

# Package ‘tfaddons’

October 14, 2022

**Type** Package

**Title** Interface to 'TensorFlow SIG Addons'

**Version** 0.10.0

**Maintainer** Turgut Abdullayev <turqut.a.314@gmail.com>

**Description** 'TensorFlow SIG Addons' <<https://www.tensorflow.org/addons>> is a repository of community contributions that conform to well-established API patterns, but implement new functionality not available in core 'TensorFlow'. 'TensorFlow' natively supports a large number of operators, layers, metrics, losses, optimizers, and more. However, in a fast moving field like Machine Learning, there are many interesting new developments that cannot be integrated into core 'TensorFlow' (because their broad applicability is not yet clear, or it is mostly used by a smaller subset of the community).

**License** Apache License 2.0

**URL** <https://github.com/henry090/tfaddons>

**BugReports** <https://github.com/henry090/tfaddons/issues>

**SystemRequirements** TensorFlow >= 2.0 (<https://www.tensorflow.org/>)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.0

**Imports** reticulate, tensorflow, rstudioapi, keras, purrr

**Suggests** knitr, rmarkdown, testthat, dplyr

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Turgut Abdullayev [aut, cre]

**Repository** CRAN

**Date/Publication** 2020-06-02 08:50:04 UTC

**R topics documented:**

activation_gelu	5
activation_hardshrink	6
activation_lisht	7
activation_mish	7
activation_rrelu	8
activation_softshrink	9
activation_sparsemax	10
activation_tanhshrink	10
attention_bahdanau	11
attention_bahdanau_monotonic	12
attention_luong	13
attention_luong_monotonic	15
attention_monotonic	16
attention_wrapper	17
attention_wrapper_state	19
callback_average_model_checkpoint	20
callback_time_stopping	21
callback_tqdm_progress_bar	22
crf_binary_score	23
crf_decode	24
crf_decode_backward	24
crf_decode_forward	25
crf_forward	25
crf_log_likelihood	26
crf_log_norm	27
crf_multitag_sequence_score	27
crf_sequence_score	28
crf_unary_score	29
decoder	29
decoder_base	30
decoder_basic	30
decoder_basic_output	31
decoder_beam_search	31
decoder_beam_search_output	32
decoder_beam_search_state	33
decoder_final_beam_search_output	34
decode_dynamic	34
extend_with_decoupled_weight_decay	35
gather_tree	36
gather_tree_from_array	37
hardmax	38
img_adjust_hsv_in_yiq	38
img_angles_to_projective_transforms	39
img_blend	40
img_compose_transforms	40
img_connected_components	41

img_cutout . . . . .	42
img_dense_image_warp . . . . .	43
img_equalize . . . . .	44
img_euclidean_dist_transform . . . . .	45
img_flat_transforms_to_matrices . . . . .	46
img_from_4D . . . . .	46
img_get_ndims . . . . .	47
img_interpolate_bilinear . . . . .	47
img_interpolate_spline . . . . .	48
img_matrices_to_flat_transforms . . . . .	49
img_mean_filter2d . . . . .	50
img_median_filter2d . . . . .	51
img_random_cutout . . . . .	52
img_random_hsv_in_yiq . . . . .	53
img_resampler . . . . .	54
img_rotate . . . . .	55
img_sharpness . . . . .	56
img_shear_x . . . . .	56
img_shear_y . . . . .	57
img_sparse_image_warp . . . . .	57
img_to_4D . . . . .	58
img_transform . . . . .	59
img_translate . . . . .	60
img_translate_xy . . . . .	61
img_translations_to_projective_transforms . . . . .	62
img_unwrap . . . . .	62
img_wrap . . . . .	63
install_tfaddons . . . . .	63
layer_activation_gelu . . . . .	64
layer_correlation_cost . . . . .	64
layer_filter_response_normalization . . . . .	65
layer_group_normalization . . . . .	67
layer_instance_normalization . . . . .	68
layer_maxout . . . . .	70
layer_multi_head_attention . . . . .	70
layer_nas_cell . . . . .	72
layer_norm_lstm_cell . . . . .	73
layer_poincare_normalize . . . . .	75
layer_sparsemax . . . . .	76
layer_weight_normalization . . . . .	77
lookahead_mechanism . . . . .	78
loss_contrastive . . . . .	79
loss_giou . . . . .	80
loss_hamming . . . . .	81
loss_lifted_struct . . . . .	82
loss_npairs . . . . .	83
loss_npairs_multilabel . . . . .	83
loss_pinball . . . . .	84

loss_sequence . . . . .	85
loss_sigmoid_focal_crossentropy . . . . .	86
loss_sparsemax . . . . .	87
loss_triplet_hard . . . . .	88
loss_triplet_semihard . . . . .	89
metrics_f1score . . . . .	90
metric_cohen_kappa . . . . .	91
metric_fbetascore . . . . .	92
metric_hamming_distance . . . . .	93
metric_mcc . . . . .	94
metric_multilabel_confusion_matrix . . . . .	95
metric_rsquare . . . . .	96
optimizer_conditional_gradient . . . . .	97
optimizer_decay_adamw . . . . .	98
optimizer_decay_sgdw . . . . .	100
optimizer_lamb . . . . .	101
optimizer_lazy_adam . . . . .	103
optimizer_moving_average . . . . .	104
optimizer_novograd . . . . .	105
optimizer_radam . . . . .	107
optimizer_swa . . . . .	108
optimizer_yogi . . . . .	110
parse_time . . . . .	111
register_all . . . . .	112
register_custom_kernels . . . . .	113
register_keras_objects . . . . .	113
safe_cumprod . . . . .	114
sampler . . . . .	114
sampler_custom . . . . .	115
sampler_greedy_embedding . . . . .	115
sampler_inference . . . . .	116
sampler_sample_embedding . . . . .	117
sampler_scheduled_embedding_training . . . . .	118
sampler_scheduled_output_training . . . . .	118
sampler_training . . . . .	119
sample_bernoulli . . . . .	120
sample_categorical . . . . .	120
skip_gram_sample . . . . .	121
skip_gram_sample_with_text_vocab . . . . .	123
tfaddons_version . . . . .	126
tile_batch . . . . .	126
viterbi_decode . . . . .	127

---

`activation_gelu`*Gelu*

---

### Description

Gaussian Error Linear Unit.

### Usage

```
activation_gelu(x, approximate = TRUE)
```

### Arguments

`x` A 'Tensor'. Must be one of the following types: 'float16', 'float32', 'float64'.  
`approximate` bool, whether to enable approximation. Returns: A 'Tensor'. Has the same type as 'x'.

### Details

Computes gaussian error linear:  $0.5 * x * (1 + \tanh(\sqrt{2 / \pi}) * (x + 0.044715 * x^3))$  or  $x * P(X \leq x) = 0.5 * x * (1 + \operatorname{erf}(x / \sqrt{2}))$ , where  $P(X) \sim N(0, 1)$ , depending on whether approximation is enabled. See [Gaussian Error Linear Units (GELUs)](<https://arxiv.org/abs/1606.08415>) and [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](<https://arxiv.org/abs/1810.04805>).

