adanet Documentation

Release [0.5.0]

AdaNet Authors

Dec 18, 2018

Package Reference

AdaNet: Fast and flexible AutoML with learning guarantees.

AdaNet is a lightweight TensorFlow-based framework for automatically learning high-quality models with minimal expert intervention. AdaNet builds on recent AutoML efforts to be fast and flexible while providing learning guarantees. Importantly, AdaNet provides a general framework for not only learning a neural network architecture, but also for learning to ensemble to obtain even better models.

This project is based on the *AdaNet algorithm*, presented in "AdaNet: Adaptive Structural Learning of Artificial Neural Networks" at ICML 2017, for learning the structure of a neural network as an ensemble of subnetworks.

AdaNet has the following goals:

- *Ease of use*: Provide familiar APIs (e.g. Keras, Estimator) for training, evaluating, and serving models.
- *Speed*: Scale with available compute and quickly produce high quality models.
- *Flexibility*: Allow researchers and practitioners to extend AdaNet to novel subnetwork architectures, search spaces, and tasks.
- *Learning guarantees*: Optimize an objective that offers theoretical learning guarantees.

The following animation shows AdaNet adaptively growing an ensemble of neural networks. At each iteration, it measures the ensemble loss for each candidate, and selects the best one to move onto the next iteration. At subsequent iterations, the blue subnetworks are frozen, and only yellow subnetworks are trained:

AdaNet was first announced on the Google AI research blog: "[Introducing AdaNet: Fast and Flexible AutoML with Learning Guarantees](https://ai.googleblog.com/2018/10/introducing-adanet-fast-and-flexible.html)".

This is not an official Google product.

CHAPTER 1

adanet

AdaNet: Fast and flexible AutoML with learning guarantees.

1.1 Estimators

High-level APIs for training, evaluating, predicting, and serving AdaNet model.

1.1.1 AutoEnsembleEstimator

class adanet.**AutoEnsembleEstimator**(*head*, *candidate_pool*, *max_iteration_steps*, *logits_fn=None*, *adanet_lambda=0.0*, *evaluator=None*, *metric_fn=None*, *force_grow=False*, *adanet_loss_decay=0.9*, *worker_wait_timeout_secs=7200*, *model_dir=None*, *config=None*)

Bases: adanet.core.estimator.Estimator

A tf.estimator.Estimator that learns to ensemble models.

Specifically, it learns to ensemble models from a candidate pool using the Adanet algorithm.

```
# A simple example of learning to ensemble linear and neural network
# models.

import adanet
import tensorflow as tf

feature_columns = ...

head = tf.contrib.estimator.multi_class_head(n_classes=3)

# Learn to ensemble linear and DNN models.
estimator = adanet.AutoEnsembleEstimator(
```

(continues on next page)

(continued from previous page)

```
    head=head,
    candidate_pool=[
        tf.estimator.LinearEstimator(
            head=head,
            feature_columns=feature_columns,
            optimizer=tf.train.FtrlOptimizer(...)),
        tf.estimator.DNNEstimator(
            head=head,
            feature_columns=feature_columns,
            optimizer=tf.train.ProximalAdagradOptimizer(...),
            hidden_units=[1000, 500, 100])],
    max_iteration_steps=50)

# Input builders
def input_fn_train:
  # Returns tf.data.Dataset of (x, y) tuple where y represents label's
  # class index.
  pass
def input_fn_eval:
  # Returns tf.data.Dataset of (x, y) tuple where y represents label's
  # class index.
  pass
def input_fn_predict:
  # Returns tf.data.Dataset of (x, None) tuple.
  pass
estimator.train(input_fn=input_fn_train, steps=100)
metrics = estimator.evaluate(input_fn=input_fn_eval, steps=10)
predictions = estimator.predict(input_fn=input_fn_predict)
```

Parameters

- **head** – A `tf.contrib.estimator.Head` instance for computing loss and evaluation metrics for every candidate.
- **candidate_pool** – List of `tf.estimator.Estimator` objects that are candidates to ensemble at each iteration. The order does not directly affect which candidates will be included in the final ensemble.
- **max_iteration_steps** – Total number of steps for which to train candidates per iteration. If *OutOfRange* or *StopIteration* occurs in the middle, training stops before *max_iteration_steps* steps.
- **logits_fn** – A function for fetching the subnetwork logits from a `tf.estimator.EstimatorSpec`, which should obey the following signature:
  - *Args*: Can only have following argument: - estimator_spec: The candidate's `tf.estimator.EstimatorSpec`.
  - *Returns*: Logits `tf.Tensor` or dict of string to logits `tf.Tensor` (for multi-head) for the candidate subnetwork extracted from the given *estimator_spec*. When *None*, it will default to returning *estimator_spec.predictions* when they are a `tf.Tensor` or the `tf.Tensor` for the key 'logits' when they are a dict of string to `tf.Tensor`.
- **adanet_lambda** – See *adanet.Estimator*.
- **evaluator** – See *adanet.Estimator*.
- **metric_fn** – See *adanet.Estimator*.

- **force_grow** – See *adanet.Estimator*.
- **adanet_loss_decay** – See *adanet.Estimator*.
- **worker_wait_timeout_secs** – See *adanet.Estimator*.
- **model_dir** – See *adanet.Estimator*.
- **config** – See *adanet.Estimator*.

Returns An *adanet.AutoEnsembleEstimator* instance.

Raises `ValueError` – If any of the candidates in *candidate_pool* are not `tf.estimator.Estimator` instances.

eval_dir(*name=None*)

Shows the directory name where evaluation metrics are dumped.

Parameters **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 string which is the path of directory contains evaluation metrics.

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 (*tf.errors.OutOfRangeError* or *StopIteration*).

Parameters

- **input_fn** – A function that constructs the input data for evaluation. See [Premade Estimators](https://tensorflow.org/guide/premade#create_input_functions) for more information. The function should construct and return one of the following: * A *tf.data.Dataset* object: Outputs of *Dataset* object must be a tuple *(features, labels)* with same constraints as below. * A tuple *(features, labels)*: Where *features* is a *tf.Tensor* or a dictionary of string feature name to *Tensor* and *labels* is a *Tensor* or a dictionary of string label name to *Tensor*. Both *features* and *labels* are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
- **steps** – Number of steps for which to evaluate model. If *None*, evaluates until *input_fn* raises an end-of-input exception.
- **hooks** – List of *tf.train.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. If there are no checkpoints in *model_dir*, evaluation is run with newly initialized *Variables* instead of ones restored from checkpoint.
- **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. For canned estimators, the dict contains the *loss* (mean loss per mini-batch) and the *average_loss* (mean loss per sample). Canned classifiers also return the *accuracy*. Canned regressors also return the *label/mean* and the *prediction/mean*.

Raises

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

export_saved_model(*export_dir_base*, *serving_input_receiver_fn*, *assets_extra=None*, *as_text=False*, *checkpoint_path=None*)

Exports inference graph as a *SavedModel* into the given dir.

