# tf.contrib.learn.Experiment

**Contents**

## Class **`Experiment`**

Defined in `tensorflow/contrib/learn/python/learn/experiment.py`.

See the guide: Learn (contrib) > Distributed training utilities

Experiment is a class containing all information needed to train a model.

After an experiment is created (by passing an Estimator and inputs for training and evaluation), an Experiment instance knows how to invoke training and eval loops in a sensible fashion for distributed training.

## Properties

### `estimator`

### `eval_metrics`

### `eval_steps`

### `train_steps`

## Methods

### `__init__`

```
__init__(
    estimator,
    train_input_fn,
    eval_input_fn,
    eval_metrics=None,
    train_steps=None,
    eval_steps=100,
    train_monitors=None,
    eval_hooks=None,
    local_eval_frequency=None,
    eval_delay_secs=120,
    continuous_eval_throttle_secs=60,
    min_eval_frequency=None,
    delay_workers_by_global_step=False,
    export_strategies=None,
    train_steps_per_iteration=None,
    checkpoint_and_export=False
)
```

Constructor for `Experiment`. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-10-23. Instructions for updating: local_eval_frequency is deprecated as local_run will be renamed to train_and_evaluate. Use min_eval_frequency and call train_and_evaluate instead. Note, however, that the default for min_eval_frequency is 1, meaning models will be evaluated every time a new checkpoint is available. In contrast, the default for local_eval_frequency is None, resulting in evaluation occurring only after training has completed. min_eval_frequency is ignored when calling the deprecated local_run.

Creates an Experiment instance. None of the functions passed to this constructor are executed at construction time. They are stored and used when a method is executed which requires it.

## Args:

- `estimator`: Object implementing Estimator interface, which could be a combination of ${tf.contrib.learn.Trainable} and ${tf.contrib.learn.Evaluable} (deprecated), or ${tf.estimator.`Estimator}.
- `train_input_fn`: function, returns features and labels for training.
- `eval_input_fn`: function, returns features and labels for evaluation. If `eval_steps` is `None`, this should be configured only to produce for a finite number of batches (generally, 1 epoch over the evaluation data).
- `eval_metrics`: `dict` of string, metric function. If `None`, default set is used. This should be `None` if the `estimator` is ${tf.estimator.Estimator}. If metrics are provided they will be *appended* to the default set.
- `train_steps`: Perform this many steps of training. `None`, the default, means train forever.
- `eval_steps`: `evaluate` runs until input is exhausted (or another exception is raised), or for `eval_steps` steps, if specified.
- `train_monitors`: A list of monitors to pass to the `Estimator`'s `fit` function.
- `eval_hooks`: A list of `SessionRunHook` hooks to pass to the `Estimator`'s `evaluate` function.
- `local_eval_frequency`: (applies only to local_run) Frequency of running eval in steps. If `None`, runs evaluation only at the end of training.
- `eval_delay_secs`: Start evaluating after waiting for this many seconds.
- `continuous_eval_throttle_secs`: Do not re-evaluate unless the last evaluation was started at least this many seconds ago for continuous_eval().
- `min_eval_frequency`: (applies only to train_and_evaluate). the minimum number of steps between evaluations. Of course, evaluation does not occur if no new snapshot is available, hence, this is the minimum. If 0, the evaluation will only happen after training. If None, defaults to 1, unless model_dir is on GCS, in which case the default is 1000.
- `delay_workers_by_global_step`: if `True` delays training workers based on global step instead of time.

- `export_strategies`: Iterable of `ExportStrategy`s, or a single one, or `None`.
- `train_steps_per_iteration`: (applies only to continuous_train_and_eval). Perform this many (integer) number of train steps for each training-evaluation iteration. With a small value, the model will be evaluated more frequently with more checkpoints saved. If `None`, will use a default value (which is smaller than `train_steps` if provided).
- `checkpoint_and_export`: (applies only to train_and_evaluate). If `True`, performs intermediate model checkpoints and exports during the training process, rather than only once model training is complete. This parameter is experimental and may be changed or removed in the future. Setting this parameter leads to the following: the value of `min_eval_frequency` will be ignored, and the number of steps between evaluations and exports will instead be determined by the Estimator configuration parameters `save_checkpoints_secs` and `save_checkpoints_steps`. Also, this parameter leads to the creation of a default `CheckpointSaverHook` instead of a `ValidationMonitor`, so the provided `train_monitors` will need to be adjusted accordingly.

Raises:

- `ValueError`: if `estimator` does not implement Estimator interface, or if export_strategies has the wrong type.