### Value

A 'Tensor'. Has the same type as 'x'.

### Computes gaussian error linear

$0.5 * x * (1 + \tanh(\sqrt{2 / \pi}) * (x + 0.044715 * x^3))$  or  $x * P(X \leq x) = 0.5 * x * (1 + \operatorname{erf}(x / \sqrt{2}))$ , where  $P(X) \sim N(0, 1)$ , depending on whether approximation is enabled.

### Examples

```
## Not run:  
library(keras)  
library(tfaddons)  
model = keras_model_sequential() %>%  
  layer_conv_2d(filters = 10, kernel_size = c(3,3), input_shape = c(28,28,1),  
               activation = activation_gelu)  
  
## End(Not run)
```

---

activation\_hardshrink *Hardshrink*

---

## Description

Hard shrink function.

## Usage

```
activation_hardshrink(x, lower = -0.5, upper = 0.5)
```

## Arguments

x	A 'Tensor'. Must be one of the following types: 'float16', 'float32', 'float64'.
lower	'float', lower bound for setting values to zeros.
upper	'float', upper bound for setting values to zeros. Returns: A 'Tensor'. Has the same type as 'x'.

## Details

Computes hard shrink function: 'x if  $x < \text{lower}$  or  $x > \text{upper}$  else 0'.

## Value

A 'Tensor'. Has the same type as 'x'.

## Computes hard shrink function

'x if  $x < \text{lower}$  or  $x > \text{upper}$  else 0'.

## Examples

```
## Not run:
library(keras)
library(tfaddons)
model = keras_model_sequential() %>%
  layer_conv_2d(filters = 10, kernel_size = c(3,3), input_shape = c(28,28,1),
                activation = activation_hardshrink)

## End(Not run)
```

---

activation_lisht	<i>Lisht</i>
------------------	--------------

---

**Description**

LiSHT: Non-Parameteric Linearly Scaled Hyperbolic Tangent Activation Function.

**Usage**

```
activation_lisht(x)
```

**Arguments**

`x` A 'Tensor'. Must be one of the following types: 'float16', 'float32', 'float64'.

**Details**

Computes linearly scaled hyperbolic tangent (LiSHT):  $x * \tanh(x)$  See [LiSHT: Non-Parameteric Linearly Scaled Hyperbolic Tangent Activation Function for Neural Networks](<https://arxiv.org/abs/1901.05894>).

**Value**

A 'Tensor'. Has the same type as 'x'.

**Examples**

```
## Not run:
library(keras)
library(tfaddons)
model = keras_model_sequential() %>%
  layer_conv_2d(filters = 10, kernel_size = c(3,3), input_shape = c(28,28,1),
               activation = activation_lisht)

## End(Not run)
```

---

activation_mish	<i>Mish</i>
-----------------	-------------

---

**Description**

Mish: A Self Regularized Non-Monotonic Neural Activation Function.

**Usage**

```
activation_mish(x)
```

**Arguments**

x A 'Tensor'. Must be one of the following types: 'float16', 'float32', 'float64'.  
Returns: A 'Tensor'. Has the same type as 'x'.

**Details**

Computes mish activation:  $x * \tanh(\text{softplus}(x))$  See [Mish: A Self Regularized Non-Monotonic Neural Activation Function](<https://arxiv.org/abs/1908.08681>).

**Value**

A 'Tensor'. Has the same type as 'x'.

---

activation_rrelu	<i>Rrelu</i>
------------------	--------------

---

**Description**

rrelu function.

**Usage**

```
activation_rrelu(
  x,
  lower = 0.125,
  upper = 0.3333333333333333,
  training = NULL,
  seed = NULL
)
```

**Arguments**

x A 'Tensor'. Must be one of the following types: 'float16', 'float32', 'float64'.  
lower 'float', lower bound for random alpha.  
upper 'float', upper bound for random alpha.  
training 'bool', indicating whether the 'call' is meant for training or inference.  
seed 'int', this sets the operation-level seed. Returns:

**Details**

Computes rrelu function: 'x if  $x > 0$  else  $\text{random}(\text{lower}, \text{upper}) * x$ ' or 'x if  $x > 0$  else  $x * (\text{lower} + \text{upper}) / 2$ ' depending on whether training is enabled. See [Empirical Evaluation of Rectified Activations in Convolutional Network](<https://arxiv.org/abs/1505.00853>).

**Value**

A 'Tensor'. Has the same type as 'x'.



**Computes rrelu function**

'x if  $x > 0$  else  $\text{random}(\text{lower}, \text{upper}) * x$ ' or 'x if  $x > 0$  else  $x * (\text{lower} + \text{upper}) / 2$ ' depending on whether training is enabled.

---

activation\_softshrink *Softshrink*

---

**Description**

Soft shrink function.

**Usage**

```
activation_softshrink(x, lower = -0.5, upper = 0.5)
```

**Arguments**

x	A 'Tensor'. Must be one of the following types: 'float16', 'float32', 'float64'.
lower	'float', lower bound for setting values to zeros.
upper	'float', upper bound for setting values to zeros. Returns: A 'Tensor'. Has the same type as 'x'.

**Details**

Computes soft shrink function: 'x - lower if  $x < \text{lower}$ , x - upper if  $x > \text{upper}$  else 0'.

**Value**

A 'Tensor'. Has the same type as 'x'.

**Computes soft shrink function**

'x - lower if  $x < \text{lower}$ , x - upper if  $x > \text{upper}$  else 0'.

---

activation\_sparsemax *Sparsemax*

---

### Description

Sparsemax activation function [1].

### Usage

```
activation_sparsemax(logits, axis = -1L)
```

### Arguments

logits	Input tensor.
axis	Integer, axis along which the sparsemax operation is applied.

### Details

For each batch ‘i’ and class ‘j’ we have  $\text{sparsemax}[i, j] = \max(\text{logits}[i, j] - \tau(\text{logits}[i, :]), 0)$   
 [1]: <https://arxiv.org/abs/1602.02068>

### Value

Tensor, output of sparsemax transformation. Has the same type and shape as ‘logits’. Raises: ValueError: In case ‘dim(logits) == 1’.

### Raises

ValueError: In case ‘dim(logits) == 1’.

---

activation\_tanhshrink *Tanhshrink*

---

### Description

Applies the element-wise function:  $x - \tanh(x)$

### Usage

```
activation_tanhshrink(x)
```

### Arguments

x	A ‘Tensor’. Must be one of the following types: ‘float16’, ‘float32’, ‘float64’.
---	--

### Value

A ‘Tensor’. Has the same type as ‘features’.

