Module: tf.compat.v1.random / tf.random

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random#top_of_page)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random#modules)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random#functions)

Public API for tf.random namespace.

Modules

[experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random/experimental) module: Public API for tf.random.experimental namespace.

Functions

[all\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/all_candidate_sampler): Generate the set of all classes.

[categorical(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/categorical): Draws samples from a categorical distribution.

[fixed\_unigram\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/fixed_unigram_candidate_sampler): Samples a set of classes using the provided (fixed) base distribution.

[gamma(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/gamma): Draws shape samples from each of the given Gamma distribution(s).

[get\_seed(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_seed): Returns the local seeds an operation should use given an op-specific seed.

[learned\_unigram\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/learned_unigram_candidate_sampler): Samples a set of classes from a distribution learned during training.

[log\_uniform\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/log_uniform_candidate_sampler): Samples a set of classes using a log-uniform (Zipfian) base distribution.

[multinomial(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/multinomial): Draws samples from a multinomial distribution. (deprecated)

[normal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal): Outputs random values from a normal distribution.

[poisson(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_poisson): Draws shape samples from each of the given Poisson distribution(s).

[set\_random\_seed(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed): Sets the graph-level random seed for the default graph.

[shuffle(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/shuffle): Randomly shuffles a tensor along its first dimension.

[stateless\_categorical(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_categorical): Draws deterministic pseudorandom samples from a categorical distribution.

[stateless\_multinomial(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random/stateless_multinomial): Draws deterministic pseudorandom samples from a multinomial distribution. (deprecated)

[stateless\_normal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_normal): Outputs deterministic pseudorandom values from a normal distribution.

[stateless\_truncated\_normal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_truncated_normal): Outputs deterministic pseudorandom values, truncated normally distributed.

[stateless\_uniform(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_uniform): Outputs deterministic pseudorandom values from a uniform distribution.

[truncated\_normal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal): Outputs random values from a truncated normal distribution.

[uniform(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform): Outputs random values from a uniform distribution.

[uniform\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform_candidate_sampler): Samples a set of classes using a uniform base distribution.

# tf.compat.v1.random.stateless\_multinomial

Draws deterministic pseudorandom samples from a multinomial distribution. (deprecated)

tf.compat.v1.random.stateless\_multinomial(  
    logits,  
    num\_samples,  
    seed,  
    output\_dtype=tf.dtypes.int64,  
    name=None  
)

Defined in [python/ops/stateless\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateless_random_ops.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.random.stateless\_categorical**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_categorical) instead.

This is a stateless version of [tf.random.categorical](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/categorical): if run twice with the same seeds, it will produce the same pseudorandom numbers. The output is consistent across multiple runs on the same hardware (and between CPU and GPU), but may change between versions of TensorFlow or on non-CPU/GPU hardware.

#### Example:

# samples has shape [1, 5], where each value is either 0 or 1 with equal  
# probability.  
samples = tf.random.stateless\_categorical(  
    tf.math.log([[10., 10.]]), 5, seed=[7, 17])

#### Args:

* **logits**: 2-D Tensor with shape [batch\_size, num\_classes]. Each slice [i, :] represents the unnormalized log-probabilities for all classes.
* **num\_samples**: 0-D. Number of independent samples to draw for each row slice.
* **seed**: A shape [2] integer Tensor of seeds to the random number generator.
* **output\_dtype**: integer type to use for the output. Defaults to int64.
* **name**: Optional name for the operation.

#### Returns:

The drawn samples of shape [batch\_size, num\_samples].

# tf.random.all\_candidate\_sampler

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/all_candidate_sampler#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/all_candidate_sampler#aliases)

Generate the set of all classes.

### Aliases:

* tf.compat.v1.nn.all\_candidate\_sampler
* tf.compat.v1.random.all\_candidate\_sampler
* tf.compat.v2.nn.all\_candidate\_sampler
* tf.compat.v2.random.all\_candidate\_sampler
* tf.nn.all\_candidate\_sampler
* tf.random.all\_candidate\_sampler

tf.random.all\_candidate\_sampler(  
    true\_classes,  
    num\_true,  
    num\_sampled,  
    unique,  
    seed=None,  
    name=None  
)

Defined in [python/ops/candidate\_sampling\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/candidate_sampling_ops.py).

Deterministically generates and returns the set of all possible classes. For testing purposes. There is no need to use this, since you might as well use full softmax or full logistic regression.

#### Args:

* **true\_classes**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **num\_true**: An int. The number of target classes per training example.
* **num\_sampled**: An int. The number of possible classes.
* **unique**: A bool. Ignored. unique.
* **seed**: An int. An operation-specific seed. Default is 0.
* **name**: A name for the operation (optional).

#### Returns:

* **sampled\_candidates**: A tensor of type int64 and shape [num\_sampled]. This operation deterministically returns the entire range [0, num\_sampled].
* **true\_expected\_count**: A tensor of type float. Same shape as true\_classes. The expected counts under the sampling distribution of each of true\_classes. All returned values are 1.0.
* **sampled\_expected\_count**: A tensor of type float. Same shape as sampled\_candidates. The expected counts under the sampling distribution of each of sampled\_candidates. All returned values are 1.0.

# tf.random.categorical

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/categorical#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/categorical#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/categorical#used_in_the_tutorials)

Draws samples from a categorical distribution.

### Aliases:

* tf.compat.v1.random.categorical
* tf.compat.v2.random.categorical
* tf.random.categorical

tf.random.categorical(  
    logits,  
    num\_samples,  
    dtype=None,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

### Used in the tutorials:

* [Text generation with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_generation)

#### Example:

# samples has shape [1, 5], where each value is either 0 or 1 with equal  
# probability.  
samples = tf.random.categorical(tf.math.log([[10., 10.]]), 5)

#### Args:

* **logits**: 2-D Tensor with shape [batch\_size, num\_classes]. Each slice [i, :] represents the unnormalized log-probabilities for all classes.
* **num\_samples**: 0-D. Number of independent samples to draw for each row slice.
* **dtype**: integer type to use for the output. Defaults to int64.
* **seed**: A Python integer. Used to create a random seed for the distribution. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: Optional name for the operation.

