

tf.contrib.estimator.DNNEstimator

Contents

Class DNNEstimator

Properties

config

model_dir

Class DNNEstimator

Inherits From: [Estimator](#)Defined in [tensorflow/contrib/estimator/python/estimator/dnn.py](#).

An estimator for TensorFlow DNN models with user-specified head.

Example:

```

sparse_feature_a = sparse_column_with_hash_bucket(...)
sparse_feature_b = sparse_column_with_hash_bucket(...)

sparse_feature_a_emb = embedding_column(sparse_id_column=sparse_feature_a,
                                         ...)
sparse_feature_b_emb = embedding_column(sparse_id_column=sparse_feature_b,
                                         ...)

estimator = DNNEstimator(
    head=tf.contrib.estimator.multi_label_head(n_classes=3),
    feature_columns=[sparse_feature_a_emb, sparse_feature_b_emb],
    hidden_units=[1024, 512, 256])

# Or estimator using the ProximalAdagradOptimizer optimizer with
# regularization.
estimator = DNNEstimator(
    head=tf.contrib.estimator.multi_label_head(n_classes=3),
    feature_columns=[sparse_feature_a_emb, sparse_feature_b_emb],
    hidden_units=[1024, 512, 256],
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=0.1,
        l1_regularization_strength=0.001
    ))

# Input builders
def input_fn_train: # returns x, y
    pass
estimator.train(input_fn=input_fn_train, steps=100)

def input_fn_eval: # returns x, y
    pass
metrics = estimator.evaluate(input_fn=input_fn_eval, steps=10)
def input_fn_predict: # returns x, None
    pass
predictions = estimator.predict(input_fn=input_fn_predict)

```

Input of **train** and **evaluate** should have following features, otherwise there will be a **KeyError** :

- if **weight_column** is not **None**, a feature with **key=weight_column** whose value is a **Tensor** .
- for each **column** in **feature_columns** :
 - if **column** is a **_CategoricalColumn**, a feature with **key=column.name** whose **value** is a **SparseTensor** .
 - if **column** is a **_WeightedCategoricalColumn**, two features: the first with **key** the id column name, the second with **key** the weight column name. Both features' **value** must be a **SparseTensor** .
 - if **column** is a **_DenseColumn**, a feature with **key=column.name** whose **value** is a **Tensor** .

Loss and predicted output are determined by the specified head.

Properties

config

model_dir

model_fn

Returns the model_fn which is bound to self.params.

Returns:

The model_fn with following signature: `def model_fn(features, labels, mode, config)`

params

Methods

`__init__`

```
__init__(
    head,
    hidden_units,
    feature_columns,
    model_dir=None,
    optimizer='Adagrad',
    activation_fn=tf.nn.relu,
    dropout=None,
    input_layer_partitioner=None,
    config=None
)
```

Initializes a `DNNClassifier` instance.

Args:

- `head`: A `_Head` instance constructed with a method such as `tf.contrib.estimator.multi_label_head`.
- `hidden_units`: Iterable of number hidden units per layer. All layers are fully connected. Ex. `[64, 32]` means first layer has 64 nodes and second one has 32.
- `feature_columns`: An iterable containing all the feature columns used by the model. All items in the set should be instances of classes derived from `_FeatureColumn`.
- `model_dir`: Directory to save model parameters, graph and etc. This can also be used to load checkpoints from the directory into a estimator to continue training a previously saved model.
- `optimizer`: An instance of `tf.Optimizer` used to train the model. Defaults to Adagrad optimizer.
- `activation_fn`: Activation function applied to each layer. If `None`, will use `tf.nn.relu`.
- `dropout`: When not `None`, the probability we will drop out a given coordinate.
- `input_layer_partitioner`: Optional. Partitioner for input layer. Defaults to `min_max_variable_partitioner` with `min_slice_size` `64 << 20`.
- `config`: `RunConfig` object to configure the runtime settings.

evaluate

```
evaluate(
    input_fn,
    steps=None,
    hooks=None,
    checkpoint_path=None,
    name=None
)
```

Evaluates the model given evaluation data `input_fn`.

For each step, calls `input_fn`, which returns one batch of data. Evaluates until: - `steps` batches are processed, or -

`input_fn` raises an end-of-input exception (`OutOfRangeError` or `StopIteration`).

Args:

- `input_fn` : Input function returning a tuple of: features - Dictionary of string feature name to `Tensor` or `SparseTensor` . labels - `Tensor` or dictionary of `Tensor` with labels.
- `steps` : Number of steps for which to evaluate model. If `None` , evaluates until `input_fn` raises an end-of-input exception.
- `hooks` : List of `SessionRunHook` subclass instances. Used for callbacks inside the evaluation call.
- `checkpoint_path` : Path of a specific checkpoint to evaluate. If `None` , the latest checkpoint in `model_dir` is used.
- `name` : Name of the evaluation if user needs to run multiple evaluations on different data sets, such as on training data vs test data. Metrics for different evaluations are saved in separate folders, and appear separately in tensorboard.

Returns:

A dict containing the evaluation metrics specified in `model_fn` keyed by name, as well as an entry `global_step` which contains the value of the global step for which this evaluation was performed.

Raises:

- `ValueError` : If `steps <= 0` .
- `ValueError` : If no model has been trained, namely `model_dir` , or the given `checkpoint_path` is empty.

`export_savedmodel`

```
export_savedmodel(  
    export_dir_base,  
    serving_input_receiver_fn,  
    assets_extra=None,  
    as_text=False,  
    checkpoint_path=None  
)
```

Exports inference graph as a SavedModel into given dir.