### continuous_eval

```
continuous_eval(
    delay_secs=None,
    throttle_delay_secs=None,
    evaluate_checkpoint_only_once=True,
    continuous_eval_predicate_fn=None,
    name='continuous'
)
```

### continuous_eval_on_train_data

```
continuous_eval_on_train_data(
    delay_secs=None,
    throttle_delay_secs=None,
    continuous_eval_predicate_fn=None,
    name='continuous_on_train_data'
)
```

### continuous_train_and_eval

```
continuous_train_and_eval(
    *args,
    **kwargs
)
```

Interleaves training and evaluation. (experimental)

THIS FUNCTION IS EXPERIMENTAL. It may change or be removed at any time, and without warning.

The frequency of evaluation is controlled by the `train_steps_per_iteration` (via constructor). The model will be first trained for `train_steps_per_iteration`, and then be evaluated in turns.

This method is intended for single machine usage.

This differs from `train_and_evaluate` as follows:

1. The procedure will have train and evaluation in turns. The model will be trained for a number of steps (usually smaller than `train_steps` if provided) and then be evaluated. `train_and_evaluate` will train the model for

`train_steps` (no small training iterations).
2. Due to the different approach this schedule takes, it leads to two differences in resource control. First, the resources (e.g., memory) used by training will be released before evaluation ( `train_and_evaluate` takes double resources). Second, more checkpoints will be saved as a checkpoint is generated at the end of each training iteration.
3. As the estimator.train starts from scratch (new graph, new states for input, etc) at each iteration, it is recommended to have the `train_steps_per_iteration` larger. It is also recommended to shuffle your input.

#### Args:

- `continuous_eval_predicate_fn` : A predicate function determining whether to continue after each iteration. `predicate_fn` takes the evaluation results as its arguments. At the beginning of evaluation, the passed eval results will be None so it's expected that the predicate function handles that gracefully. When `predicate_fn` is not specified, this will run in an infinite loop or exit when global_step reaches `train_steps` .

#### Returns:

A tuple of the result of the `evaluate` call to the `Estimator` and the export results using the specified `ExportStrategy` .

#### Raises:

- `ValueError` : if `continuous_eval_predicate_fn` is neither None nor callable.

### `evaluate`

```
evaluate(
    delay_secs=None,
    name=None
)
```

Evaluate on the evaluation data.

Runs evaluation on the evaluation data and returns the result. Runs for `self._eval_steps` steps, or if it's `None` , then run until input is exhausted or another exception is raised. Start the evaluation after `delay_secs` seconds, or if it's `None` , defaults to using `self._eval_delay_secs` seconds.

#### Args:

- `delay_secs` : Start evaluating after this many seconds. If `None` , defaults to using `self._eval_delays_secs` .
- `name` : Gives the name to the evauation for the case multiple evaluation is run for the same experiment.

#### Returns:

The result of the `evaluate` call to the `Estimator` .

### `extend_train_hooks`

```
extend_train_hooks(additional_hooks)
```

Extends the hooks for training.

### `local_run`

```
local_run()
```

DEPRECATED FUNCTION

THIS FUNCTION IS DEPRECATED. It will be removed after 2016-10-23. Instructions for updating: local_run will be renamed to train_and_evaluate and the new default behavior will be to run evaluation every time there is a new checkpoint.

### `reset_export_strategies`

```
reset_export_strategies(new_export_strategies=None)
```

Resets the export strategies with the `new_export_strategies`.

#### Args:

- `new_export_strategies`: A new list of `ExportStrategy`s, or a single one, or None.

#### Returns:

The old export strategies.

### `run_std_server`

```
run_std_server()
```

Starts a TensorFlow server and joins the serving thread.

Typically used for parameter servers.

#### Raises:

- `ValueError`: if not enough information is available in the estimator's config to create a server.

### `test`

```
test()
```

Tests training, evaluating and exporting the estimator for a single step.

#### Returns:

The result of the `evaluate` call to the `Estimator`.

### `train`

```
train(delay_secs=None)
```

Fit the estimator using the training data.

Train the estimator for `self._train_steps` steps, after waiting for `delay_secs` seconds. If `self._train_steps` is `None`, train forever.

#### Args:

* `delay_secs`: Start training after this many seconds.

#### Returns:

The trained estimator.

### `train_and_evaluate`

```
train_and_evaluate()
```

Interleaves training and evaluation.

The frequency of evaluation is controlled by the constructor arg `min_eval_frequency`. When this parameter is 0, evaluation happens only after training has completed. Note that evaluation cannot happen more frequently than checkpoints are taken. If no new snapshots are available when evaluation is supposed to occur, then evaluation doesn't happen for another `min_eval_frequency` steps (assuming a checkpoint is available at that point). Thus, settings `min_eval_frequency` to 1 means that the model will be evaluated everytime there is a new checkpoint.

This is particular useful for a "Master" task in the cloud, whose responsibility it is to take checkpoints, evaluate those checkpoints, and write out summaries. Participating in training as the supervisor allows such a task to accomplish the first and last items, while performing evaluation allows for the second.

#### Returns:

The result of the `evaluate` call to the `Estimator` as well as the export results using the specified `ExportStrategy`.