tf.estimator.train_and_evaluate

```
train_and_evaluate(
    estimator,
    train_spec,
    eval_spec
)
```

Defined in `tensorflow/python/estimator/training.py`.

Train and evaluate the `estimator`.

This utility function trains, evaluates, and (optionally) exports the model by using the given `estimator`. All training related specification is held in `train_spec`, including training `input_fn` and training max steps, etc. All evaluation and export related specification is held in `eval_spec`, including evaluation `input_fn`, steps, etc.

This utility function provides consistent behavior for both local (non-distributed) and distributed configurations. Currently, the only supported distributed training configuration is between-graph replication.

Overfitting: In order to avoid overfitting, it is recommended to set up the training `input_fn` to shuffle the training data properly. It is also recommended to train the model a little longer, say multiple epochs, before performing evaluation, as the input pipeline starts from scratch for each training. It is particularly important for local training and evaluation.

Stop condition: In order to support both distributed and non-distributed configuration reliably, the only supported stop condition for model training is `train_spec.max_steps`. If `train_spec.max_steps` is `None`, the model is trained forever. *Use with care* if model stop condition is different. For example, assume that the model is expected to be trained with one epoch of training data, and the training `input_fn` is configured to throw `OutOfRangeError` after going through one epoch, which stops the `Estimator.train`. For a three-training-worker distributed configuration, each training worker is likely to go through the whole epoch independently. So, the model will be trained with three epochs of training data instead of one epoch.

Example of local (non-distributed) training:

```
# Set up feature columns.
categorial_feature_a = categorial_column_with_hash_bucket(...)
categorial_feature_a_emb = embedding_column(
    categorical_column=categorial_feature_a, ...)
...  # other feature columns

estimator = DNNClassifier(
    feature_columns=[categorial_feature_a_emb, ...],
    hidden_units=[1024, 512, 256])

# Or set up the model directory
#   estimator = DNNClassifier(
#       config=tf.estimator.RunConfig(
#           model_dir='/my_model', save_summary_steps=100),
#       feature_columns=[categorial_feature_a_emb, ...],
#       hidden_units=[1024, 512, 256])

# Input pipeline for train and evaluate.
def train_input_fn: # returns x, y
  # please shuffle the data.
  pass
def eval_input_fn_eval: # returns x, y
  pass

train_spec = tf.estimator.TrainSpec(input_fn=train_input_fn, max_steps=1000)
eval_spec = tf.estimator.EvalSpec(input_fn=eval_input_fn)

tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

Example of distributed training:

Regarding the example of distributed training, the code above can be used without a change (Please do make sure that the `RunConfig.model_dir` for all workers is set to the same directory, i.e., a shared file system all workers can read and write). The only extra work to do is setting the environment variable `TF_CONFIG` properly for each worker correspondingly.

Also see: https://www.tensorflow.org/deploy/distributed

Setting environment variable depends on the platform. For example, on Linux, it can be done as follows (`$` is the shell prompt):

```
$ TF_CONFIG='<replace_with_real_content>' python train_model.py
```

For the content in `TF_CONFIG`, assume that the training cluster spec looks like:

```
cluster = {"chief": ["host0:2222"],
           "worker": ["host1:2222", "host2:2222", "host3:2222"],
           "ps": ["host4:2222", "host5:2222"]}
```

Example of `TF_CONFIG` for chief training worker (must have one and only one):

```
# This should be a JSON string, which is set as environment variable. Usually
# the cluster manager handles that.
TF_CONFIG='{
    "cluster": {
        "chief": ["host0:2222"],
        "worker": ["host1:2222", "host2:2222", "host3:2222"],
        "ps": ["host4:2222", "host5:2222"]
    },
    "task": {"type": "chief", "index": 0}
}'
```

Note that the chief worker also does the model training job, similar to other non-chief training workers (see next paragraph). In addition to the model training, it manages some extra work, e.g., checkpoint saving and restoring, writing summaries, etc.

Example of `TF_CONFIG` for non-chief training worker (optional, could be multiple):

```
# This should be a JSON string, which is set as environment variable. Usually
# the cluster manager handles that.
TF_CONFIG='{
    "cluster": {
        "chief": ["host0:2222"],
        "worker": ["host1:2222", "host2:2222", "host3:2222"],
        "ps": ["host4:2222", "host5:2222"]
    },
    "task": {"type": "worker", "index": 0}
}'
```

where the `task.index` should be set as 0, 1, 2, in this example, respectively for non-chief training workers.

Example of `TF_CONFIG` for parameter server, aka ps (could be multiple):

```
# This should be a JSON string, which is set as environment variable. Usually
# the cluster manager handles that.
TF_CONFIG='{
    "cluster": {
        "chief": ["host0:2222"],
        "worker": ["host1:2222", "host2:2222", "host3:2222"],
        "ps": ["host4:2222", "host5:2222"]
    },
    "task": {"type": "ps", "index": 0}
}'
```

where the `task.index` should be set as 0 and 1, in this example, respectively for parameter servers.

Example of `TF_CONFIG` for evaluator task. Evaluator is a special task that is not part of the training cluster. There could be only one. It is used for model evaluation.

```
# This should be a JSON string, which is set as environment variable. Usually
# the cluster manager handles that.
TF_CONFIG='{
    "cluster": {
        "chief": ["host0:2222"],
        "worker": ["host1:2222", "host2:2222", "host3:2222"],
        "ps": ["host4:2222", "host5:2222"]
    },
    "task": {"type": "evaluator", "index": 0}
}'
```

Args:

- `estimator`: An `Estimator` instance to train and evaluate.
- `train_spec`: A `TrainSpec instance to specify the training specification.
- `eval_spec`: A `EvalSpec instance to specify the evaluation and export specification.

Raises:

- `ValueError`: if environment variable `TF_CONFIG` is incorrectly set.