This method builds a new graph by first calling the `serving_input_receiver_fn` to obtain feature `Tensor` s, and then calling this `Estimator` 's `model_fn` to generate the model graph based on those features. It restores the given checkpoint (or, lacking that, the most recent checkpoint) into this graph in a fresh session. Finally it creates a timestamped export directory below the given `export_dir_base`, and writes a `SavedModel` into it containing a single `MetaGraphDef` saved from this session.

The exported `MetaGraphDef` will provide one `SignatureDef` for each element of the `export_outputs` dict returned from the `model_fn`, named using the same keys. One of these keys is always `signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY`, indicating which signature will be served when a serving request does not specify one. For each signature, the outputs are provided by the corresponding `ExportOutput` s, and the inputs are always the input receivers provided by the `serving_input_receiver_fn`.

Extra assets may be written into the SavedModel via the `extra_assets` argument. This should be a dict, where each key gives a destination path (including the filename) relative to the `assets.extra` directory. The corresponding value gives the full path of the source file to be copied. For example, the simple case of copying a single file without renaming it is specified as `{ 'my_asset_file.txt': ' /path/to/my_asset_file.txt' }` .

Args:

- `export_dir_base` : A string containing a directory in which to create timestamped subdirectories containing exported SavedModels.
- `serving_input_receiver_fn` : A function that takes no argument and returns a `ServingInputReceiver` .
- `assets_extra` : A dict specifying how to populate the assets.extra directory within the exported SavedModel, or `None` if no extra assets are needed.
- `as_text` : whether to write the SavedModel proto in text format.
- `checkpoint_path` : The checkpoint path to export. If `None` (the default), the most recent checkpoint found within the model directory is chosen.

Returns:

The string path to the exported directory.

Raises:

- `ValueError` : if no `serving_input_receiver_fn` is provided, no `export_outputs` are provided, or no checkpoint can be found.

get_variable_names

```
get_variable_names()
```

Returns list of all variable names in this model.

Returns:

List of names.

Raises:

- `ValueError` : If the Estimator has not produced a checkpoint yet.

get_variable_value

```
get_variable_value(name)
```

Returns value of the variable given by name.

Args:

- `name` : string or a list of string, name of the tensor.

Returns:

Numpy array - value of the tensor.

Raises:

- `ValueError` : If the Estimator has not produced a checkpoint yet.

latest_checkpoint

```
latest_checkpoint()
```

Finds the filename of latest saved checkpoint file in `model_dir`.

Returns:

The full path to the latest checkpoint or `None` if no checkpoint was found.

predict

```
predict(  
    input_fn,  
    predict_keys=None,  
    hooks=None,  
    checkpoint_path=None  
)
```

Yields predictions for given features.

Args:

- `input_fn` : Input function returning features which is a dictionary of string feature name to `Tensor` or `SparseTensor`. If it returns a tuple, first item is extracted as features. Prediction continues until `input_fn` raises an end-of-input exception (`OutOfRangeError` or `StopIteration`).
- `predict_keys` : list of `str`, name of the keys to predict. It is used if the `EstimatorSpec.predictions` is a `dict`. If `predict_keys` is used then rest of the predictions will be filtered from the dictionary. If `None`, returns all.
- `hooks` : List of `SessionRunHook` subclass instances. Used for callbacks inside the prediction call.
- `checkpoint_path` : Path of a specific checkpoint to predict. If `None`, the latest checkpoint in `model_dir` is used.

Yields:

Evaluated values of `predictions` tensors.

Raises:

- `ValueError` : Could not find a trained model in `model_dir`.
- `ValueError` : if batch length of predictions are not same.
- `ValueError` : If there is a conflict between `predict_keys` and `predictions`. For example if `predict_keys` is not `None` but `EstimatorSpec.predictions` is not a `dict`.

train

```

train(
    input_fn,
    hooks=None,
    steps=None,
    max_steps=None,
    saving_listeners=None
)

```

Trains a model given training data `input_fn`.

Args:

- `input_fn`: Input function returning a tuple of: features - `Tensor` or dictionary of string feature name to `Tensor`. labels - `Tensor` or dictionary of `Tensor` with labels.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the training loop.
- `steps`: Number of steps for which to train model. If `None`, train forever or train until `input_fn` generates the `OutOfRange` error or `StopIteration` exception. 'steps' works incrementally. If you call two times `train(steps=10)` then training occurs in total 20 steps. If `OutOfRange` or `StopIteration` occurs in the middle, training stops before 20 steps. If you don't want to have incremental behavior please set `max_steps` instead. If set, `max_steps` must be `None`.
- `max_steps`: Number of total steps for which to train model. If `None`, train forever or train until `input_fn` generates the `OutOfRange` error or `StopIteration` exception. If set, `steps` must be `None`. If `OutOfRange` or `StopIteration` occurs in the middle, training stops before `max_steps` steps. Two calls to `train(steps=100)` means 200 training iterations. On the other hand, two calls to `train(max_steps=100)` means that the second call will not do any iteration since first call did all 100 steps.
- `saving_listeners`: list of `CheckpointSaverListener` objects. Used for callbacks that run immediately before or after checkpoint savings.

Returns:

`self`, for chaining.

Raises:

- `ValueError`: If both `steps` and `max_steps` are not `None`.
- `ValueError`: If either `steps` or `max_steps` is ≤ 0 .

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 2, 2017.

Stay Connected

[Blog](#)

[GitHub](#)

[Twitter](#)

Support

[Issue Tracker](#)

[Release Notes](#)

[Stack Overflow](#)

English

[Terms](#) | [Privacy](#)