#### Returns:

The drawn samples of shape [batch\_size, num\_samples].

# tf.random.fixed\_unigram\_candidate\_sampler

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/fixed_unigram_candidate_sampler#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/fixed_unigram_candidate_sampler#aliases)

Samples a set of classes using the provided (fixed) base distribution.

### Aliases:

* tf.compat.v1.nn.fixed\_unigram\_candidate\_sampler
* tf.compat.v1.random.fixed\_unigram\_candidate\_sampler
* tf.compat.v2.nn.fixed\_unigram\_candidate\_sampler
* tf.compat.v2.random.fixed\_unigram\_candidate\_sampler
* tf.nn.fixed\_unigram\_candidate\_sampler
* tf.random.fixed\_unigram\_candidate\_sampler

tf.random.fixed\_unigram\_candidate\_sampler(  
    true\_classes,  
    num\_true,  
    num\_sampled,  
    unique,  
    range\_max,  
    vocab\_file='',  
    distortion=1.0,  
    num\_reserved\_ids=0,  
    num\_shards=1,  
    shard=0,  
    unigrams=(),  
    seed=None,  
    name=None  
)

Defined in [python/ops/candidate\_sampling\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/candidate_sampling_ops.py).

This operation randomly samples a tensor of sampled classes (sampled\_candidates) from the range of integers [0, range\_max).

The elements of sampled\_candidates are drawn without replacement (if unique=True) or with replacement (if unique=False) from the base distribution.

The base distribution is read from a file or passed in as an in-memory array. There is also an option to skew the distribution by applying a distortion power to the weights.

In addition, this operation returns tensors true\_expected\_count and sampled\_expected\_countrepresenting the number of times each of the target classes (true\_classes) and the sampled classes (sampled\_candidates) is expected to occur in an average tensor of sampled classes. These values correspond to Q(y|x) defined in [this document](http://www.tensorflow.org/extras/candidate_sampling.pdf). If unique=True, then these are post-rejection probabilities and we compute them approximately.

#### Args:

* **true\_classes**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **num\_true**: An int. The number of target classes per training example.
* **num\_sampled**: An int. The number of classes to randomly sample.
* **unique**: A bool. Determines whether all sampled classes in a batch are unique.
* **range\_max**: An int. The number of possible classes.
* **vocab\_file**: Each valid line in this file (which should have a CSV-like format) corresponds to a valid word ID. IDs are in sequential order, starting from num\_reserved\_ids. The last entry in each line is expected to be a value corresponding to the count or relative probability. Exactly one of vocab\_file and unigrams needs to be passed to this operation.
* **distortion**: The distortion is used to skew the unigram probability distribution. Each weight is first raised to the distortion's power before adding to the internal unigram distribution. As a result,distortion = 1.0 gives regular unigram sampling (as defined by the vocab file), and distortion = 0.0 gives a uniform distribution.
* **num\_reserved\_ids**: Optionally some reserved IDs can be added in the range [0, num\_reserved\_ids) by the users. One use case is that a special unknown word token is used as ID 0. These IDs will have a sampling probability of 0.
* **num\_shards**: A sampler can be used to sample from a subset of the original range in order to speed up the whole computation through parallelism. This parameter (together with shard) indicates the number of partitions that are being used in the overall computation.
* **shard**: A sampler can be used to sample from a subset of the original range in order to speed up the whole computation through parallelism. This parameter (together with num\_shards) indicates the particular partition number of the operation, when partitioning is being used.
* **unigrams**: A list of unigram counts or probabilities, one per ID in sequential order. Exactly one of vocab\_file and unigrams should be passed to this operation.
* **seed**: An int. An operation-specific seed. Default is 0.
* **name**: A name for the operation (optional).

#### Returns:

* **sampled\_candidates**: A tensor of type int64 and shape [num\_sampled]. The sampled classes.
* **true\_expected\_count**: A tensor of type float. Same shape as true\_classes. The expected counts under the sampling distribution of each of true\_classes.
* **sampled\_expected\_count**: A tensor of type float. Same shape as sampled\_candidates. The expected counts under the sampling distribution of each of sampled\_candidates.

# tf.random.gamma

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/gamma#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/gamma#aliases)

Draws shape samples from each of the given Gamma distribution(s).

### Aliases:

* tf.compat.v1.random.gamma
* tf.compat.v1.random\_gamma
* tf.compat.v2.random.gamma
* tf.random.gamma

tf.random.gamma(  
    shape,  
    alpha,  
    beta=None,  
    dtype=tf.dtypes.float32,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

alpha is the shape parameter describing the distribution(s), and beta is the inverse scale parameter(s).

**Note:** Because internal calculations are done using **float64** and casting has **floor** semantics, we must manually map zero outcomes to the smallest possible positive floating-point value, i.e., **np.finfo(dtype).tiny**. This means that **np.finfo(dtype).tiny** occurs more frequently than it otherwise should. This bias can only happen for small values of **alpha**, i.e., **alpha << 1** or large values of **beta**, i.e., **beta >> 1**.