---

attention\_bahdanau     *Bahdanau Attention*

---

## Description

Implements Bahdanau-style (additive) attention

## Usage

```
attention_bahdanau(
    object,
    units,
    memory = NULL,
    memory_sequence_length = NULL,
    normalize = FALSE,
    probability_fn = "softmax",
    kernel_initializer = "glorot_uniform",
    dtype = NULL,
    name = "BahdanauAttention",
    ...
)
```

## Arguments

object	Model or layer object
units	The depth of the query mechanism.
memory	The memory to query; usually the output of an RNN encoder. This tensor should be shaped [batch_size, max_time, ...].
memory_sequence_length	(optional): Sequence lengths for the batch entries in memory. If provided, the memory tensor rows are masked with zeros for values past the respective sequence lengths.
normalize	boolean. Whether to normalize the energy term.
probability_fn	(optional) string, the name of function to convert the attention score to probabilities. The default is softmax which is tf.nn.softmax. Other options is hardmax, which is hardmax() within this module. Any other value will result into validation error. Default to use softmax.
kernel_initializer	(optional), the name of the initializer for the attention kernel.
dtype	The data type for the query and memory layers of the attention mechanism.
name	Name to use when creating ops.
...	A list that contains other common arguments for layer creation.

**Details**

This attention has two forms. The first is Bahdanau attention, as described in: Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." ICLR 2015. <https://arxiv.org/abs/1409.0473> The second is the normalized form. This form is inspired by the weight normalization article: Tim Salimans, Diederik P. Kingma. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks." <https://arxiv.org/abs/1602.07868> To enable the second form, construct the object with parameter 'normalize=TRUE'.

**Value**

None

---

attention\_bahdanau\_monotonic  
*Bahdanau Monotonic Attention*

---

**Description**

Monotonic attention mechanism with Bahadanau-style energy function.

**Usage**

```
attention_bahdanau_monotonic(
  object,
  units,
  memory = NULL,
  memory_sequence_length = NULL,
  normalize = FALSE,
  sigmoid_noise = 0,
  sigmoid_noise_seed = NULL,
  score_bias_init = 0,
  mode = "parallel",
  kernel_initializer = "glorot_uniform",
  dtype = NULL,
  name = "BahdanauMonotonicAttention",
  ...
)
```

**Arguments**

object	Model or layer object
units	The depth of the query mechanism.
memory	The memory to query; usually the output of an RNN encoder. This tensor should be shaped [batch_size, max_time, ...].

memory_sequence_length	(optional): Sequence lengths for the batch entries in memory. If provided, the memory tensor rows are masked with zeros for values past the respective sequence lengths.
normalize	Python boolean. Whether to normalize the energy term.
sigmoid_noise	Standard deviation of pre-sigmoid noise. See the docstring for ‘_monotonic_probability_fn’ for more information.
sigmoid_noise_seed	(optional) Random seed for pre-sigmoid noise.
score_bias_init	Initial value for score bias scalar. It’s recommended to initialize this to a negative value when the length of the memory is large.
mode	How to compute the attention distribution. Must be one of ‘recursive’, ‘parallel’, or ‘hard’. See the docstring for <code>tfa.seq2seq.monotonic_attention</code> for more information.
kernel_initializer	(optional), the name of the initializer for the attention kernel.
dtype	The data type for the query and memory layers of the attention mechanism.
name	Name to use when creating ops.
...	A list that contains other common arguments for layer creation.

### Details

This type of attention enforces a monotonic constraint on the attention distributions; that is once the model attends to a given point in the memory it can’t attend to any prior points at subsequence output timesteps. It achieves this by using the `_monotonic_probability_fn` instead of softmax to construct its attention distributions. Since the attention scores are passed through a sigmoid, a learnable scalar bias parameter is applied after the score function and before the sigmoid. Otherwise, it is equivalent to BahdanauAttention. This approach is proposed in

Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, Douglas Eck, "Online and Linear-Time Attention by Enforcing Monotonic Alignments." ICML 2017. <https://arxiv.org/abs/1704.00784>

### Value

None

---

attention_luong	<i>Implements Luong-style (multiplicative) attention scoring.</i>
-----------------	---

---

### Description

Implements Luong-style (multiplicative) attention scoring.

**Usage**

```
attention_luong(
    object,
    units,
    memory = NULL,
    memory_sequence_length = NULL,
    scale = FALSE,
    probability_fn = "softmax",
    dtype = NULL,
    name = "LuongAttention",
    ...
)
```

**Arguments**

<code>object</code>	Model or layer object
<code>units</code>	The depth of the attention mechanism.
<code>memory</code>	The memory to query; usually the output of an RNN encoder. This tensor should be shaped <code>[batch_size, max_time, ...]</code> .
<code>memory_sequence_length</code>	(optional): Sequence lengths for the batch entries in memory. If provided, the memory tensor rows are masked with zeros for values past the respective sequence lengths.
<code>scale</code>	boolean. Whether to scale the energy term.
<code>probability_fn</code>	(optional) string, the name of function to convert the attention score to probabilities. The default is <code>softmax</code> which is <code>tf.nn.softmax</code> . Other options is <code>hardmax</code> , which is <code>hardmax()</code> within this module. Any other value will result into validation error. Default to use <code>softmax</code> .
<code>dtype</code>	The data type for the memory layer of the attention mechanism.
<code>name</code>	Name to use when creating ops.
<code>...</code>	A list that contains other common arguments for layer creation.

**Details**

This attention has two forms. The first is standard Luong attention, as described in: Minh-Thang Luong, Hieu Pham, Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. EMNLP 2015. The second is the scaled form inspired partly by the normalized form of Bahdanau attention. To enable the second form, construct the object with parameter `'scale=TRUE'`.

**Value**

None

---

 attention\_luong\_monotonic

*Monotonic attention mechanism with Luong-style energy function.*


---

## Description

Monotonic attention mechanism with Luong-style energy function.

## Usage

```
attention_luong_monotonic(
    object,
    units,
    memory = NULL,
    memory_sequence_length = NULL,
    scale = FALSE,
    sigmoid_noise = 0,
    sigmoid_noise_seed = NULL,
    score_bias_init = 0,
    mode = "parallel",
    dtype = NULL,
    name = "LuongMonotonicAttention",
    ...
)
```

## Arguments

object	Model or layer object
units	The depth of the query mechanism.
memory	The memory to query; usually the output of an RNN encoder. This tensor should be shaped [batch_size, max_time, ...].
memory_sequence_length	(optional): Sequence lengths for the batch entries in memory. If provided, the memory tensor rows are masked with zeros for values past the respective sequence lengths.
scale	boolean. Whether to scale the energy term.
sigmoid_noise	Standard deviation of pre-sigmoid noise. See the docstring for ‘_monotonic_probability_fn’ for more information.
sigmoid_noise_seed	(optional) Random seed for pre-sigmoid noise.
score_bias_init	Initial value for score bias scalar. It’s recommended to initialize this to a negative value when the length of the memory is large.
mode	How to compute the attention distribution. Must be one of ‘recursive’, ‘parallel’, or ‘hard’. See the docstring for <code>tfa.seq2seq.monotonic_attention</code> for more information.

dtype	The data type for the query and memory layers of the attention mechanism.
name	Name to use when creating ops.
...	A list that contains other common arguments for layer creation.