For a detailed guide, see [Using SavedModel with Estimators](https://tensorflow.org/guide/saved_model#using_savedmodel_with_estimators).

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 *tf.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 *tf.saved_model.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 *tf.estimator.export.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 *assets_extra* 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’}*.

Parameters

- **export_dir_base** – A string containing a directory in which to create timestamped subdirectories containing exported ‘SavedModel‘s.
- **serving_input_receiver_fn** – A function that takes no argument and returns a *tf.estimator.export.ServingInputReceiver* or *tf.estimator.export.TensorServingInputReceiver*.
- **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.

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

Exports inference graph as a *SavedModel* into the given dir.

Note that *export_to_savedmodel* will be renamed to *export_saved_model* in TensorFlow 2.0. At that time, *export_to_savedmodel* without the additional underscore will be available only through tf.compat.v1.

Please see *tf.estimator.Estimator.export_saved_model* for more information.

There is one additional arg versus the new method:

strip_default_attrs: This parameter is going away in TF 2.0, and the new behavior will automatically strip all default attributes. Boolean. If *True*, default-valued attributes will be removed from the 'NodeDef's. For a detailed guide, see [Stripping Default-Valued Attributes](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md#stripping-default-valued-attributes).

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(*name*)
Returns value of the variable given by name.

Parameters **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()
Finds the filename of the latest saved checkpoint file in *model_dir*.

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

model_fn
Returns the *model_fn* which is bound to *self.params*.

Returns *def model_fn(features, labels, mode, config)*

Return type The *model_fn* with following signature

predict(*input_fn*, *predict_keys=None*, *hooks=None*, *checkpoint_path=None*, *yield_single_examples=True*)
Yields predictions for given features.

Please note that interleaving two predict outputs does not work. See: [issue/20506](https://github.com/tensorflow/tensorflow/issues/20506#issuecomment-422208517)

Parameters

- **input_fn** – A function that constructs the features. Prediction continues until *input_fn* raises an end-of-input exception (*tf.errors.OutOfRangeError* or *StopIteration*). See [Premade Estimators](https://tensorflow.org/guide/premade_estimators#create_input_functions) for more information. The function should construct and return one of the following:
  - A *tf.data.Dataset* object: Outputs of *Dataset* object must have same constraints as below.
  - features: A *tf.Tensor* or a dictionary of string feature name to *Tensor*. features are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
  - A tuple, in which case the first item is extracted as features.
- **predict_keys** – list of *str*, name of the keys to predict. It is used if the *tf.estimator.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 *tf.train.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. If there are no checkpoints in *model_dir*, prediction is run with newly initialized *Variables* instead of ones restored from checkpoint.
- **yield_single_examples** – If *False*, yields the whole batch as returned by the *model_fn* instead of decomposing the batch into individual elements. This is useful if *model_fn* returns some tensors whose first dimension is not equal to the batch size.

Yields Evaluated values of *predictions* tensors.

Raises

- `ValueError` – Could not find a trained model in *model_dir*.
- `ValueError` – If batch length of predictions is not the same and *yield_single_examples* is *True*.
- `ValueError` – If there is a conflict between *predict_keys* and *predictions*. For example if *predict_keys* is not *None* but *tf.estimator.EstimatorSpec.predictions* is not a *dict*.

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

Trains a model given training data *input_fn*.

Parameters

- **input_fn** – A function that provides input data for training as minibatches. See [Premade Estimators](https://tensorflow.org/guide/premade_estimators#create_input_functions) for more information. The function should construct and return one of the following: * A *tf.data.Dataset* object: Outputs of *Dataset* object must be a tuple *(features, labels)* with same constraints as below. * A tuple *(features, labels)*: Where *features* is a *tf.Tensor* or a dictionary of string feature name to *Tensor* and *labels* is a *Tensor* or a dictionary of string label name to *Tensor*. Both *features* and *labels* are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
- **hooks** – List of *tf.train.SessionRunHook* subclass instances. Used for callbacks inside the training loop.
- **steps** – Number of steps for which to train the model. If *None*, train forever or train until *input_fn* generates the *tf.errors.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 *tf.errors.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* <= *0*.

1.1.2 Estimator

class adanet.**Estimator**(*head*, *subnetwork_generator*, *max_iteration_steps*, *mixture_weight_type='scalar'*, *mixture_weight_initializer=None*, *warm_start_mixture_weights=False*, *adanet_lambda=0.0*, *adanet_beta=0.0*, *evaluator=None*, *report_materializer=None*, *use_bias=False*, *metric_fn=None*, *force_grow=False*, *replicate_ensemble_in_training=False*, *adanet_loss_decay=0.9*, *worker_wait_timeout_secs=7200*, *model_dir=None*, *report_dir=None*, *config=None*, ***kwargs*)

Bases: tensorflow.python.estimator.estimator.Estimator

The AdaNet algorithm implemented as a tf.estimator.Estimator.

AdaNet is as defined in the paper: https://arxiv.org/abs/1607.01097.

The AdaNet algorithm uses a weak learning algorithm to iteratively generate a set of candidate subnetworks that attempt to minimize the loss function defined in Equation (4) as part of an ensemble. At the end of each iteration, the best candidate is chosen based on its ensemble's complexity-regularized train loss. New subnetworks are allowed to use any subnetwork weights within the previous iteration's ensemble in order to improve upon them. If the complexity-regularized loss of the new ensemble, as defined in Equation (4), is less than that of the previous iteration's ensemble, the AdaNet algorithm continues onto the next iteration.

AdaNet attempts to minimize the following loss function to learn the mixture weights 'w' of each subnetwork 'h' in the ensemble with differentiable convex non-increasing surrogate loss function Phi:

Equation (4):

$$F(w) = \frac{1}{m}\sum_{i=1}^{m}\Phi\left(\sum_{j=1}^{N} w_j h_j(x_i), y_i\right) + \sum_{j=1}^{N}\left(\lambda r(h_j) + \beta\right)|w_j|$$

with $\lambda >= 0$ and $\beta >= 0$.

This implementation uses an *adanet.subnetwork.Generator* as its weak learning algorithm for generating candidate subnetworks. These are trained in parallel using a single graph per iteration. At the end of each iteration, the estimator saves the sub-graph of the best subnetwork ensemble and its weights as a separate checkpoint. At the beginning of the next iteration, the estimator imports the previous iteration's frozen graph and adds ops for the next candidates as part of a new graph and session. This allows the estimator have the performance of Tensorflow's static graph constraint (minus the performance hit of reconstructing a graph between iterations), while having the flexibility of having a dynamic graph.

NOTE: Subclassing tf.estimator.Estimator is only necessary to work with tf.estimator.train_and_evaluate() which asserts that the estimator argument is a tf.estimator.Estimator subclass. However, all training is delegated to a separate tf.estimator.Estimator instance. It is responsible for supporting both local and distributed training. As such, the *adanet.Estimator* is only responsible for bookkeeping across iterations.