The samples are differentiable w.r.t. alpha and beta. The derivatives are computed using the approach described in the paper

[Michael Figurnov, Shakir Mohamed, Andriy Mnih. Implicit Reparameterization Gradients, 2018](https://arxiv.org/abs/1805.08498)

#### Example:

samples = tf.random.gamma([10], [0.5, 1.5])  
# samples has shape [10, 2], where each slice [:, 0] and [:, 1] represents  
# the samples drawn from each distribution  
  
samples = tf.random.gamma([7, 5], [0.5, 1.5])  
# samples has shape [7, 5, 2], where each slice [:, :, 0] and [:, :, 1]  
# represents the 7x5 samples drawn from each of the two distributions  
  
alpha = tf.constant([[1.],[3.],[5.]])  
beta = tf.constant([[3., 4.]])  
samples = tf.random.gamma([30], alpha=alpha, beta=beta)  
# samples has shape [30, 3, 2], with 30 samples each of 3x2 distributions.  
  
loss = tf.reduce\_mean(tf.square(samples))  
dloss\_dalpha, dloss\_dbeta = tf.gradients(loss, [alpha, beta])  
# unbiased stochastic derivatives of the loss function  
alpha.shape == dloss\_dalpha.shape  # True  
beta.shape == dloss\_dbeta.shape  # True

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output samples to be drawn per alpha/beta-parameterized distribution.
* **alpha**: A Tensor or Python value or N-D array of type dtype. alpha provides the shape parameter(s) describing the gamma distribution(s) to sample. Must be broadcastable with beta.
* **beta**: A Tensor or Python value or N-D array of type dtype. Defaults to 1. beta provides the inverse scale parameter(s) of the gamma distribution(s) to sample. Must be broadcastable with alpha.
* **dtype**: The type of alpha, beta, and the output: float16, float32, or float64.
* **seed**: A Python integer. Used to create a random seed for the distributions. See[tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: Optional name for the operation.

#### Returns:

* **samples**: a Tensor of shape tf.concat([shape, tf.shape(alpha + beta)], axis=0)with values of type dtype.

# tf.random.learned\_unigram\_candidate\_sampler

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/learned_unigram_candidate_sampler#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/learned_unigram_candidate_sampler#aliases)

Samples a set of classes from a distribution learned during training.

### Aliases:

* tf.compat.v1.nn.learned\_unigram\_candidate\_sampler
* tf.compat.v1.random.learned\_unigram\_candidate\_sampler
* tf.compat.v2.nn.learned\_unigram\_candidate\_sampler
* tf.compat.v2.random.learned\_unigram\_candidate\_sampler
* tf.nn.learned\_unigram\_candidate\_sampler
* tf.random.learned\_unigram\_candidate\_sampler

tf.random.learned\_unigram\_candidate\_sampler(  
    true\_classes,  
    num\_true,  
    num\_sampled,  
    unique,  
    range\_max,  
    seed=None,  
    name=None  
)

Defined in [python/ops/candidate\_sampling\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/candidate_sampling_ops.py).

This operation randomly samples a tensor of sampled classes (sampled\_candidates) from the range of integers [0, range\_max).

The elements of sampled\_candidates are drawn without replacement (if unique=True) or with replacement (if unique=False) from the base distribution.

The base distribution for this operation is constructed on the fly during training. It is a unigram distribution over the target classes seen so far during training. Every integer in [0, range\_max)begins with a weight of 1, and is incremented by 1 each time it is seen as a target class. The base distribution is not saved to checkpoints, so it is reset when the model is reloaded.

In addition, this operation returns tensors true\_expected\_count and sampled\_expected\_countrepresenting the number of times each of the target classes (true\_classes) and the sampled classes (sampled\_candidates) is expected to occur in an average tensor of sampled classes. These values correspond to Q(y|x) defined in [this document](http://www.tensorflow.org/extras/candidate_sampling.pdf). If unique=True, then these are post-rejection probabilities and we compute them approximately.

#### Args:

* **true\_classes**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **num\_true**: An int. The number of target classes per training example.
* **num\_sampled**: An int. The number of classes to randomly sample.
* **unique**: A bool. Determines whether all sampled classes in a batch are unique.
* **range\_max**: An int. The number of possible classes.
* **seed**: An int. An operation-specific seed. Default is 0.
* **name**: A name for the operation (optional).

#### Returns:

* **sampled\_candidates**: A tensor of type int64 and shape [num\_sampled]. The sampled classes.
* **true\_expected\_count**: A tensor of type float. Same shape as true\_classes. The expected counts under the sampling distribution of each of true\_classes.
* **sampled\_expected\_count**: A tensor of type float. Same shape as sampled\_candidates. The expected counts under the sampling distribution of each of sampled\_candidates.

# tf.random.log\_uniform\_candidate\_sampler

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/log_uniform_candidate_sampler#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/log_uniform_candidate_sampler#aliases)

Samples a set of classes using a log-uniform (Zipfian) base distribution.

### Aliases:

* tf.compat.v1.nn.log\_uniform\_candidate\_sampler
* tf.compat.v1.random.log\_uniform\_candidate\_sampler
* tf.compat.v2.random.log\_uniform\_candidate\_sampler
* tf.random.log\_uniform\_candidate\_sampler

tf.random.log\_uniform\_candidate\_sampler(  
    true\_classes,  
    num\_true,  
    num\_sampled,  
    unique,  
    range\_max,  
    seed=None,  
    name=None  
)

Defined in [python/ops/candidate\_sampling\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/candidate_sampling_ops.py).

This operation randomly samples a tensor of sampled classes (sampled\_candidates) from the range of integers [0, range\_max).

The elements of sampled\_candidates are drawn without replacement (if unique=True) or with replacement (if unique=False) from the base distribution.

The base distribution for this operation is an approximately log-uniform or Zipfian distribution:

P(class) = (log(class + 2) - log(class + 1)) / log(range\_max + 1)

This sampler is useful when the target classes approximately follow such a distribution - for example, if the classes represent words in a lexicon sorted in decreasing order of frequency. If your classes are not ordered by decreasing frequency, do not use this op.