### Details

This type of attention enforces a monotonic constraint on the attention distributions; that is once the model attends to a given point in the memory it can't attend to any prior points at subsequent output timesteps. It achieves this by using the `_monotonic_probability_fn` instead of `softmax` to construct its attention distributions. Otherwise, it is equivalent to `LuongAttention`. This approach is proposed in [Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, Douglas Eck, "Online and Linear-Time Attention by Enforcing Monotonic Alignments." ICML 2017.](<https://arxiv.org/abs/1704.00784>)

### Value

None

---

attention\_monotonic     *Monotonic attention*

---

### Description

Compute monotonic attention distribution from choosing probabilities.

### Usage

```
attention_monotonic(p_choose_i, previous_attention, mode)
```

### Arguments

p_choose_i	Probability of choosing input sequence/memory element i. Should be of shape (batch_size, input_sequence_length), and should all be in the range [0, 1].
previous_attention	The attention distribution from the previous output timestep. Should be of shape (batch_size, input_sequence_length). For the first output timestep, previous_attention[n] should be [1, 0, 0, ..., 0] for all n in [0, ... batch_size - 1].
mode	How to compute the attention distribution. Must be one of 'recursive', 'parallel', or 'hard'. 'recursive' uses <code>tf\$scan</code> to recursively compute the distribution. This is slowest but is exact, general, and does not suffer from numerical instabilities. 'parallel' uses parallelized cumulative-sum and cumulative-product operations to compute a closed-form solution to the recurrence relation defining the attention distribution. This makes it more efficient than 'recursive', but it requires numerical checks which make the distribution non-exact. This can be a problem in particular when <code>input_sequence_length</code> is long and/or <code>p_choose_i</code> has entries very close to 0 or 1. * 'hard' requires that the probabilities in <code>p_choose_i</code> are all either 0 or 1, and subsequently uses a more efficient and exact solution.



### Details

Monotonic attention implies that the input sequence is processed in an explicitly left-to-right manner when generating the output sequence. In addition, once an input sequence element is attended to at a given output timestep, elements occurring before it cannot be attended to at subsequent output timesteps. This function generates attention distributions according to these assumptions. For more information, see ‘Online and Linear-Time Attention by Enforcing Monotonic Alignments‘.

### Value

A tensor of shape (batch\_size, input\_sequence\_length) representing the attention distributions for each sequence in the batch.

### Raises

ValueError: mode is not one of ‘recursive’, ‘parallel’, ‘hard’.

---

attention_wrapper	<i>Attention Wrapper</i>
-------------------	--------------------------

---

### Description

Attention Wrapper

### Usage

```
attention_wrapper(  
    object,  
    cell,  
    attention_mechanism,  
    attention_layer_size = NULL,  
    alignment_history = FALSE,  
    cell_input_fn = NULL,  
    output_attention = TRUE,  
    initial_cell_state = NULL,  
    name = NULL,  
    attention_layer = NULL,  
    attention_fn = NULL,  
    ...  
)
```

### Arguments

object	Model or layer object
cell	An instance of RNNCell.
attention_mechanism	A list of AttentionMechanism instances or a single instance.

<code>attention_layer_size</code>	A list of Python integers or a single Python integer, the depth of the attention (output) layer(s). If 'NULL' (default), use the context as attention at each time step. Otherwise, feed the context and cell output into the attention layer to generate attention at each time step. If <code>attention_mechanism</code> is a list, <code>attention_layer_size</code> must be a list of the same length. If <code>attention_layer</code> is set, this must be 'NULL'. If <code>attention_fn</code> is set, it must guaranteed that the outputs of 'attention_fn' also meet the above requirements.
<code>alignment_history</code>	Python boolean, whether to store alignment history from all time steps in the final output state (currently stored as a time major TensorArray on which you must call <code>stack()</code> ).
<code>cell_input_fn</code>	(optional) A callable. The default is: <code>lambda inputs, attention: tf\$concat(list(inputs, attention), -1)</code> .
<code>output_attention</code>	Python bool. If True (default), the output at each time step is the attention value. This is the behavior of Luong-style attention mechanisms. If False, the output at each time step is the output of cell. This is the behavior of Bhadanau-style attention mechanisms. In both cases, the attention tensor is propagated to the next time step via the state and is used there. This flag only controls whether the attention mechanism is propagated up to the next cell in an RNN stack or to the top RNN output.
<code>initial_cell_state</code>	The initial state value to use for the cell when the user calls <code>get_initial_state()</code> . Note that if this value is provided now, and the user uses a <code>batch_size</code> argument of <code>get_initial_state</code> which does not match the batch size of <code>initial_cell_state</code> , proper behavior is not guaranteed.
<code>name</code>	Name to use when creating ops.
<code>attention_layer</code>	A list of <code>tf\$keras\$layers\$Layer</code> instances or a single <code>tf\$keras\$layers\$Layer</code> instance taking the context and cell output as inputs to generate attention at each time step. If 'NULL' (default), use the context as attention at each time step. If <code>attention_mechanism</code> is a list, <code>attention_layer</code> must be a list of the same length. If <code>attention_layers_size</code> is set, this must be 'NULL'.
<code>attention_fn</code>	An optional callable function that allows users to provide their own customized attention function, which takes input ( <code>attention_mechanism</code> , <code>cell_output</code> , <code>attention_state</code> , <code>attention_layer</code> ) and outputs ( <code>attention</code> , <code>alignments</code> , <code>next_attention_state</code> ). If provided, the <code>attention_layer_size</code> should be the size of the outputs of <code>attention_fn</code> .
<code>...</code>	Other keyword arguments to pass

**Value**

None

**Note**

If you are using the 'decoder\_beam\_search' with a cell wrapped in 'AttentionWrapper', then you must ensure that: - The encoder output has been tiled to 'beam\_width' via 'tile\_batch' (NOT 'tf\$tile'). - The 'batch\_size' argument passed to the 'get\_initial\_state' method of this wrapper is equal to 'true\_batch\_size \* beam\_width'. - The initial state created with 'get\_initial\_state' above contains a 'cell\_state' value containing properly tiled final state from the encoder.

---

attention\_wrapper\_state

*Attention Wrapper State*

---

**Description**

'namedlist' storing the state of a 'attention\_wrapper'.

**Usage**

```
attention_wrapper_state(
    object,
    cell_state,
    attention,
    alignments,
    alignment_history,
    attention_state
)
```

**Arguments**

object	Model or layer object
cell_state	The state of the wrapped RNNCell at the previous time step.
attention	The attention emitted at the previous time step.
alignments	A single or tuple of Tensor(s) containing the alignments emitted at the previous time step for each attention mechanism.
alignment_history	(if enabled) a single or tuple of TensorArray(s) containing alignment matrices from all time steps for each attention mechanism. Call stack() on each to convert to a Tensor.
attention_state	A single or tuple of nested objects containing attention mechanism state for each attention mechanism. The objects may contain Tensors or TensorArrays.