Parameters

- **head** – A tf.contrib.estimator.Head instance for computing loss and evaluation metrics for every candidate.
- **subnetwork_generator** – The *adanet.subnetwork.Generator* which defines the candidate subnetworks to train and evaluate at every AdaNet iteration.
- **max_iteration_steps** – Total number of steps for which to train candidates per iteration. If OutOfRange or StopIteration occurs in the middle, training stops before *max_iteration_steps* steps.

- **mixture_weight_type** – The *adanet.MixtureWeightType* defining which mixture weight type to learn in the linear combination of subnetwork outputs:
  - SCALAR: creates a rank 0 tensor mixture weight . It performs an element- wise multiplication with its subnetwork's logits. This mixture weight is the simplest to learn, the quickest to train, and most likely to generalize well.
  - VECTOR: creates a tensor with shape [k] where k is the ensemble's logits dimension as defined by *head*. It is similar to *SCALAR* in that it performs an element-wise multiplication with its subnetwork's logits, but is more flexible in learning a subnetworks's preferences per class.
  - MATRIX: creates a tensor of shape [a, b] where a is the number of outputs from the subnetwork's *last_layer* and b is the number of outputs from the ensemble's *logits*. This weight matrix-multiplies the subnetwork's *last_layer*. This mixture weight offers the most flexibility and expressivity, allowing subnetworks to have outputs of different dimensionalities. However, it also has the most trainable parameters (a*b), and is therefore the most sensitive to learning rates and regularization.
- **mixture_weight_initializer** – The initializer for mixture_weights. When *None*, the default is different according to *mixture_weight_type*:
  - SCALAR: initializes to 1/N where N is the number of subnetworks in the ensemble giving a uniform average.
  - VECTOR: initializes each entry to 1/N where N is the number of subnetworks in the ensemble giving a uniform average.
  - MATRIX: uses `tf.zeros_initializer()`.
- **warm_start_mixture_weights** – Whether, at the beginning of an iteration, to initialize the mixture weights of the subnetworks from the previous ensemble to their learned value at the previous iteration, as opposed to retraining them from scratch. Takes precedence over the value for *mixture_weight_initializer* for subnetworks from previous iterations.
- **adanet_lambda** – Float multiplier 'lambda' for applying L1 regularization to subnetworks' mixture weights 'w' in the ensemble proportional to their complexity. See Equation (4) in the AdaNet paper.
- **adanet_beta** – Float L1 regularization multiplier 'beta' to apply equally to all subnetworks' weights 'w' in the ensemble regardless of their complexity. See Equation (4) in the AdaNet paper.
- **evaluator** – An *adanet.Evaluator* for candidate selection after all subnetworks are done training. When *None*, candidate selection uses a moving average of their *adanet.Ensemble* AdaNet loss during training instead. In order to use the *AdaNet algorithm* as described in [Cortes et al., '17], the given *adanet.Evaluator* must be created with the same dataset partition used during training. Otherwise, this framework will perform *AdaNet.HoldOut* which uses a holdout set for candidate selection, but does not benefit from learning guarantees.
- **report_materializer** – An *adanet.ReportMaterializer*. Its reports are made available to the *subnetwork_generator* at the next iteration, so that it can adapt its search space. When *None*, the *subnetwork_generator* `generate_candidates()` method will receive empty Lists for their *previous_ensemble_reports* and *all_reports* arguments.
- **use_bias** – Whether to add a bias term to the ensemble's logits. Adding a bias allows the ensemble to learn a shift in the data, often leading to more stable training and better predictions.

- **metric_fn** – A function for adding custom evaluation metrics, which should obey the following signature:
  - *Args*: Can only have the following three arguments in any order: - *predictions*: Predictions *Tensor* or dict of *Tensor* created by

    given *head*.

    * *features*: Input *dict* of *Tensor* objects created by *input_fn* which is given to *estimator.evaluate* as an argument.
    * *labels*: Labels *Tensor* or dict of *Tensor* (for multi-head) created by *input_fn* which is given to *estimator.evaluate* as an argument.

  - *Returns*: Dict of metric results keyed by name. Final metrics are a union of this and *head's* existing metrics. If there is a name conflict between this and *head's existing metrics, this will override the existing one. The values of the dict are the results of calling a metric function, namely a '(metric_tensor, update_op)* tuple.
- **force_grow** – Boolean override that forces the ensemble to grow by one subnetwork at the end of each iteration. Normally at the end of each iteration, AdaNet selects the best candidate ensemble according to its performance on the AdaNet objective. In some cases, the best ensemble is the *previous_ensemble* as opposed to one that includes a newly trained subnetwork. When *True*, the algorithm will not select the *previous_ensemble* as the best candidate, and will ensure that after n iterations the final ensemble is composed of n subnetworks.
- **replicate_ensemble_in_training** – Whether to rebuild the frozen subnetworks of the ensemble in training mode, which can change the outputs of the frozen subnetworks in the ensemble. When *False* and during candidate training, the frozen subnetworks in the ensemble are in prediction mode, so training-only ops like dropout are not applied to them. When *True* and training the candidates, the frozen subnetworks will be in training mode as well, so they will apply training-only ops like dropout. This argument is useful for regularizing learning mixture weights, or for making training-only side inputs available in subsequent iterations. For most use-cases, this should be *False*.
- **adanet_loss_decay** – Float decay for the exponential-moving-average of the AdaNet objective throughout training. This moving average is a data- driven way tracking the best candidate with only the training set.
- **worker_wait_timeout_secs** – Float number of seconds for workers to wait for chief to prepare the next iteration during distributed training. This is needed to prevent workers waiting indefinitely for a chief that may have crashed or been turned down. When the timeout is exceeded, the worker exits the train loop. In situations where the chief job is much slower than the worker jobs, this timeout should be increased.
- **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.
- **report_dir** – Directory where the *adanet.subnetwork.MaterializedReport's materialized by 'report_materializer* would be saved. If *report_materializer* is None, this will not save anything. If *None* or empty string, defaults to "<model_dir>/report".
- **config** – *RunConfig* object to configure the runtime settings.
- ****kwargs** – Extra keyword args passed to the parent.

Returns An *Estimator* instance.

Raises

- `ValueError` – If *subnetwork_generator* is *None*.
- `ValueError` – If *max_iteration_steps* is <= 0.

eval_dir(*name=None*)
Shows the directory name where evaluation metrics are dumped.

Parameters **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 string which is the path of directory contains evaluation metrics.

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 (*tf.errors.OutOfRangeError* or *StopIteration*).

Parameters

- **input_fn** – A function that constructs the input data for evaluation. See [Premade Estimators](https://tensorflow.org/guide/premade#create_input_functions) for more information. The function should construct and return one of the following: * A *tf.data.Dataset* object: Outputs of *Dataset* object must be a tuple *(features, labels)* with same constraints as below. * A tuple *(features, labels)*: Where *features* is a *tf.Tensor* or a dictionary of string feature name to *Tensor* and *labels* is a *Tensor* or a dictionary of string label name to *Tensor*. Both *features* and *labels* are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
- **steps** – Number of steps for which to evaluate model. If *None*, evaluates until *input_fn* raises an end-of-input exception.
- **hooks** – List of *tf.train.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. If there are no checkpoints in *model_dir*, evaluation is run with newly initialized *Variables* instead of ones restored from checkpoint.
- **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. For canned estimators, the dict contains the *loss* (mean loss per mini-batch) and the *average_loss* (mean loss per sample). Canned classifiers also return the *accuracy*. Canned regressors also return the *label/mean* and the *prediction/mean*.