In addition, this operation returns tensors true\_expected\_count and sampled\_expected\_countrepresenting the number of times each of the target classes (true\_classes) and the sampled classes (sampled\_candidates) is expected to occur in an average tensor of sampled classes. These values correspond to Q(y|x) defined in [this document](http://www.tensorflow.org/extras/candidate_sampling.pdf). If unique=True, then these are post-rejection probabilities and we compute them approximately.

#### Args:

* **true\_classes**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **num\_true**: An int. The number of target classes per training example.
* **num\_sampled**: An int. The number of classes to randomly sample.
* **unique**: A bool. Determines whether all sampled classes in a batch are unique.
* **range\_max**: An int. The number of possible classes.
* **seed**: An int. An operation-specific seed. Default is 0.
* **name**: A name for the operation (optional).

#### Returns:

* **sampled\_candidates**: A tensor of type int64 and shape [num\_sampled]. The sampled classes.
* **true\_expected\_count**: A tensor of type float. Same shape as true\_classes. The expected counts under the sampling distribution of each of true\_classes.
* **sampled\_expected\_count**: A tensor of type float. Same shape as sampled\_candidates. The expected counts under the sampling distribution of each of sampled\_candidates.

# tf.random.normal

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal#used_in_the_guide)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal#used_in_the_tutorials)

Outputs random values from a normal distribution.

### Aliases:

* tf.compat.v1.random.normal
* tf.compat.v1.random\_normal
* tf.compat.v2.random.normal
* tf.random.normal

tf.random.normal(  
    shape,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)

### Used in the tutorials:

* [Convolutional Variational Autoencoder](https://www.tensorflow.org/beta/tutorials/generative/cvae)
* [Custom training: basics](https://www.tensorflow.org/beta/tutorials/eager/custom_training)
* [Deep Convolutional Generative Adversarial Network](https://www.tensorflow.org/beta/tutorials/generative/dcgan)

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution.
* **dtype**: The type of the output.
* **seed**: A Python integer. Used to create a random seed for the distribution. See[tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random normal values.

# tf.random.poisson

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/poisson#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/poisson#aliases)

Draws shape samples from each of the given Poisson distribution(s).

### Aliases:

* tf.compat.v2.random.poisson
* tf.random.poisson

tf.random.poisson(  
    shape,  
    lam,  
    dtype=tf.dtypes.float32,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

lam is the rate parameter describing the distribution(s).

#### Example:

samples = tf.random.poisson([10], [0.5, 1.5])  
# samples has shape [10, 2], where each slice [:, 0] and [:, 1] represents  
# the samples drawn from each distribution  
  
samples = tf.random.poisson([7, 5], [12.2, 3.3])  
# samples has shape [7, 5, 2], where each slice [:, :, 0] and [:, :, 1]  
# represents the 7x5 samples drawn from each of the two distributions

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output samples to be drawn per "rate"-parameterized distribution.
* **lam**: A Tensor or Python value or N-D array of type dtype. lam provides the rate parameter(s) describing the poisson distribution(s) to sample.
* **dtype**: The type of the output: float16, float32, float64, int32 or int64.
* **seed**: A Python integer. Used to create a random seed for the distributions. See[tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: Optional name for the operation.

#### Returns:

* **samples**: a Tensor of shape tf.concat([shape, tf.shape(lam)], axis=0) with values of type dtype.

# tf.random.set\_seed

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/set_seed#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/set_seed#aliases)

Sets the graph-level random seed.

### Aliases:

* tf.compat.v2.random.set\_seed
* tf.random.set\_seed

tf.random.set\_seed(seed)

Defined in [python/framework/random\_seed.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/random_seed.py).

Operations that rely on a random seed actually derive it from two seeds: the graph-level and operation-level seeds. This sets the graph-level seed.

Its interactions with operation-level seeds is as follows:

1. If neither the graph-level nor the operation seed is set: A random seed is used for this op.
2. If the graph-level seed is set, but the operation seed is not: The system deterministically picks an operation seed in conjunction with the graph-level seed so that it gets a unique random sequence.
3. If the graph-level seed is not set, but the operation seed is set: A default graph-level seed and the specified operation seed are used to determine the random sequence.
4. If both the graph-level and the operation seed are set: Both seeds are used in conjunction to determine the random sequence.

To illustrate the user-visible effects, consider these examples:

To generate different sequences across sessions, set neither graph-level nor op-level seeds:

a = tf.random.uniform([1])  
b = tf.random.normal([1])  
  
print("Session 1")  
with tf.compat.v1.Session() as sess1:  
  print(sess1.run(a))  # generates 'A1'  
  print(sess1.run(a))  # generates 'A2'  
  print(sess1.run(b))  # generates 'B1'  
  print(sess1.run(b))  # generates 'B2'  
  
print("Session 2")  
with tf.compat.v1.Session() as sess2:  
  print(sess2.run(a))  # generates 'A3'  
  print(sess2.run(a))  # generates 'A4'  
  print(sess2.run(b))  # generates 'B3'  
  print(sess2.run(b))  # generates 'B4'

To generate the same repeatable sequence for an op across sessions, set the seed for the op:

a = tf.random.uniform([1], seed=1)  
b = tf.random.normal([1])  
  
# Repeatedly running this block with the same graph will generate the same  
# sequence of values for 'a', but different sequences of values for 'b'.  
print("Session 1")  
with tf.compat.v1.Session() as sess1:  
  print(sess1.run(a))  # generates 'A1'  
  print(sess1.run(a))  # generates 'A2'  
  print(sess1.run(b))  # generates 'B1'  
  print(sess1.run(b))  # generates 'B2'  
  
print("Session 2")  
with tf.compat.v1.Session() as sess2:  
  print(sess2.run(a))  # generates 'A1'  
  print(sess2.run(a))  # generates 'A2'  
  print(sess2.run(b))  # generates 'B3'  
  print(sess2.run(b))  # generates 'B4'