**Value**

None

---

```
callback_average_model_checkpoint
    Average Model Checkpoint
```

---

### Description

Save the model after every epoch.

### Usage

```
callback_average_model_checkpoint(
    filepath,
    update_weights,
    monitor = "val_loss",
    verbose = 0,
    save_best_only = FALSE,
    save_weights_only = FALSE,
    mode = "auto",
    save_freq = "epoch",
    ...
)
```

### Arguments

filepath	string, path to save the model file.
update_weights	bool, whether to update weights or not
monitor	quantity to monitor.
verbose	verbosity mode, 0 or 1.
save_best_only	if 'save_best_only=TRUE', the latest best model according to the quantity monitored will not be overwritten. If 'filepath' doesn't contain formatting options like 'epoch' then 'filepath' will be overwritten by each new better model.
save_weights_only	if TRUE, then only the model's weights will be saved ('model\$save_weights(filepath)'), else the full model is saved ('model\$save(filepath)').
mode	one of auto, min, max. If 'save_best_only=TRUE', the decision to overwrite the current save file is made based on either the maximization or the minimization of the monitored quantity. For 'val_acc', this should be 'max', for 'val_loss' this should be 'min', etc. In 'auto' mode, the direction is automatically inferred from the name of the monitored quantity.
save_freq	'epoch' or integer. When using 'epoch', the callback saves the model after each epoch. When using integer, the callback saves the model at end of a batch at which this many samples have been seen since last saving. Note that if the saving isn't aligned to epochs, the monitored metric may potentially be less reliable (it could reflect as little as 1 batch, since the metrics get reset every epoch). Defaults to 'epoch'.
...	Additional arguments for backwards compatibility. Possible key is 'period'.

**Details**

The callback that should be used with optimizers that extend AverageWrapper, i.e., MovingAverage and StochasticAverage optimizers. It saves and, optionally, assigns the averaged weights.

**Value**

None

**For example**

if 'filepath' is 'weights.epoch:02d-val\_loss:.2f.hdf5', then the model checkpoints will be saved with the epoch number and the validation loss in the filename.

---

callback\_time\_stopping  
*Time Stopping*

---

**Description**

Time Stopping

**Usage**

```
callback_time_stopping(seconds = 86400, verbose = 0)
```

**Arguments**

seconds            maximum amount of time before stopping. Defaults to 86400 (1 day).  
verbose            verbosity mode. Defaults to 0.

**Details**

Stop training when a specified amount of time has passed.

**Value**

None

**Examples**

```
## Not run:  
model %>% fit(  
  x_train, y_train,  
  batch_size = 128,  
  epochs = 4,  
  validation_split = 0.2,  
  verbose = 0,
```

```

callbacks = callback_time_stopping(seconds = 6, verbose = 1)
)

## End(Not run)

```

---

```

callback_tqdm_progress_bar
      TQDM Progress Bar

```

---

### Description

TQDM Progress Bar

### Usage

```

callback_tqdm_progress_bar(
  metrics_separator = " - ",
  overall_bar_format = NULL,
  epoch_bar_format = "{n_fmt}/{total_fmt}{bar} ETA: {remaining}s - {desc}",
  update_per_second = 10,
  leave_epoch_progress = TRUE,
  leave_overall_progress = TRUE,
  show_epoch_progress = TRUE,
  show_overall_progress = TRUE
)

```

### Arguments

`metrics_separator`  
(string) Custom separator between metrics. Defaults to ' - '

`overall_bar_format`  
(string format) Custom bar format for overall (outer) progress bar, see <https://github.com/tqdm/tqdm#parameters> for more detail. By default: 'l\_barbar n\_fmt/total\_fmt ETA: remainings, rate\_fmtpostfix'

`epoch_bar_format`  
(string format) Custom bar format for epoch (inner) progress bar, see <https://github.com/tqdm/tqdm#parameters> for more detail.

`update_per_second`  
(int) Maximum number of updates in the epochs bar per second, this is to prevent small batches from slowing down training. Defaults to 10.

`leave_epoch_progress`  
(bool) TRUE to leave epoch progress bars

`leave_overall_progress`  
(bool) TRUE to leave overall progress bar

```

show_epoch_progress
    (bool) FALSE to hide epoch progress bars
show_overall_progress
    (bool) FALSE to hide overall progress bar

```

**Details**

TQDM Progress Bar for Tensorflow Keras.

**Value**

None

**Examples**

```

## Not run:
model %>% fit(
  x_train, y_train,
  batch_size = 128,
  epochs = 4,
  validation_split = 0.2,
  verbose = 0,
  callbacks = callback_tqdm_progress_bar()
)

## End(Not run)

```

---

crf_binary_score	<i>CRF binary score</i>
------------------	-------------------------

---

**Description**

Computes the binary scores of tag sequences.

**Usage**

```
crf_binary_score(tag_indices, sequence_lengths, transition_params)
```

**Arguments**

```

tag_indices      A [batch_size, max_seq_len] matrix of tag indices.
sequence_lengths A [batch_size] vector of true sequence lengths.
transition_params A [num_tags, num_tags] matrix of binary potentials.

```

**Value**

binary\_scores: A [batch\_size] vector of binary scores.

---

crf\_decode

*CRF decode*

---

**Description**

Decode the highest scoring sequence of tags.

**Usage**

```
crf_decode(potentials, transition_params, sequence_length)
```

**Arguments**

potentials A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials.

transition\_params

A [num\_tags, num\_tags] matrix of binary potentials.

sequence\_length

A [batch\_size] vector of true sequence lengths.

**Value**

decode\_tags: A [batch\_size, max\_seq\_len] matrix, with dtype 'tf.int32'. Contains the highest scoring tag indices. best\_score: A [batch\_size] vector, containing the score of 'decode\_tags'.

---

crf\_decode\_backward

*CRF decode backward*

---

**Description**

Computes backward decoding in a linear-chain CRF.

**Usage**

```
crf_decode_backward(inputs, state)
```

**Arguments**

inputs A [batch\_size, num\_tags] matrix of backpointer of next step (in time order).

state

A [batch\_size, 1] matrix of tag index of next step.

**Value**

new\_tags: A [batch\_size, num\_tags] tensor containing the new tag indices.



---

crf_decode_forward	<i>CRF decode forward</i>
--------------------	---------------------------

---

**Description**

Computes forward decoding in a linear-chain CRF.

**Usage**

```
crf_decode_forward(inputs, state, transition_params, sequence_lengths)
```

**Arguments**

inputs	A [batch_size, num_tags] matrix of unary potentials.
state	A [batch_size, num_tags] matrix containing the previous step's score values.
transition_params	A [num_tags, num_tags] matrix of binary potentials.
sequence_lengths	A [batch_size] vector of true sequence lengths.

**Value**

backpointers: A [batch\_size, num\_tags] matrix of backpointers. new\_state: A [batch\_size, num\_tags] matrix of new score values.