Raises

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

export_saved_model(*export_dir_base*, *serving_input_receiver_fn*, *assets_extra=None*, *as_text=False*, *checkpoint_path=None*)
Exports inference graph as a *SavedModel* into the given dir.

For a detailed guide, see [Using SavedModel with Estimators](https://tensorflow.org/guide/saved_model#using_savedmodel_with_estimators).

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 *tf.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 *tf.saved_model.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 *tf.estimator.export.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 *assets_extra* 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'}*.

Parameters

- **export_dir_base** – A string containing a directory in which to create timestamped subdirectories containing exported 'SavedModel's.
- **serving_input_receiver_fn** – A function that takes no argument and returns a *tf.estimator.export.ServingInputReceiver* or *tf.estimator.export.TensorServingInputReceiver*.
- **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.

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

Exports inference graph as a *SavedModel* into the given dir.

Note that *export_to_savedmodel* will be renamed to *export_saved_model* in TensorFlow 2.0. At that time, *export_to_savedmodel* without the additional underscore will be available only through tf.compat.v1.

Please see *tf.estimator.Estimator.export_saved_model* for more information.

There is one additional arg versus the new method:

strip_default_attrs: This parameter is going away in TF 2.0, and the new behavior will automatically strip all default attributes. Boolean. If *True*, default-valued attributes will be removed from the 'NodeDef's. For a detailed guide, see [Stripping Default-Valued Attributes](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md#stripping-default-valued-attributes).

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(*name*)
Returns value of the variable given by name.

Parameters **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()
Finds the filename of the latest saved checkpoint file in *model_dir*.

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

model_fn
Returns the *model_fn* which is bound to *self.params*.

Returns *def model_fn(features, labels, mode, config)*

Return type The *model_fn* with following signature

predict(*input_fn*, *predict_keys=None*, *hooks=None*, *checkpoint_path=None*, *yield_single_examples=True*)
Yields predictions for given features.

Please note that interleaving two predict outputs does not work. See: [issue/20506](https://github.com/tensorflow/tensorflow/issues/20506#issuecomment-422208517)

Parameters

- **input_fn** – A function that constructs the features. Prediction continues until *input_fn* raises an end-of-input exception (*tf.errors.OutOfRangeError* or *StopIteration*). See [Premade Estimators](https://tensorflow.org/guide/premade_estimators#create_input_functions) for more information. The function should construct and return one of the following:
  - A *tf.data.Dataset* object: Outputs of *Dataset* object must have same constraints as below.
  - features: A *tf.Tensor* or a dictionary of string feature name to *Tensor*. features are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
  - A tuple, in which case the first item is extracted as features.
- **predict_keys** – list of *str*, name of the keys to predict. It is used if the *tf.estimator.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 *tf.train.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. If there are no checkpoints in *model_dir*, prediction is run with newly initialized *Variables* instead of ones restored from checkpoint.
- **yield_single_examples** – If *False*, yields the whole batch as returned by the *model_fn* instead of decomposing the batch into individual elements. This is useful if *model_fn* returns some tensors whose first dimension is not equal to the batch size.

Yields Evaluated values of *predictions* tensors.

Raises

- `ValueError` – Could not find a trained model in *model_dir*.
- `ValueError` – If batch length of predictions is not the same and *yield_single_examples* is *True*.
- `ValueError` – If there is a conflict between *predict_keys* and *predictions*. For example if *predict_keys* is not *None* but *tf.estimator.EstimatorSpec.predictions* is not a *dict*.

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

Trains a model given training data *input_fn*.

Parameters

- **`input_fn`** – A function that provides input data for training as minibatches. See [Premade Estimators](https://tensorflow.org/guide/premade_estimators#create_input_functions) for more information. The function should construct and return one of the following: * A *tf.data.Dataset* object: Outputs of *Dataset* object must be a tuple *(features, labels)* with same constraints as below. * A tuple *(features, labels)*: Where *features* is a *tf.Tensor* or a dictionary of string feature name to *Tensor* and *labels* is a *Tensor* or a dictionary of string label name to *Tensor*. Both *features* and *labels* are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
- **`hooks`** – List of *tf.train.SessionRunHook* subclass instances. Used for callbacks inside the training loop.
- **`steps`** – Number of steps for which to train the model. If *None*, train forever or train until *input_fn* generates the *tf.errors.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 *tf.errors.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* <= *0*.

1.1.3 TPUEstimator

class adanet.**TPUEstimator**(*head*, *subnetwork_generator*, *max_iteration_steps*, *mixture_weight_type='scalar'*, *mixture_weight_initializer=None*, *warm_start_mixture_weights=False*, *adanet_lambda=0.0*, *adanet_beta=0.0*, *evaluator=None*, *report_materializer=None*, *use_bias=False*, *metric_fn=None*, *force_grow=False*, *replicate_ensemble_in_training=False*, *adanet_loss_decay=0.9*, *worker_wait_timeout_secs=7200*, *model_dir=None*, *report_dir=None*, *config=None*, *use_tpu=True*, *train_batch_size=None*, *eval_batch_size=None*)

Bases: adanet.core.estimator.Estimator, tensorflow.contrib.tpu.python.tpu.tpu_estimator.TPUEstimator

An adanet.Estimator capable of running on TPU.

If running on TPU, all summary calls are rewired to be no-ops during training.

WARNING: this API is highly experimental, unstable, and can change without warning.

eval_dir(*name=None*)

Shows the directory name where evaluation metrics are dumped.

Parameters **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 string which is the path of directory contains evaluation metrics.

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 (*tf.errors.OutOfRangeError* or *StopIteration*).

Parameters

- **input_fn** – A function that constructs the input data for evaluation. See [Premade Estimators](https://tensorflow.org/guide/premade#create_input_functions) for more information. The function should construct and return one of the following: * A *tf.data.Dataset* object: Outputs of *Dataset* object must be a tuple *(features, labels)* with same constraints as below. * A tuple *(features, labels)*: Where *features* is a *tf.Tensor* or a dictionary of string feature name to *Tensor* and *labels* is a *Tensor* or a dictionary of string label name to *Tensor*. Both *features* and *labels* are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
- **steps** – Number of steps for which to evaluate model. If *None*, evaluates until *input_fn* raises an end-of-input exception.
- **hooks** – List of *tf.train.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. If there are no checkpoints in *model_dir*, evaluation is run with newly initialized *Variables* instead of ones restored from checkpoint.
- **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. For canned estimators, the dict contains the *loss* (mean loss per mini-batch) and the *average_loss* (mean loss per sample). Canned classifiers also return the *accuracy*. Canned regressors also return the *label/mean* and the *prediction/mean*.

