tf.contrib.gan.estimator.GANEstimator

Contents

Class **`GANEstimator`**

Inherits From: **`Estimator`**

Aliases:

- Class **`tf.contrib.gan.estimator.GANEstimator`**
- Class **`tf.contrib.gan.estimator.gan_estimator.GANEstimator`**

Defined in **`tensorflow/contrib/gan/python/estimator/python/gan_estimator_impl.py`**.

An estimator for Generative Adversarial Networks (GANs).

This Estimator is backed by TFGAN.

Example:

```
    import tensorflow as tf
    tfgan = tf.contrib.gan

    # See TFGAN's `train.py` for a description of the generator and
    # discriminator API.
    def generator_fn(generator_inputs):
      ...
      return generated_data

    def discriminator_fn(data, conditioning):
      ...
      return logits

    # Create GAN estimator.
    gan_estimator = estimator.GANEstimator(
        model_dir,
        generator_fn=generator_fn,
        discriminator_fn=discriminator_fn,
        generator_loss_fn=tfgan.losses.wasserstein_generator_loss,
        discriminator_loss_fn=tfgan.losses.wasserstein_discriminator_loss,
        generator_optimizer=tf.train.AdamOptimizier(0.1, 0.5),
        discriminator_optimizer=tf.train.AdamOptimizier(0.1, 0.5))

    # Train estimator.
    gan_estimator.train(train_input_fn, steps)

    # Evaluate resulting estimator.
```

~~~
    # Evaluate resulting estimator.
    gan_estimator.evaluate(eval_input_fn)

    # Generate samples from generator.
    predictions = np.array([
        x for x in gan_estimator.predict(predict_input_fn)])

## Properties

<h3 id="config"><code>config</code></h3>

<h3 id="model_dir"><code>model_dir</code></h3>

<h3 id="model_fn"><code>model_fn</code></h3>

Returns the model_fn which is bound to self.params.

#### Returns:

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

<h3 id="params"><code>params</code></h3>

## Methods

<h3 id="__init__"><code>__init__</code></h3>

``` python
__init__(
    model_dir=None,
    generator_fn=None,
    discriminator_fn=None,
    generator_loss_fn=None,
    discriminator_loss_fn=None,
    generator_optimizer=None,
    discriminator_optimizer=None,
    add_summaries=None,
    use_loss_summaries=True,
    config=None
)
~~~

Initializes a GANEstimator instance.

Args:

- `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.
- `generator_fn`: A python function that takes a Tensor, Tensor list, or Tensor dictionary as inputs and returns the outputs of the GAN generator. See `TFGAN` for more details and examples.
- `discriminator_fn`: A python function that takes the output of `generator_fn` or real data in the GAN setup, and `generator_inputs`. Outputs a Tensor in the range [-inf, inf]. See `TFGAN` for more details and examples.
- `generator_loss_fn`: The loss function on the generator. Takes a `GANModel` tuple.
- `discriminator_loss_fn`: The loss function on the discriminator. Takes a `GANModel` tuple.
- `generator_optimizer`: The optimizer for generator updates, or a function that takes no arguments and returns an optimizer. This function will be called when the default graph is the `GANEstimator`'s graph, so utilities like `tf.contrib.framework.get_or_create_global_step` will work.
- `discriminator_optimizer`: Same as `generator_optimizer`, but for the discriminator updates.
- `add_summaries`: `None`, a single `SummaryType`, or a list of `SummaryType`.

- `use_loss_summaries`: If `True`, add loss summaries. If `False`, does not. If `None`, uses defaults.
- `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.