---

crf_forward	<i>CRF forward</i>
-------------	--------------------

---

**Description**

Computes the alpha values in a linear-chain CRF.

**Usage**

```
crf_forward(inputs, state, transition_params, sequence_lengths)
```

**Arguments**

inputs	A [batch_size, num_tags] matrix of unary potentials.
state	A [batch_size, num_tags] matrix containing the previous alpha values.
transition_params	A [num_tags, num_tags] matrix of binary potentials. This matrix is expanded into a [1, num_tags, num_tags] in preparation for the broadcast summation occurring within the cell.
sequence_lengths	A [batch_size] vector of true sequence lengths.

**Details**

See <http://www.cs.columbia.edu/~mcollins/fb.pdf> for reference.

**Value**

new\_alphas: A [batch\_size, num\_tags] matrix containing the new alpha values.

---

crf_log_likelihood	<i>CRF log likelihood</i>
--------------------	---------------------------

---

**Description**

Computes the log-likelihood of tag sequences in a CRF.

**Usage**

```
crf_log_likelihood(
  inputs,
  tag_indices,
  sequence_lengths,
  transition_params = NULL
)
```

**Arguments**

inputs	A [batch_size, max_seq_len, num_tags] tensor of unary potentials to use as input to the CRF layer.
tag_indices	A [batch_size, max_seq_len] matrix of tag indices for which we compute the log-likelihood.
sequence_lengths	A [batch_size] vector of true sequence lengths.
transition_params	A [num_tags, num_tags] transition matrix, if available.

**Value**

log\_likelihood: A [batch\_size] Tensor containing the log-likelihood of each example, given the sequence of tag indices. transition\_params: A [num\_tags, num\_tags] transition matrix. This is either provided by the caller or created in this function.

---

crf_log_norm	<i>CRF log norm</i>
--------------	---------------------

---

**Description**

Computes the normalization for a CRF.

**Usage**

```
crf_log_norm(inputs, sequence_lengths, transition_params)
```

**Arguments**

`inputs` A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials to use as input to the CRF layer.

`sequence_lengths` A [batch\_size] vector of true sequence lengths.

`transition_params` A [num\_tags, num\_tags] transition matrix.

**Value**

`log_norm`: A [batch\_size] vector of normalizers for a CRF.

---

crf_multitag_sequence_score	<i>CRF multitag sequence score</i>
-----------------------------	------------------------------------

---

**Description**

Computes the unnormalized score of all tag sequences matching

**Usage**

```
crf_multitag_sequence_score(  
  inputs,  
  tag_bitmap,  
  sequence_lengths,  
  transition_params  
)
```

**Arguments**

inputs	A [batch_size, max_seq_len, num_tags] tensor of unary potentials to use as input to the CRF layer.
tag_bitmap	A [batch_size, max_seq_len, num_tags] boolean tensor representing all active tags at each index for which to calculate the unnormalized score.
sequence_lengths	A [batch_size] vector of true sequence lengths.
transition_params	A [num_tags, num_tags] transition matrix.

**Details**

tag\_bitmap. tag\_bitmap enables more than one tag to be considered correct at each time step. This is useful when an observed output at a given time step is consistent with more than one tag, and thus the log likelihood of that observation must take into account all possible consistent tags. Using one-hot vectors in tag\_bitmap gives results identical to crf\_sequence\_score.

**Value**

sequence\_scores: A [batch\_size] vector of unnormalized sequence scores.

---

crf_sequence_score	<i>CRF sequence score</i>
--------------------	---------------------------

---

**Description**

Computes the unnormalized score for a tag sequence.

**Usage**

```
crf_sequence_score(inputs, tag_indices, sequence_lengths, transition_params)
```

**Arguments**

inputs	A [batch_size, max_seq_len, num_tags] tensor of unary potentials to use as input to the CRF layer.
tag_indices	A [batch_size, max_seq_len] matrix of tag indices for which we compute the unnormalized score.
sequence_lengths	A [batch_size] vector of true sequence lengths.
transition_params	A [num_tags, num_tags] transition matrix. Returns:

**Value**

sequence\_scores: A [batch\_size] vector of unnormalized sequence scores.

---

crf_unary_score	<i>CRF unary score</i>
-----------------	------------------------

---

**Description**

Computes the unary scores of tag sequences.

**Usage**

```
crf_unary_score(tag_indices, sequence_lengths, inputs)
```

**Arguments**

tag_indices	A [batch_size, max_seq_len] matrix of tag indices.
sequence_lengths	A [batch_size] vector of true sequence lengths.
inputs	A [batch_size, max_seq_len, num_tags] tensor of unary potentials.

**Value**

unary\_scores: A [batch\_size] vector of unary scores.

---

decoder	<i>An RNN Decoder abstract interface object.</i>
---------	--

---

**Description**

An RNN Decoder abstract interface object.

**Usage**

```
decoder(...)
```

**Arguments**

...	arguments to pass
-----	-------------------

**Details**

- inputs: (structure of) tensors and TensorArrays that is passed as input to the RNNCell composing the decoder, at each time step.
- state: (structure of) tensors and TensorArrays that is passed to the RNNCell instance as the state.
- finished: boolean tensor telling whether each sequence in the batch is finished.
- training: boolean whether it should behave in training mode or in inference mode.
- outputs: Instance of BasicDecoderOutput. Result of the decoding, at each time step.

**Value**

None

---

decoder_base	<i>Base Decoder</i>
--------------	---------------------

---

**Description**

An RNN Decoder that is based on a Keras layer.

**Usage**

```
decoder_base(object, cell, sampler, output_layer = NULL, ...)
```

**Arguments**

object	Model or layer object
cell	An RNNCell instance.
sampler	A Sampler instance.
output_layer	(Optional) An instance of <code>tf.layers.Layer</code> , i.e., <code>tf.layers.Dense</code> . Optional layer to apply to the RNN output prior to storing the result or sampling.
...	Other keyword arguments for layer creation.

**Value**

None

---

decoder_basic	<i>Basic Decoder</i>
---------------	----------------------

---

**Description**

Basic Decoder

**Usage**

```
decoder_basic(object, cell, sampler, output_layer = NULL, ...)
```

**Arguments**

object	Model or layer object
cell	An RNNCell instance.
sampler	A Sampler instance.
output_layer	(Optional) An instance of <code>tf.layers.Layer</code> , i.e., <code>tf.layers.Dense</code> . Optional layer to apply to the RNN output prior to storing the result or sampling.
...	Other keyword arguments for layer creation.