Raises

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

export_saved_model(*export_dir_base*, *serving_input_receiver_fn*, *assets_extra=None*, *as_text=False*, *checkpoint_path=None*)

Exports inference graph as a *SavedModel* into the given dir.

For a detailed guide, see [Using SavedModel with Estimators](https://tensorflow.org/guide/saved_model#using_savedmodel_with_estimators).

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 *tf.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 *tf.saved_model.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 *tf.estimator.export.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 *assets_extra* 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'}*.

Parameters

- **export_dir_base** – A string containing a directory in which to create timestamped subdirectories containing exported 'SavedModel's.
- **serving_input_receiver_fn** – A function that takes no argument and returns a *tf.estimator.export.ServingInputReceiver* or *tf.estimator.export.TensorServingInputReceiver*.
- **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.

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

Exports inference graph as a *SavedModel* into the given dir.

Note that *export_to_savedmodel* will be renamed to *export_saved_model* in TensorFlow 2.0. At that time, *export_to_savedmodel* without the additional underscore will be available only through tf.compat.v1.

Please see *tf.estimator.Estimator.export_saved_model* for more information.

There is one additional arg versus the new method:

strip_default_attrs: This parameter is going away in TF 2.0, and the new behavior will automatically strip all default attributes. Boolean. If *True*, default-valued attributes will be removed from the 'NodeDef's. For a detailed guide, see [Stripping Default-Valued Attributes](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md#stripping-default-valued-attributes).

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(*name*)
Returns value of the variable given by name.

Parameters **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()
Finds the filename of the latest saved checkpoint file in *model_dir*.

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

model_fn
Returns the *model_fn* which is bound to *self.params*.

Returns *def model_fn(features, labels, mode, config)*

Return type The *model_fn* with following signature

predict(*input_fn*, *predict_keys=None*, *hooks=None*, *checkpoint_path=None*, *yield_single_examples=True*)
Yields predictions for given features.

Please note that interleaving two predict outputs does not work. See: [issue/20506](https://github.com/tensorflow/tensorflow/issues/20506#issuecomment-422208517)

Parameters

- **input_fn** – A function that constructs the features. Prediction continues until *input_fn* raises an end-of-input exception (*tf.errors.OutOfRangeError* or *StopIteration*). See [Premade Estimators](https://tensorflow.org/guide/premade_estimators#create_input_functions) for more information. The function should construct and return one of the following:
  - A *tf.data.Dataset* object: Outputs of *Dataset* object must have same constraints as below.
  - features: A *tf.Tensor* or a dictionary of string feature name to *Tensor*. features are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
  - A tuple, in which case the first item is extracted as features.

- **predict_keys** – list of *str*, name of the keys to predict. It is used if the *tf.estimator.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 *tf.train.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. If there are no checkpoints in *model_dir*, prediction is run with newly initialized *Variables* instead of ones restored from checkpoint.
- **yield_single_examples** – If *False*, yields the whole batch as returned by the *model_fn* instead of decomposing the batch into individual elements. This is useful if *model_fn* returns some tensors whose first dimension is not equal to the batch size.

Yields Evaluated values of *predictions* tensors.

Raises

- `ValueError` – Could not find a trained model in *model_dir*.
- `ValueError` – If batch length of predictions is not the same and *yield_single_examples* is *True*.
- `ValueError` – If there is a conflict between *predict_keys* and *predictions*. For example if *predict_keys* is not *None* but *tf.estimator.EstimatorSpec.predictions* is not a *dict*.

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

Trains a model given training data *input_fn*.

Parameters

- **input_fn** – A function that provides input data for training as minibatches. See [Premade Estimators](https://tensorflow.org/guide/premade_estimators#create_input_functions) for more information. The function should construct and return one of the following: * A *tf.data.Dataset* object: Outputs of *Dataset* object must be a tuple *(features, labels)* with same constraints as below. * A tuple *(features, labels)*: Where *features* is a *tf.Tensor* or a dictionary of string feature name to *Tensor* and *labels* is a *Tensor* or a dictionary of string label name to *Tensor*. Both *features* and *labels* are consumed by *model_fn*. They should satisfy the expectation of *model_fn* from inputs.
- **hooks** – List of *tf.train.SessionRunHook* subclass instances. Used for callbacks inside the training loop.
- **steps** – Number of steps for which to train the model. If *None*, train forever or train until *input_fn* generates the *tf.errors.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 *tf.errors.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* <= *0*.

1.2 Ensembles

Collections representing learned combinations of subnetworks.

1.2.1 MixtureWeightType

class adanet.**MixtureWeightType**

Mixture weight types available for learning subnetwork contributions.

The following mixture weight types are defined:

- *SCALAR*: Produces a rank 0 *Tensor* mixture weight.
- *VECTOR*: Produces a rank 1 *Tensor* mixture weight.
- *MATRIX*: Produces a rank 2 *Tensor* mixture weight.

1.2.2 WeightedSubnetwork

class adanet.**WeightedSubnetwork**

An AdaNet weighted subnetwork.

A weighted subnetwork is a weight 'w' applied to a subnetwork's last layer 'u'. The results is the weighted subnetwork's logits, regularized by its complexity.

Parameters

- **name** – String name of *subnetwork* as defined by its *adanet.subnetwork.Builder*.
- **iteration_number** – Integer iteration when the subnetwork was created.
- **weight** – The weight `tf.Tensor` or dict of string to weight `tf.Tensor` (for multi-head) to apply to this subnetwork. The AdaNet paper refers to this weight as 'w' in Equations (4), (5), and (6).
- **logits** – The output `tf.Tensor` or dict of string to weight `tf.Tensor` (for multi-head) after the matrix multiplication of *weight* and the subnetwork's `last_layer()`. The output's shape is [batch_size, logits_dimension]. It is equivalent to a linear logits layer in a neural network.
- **subnetwork** – The *adanet.subnetwork.Subnetwork* to weight.

Returns An *adanet.WeightedSubnetwork* object.

1.2.3 Ensemble

class adanet.**Ensemble**

An AdaNet ensemble.

An ensemble is a collection of subnetworks which forms a neural network through the weighted sum of their outputs. It is represented by 'f' throughout the AdaNet paper. Its component subnetworks' weights are complexity regularized (Gamma) as defined in Equation (4).

Parameters

- **weighted_subnetworks** – List of *adanet.WeightedSubnetwork* instances that form this ensemble. Ordered from first to most recent.
- **bias** – Bias term `tf.Tensor` or dict of string to bias term `tf.Tensor` (for multi-head) for the ensemble's logits.
- **logits** – Logits `tf.Tensor` or dict of string to logits `tf.Tensor` (for multi-head). The result of the function 'f' as defined in Section 5.1 which is the sum of the logits of all *adanet.WeightedSubnetwork* instances in ensemble.

Returns An *adanet.Ensemble* instance.

1.3 Evaluator

Measures *adanet.Ensemble* performance on a given dataset.

