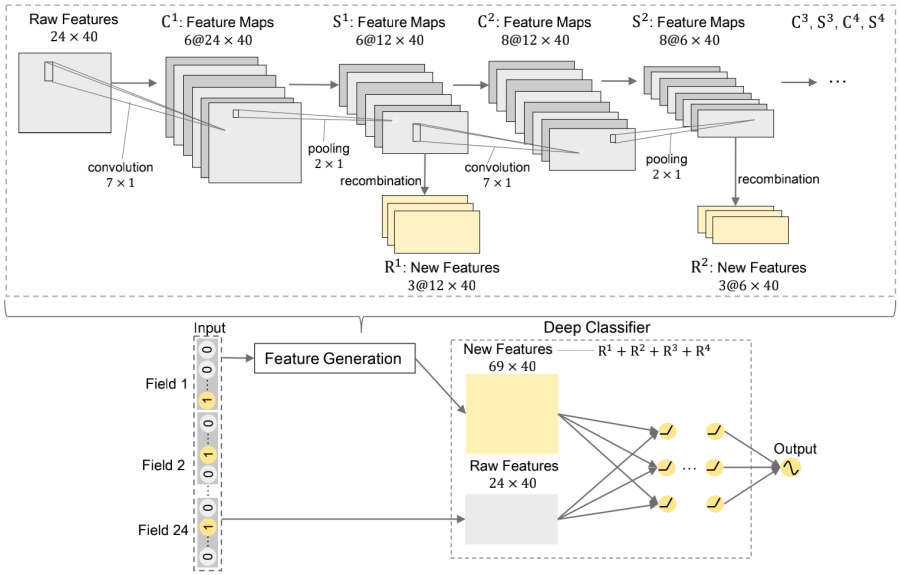


Figure 2: An overview of Feature Generation by Convolutional Neural Network Model (The hyper-parameters in the figure are the best setting of FGCNN on Avazu Dataset)

4.5 Layers

4.5.1 FM

Factorization Machine to model order-2 feature interactions.

Call arguments:

- x: A 3D tensor.

Input shape:

- 3D tensor with shape: (batch_size, field_size, embedding_size)

Output shape:

- 2D tensor with shape: (batch_size, 1)

References:

- [1] Rendle S. Factorization machines[C]//2010 IEEE International Conference on Data Mining. IEEE, 2010: 995-1000.
- [2] Guo H, Tang R, Ye Y, et al. Deepfm: An end-to-end wide & deep learning framework for CTR prediction[J]. arXiv preprint arXiv:1804.04950, 2018.

4.5.2 AFM

Attentional Factorization Machine (AFM), which learns the importance of each feature interaction from data via a neural attention network.

Arguments:

- hidden_factor: int, (default=16)
- activation_function : str, (default=' relu')
- kernel_regularizer : str or object, (default=None)
- dropout_rate: float, (default=0)

Call arguments:

- x: A list of 3D tensor.

Input shape:

- A list of 3D tensor with shape: (batch_size, 1, embedding_size)

Output shape:

- 2D tensor with shape: (batch_size, 1)

References:

- [1] Xiao J, Ye H, He X, et al. Attentional factorization machines: Learning the weight of feature interactions via attention networks[J]. arXiv preprint arXiv:1708.04617, 2017.
- [2] https://github.com/hexiangnan/attentional_factorization_machine

4.5.3 CIN

Compressed Interaction Network (CIN), with the following considerations: (1) interactions are applied at vector-wise level, not at bit-wise level; (2) high-order feature interactions is measured explicitly; (3) the complexity of network will not grow exponentially with the degree of interactions.

Arguments:

- cross_layer_size: tuple of int, (default = (128, 128,))
- activation: str, (default=' relu')
- use_residual: bool, (default=False)
- use_bias: bool, (default=False)
- direct: bool, (default=False)
- reduce_D:bool, (default=False)

Call arguments:

- x: A 3D tensor.

Input shape:

- A 3D tensor with shape: (batch_size, num_fields, embedding_size)

Output shape:

- 2D tensor with shape: (batch_size, *)

References:

- [1] Lian J, Zhou X, Zhang F, et al. xdeepfm: Combining explicit and implicit feature interactions for recommender systems[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 1754-1763.
- [2] https://github.com/Leavingseason/xDeepFM

4.5.4 MultiheadAttention

A multihead self-attentive nets with residual connections to explicitly model the feature interactions.

Arguments:

- num_head: int, (default=1)
- dropout_rate: float, (default=0)
- use_residual: bool, (default=True)

Call arguments:

- x: A 3D tensor.

Input shape:

- 3D tensor with shape: (batch_size, field_size, embedding_size)

Output shape:

- 3D tensor with shape: (batch_size, field_size, embedding_size*num_head)

References:

- [1] Song W, Shi C, Xiao Z, et al. Autoint: Automatic feature interaction learning via self-attentive neural networks[C]//Proceedings of the 28th ACM International Conference on Information and Knowledge Management. 2019: 1161-1170.
- [2] https://github.com/shichence/AutoInt

4.5.5 FGCNN

Feature Generation nets leverages the strength of CNN to generate local patterns and recombine them to generate new features.