**Value**

None

---

decoder\_basic\_output    *Basic decoder output*

---

**Description**

Basic decoder output

**Usage**

```
decoder_basic_output(rnn_output, sample_id)
```

**Arguments**

rnn_output	the output of RNN cell
sample_id	the 'id' of the sample

**Value**

None

---

decoder\_beam\_search    *BeamSearch sampling decoder*

---

**Description**

BeamSearch sampling decoder

**Usage**

```
decoder_beam_search(  
    object,  
    cell,  
    beam_width,  
    embedding_fn = NULL,  
    output_layer = NULL,  
    length_penalty_weight = 0,  
    coverage_penalty_weight = 0,  
    reorder_tensor_arrays = TRUE,  
    ...  
)
```

**Arguments**

object	Model or layer object
cell	An RNNCell instance.
beam_width	integer, the number of beams.
embedding_fn	A callable that takes a vector tensor of ids (argmax ids).
output_layer	(Optional) An instance of <code>tf.keras.layers.Layer</code> , i.e., <code>tf.keras.layers.Dense</code> . Optional layer to apply to the RNN output prior to storing the result or sampling.
length_penalty_weight	Float weight to penalize length. Disabled with 0.0.
coverage_penalty_weight	Float weight to penalize the coverage of source sentence. Disabled with 0.0.
reorder_tensor_arrays	If 'TRUE', TensorArrays' elements within the cell state will be reordered according to the beam search path. If the TensorArray can be reordered, the stacked form will be returned. Otherwise, the TensorArray will be returned as is. Set this flag to False if the cell state contains TensorArrays that are not amenable to reordering.
...	A list, other keyword arguments for initialization.

**Value**

None

**Note**

If you are using the 'BeamSearchDecoder' with a cell wrapped in 'AttentionWrapper', then you must ensure that: - The encoder output has been tiled to 'beam\_width' via 'tile\_batch()' (NOT 'tf.tile'). - The 'batch\_size' argument passed to the 'get\_initial\_state' method of this wrapper is equal to 'true\_batch\_size \* beam\_width'. - The initial state created with 'get\_initial\_state' above contains a 'cell\_state' value containing properly tiled final state from the encoder.

---

decoder\_beam\_search\_output

*Beam Search Decoder Output*


---

**Description**

Beam Search Decoder Output

**Usage**

```
decoder_beam_search_output(scores, predicted_ids, parent_ids)
```



**Arguments**

scores	calculate the scores for each beam
predicted_ids	The final prediction. A tensor of shape '[batch_size, T, beam_width]' (or '[T, batch_size, beam_width]' if 'output_time_major' is 'TRUE'). Beams are ordered from best to worst.
parent_ids	The parent ids of shape '[max_time, batch_size, beam_width]'.

**Value**

None

---

decoder\_beam\_search\_state  
*Beam Search Decoder State*

---

**Description**

Beam Search Decoder State

**Usage**

```
decoder_beam_search_state(
    cell_state,
    log_probs,
    finished,
    lengths,
    accumulated_attention_probs
)
```

**Arguments**

cell_state	cell_state
log_probs	log_probs
finished	finished
lengths	lengths
accumulated_attention_probs	accumulated_attention_probs

**Value**

None

---

decoder\_final\_beam\_search\_output  
*Final Beam Search Decoder Output*

---

**Description**

Final outputs returned by the beam search after all decoding is finished.

**Usage**

```
decoder_final_beam_search_output(predicted_ids, beam_search_decoder_output)
```

**Arguments**

`predicted_ids` The final prediction. A tensor of shape `'[batch_size, T, beam_width]'` (or `'[T, batch_size, beam_width]'` if `'output_time_major'` is `TRUE`). Beams are ordered from best to worst.

`beam_search_decoder_output`  
 An instance of `'BeamSearchDecoderOutput'` that describes the state of the beam search.

**Value**

None

---

decode\_dynamic      *Dynamic decode*

---

**Description**

Perform dynamic decoding with `'decoder'`.

**Usage**

```
decode_dynamic(  
  decoder,  
  output_time_major = FALSE,  
  impute_finished = FALSE,  
  maximum_iterations = NULL,  
  parallel_iterations = 32L,  
  swap_memory = FALSE,  
  training = NULL,  
  scope = NULL,  
  ...  
)
```

**Arguments**

decoder	A 'Decoder' instance.
output_time_major	boolean. Default: 'FALSE' (batch major). If 'TRUE', outputs are returned as time major tensors (this mode is faster). Otherwise, outputs are returned as batch major tensors (this adds extra time to the computation).
impute_finished	boolean. If 'TRUE', then states for batch entries which are marked as finished get copied through and the corresponding outputs get zeroed out. This causes some slowdown at each time step, but ensures that the final state and outputs have the correct values and that backprop ignores time steps that were marked as finished.
maximum_iterations	'int32' scalar, maximum allowed number of decoding steps. Default is 'NULL' (decode until the decoder is fully done).
parallel_iterations	Argument passed to 'tf\$while_loop'.
swap_memory	Argument passed to 'tf\$while_loop'.
training	boolean. Indicates whether the layer should behave in training mode or in inference mode. Only relevant when 'dropout' or 'recurrent_dropout' is used.
scope	Optional variable scope to use.
...	A list, other keyword arguments for dynamic_decode. It might contain arguments for 'BaseDecoder' to initialize, which takes all tensor inputs during 'call()'.

**Details**

Calls 'initialize()' once and 'step()' repeatedly on the Decoder object.

**Value**

'(final\_outputs, final\_state, final\_sequence\_lengths)'.

**Raises**

TypeError: if 'decoder' is not an instance of 'Decoder'. ValueError: if 'maximum\_iterations' is provided but is not a scalar.

---

extend\_with\_decoupled\_weight\_decay

*Factory function returning an optimizer class with decoupled weight decay*

---

**Description**

Factory function returning an optimizer class with decoupled weight decay

**Usage**

```
extend_with_decoupled_weight_decay(base_optimizer)
```

**Arguments**

`base_optimizer` An optimizer class that inherits from `tf$optimizers$Optimizer`.

**Details**

The API of the new optimizer class slightly differs from the API of the base optimizer:

- The first argument to the constructor is the weight decay rate.
- `minimize` and `apply_gradients` accept the optional keyword argument `decay_var_list`, which specifies the variables that should be decayed. If NULLs, all variables that are optimized are decayed.

**Value**

A new optimizer class that inherits from `DecoupledWeightDecayExtension` and `base_optimizer`.

**Note**

Note: this extension decays weights BEFORE applying the update based on the gradient, i.e. this extension only has the desired behaviour for optimizers which do not depend on the value of 'var' in the update step! Note: when applying a decay to the learning rate, be sure to manually apply the decay to the 'weight\_decay' as well.

**Examples**

```
## Not run:

### MyAdamW is a new class
MyAdamW = extend_with_decoupled_weight_decay(tf$keras$optimizers$Adam)
### Create a MyAdamW object
optimizer = MyAdamW(weight_decay = 0.001, learning_rate = 0.001)
#### update var1, var2 but only decay var1
optimizer$minimize(loss, var_list = list(var1, var2), decay_variables = list(var1))

## End(Not run)
```

---

gather\_tree

*Gather tree*


---

**Description**

Gather tree

**Usage**

```
gather_tree(step_ids, parent_ids, max_sequence_lengths, end_token)
```

**Arguments**

`step_ids` requires the step id

`parent_ids` The parent ids of shape `'[max_time, batch_size, beam_width]'`.