To make the random sequences generated by all ops be repeatable across sessions, set a graph-level seed:

tf.random.set\_seed(1234)  
a = tf.random.uniform([1])  
b = tf.random.normal([1])  
  
# Repeatedly running this block with the same graph will generate the same  
# sequences of 'a' and 'b'.  
print("Session 1")  
with tf.compat.v1.Session() as sess1:  
  print(sess1.run(a))  # generates 'A1'  
  print(sess1.run(a))  # generates 'A2'  
  print(sess1.run(b))  # generates 'B1'  
  print(sess1.run(b))  # generates 'B2'  
  
print("Session 2")  
with tf.compat.v1.Session() as sess2:  
  print(sess2.run(a))  # generates 'A1'  
  print(sess2.run(a))  # generates 'A2'  
  print(sess2.run(b))  # generates 'B1'  
  print(sess2.run(b))  # generates 'B2'

#### Args:

* **seed**: integer.

# tf.random.shuffle

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/shuffle#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/shuffle#aliases)

Randomly shuffles a tensor along its first dimension.

### Aliases:

* tf.compat.v1.random.shuffle
* tf.compat.v1.random\_shuffle
* tf.compat.v2.random.shuffle
* tf.random.shuffle

tf.random.shuffle(  
    value,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

The tensor is shuffled along dimension 0, such that each value[j] is mapped to one and only one output[i]. For example, a mapping that might occur for a 3x2 tensor is:

[[1, 2],       [[5, 6],  
 [3, 4],  ==>   [1, 2],  
 [5, 6]]        [3, 4]]

#### Args:

* **value**: A Tensor to be shuffled.
* **seed**: A Python integer. Used to create a random seed for the distribution. See[tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of same shape and type as value, shuffled along its first dimension.

# tf.random.stateless\_categorical

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_categorical#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_categorical#aliases)

Draws deterministic pseudorandom samples from a categorical distribution.

### Aliases:

* tf.compat.v1.random.stateless\_categorical
* tf.compat.v2.random.stateless\_categorical
* tf.random.stateless\_categorical

tf.random.stateless\_categorical(  
    logits,  
    num\_samples,  
    seed,  
    dtype=tf.dtypes.int64,  
    name=None  
)

Defined in [python/ops/stateless\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateless_random_ops.py).

This is a stateless version of tf.categorical: if run twice with the same seeds, it will produce the same pseudorandom numbers. The output is consistent across multiple runs on the same hardware (and between CPU and GPU), but may change between versions of TensorFlow or on non-CPU/GPU hardware.

#### Example:

# samples has shape [1, 5], where each value is either 0 or 1 with equal  
# probability.  
samples = tf.random.stateless\_categorical(  
    tf.math.log([[10., 10.]]), 5, seed=[7, 17])

#### Args:

* **logits**: 2-D Tensor with shape [batch\_size, num\_classes]. Each slice [i, :] represents the unnormalized log-probabilities for all classes.
* **num\_samples**: 0-D. Number of independent samples to draw for each row slice.
* **seed**: A shape [2] integer Tensor of seeds to the random number generator.
* **dtype**: integer type to use for the output. Defaults to int64.
* **name**: Optional name for the operation.

#### Returns:

The drawn samples of shape [batch\_size, num\_samples].

# tf.random.stateless\_normal

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_normal#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_normal#aliases)

Outputs deterministic pseudorandom values from a normal distribution.

### Aliases:

* tf.compat.v1.random.stateless\_normal
* tf.compat.v2.random.stateless\_normal
* tf.random.stateless\_normal

tf.random.stateless\_normal(  
    shape,  
    seed,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Defined in [python/ops/stateless\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateless_random_ops.py).

This is a stateless version of [tf.random.normal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal): if run twice with the same seeds, it will produce the same pseudorandom numbers. The output is consistent across multiple runs on the same hardware (and between CPU and GPU), but may change between versions of TensorFlow or on non-CPU/GPU hardware.

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **seed**: A shape [2] integer Tensor of seeds to the random number generator.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution.
* **dtype**: The type of the output.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random normal values.

# tf.random.stateless\_truncated\_normal

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_truncated_normal#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_truncated_normal#aliases)

Outputs deterministic pseudorandom values, truncated normally distributed.

### Aliases:

* tf.compat.v1.random.stateless\_truncated\_normal
* tf.compat.v2.random.stateless\_truncated\_normal
* tf.random.stateless\_truncated\_normal

tf.random.stateless\_truncated\_normal(  
    shape,  
    seed,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Defined in [python/ops/stateless\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateless_random_ops.py).

This is a stateless version of [tf.random.truncated\_normal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal): if run twice with the same seeds, it will produce the same pseudorandom numbers. The output is consistent across multiple runs on the same hardware (and between CPU and GPU), but may change between versions of TensorFlow or on non-CPU/GPU hardware.

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **seed**: A shape [2] integer Tensor of seeds to the random number generator.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the truncated normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution, before truncation.
* **dtype**: The type of the output.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random truncated normal values.

# tf.random.stateless\_truncated\_normal

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_truncated_normal#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/stateless_truncated_normal#aliases)

Outputs deterministic pseudorandom values, truncated normally distributed.

### Aliases:

* tf.compat.v1.random.stateless\_truncated\_normal
* tf.compat.v2.random.stateless\_truncated\_normal
* tf.random.stateless\_truncated\_normal

tf.random.stateless\_truncated\_normal(  
    shape,  
    seed,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Defined in [python/ops/stateless\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateless_random_ops.py).