1.3.1 Evaluator

class adanet.**Evaluator**(*input_fn*, *steps=None*)

Evaluates candidate ensemble performance.

Parameters

- **input_fn** – Input function returning a tuple of: features - Dictionary of string feature name to *Tensor*. labels - *Tensor* of labels.
- **steps** – Number of steps for which to evaluate the ensembles. If an *OutOfRangeError* occurs, evaluation stops. If set to None, will iterate the dataset until all inputs are exhausted.

Returns An *adanet.Evaluator* instance.

evaluate_adanet_losses(*sess*, *adanet_losses*)

Evaluates the given AdaNet objectives on the data from *input_fn*.

The candidates are fed the same batches of features and labels as provided by *input_fn*, and their losses are computed and summed over *steps* batches.

Parameters

- **sess** – *Session* instance with most recent variable values loaded.
- **adanet_losses** – List of AdaNet loss *Tensors*.

Returns List of evaluated AdaNet losses.

input_fn

Return the input_fn.

steps

Return the number of evaluation steps.

1.4 Summary

Extends `tf.summary` to power AdaNet's TensorBoard integration.

1.4.1 Summary

class adanet.**Summary**

Interface for writing summaries to Tensorboard.

audio(*name*, *tensor*, *sample_rate*, *max_outputs=3*, *family=None*)

Outputs a *tf.Summary* protocol buffer with audio.

The summary has up to *max_outputs* summary values containing audio. The audio is built from *tensor* which must be 3-D with shape *[batch_size, frames, channels]* or 2-D with shape *[batch_size, frames]*. The values are assumed to be in the range of *[-1.0, 1.0]* with a sample rate of *sample_rate*.

The *tag* in the outputted tf.Summary.Value protobufs is generated based on the name, with a suffix depending on the max_outputs setting:

- If *max_outputs* is 1, the summary value tag is '*name*/audio'.
- If *max_outputs* is greater than 1, the summary value tags are

generated sequentially as '*name*/audio/0', '*name*/audio/1', etc

Parameters

- **name** – A name for the generated node. Will also serve as a series name in TensorBoard.
- **tensor** – A 3-D *float32 Tensor* of shape *[batch_size, frames, channels]* or a 2-D *float32 Tensor* of shape *[batch_size, frames]*.
- **sample_rate** – A Scalar *float32 Tensor* indicating the sample rate of the signal in hertz.
- **max_outputs** – Max number of batch elements to generate audio for.
- **family** – Optional; if provided, used as the prefix of the summary tag name, which controls the tab name used for display on Tensorboard.

Returns A scalar *Tensor* of type *string*. The serialized *tf.Summary* protocol buffer.

histogram(*name*, *values*, *family=None*)

Outputs a *tf.Summary* protocol buffer with a histogram.

Adding a histogram summary makes it possible to visualize your data's distribution in TensorBoard. You can see a detailed explanation of the TensorBoard histogram dashboard [here](https://www.tensorflow.org/get_started/tensorboard_histograms).

The generated [*tf.Summary*](tensorflow/core/framework/summary.proto) has one summary value containing a histogram for *values*.

This op reports an *InvalidArgument* error if any value is not finite.

Parameters

- **name** – A name for the generated node. Will also serve as a series name in TensorBoard.
- **values** – A real numeric *Tensor*. Any shape. Values to use to build the histogram.
- **family** – Optional; if provided, used as the prefix of the summary tag name, which controls the tab name used for display on Tensorboard.

Returns A scalar *Tensor* of type *string*. The serialized *tf.Summary* protocol buffer.

image(*name*, *tensor*, *max_outputs=3*, *family=None*)
Outputs a *tf.Summary* protocol buffer with images.

The summary has up to *max_outputs* summary values containing images. The images are built from *tensor* which must be 4-D with shape *[batch_size, height, width, channels]* and where *channels* can be:

- 1: *tensor* is interpreted as Grayscale.
- 3: *tensor* is interpreted as RGB.
- 4: *tensor* is interpreted as RGBA.

The images have the same number of channels as the input tensor. For float input, the values are normalized one image at a time to fit in the range *[0, 255]*. *uint8* values are unchanged. The op uses two different normalization algorithms:

- If the input values are all positive, they are rescaled so the largest

one is 255. * If any input value is negative, the values are shifted so input value 0.0

is at 127. They are then rescaled so that either the smallest value is 0, or the largest one is 255.

The *tag* in the outputted tf.Summary.Value protobufs is generated based on the name, with a suffix depending on the max_outputs setting:

- If *max_outputs* is 1, the summary value tag is '*name*/image'.
- If *max_outputs* is greater than 1, the summary value tags are

generated sequentially as '*name*/image/0', '*name*/image/1', etc.

Parameters

- **name** – A name for the generated node. Will also serve as a series name in TensorBoard.
- **tensor** – A 4-D *uint8* or *float32 Tensor* of shape *[batch_size, height, width, channels]* where *channels* is 1, 3, or 4.
- **max_outputs** – Max number of batch elements to generate images for.
- **family** – Optional; if provided, used as the prefix of the summary tag name, which controls the tab name used for display on Tensorboard.

Returns A scalar *Tensor* of type *string*. The serialized *tf.Summary* protocol buffer.

scalar(*name*, *tensor*, *family=None*)
Outputs a *tf.Summary* protocol buffer containing a single scalar value.

The generated tf.Summary has a Tensor.proto containing the input Tensor.

Parameters

- **name** – A name for the generated node. Will also serve as the series name in TensorBoard.
- **tensor** – A real numeric Tensor containing a single value.
- **family** – Optional; if provided, used as the prefix of the summary tag name, which controls the tab name used for display on Tensorboard.

Returns A scalar *Tensor* of type *string*. Which contains a *tf.Summary* protobuf.

Raises `ValueError` – If tensor has the wrong shape or type.

1.5 ReportMaterializer

1.5.1 ReportMaterializer

class adanet.**ReportMaterializer**(*input_fn*, *steps=None*)

Materializes reports.

Specifically it materializes a subnetwork's *adanet.subnetwork.Report* instances into *adanet.subnetwork.MaterializedReport* instances.

Requires an input function *input_fn* that returns a tuple of:

- features: Dictionary of string feature name to *Tensor*.
- labels: *Tensor* of labels.

Parameters

- **input_fn** – The input function.
- **steps** – Number of steps for which to materialize the ensembles. If an *OutOfRangeError* occurs, materialization stops. If set to None, will iterate the dataset until all inputs are exhausted.

Returns A *ReportMaterializer* instance.

input_fn

Returns the input_fn that materialize_subnetwork_reports would run on.

Even though this property appears to be unused, it would be used to build the AdaNet model graph inside AdaNet estimator.train(). After the graph is built, the queue_runners are started and the initializers are run, AdaNet estimator.train() passes its tf.Session as an argument to materialize_subnetwork_reports(), thus indirectly making input_fn available to materialize_subnetwork_reports.

materialize_subnetwork_reports(*sess*, *iteration_number*, *subnetwork_reports*, *included_subnetwork_names*)

Materializes the Tensor objects in subnetwork_reports using sess.