Arguments:

- filters: int, the filters of convolutional layer
- kernel_height: int, the height of kernel_size of convolutional layer
- new_filters: int, the number of new features' map in recombination layer
- pool_height: int, the height of pool_size of pooling layer
- activation: str, (default='tanh')

Call arguments:

- x: A 4D tensor.

Input shape:

- 4D tensor with shape: (batch_size, field_size, embedding_size, 1)

Output shape:

- pooling_output - 4D tensor
- new_features - 3D tensor with shape: (batch_size, field_size*new_filters, embedding_size)

References:

- [1] Liu B, Tang R, Chen Y, et al. Feature generation by convolutional neural network for click-through rate prediction[C]//The World Wide Web Conference. 2019: 1119-1129.

4.5.6 SENET

SENET layer can dynamically increase the weights of important features and decrease the weights of uninformative features to let the model pay more attention to more important features.

Arguments:

- pooling_op: str, (default='mean'). Pooling methods to squeeze the original embedding E into a statistic vector Z
- reduction_ratio: float, (default=3). Hyper-parameter for dimensionality-reduction

Call arguments:

- x: A 3D tensor.

Input shape:

- 3D tensor with shape: (batch_size, field_size, embedding_size)

Output shape:

- 3D tensor with shape: (batch_size, field_size, embedding_size)

References:

- [1] Huang T, Zhang Z, Zhang J. FiBiNET: combining feature importance and bilinear feature interaction for click-through rate prediction[C]//Proceedings of the 13th ACM Conference on Recommender Systems. 2019: 169-177.

4.5.7 BilinearInteraction

The Bilinear-Interaction layer combines the inner product and Hadamard product to learn the feature interactions.

Arguments:

- bilinear_type: str, (default=' field_interaction'). The type of bilinear functions
  - `field_interaction`
  - `field_all`
  - `field_each`

Call arguments:

- x: A 3D tensor.

Input shape:

- 3D tensor with shape: (batch_size, field_size, embedding_size)

Output shape:

- 3D tensor with shape: (batch_size, *, embedding_size)

References:

- [1] Huang T, Zhang Z, Zhang J. FiBiNET: combining feature importance and bilinear feature interaction for click-through rate prediction[C]//Proceedings of the 13th ACM Conference on Recommender Systems. 2019: 169-177.

4.5.8 Cross

The cross network is composed of cross layers to apply explicit feature crossing in an efficient way.

Arguments:

- num_cross_layer: int, (default=2). The number of cross layers

Call arguments:

- x: A 2D tensor.

Input shape:

- 2D tensor with shape: (batch_size, field_size)

Output shape:

- 2D tensor with shape: (batch_size, field_size)

References:

- [1] Wang R, Fu B, Fu G, et al. Deep & cross network for ad click predictions[M]//Proceedings of the ADKDD' 17. 2017: 1-7.

4.5.9 InnerProduct

InnerProduct layer used in PNN

Call arguments:

- x: A list of 3D tensor.

Input shape:

- A list of 3D tensor with shape (batch_size, 1, embedding_size)

Output shape:

- 2D tensor with shape: (batch_size, num_fields*(num_fields-1)/2)

References:

- [1] Qu Y, Cai H, Ren K, et al. Product-based neural networks for user response prediction[C]//2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016: 1149-1154.
- [2] Qu Y, Fang B, Zhang W, et al. Product-based neural networks for user response prediction over multi-field categorical data[J]. ACM Transactions on Information Systems (TOIS), 2018, 37(1): 1-35.
- [3] https://github.com/Atomu2014/product-nets

4.5.10 OuterProduct

OuterProduct layer used in PNN

Arguments:

- outer_product_kernel_type: str, (default=' mat'). The type of outer product kernel
  - `mat`
  - `vec`
  - `num` **Call arguments**:

- x: A list of 3D tensor.

Input shape:

- A list of 3D tensor with shape (batch_size, 1, embedding_size)

Output shape:

- 2D tensor with shape: (batch_size, num_fields*(num_fields-1)/2)

References:

- [1] Qu Y, Cai H, Ren K, et al. Product-based neural networks for user response prediction[C]//2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016: 1149-1154.
- [2] Qu Y, Fang B, Zhang W, et al. Product-based neural networks for user response prediction over multi-field categorical data[J]. ACM Transactions on Information Systems (TOIS), 2018, 37(1): 1-35.
- [3] https://github.com/Atomu2014/product-nets

4.6 deeptables

4.6.1 deeptables package

Subpackages

deeptables.datasets package

Submodules

deeptables.datasets.dsutils module

Module contents

deeptables.eda package

Submodules

deeptables.eda.utils module

Module contents

deeptables.ensemble package

deeptables.preprocessing.utils module

Module contents

deeptables.utils package

Submodules

deeptables.utils.batch_trainer module

deeptables.utils.consts module

deeptables.utils.dart_early_stopping module

deeptables.utils.dt_logging module

deeptables.utils.gpu module

deeptables.utils.quicktest module

Module contents

Module contents

4.7 FAQ

4.7.1 How…

CHAPTER 5

Indices and tables

- genindex
- modindex
- search