This is a stateless version of [tf.random.truncated\_normal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal): if run twice with the same seeds, it will produce the same pseudorandom numbers. The output is consistent across multiple runs on the same hardware (and between CPU and GPU), but may change between versions of TensorFlow or on non-CPU/GPU hardware.

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **seed**: A shape [2] integer Tensor of seeds to the random number generator.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the truncated normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution, before truncation.
* **dtype**: The type of the output.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random truncated normal values.

# tf.random.truncated\_normal

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal#used_in_the_guide)

Outputs random values from a truncated normal distribution.

### Aliases:

* tf.compat.v1.random.truncated\_normal
* tf.compat.v1.truncated\_normal
* tf.compat.v2.random.truncated\_normal
* tf.random.truncated\_normal

tf.random.truncated\_normal(  
    shape,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

### Used in the guide:

* [Ragged Tensors](https://www.tensorflow.org/beta/guide/ragged_tensors)

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the truncated normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution, before truncation.
* **dtype**: The type of the output.
* **seed**: A Python integer. Used to create a random seed for the distribution. See[tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random truncated normal values

.

# tf.random.uniform

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform#used_in_the_guide)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform#used_in_the_tutorials)

Outputs random values from a uniform distribution.

### Aliases:

* tf.compat.v1.random.uniform
* tf.compat.v1.random\_uniform
* tf.compat.v2.random.uniform
* tf.random.uniform

tf.random.uniform(  
    shape,  
    minval=0,  
    maxval=None,  
    dtype=tf.dtypes.float32,  
    seed=None,  
    name=None  
)

Defined in [python/ops/random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/random_ops.py).

### Used in the guide:

* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)
* [Tensors and Operations](https://www.tensorflow.org/beta/tutorials/eager/basics)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

The generated values follow a uniform distribution in the range [minval, maxval). The lower bound minval is included in the range, while the upper bound maxval is excluded.

For floats, the default range is [0, 1). For ints, at least maxval must be specified explicitly.

In the integer case, the random integers are slightly biased unless maxval - minval is an exact power of two. The bias is small for values of maxval - minval significantly smaller than the range of the output (either 2\*\*32 or 2\*\*64).

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **minval**: A 0-D Tensor or Python value of type dtype. The lower bound on the range of random values to generate. Defaults to 0.
* **maxval**: A 0-D Tensor or Python value of type dtype. The upper bound on the range of random values to generate. Defaults to 1 if dtype is floating point.
* **dtype**: The type of the output: float16, float32, float64, int32, or int64.
* **seed**: A Python integer. Used to create a random seed for the distribution. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random uniform values.

#### Raises:

* **ValueError**: If dtype is integral and maxval is not specified.

# tf.random.uniform\_candidate\_sampler

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform_candidate_sampler#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform_candidate_sampler#aliases)

Samples a set of classes using a uniform base distribution.

### Aliases:

* tf.compat.v1.nn.uniform\_candidate\_sampler
* tf.compat.v1.random.uniform\_candidate\_sampler
* tf.compat.v2.random.uniform\_candidate\_sampler
* tf.random.uniform\_candidate\_sampler

tf.random.uniform\_candidate\_sampler(  
    true\_classes,  
    num\_true,  
    num\_sampled,  
    unique,  
    range\_max,  
    seed=None,  
    name=None  
)

Defined in [python/ops/candidate\_sampling\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/candidate_sampling_ops.py).

This operation randomly samples a tensor of sampled classes (sampled\_candidates) from the range of integers [0, range\_max).

The elements of sampled\_candidates are drawn without replacement (if unique=True) or with replacement (if unique=False) from the base distribution.

The base distribution for this operation is the uniform distribution over the range of integers [0, range\_max).

In addition, this operation returns tensors true\_expected\_count and sampled\_expected\_countrepresenting the number of times each of the target classes (true\_classes) and the sampled classes (sampled\_candidates) is expected to occur in an average tensor of sampled classes. These values correspond to Q(y|x) defined in [this document](http://www.tensorflow.org/extras/candidate_sampling.pdf). If unique=True, then these are post-rejection probabilities and we compute them approximately.

#### Args:

* **true\_classes**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **num\_true**: An int. The number of target classes per training example.
* **num\_sampled**: An int. The number of classes to randomly sample. The sampled\_candidatesreturn value will have shape [num\_sampled]. If unique=True, num\_sampled must be less than or equal to range\_max.
* **unique**: A bool. Determines whether all sampled classes in a batch are unique.
* **range\_max**: An int. The number of possible classes.
* **seed**: An int. An operation-specific seed. Default is 0.
* **name**: A name for the operation (optional).

#### Returns:

* **sampled\_candidates**: A tensor of type int64 and shape [num\_sampled]. The sampled classes, either with possible duplicates (unique=False) or all unique (unique=True). In either case, sampled\_candidates is independent of the true classes.
* **true\_expected\_count**: A tensor of type float. Same shape as true\_classes. The expected counts under the sampling distribution of each of true\_classes.
* **sampled\_expected\_count**: A tensor of type float. Same shape as sampled\_candidates. The expected counts under the sampling distribution of each of sampled\_candidates.

Module: tf.random.experimental

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental#top_of_page)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental#functions)

Public API for tf.random.experimental namespace.

Classes

[class Generator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/Generator): Random-number generator.

Functions

[create\_rng\_state(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/create_rng_state): Creates a RNG state.

[get\_global\_generator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/get_global_generator)

[set\_global\_generator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/set_global_generator): Replaces the global generator with another Generator object.

# tf.random.experimental.create\_rng\_state

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/create_rng_state#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/create_rng_state#aliases)

Creates a RNG state.