This converts the Tensors in subnetwork_reports to ndarrays, logs the progress, converts the ndarrays to python primitives, then packages them into *adanet.subnetwork.MaterializedReports*.

Parameters

- **sess** – *Session* instance with most recent variable values loaded.
- **iteration_number** – Integer iteration number.
- **subnetwork_reports** – Dict mapping string names to *subnetwork.Report* objects to be materialized.
- **included_subnetwork_names** – List of string names of the ‘subnetwork.Report‘s that are included in the final ensemble.

Returns List of *adanet.subnetwork.MaterializedReport* objects.

steps

Return the number of steps.

CHAPTER 2

adanet.subnetwork

Low-level APIs for defining custom subnetworks and search spaces.

2.1 Generators

Interfaces and containers for defining subnetworks, search spaces, and search algorithms.

2.1.1 Subnetwork

class adanet.subnetwork.**Subnetwork**

An AdaNet subnetwork.

In the AdaNet paper, an *adanet.subnetwork.Subnetwork* is are called a 'subnetwork', and indicated by 'h'. A collection of weighted subnetworks form an AdaNet ensemble.

Parameters

- **last_layer** – tf.Tensor output or dict of string to tf.Tensor outputs (for multi-head) of the last layer of the subnetwork, i.e the layer before the logits layer. When the mixture weight type is MATRIX, the AdaNet algorithm takes care of computing ensemble mixture weights matrices (one per subnetwork) that multiply the various last layers of the ensemble's subnetworks, and regularize them using their subnetwork's complexity. This field is represented by 'h' in the AdaNet paper.
- **logits** – tf.Tensor logits or dict of string to tf.Tensor logits (for multi-head) for training the subnetwork. These logits are not used in the ensemble's outputs if the mixture weight type is MATRIX, instead AdaNet learns its own logits (mixture weights) from the subnetwork's *last_layers* with complexity regularization. The logits are used in the ensemble only when the mixture weights type is SCALAR or VECTOR. Even though the logits are not used in the ensemble in some cases, they should always be supplied as adanet uses the logits to train the subnetworks.

- **complexity** – A scalar `tf.Tensor` representing the complexity of the subnetwork's architecture. It is used for choosing the best subnetwork at each iteration, and for regularizing the weighted outputs of more complex subnetworks.
- **persisted_tensors** – DEPRECATED. See *shared*. Optional nested dictionary of string to `tf.Tensor` to persist across iterations. At the end of an iteration, the `tf.Tensor` instances will be available to subnetworks in the next iterations, whereas others that are not part of the *Subnetwork* will be pruned. This allows later *adanet.subnetwork.Subnetwork* instances to dynamically build upon arbitrary `tf.Tensors` from previous *adanet.subnetwork.Subnetwork* instances.
- **shared** – Optional Python object(s), primitive(s), or function(s) to share with subnetworks within the same iteration or in future iterations.

Returns A validated *adanet.subnetwork.Subnetwork* object.

Raises

- `ValueError` – If last_layer is None.
- `ValueError` – If logits is None.
- `ValueError` – If logits is a dict but last_layer is not.
- `ValueError` – If last_layer is a dict but logits is not.
- `ValueError` – If complexity is None.
- `ValueError` – If persisted_tensors is present but not a dictionary.
- `ValueError` – If persisted_tensors contains an empty nested dictionary.

2.1.2 TrainOpSpec

class adanet.subnetwork.**TrainOpSpec**

A data structure for specifying training operations.

Parameters

- **train_op** – Op for the training step.
- **chief_hooks** – Iterable of `tf.train.SessionRunHook` objects to run on the chief worker during training.
- **hooks** – Iterable of `tf.train.SessionRunHook` objects to run on all workers during training.

Returns A *adanet.subnetwork.TrainOpSpec* object.

2.1.3 Builder

class adanet.subnetwork.**Builder**

Bases: `object`

Interface for a subnetwork builder.

Given features, labels, and the best ensemble of subnetworks at iteration t-1, a *Builder* creates a *Subnetwork* to add to a candidate ensemble at iteration t. These candidate ensembles are evaluated against one another at the end of the iteration, and the best one is selected based on its complexity-regularized loss.

build_mixture_weights_train_op(*loss*, *var_list*, *logits*, *labels*, *iteration_step*, *summary*)
Returns an op for training the ensemble's mixture weights.

Allows AdaNet to learn the mixture weights of each subnetwork according to Equation (6).

This method will be called once after *build_subnetwork*.

Accessing the global step via `tf.train.get_or_create_global_step()` or `tf.train.get_global_step()` within this scope will return an incrementable iteration step since the beginning of the iteration.

Parameters

- **loss** – A `tf.Tensor` containing the ensemble's loss to minimize.
- **var_list** – List of ensemble mixture weight *tf.Variables* to update as become part of the training operation.
- **logits** – The ensemble's logits `tf.Tensor` from applying the mixture weights and bias to the ensemble's subnetworks.
- **labels** – Labels `tf.Tensor` or a dictionary of string label name to `tf.Tensor` (for multi-head).
- **iteration_step** – Integer `tf.Tensor` representing the step since the beginning of the current iteration, as opposed to the global step.
- **summary** – An *adanet.Summary* for scoping summaries to individual subnetworks in Tensorboard. Using `tf.summary` within this scope will use this *adanet.Summary* under the hood.

Returns Either a train op or an *adanet.subnetwork.TrainOpSpec*.

build_subnetwork(*features*, *labels*, *logits_dimension*, *training*, *iteration_step*, *summary*, *previous_ensemble=None*)
Returns the candidate *Subnetwork* to add to the ensemble.

This method will be called only once, before *build_subnetwork_train_op()* and *build_mixture_weights_train_op()* are called. This method should construct the candidate subnetwork's graph operations and variables.

Accessing the global step via `tf.train.get_or_create_global_step()` or `tf.train.get_global_step()` within this scope will return an incrementable iteration step since the beginning of the iteration.

Parameters

- **features** – Input *dict* of `tf.Tensor` objects.
- **labels** – Labels `tf.Tensor` or a dictionary of string label name to `tf.Tensor` (for multi-head). Can be *None*.
- **logits_dimension** – Size of the last dimension of the logits `tf.Tensor`. Typically, logits have for shape *[batch_size, logits_dimension]*.
- **training** – A python boolean indicating whether the graph is in training mode or prediction mode.
- **iteration_step** – Integer `tf.Tensor` representing the step since the beginning of the current iteration, as opposed to the global step.
- **summary** – An *adanet.Summary* for scoping summaries to individual subnetworks in Tensorboard. Using `tf.summary()` within this scope will use this *adanet.Summary* under the hood.

- **previous_ensemble** – The best *adanet.Ensemble* from iteration t-1. The created subnetwork will extend the previous ensemble to form the *adanet.Ensemble* at iteration t.

Returns An *adanet.subnetwork.Subnetwork* instance.

build_subnetwork_report()

Returns a *subnetwork.Report* to materialize and record.

This method will be called once after *build_subnetwork()*. Do NOT depend on variables created in *build_subnetwork_train_op()* or *build_mixture_weights_train_op()*, because they are not called before *build_subnetwork_report()* is called.

If it returns None, AdaNet records the name and standard eval metrics.

build_subnetwork_train_op(*subnetwork*, *loss*, *var_list*, *labels*, *iteration_step*, *summary*, *previous_ensemble*)

Returns an op for training a new subnetwork.

This method will be called once after *build_subnetwork()*.

Accessing the global step via `tf.train.get_or_create_global_step()` or `tf.train.get_global_step()` within this scope will return an incrementable iteration step since the beginning of the iteration.

Parameters

- **subnetwork** – Newest subnetwork, that is not part of the *previous_ensemble*.
- **loss** – A `tf.Tensor` containing the subnetwork's loss to minimize.
- **var_list** – List of subnetwork `tf.Variable` parameters to update as part of the training operation.
- **labels** – Labels `tf.Tensor` or a dictionary of string label name to `tf.Tensor` (for multi-head).
- **iteration_step** – Integer `tf.Tensor` representing the step since the beginning of the current iteration, as opposed to the global step.
- **summary** – An *adanet.Summary* for scoping summaries to individual subnetworks in Tensorboard. Using *tf.summary* within this scope will use this *adanet.Summary* under the hood.
- **previous_ensemble** – The best *Ensemble* from iteration t-1. The created subnetwork will extend the previous ensemble to form the *Ensemble* at iteration t. Is None for iteration 0.

Returns Either a train op or an *adanet.subnetwork.TrainOpSpec*.

name

Returns the unique name of this subnetwork within an iteration.

prune_previous_ensemble(*previous_ensemble*)

Specifies which subnetworks from the previous ensemble to keep.

The selected subnetworks from the previous ensemble will be kept in the candidate ensemble that includes this subnetwork.

By default, none of the previous ensemble subnetworks are pruned.

Parameters **previous_ensemble** – *adanet.Ensemble* object.

Returns List of integer indices of *weighted_subnetworks* to keep.

2.1.4 Generator

class adanet.subnetwork.**Generator**
Bases: object

Interface for a candidate subnetwork generator.

Given the ensemble of subnetworks at iteration t-1, this object is responsible for generating the set of candidate subnetworks for iteration t that minimize the objective as part of an ensemble.

generate_candidates(*previous_ensemble*, *iteration_number*, *previous_ensemble_reports*, *all_reports*)
Generates *adanet.subnetwork.Builder* instances for an iteration.

NOTE: Every call to *generate_candidates()* must be deterministic for the given arguments.

Parameters

- **previous_ensemble** – The best *adanet.Ensemble* from iteration t-1. DEPRECATED. We are transitioning away from the use of previous_ensemble in generate_candidates. New Generators should *not* use previous_ensemble in their implementation of generate_candidates – please only use iteration_number, previous_ensemble_reports and all_reports.
- **iteration_number** – Python integer AdaNet iteration t, starting from 0.
- **previous_ensemble_reports** – List of *adanet.subnetwork.MaterializedReport* instances corresponding to the Builders composing *adanet.Ensemble* from iteration t-1. The first element in the list corresponds to the Builder added in the first iteration. If a *adanet.subnetwork.MaterializedReport* is not supplied to the estimator, previous_ensemble_report is *None*.
- **all_reports** – List of *adanet.subnetwork.MaterializedReport* instances. If an adanet.subnetwork.ReportMaterializer is not supplied to the estimator, *all_reports* is *None*. If adanet.subnetwork.ReportMaterializer is supplied to the estimator and t=0, *all_reports* is an empty List. Otherwise, *all_reports* is a sequence of Lists. Each element of the sequence is a List containing all the *adanet.subnetwork.MaterializedReport* instances in an AdaNet iteration, starting from iteration 0, and ending at iteration t-1.

Returns A list of *adanet.subnetwork.Builder* instances.

2.2 Reports

Containers for metadata about trained subnetworks.

2.2.1 Report

class adanet.subnetwork.**Report**
A container for data to be collected about a *Subnetwork*.

Parameters

- **hparams** – A dict mapping strings to python strings, ints, bools, or floats. It is meant to contain the constants that define the *adanet.subnetwork.Builder*, such as dropout, number of layers, or initial learning rate.

- **attributes** – A dict mapping strings to rank 0 Tensors of dtype string, int32, or float32. It is meant to contain properties that may or may not change over the course of training the *adanet.subnetwork.Subnetwork*, such as the number of parameters, the Lipschitz constant, the L_2 norm of the weights, or learning rate at materialization time.
- **metrics** – Dict of metric results keyed by name. The values of the dict are the results of calling a metric function, namely a *(metric_tensor, update_op)* tuple. *metric_tensor* should be evaluated without any impact on state (typically is a pure computation results based on variables.). For example, it should not trigger the *update_op* or requires any input fetching. This is meant to contain metrics of interest, such as the training loss, complexity regularized loss, or standard deviation of the last layer outputs.

Returns A validated *adanet.subnetwork.Report* object.

Raises `ValueError` – If validation fails.

2.2.2 MaterializedReport

class adanet.subnetwork.**MaterializedReport**

Data collected about a *adanet.subnetwork.Subnetwork*.

Parameters

- **iteration_number** – A python integer for the AdaNet iteration number, starting from 0.
- **name** – A string, which is either the name of the corresponding Builder, or "previous_ensemble" if it refers to the previous_ensemble.
- **hparams** – A dict mapping strings to python strings, ints, or floats. These are constants passed from the author of the *adanet.subnetwork.Builder* that was used to construct this *adanet.subnetwork.Subnetwork*. It is meant to contain the arguments that defined the *adanet.subnetwork.Builder*, such as dropout, number of layers, or initial learning rate.
- **attributes** – A dict mapping strings to python strings, ints, bools, or floats. These are python primitives that come from materialized Tensors; these Tensors were defined by the author of the *adanet.subnetwork.Builder* that was used to construct this *adanet.subnetwork.Subnetwork*. It is meant to contain properties that may or may not change over the course of training the *adanet.subnetwork.Subnetwork*, such as the number of parameters, the Lipschitz constant, or the L_2 norm of the weights.
- **metrics** – A dict mapping strings to python strings, ints, or floats. These are python primitives that come from metrics that were evaluated on the trained *adanet.subnetwork.Subnetwork* over some dataset; these metrics were defined by the author of the *adanet.subnetwork.Builder* that was used to construct this *adanet.subnetwork.Subnetwork*. It is meant to contain performance metrics or measures that could predict generalization, such as the training loss, complexity regularized loss, or standard deviation of the last layer outputs.
- **included_in_final_ensemble** – A boolean denoting whether the associated *adanet.subnetwork.Subnetwork* was included in the ensemble at the end of the AdaNet iteration.

Returns An *adanet.subnetwork.MaterializedReport* object.

CHAPTER 3

Indices and tables

- genindex
- modindex

Python Module Index

a

Index

M

N

P

R

S

T

W