### Aliases:

* tf.compat.v1.random.experimental.create\_rng\_state
* tf.compat.v2.random.experimental.create\_rng\_state
* tf.random.experimental.create\_rng\_state

tf.random.experimental.create\_rng\_state(  
    seed,  
    algorithm  
)

Defined in [python/ops/stateful\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateful_random_ops.py).

#### Args:

* **seed**: an integer or 1-D tensor.
* **algorithm**: an integer representing the RNG algorithm.

#### Returns:

a 1-D tensor whose size depends on the algorithm.

# tf.random.experimental.Generator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/Generator#top_of_page)
* [Class Generator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/Generator#class_generator)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/Generator#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/Generator#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/Generator#properties)

## Class Generator

Random-number generator.

### Aliases:

* Class tf.compat.v1.random.experimental.Generator
* Class tf.compat.v2.random.experimental.Generator
* Class tf.random.experimental.Generator

Defined in [python/ops/stateful\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateful_random_ops.py).

It uses Variable to manage its internal state, and allows choosing an Random-Number-Generation (RNG) algorithm.

CPU, GPU and TPU with the same algorithm and seed will generate the same integer random numbers. Float-point results (such as the output of normal) may have small numerical discrepancies between CPU and GPU.

## \_\_init\_\_

\_\_init\_\_(  
    copy\_from=None,  
    state=None,  
    alg=None  
)

Creates a generator.

The new generator will be initialized by one of the following ways, with decreasing precedence: (1) If copy\_from is not None, the new generator is initialized by copying information from another generator. (3) If state and alg are not None (they must be set together), the new generator is initialized by a state.

#### Args:

* **copy\_from**: a generator to be copied from.
* **state**: a vector of dtype STATE\_TYPE representing the initial state of the RNG, whose length and semantics are algorithm-specific.
* **alg**: the RNG algorithm. Possible values are RNG\_ALG\_PHILOX for the Philox algorithm and RNG\_ALG\_THREEFRY for the ThreeFry algorithm (see paper 'Parallel Random Numbers: As Easy as 1, 2, 3' [https://www.thesalmons.org/john/random123/papers/random123sc11.pdf]).

## Properties

### algorithm

The RNG algorithm.

### key

The 'key' part of the state of a counter-based RNG.

For a counter-base RNG algorithm such as Philox and ThreeFry (as described in paper 'Parallel Random Numbers: As Easy as 1, 2, 3' [https://www.thesalmons.org/john/random123/papers/random123sc11.pdf]), the RNG state consists of two parts: counter and key. The output is generated via the formula: output=hash(key, counter), i.e. a hashing of the counter parametrized by the key. Two RNGs with two different keys can be thought as generating two independent random-number streams (a stream is formed by increasing the counter).

#### Returns:

A scalar which is the 'key' part of the state, if the RNG algorithm is counter-based; otherwise it raises a ValueError.

### state

The internal state of the RNG.

## Methods

### binomial

binomial(  
    shape,  
    counts,  
    probs,  
    dtype=tf.dtypes.int32,  
    name=None  
)

Outputs random values from a binomial distribution.

The generated values follow a binomial distribution with specified count and probability of success parameters.

#### Example:

counts = [10., 20.]  
# Probability of success.  
probs = [0.8, 0.9]  
  
rng = tf.random.experimental.Generator(seed=234)  
binomial\_samples = rng.binomial(shape=[2], counts=counts, probs=probs)

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **counts**: A 0/1-D Tensor or Python value`. The counts of the binomial distribution.
* **probs**: A 0/1-D Tensor or Python value`. The probability of success for the binomial distribution.
* **dtype**: The type of the output. Default: tf.int32
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random binomial values.

### from\_key\_counter

@classmethod  
from\_key\_counter(  
    cls,  
    key,  
    counter,  
    alg  
)

Creates a generator from a key and a counter.

This constructor only applies if the algorithm is a counter-based algorithm. See method key for the meaning of "key" and "counter".

#### Args:

* **key**: the key for the RNG, a scalar of type STATE\_TYPE.
* **counter**: a vector of dtype STATE\_TYPE representing the initial counter for the RNG, whose length is algorithm-specific.,
* **alg**: the RNG algorithm. If None, it will be auto-selected. See \_\_init\_\_ for its possible values.

#### Returns:

The new generator.

### from\_non\_deterministic\_state

@classmethod  
from\_non\_deterministic\_state(  
    cls,  
    alg=None  
)

Creates a generator by non-deterministically initializing its state.

The source of the non-determinism will be platform- and time-dependent.

#### Args:

* **alg**: (optional) the RNG algorithm. If None, it will be auto-selected. See \_\_init\_\_ for its possible values.

#### Returns:

The new generator.

### from\_seed

@classmethod  
from\_seed(  
    cls,  
    seed,  
    alg=None  
)

Creates a generator from a seed.

A seed is a 1024-bit unsigned integer represented either as a Python integer or a vector of integers. Seeds shorter than 1024-bit will be padded. The padding, the internal structure of a seed and the way a seed is converted to a state are all opaque (unspecified). The only semantics specification of seeds is that two different seeds are likely to produce two independent generators (but no guarantee).

#### Args:

* **seed**: the seed for the RNG.
* **alg**: (optional) the RNG algorithm. If None, it will be auto-selected. See \_\_init\_\_ for its possible values.

#### Returns:

The new generator.

### from\_state

@classmethod  
from\_state(  
    cls,  
    state,  
    alg  
)

Creates a generator from a state.

See \_\_init\_\_ for description of state and alg.

#### Args:

* **state**: the new state.
* **alg**: the RNG algorithm.

#### Returns:

The new generator.

### make\_seeds

make\_seeds(count=1)

Generates seeds for stateless random ops.

#### For example:

seeds = get\_global\_generator().make\_seeds(count=10)  
for i in range(10):  
  seed = seeds[:, i]  
  numbers = stateless\_random\_normal(shape=[2, 3], seed=seed)  
  ...

#### Args:

* **count**: the number of seed pairs (note that stateless random ops need a pair of seeds to invoke).

#### Returns:

A tensor of shape [2, count] and dtype int64.

### normal

normal(  
    shape,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Outputs random values from a normal distribution.

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution.
* **dtype**: The type of the output.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random normal values.

### reset

reset(state)

Resets the generator by a new state.

See \_\_init\_\_ for the meaning of "state".

#### Args:

* **state**: the new state.

### reset\_from\_key\_counter

reset\_from\_key\_counter(  
    key,  
    counter  
)

Resets the generator by a new key-counter pair.

See from\_key\_counter for the meaning of "key" and "counter".

#### Args:

* **key**: the new key.
* **counter**: the new counter.

### reset\_from\_seed

reset\_from\_seed(seed)

Resets the generator by a new seed.

See from\_seed for the meaning of "seed".

#### Args:

* **seed**: the new seed.

### skip

skip(delta)

Advance the counter of a counter-based RNG.

#### Args:

* **delta**: the amount of advancement. The state of the RNG after skip(n) will be the same as that after normal([n]) (or any other distribution). The actual increment added to the counter is an unspecified implementation detail.

### split

split(count=1)

Returns a list of independent Generator objects.

Two generators are independent of each other in the sense that the random-number streams they generate don't have statistically detectable correlations. The new generators are also independent of the old one. The old generator's state will be changed (like other random-number generating methods), so two calls of split will return different new generators.

#### For example:

gens = get\_global\_generator().split(count=10)  
for gen in gens:  
  numbers = gen.normal(shape=[2, 3])  
  # ...  
gens2 = get\_global\_generator().split(count=10)  
# gens2 will be different from gens

The new generators will be put on the current device (possible different from the old generator's), for example:

with tf.device("/device:CPU:0"):  
  gen = Generator(seed=1234)  # gen is on CPU  
with tf.device("/device:GPU:0"):  
  gens = gen.split(count=10)  # gens are on GPU

#### Args:

* **count**: the number of generators to return.

#### Returns:

A list (length count) of Generator objects independent of each other. The new generators have the same RNG algorithm as the old one.

### truncated\_normal

truncated\_normal(  
    shape,  
    mean=0.0,  
    stddev=1.0,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Outputs random values from a truncated normal distribution.

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **mean**: A 0-D Tensor or Python value of type dtype. The mean of the truncated normal distribution.
* **stddev**: A 0-D Tensor or Python value of type dtype. The standard deviation of the normal distribution, before truncation.
* **dtype**: The type of the output.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random truncated normal values.

### uniform

uniform(  
    shape,  
    minval=0,  
    maxval=None,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Outputs random values from a uniform distribution.

The generated values follow a uniform distribution in the range [minval, maxval). The lower bound minval is included in the range, while the upper bound maxval is excluded. (For float numbers especially low-precision types like bfloat16, because of rounding, the result may sometimes include maxval.)

For floats, the default range is [0, 1). For ints, at least maxval must be specified explicitly.

In the integer case, the random integers are slightly biased unless maxval - minval is an exact power of two. The bias is small for values of maxval - minval significantly smaller than the range of the output (either 2\*\*32 or 2\*\*64).

#### Args:

* **shape**: A 1-D integer Tensor or Python array. The shape of the output tensor.
* **minval**: A 0-D Tensor or Python value of type dtype. The lower bound on the range of random values to generate. Defaults to 0.
* **maxval**: A 0-D Tensor or Python value of type dtype. The upper bound on the range of random values to generate. Defaults to 1 if dtype is floating point.
* **dtype**: The type of the output.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the specified shape filled with random uniform values.

#### Raises:

* **ValueError**: If dtype is integral and maxval is not specified.

### uniform\_full\_int

uniform\_full\_int(  
    shape,  
    dtype=tf.dtypes.uint64,  
    name=None  
)

Uniform distribution on an integer type's entire range.

The other method uniform only covers the range [minval, maxval), which cannot be dtype's full range because maxval is of type dtype.

#### Args:

* **shape**: the shape of the output.
* **dtype**: (optional) the integer type, default to uint64.
* **name**: (optional) the name of the node.

#### Returns:

A tensor of random numbers of the required shape.

# tf.random.experimental.get\_global\_generator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/get_global_generator#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/get_global_generator#aliases)

### Aliases:

* tf.compat.v1.random.experimental.get\_global\_generator
* tf.compat.v2.random.experimental.get\_global\_generator
* tf.random.experimental.get\_global\_generator

tf.random.experimental.get\_global\_generator()

Defined in [python/ops/stateful\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateful_random_ops.py).

# tf.random.experimental.set\_global\_generator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/set_global_generator#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/experimental/set_global_generator#aliases)

Replaces the global generator with another Generator object.

### Aliases:

* tf.compat.v1.random.experimental.set\_global\_generator
* tf.compat.v2.random.experimental.set\_global\_generator
* tf.random.experimental.set\_global\_generator

tf.random.experimental.set\_global\_generator(generator)

Defined in [python/ops/stateful\_random\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/stateful_random_ops.py).

This function creates a new Generator object (and the Variable object within), which does not work well with tf.function because (1) tf.function puts restrictions on Variable creation thus reset\_global\_generator can't be freely used inside tf.function; (2) redirecting a global variable to a new object is problematic with tf.function because the old object may be captured by a 'tf.function'ed function and still be used by it. A 'tf.function'ed function only keeps weak references to variables, so deleting a variable and then calling that function again may raise an error, as demonstrated by random\_test.py/RandomTest.testResetGlobalGeneratorBadWithDefun .

#### Args:

* **generator**: the new Generator object.