`max_sequence_lengths` get max\_sequence\_length across all beams for each batch.

`end_token` `'int32'` scalar, the token that marks end of decoding.

**Value**

None

---

gather\_tree\_from\_array

*Gather tree from array*

---

**Description**

Calculates the full beams for `'TensorArray'`s.

**Usage**

```
gather_tree_from_array(t, parent_ids, sequence_length)
```

**Arguments**

`t` A stacked `'TensorArray'` of size `'max_time'` that contains `'Tensor'`s of shape `'[batch_size, beam_width, s]'` or `'[batch_size * beam_width, s]'` where `'s'` is the depth shape.

`parent_ids` The parent ids of shape `'[max_time, batch_size, beam_width]'`.

`sequence_length` The sequence length of shape `'[batch_size, beam_width]'`.

**Value**

A `'Tensor'` which is a stacked `'TensorArray'` of the same size and type as `'t'` and where beams are sorted in each `'Tensor'` according to `'parent_ids'`.

hardmax *Hardmax*

---

### Description

Returns batched one-hot vectors.

### Usage

```
hardmax(logits, name = NULL)
```

### Arguments

logits	A batch tensor of logit values.
name	Name to use when creating ops.

### Details

The depth index containing the '1' is that of the maximum logit value.

### Value

A batched one-hot tensor.

---

img\_adjust\_hsv\_in\_yiq *Adjust hsv in yiq*

---

### Description

Adjust hue, saturation, value of an RGB image in YIQ color space.

### Usage

```
img_adjust_hsv_in_yiq(  
  image,  
  delta_hue = 0,  
  scale_saturation = 1,  
  scale_value = 1,  
  name = NULL  
)
```

**Arguments**

image	RGB image or images. Size of the last dimension must be 3.
delta_hue	float, the hue rotation amount, in radians.
scale_saturation	float, factor to multiply the saturation by.
scale_value	float, factor to multiply the value by.
name	A name for this operation (optional).

**Details**

This is a convenience method that converts an RGB image to float representation, converts it to YIQ, rotates the color around the Y channel by delta\_hue in radians, scales the chrominance channels (I, Q) by scale\_saturation, scales all channels (Y, I, Q) by scale\_value, converts back to RGB, and then back to the original data type. 'image' is an RGB image. The image hue is adjusted by converting the image to YIQ, rotating around the luminance channel (Y) by 'delta\_hue' in radians, multiplying the chrominance channels (I, Q) by 'scale\_saturation', and multiplying all channels (Y, I, Q) by 'scale\_value'. The image is then converted back to RGB.

**Value**

Adjusted image(s), same shape and dtype as 'image'.

---

```
img_angles_to_projective_transforms
    Angles to projective transforms
```

---

**Description**

Returns projective transform(s) for the given angle(s).

**Usage**

```
img_angles_to_projective_transforms(
    angles,
    image_height,
    image_width,
    name = NULL
)
```

**Arguments**

angles	A scalar angle to rotate all images by, or (for batches of images) a vector with an angle to rotate each image in the batch. The rank must be statically known (the shape is not 'TensorShape(NULL)').
image_height	Height of the image(s) to be transformed.
image_width	Width of the image(s) to be transformed.
name	name of the op.

**Value**

A tensor of shape (num\_images, 8). Projective transforms which can be given to 'transform' op.

---



*Blend*


---

**Description**

Blend image1 and image2 using 'factor'.

**Usage**

```
img_blend(image1, image2, factor)
```

**Arguments**

image1	An image Tensor of shape (num_rows, num_columns, num_channels) (HWC), or (num_rows, num_columns) (HW), or (num_channels, num_rows, num_columns).
image2	An image Tensor of shape (num_rows, num_columns, num_channels) (HWC), or (num_rows, num_columns) (HW), or (num_channels, num_rows, num_columns).
factor	A floating point value or Tensor of type tf.float32 above 0.0.

**Details**

Factor can be above 0.0. A value of 0.0 means only image1 is used. A value of 1.0 means only image2 is used. A value between 0.0 and 1.0 means we linearly interpolate the pixel values between the two images. A value greater than 1.0 "extrapolates" the difference between the two pixel values, and we clip the results to values between 0 and 255.

**Value**

A blended image Tensor of tf\$float32.

---

img\_compose\_transforms

*Compose transforms*


---

**Description**

Composes the transforms tensors.

**Usage**

```
img_compose_transforms(transforms, name = NULL)
```



**Arguments**

transforms	List of image projective transforms to be composed. Each transform is length 8 (single transform) or shape (N, 8) (batched transforms). The shapes of all inputs must be equal, and at least one input must be given.
name	The name for the op.

**Value**

A composed transform tensor. When passed to ‘transform’ op, equivalent to applying each of the given transforms to the image in order.

---

img\_connected\_components

*Connected components*

---

**Description**

Labels the connected components in a batch of images.

**Usage**

```
img_connected_components(images, name = NULL)
```

**Arguments**

images	A 2D (H, W) or 3D (N, H, W) Tensor of image (integer, floating point and boolean types are supported).
name	The name of the op.

**Details**

A component is a set of pixels in a single input image, which are all adjacent and all have the same non-zero value. The components using a squared connectivity of one (all equal entries are joined with their neighbors above,below, left, and right). Components across all images have consecutive ids 1 through n. Components are labeled according to the first pixel of the component appearing in row-major order (lexicographic order by image\_index\_in\_batch, row, col). Zero entries all have an output id of 0. This op is equivalent with ‘scipy.ndimage.measurements.label’ on a 2D array with the default structuring element (which is the connectivity used here).

**Value**

Components with the same shape as ‘images’. entries that evaluate to FALSE (e.g. 0/0.0f, FALSE) in ‘images’ have value 0, and all other entries map to a component id > 0.

**Raises**

TypeError: if ‘images’ is not 2D or 3D.

img\_cutout

*Cutout***Description**

Apply cutout (<https://arxiv.org/abs/1708.04552>) to images.

**Usage**

```
img_cutout(
    images,
    mask_size,
    offset = list(0, 0),
    constant_values = 0,
    data_format = "channels_last"
)
```

**Arguments**

images	A tensor of shape (batch_size, height, width, channels) (NHWC), (batch_size, channels, height, width)(NCHW).
mask_size	Specifies how big the zero mask that will be generated is that is applied to the images. The mask will be of size (mask_height x mask_width). Note: mask_size should be divisible by 2.
offset	A list of (height, width) or (batch_size, 2)
constant_values	What pixel value to fill in the images in the area that has the cutout mask applied to it.
data_format	A string, one of 'channels_last' (default) or 'channels_first'. The ordering of the dimensions in the inputs. 'channels_last' corresponds to inputs with shape '(batch_size, ..., channels)' while 'channels_first' corresponds to inputs with shape '(batch_size, channels, ...)'

**Details**

This operation applies a (mask\_height x mask\_width) mask of zeros to a location within 'img' specified by the offset. The pixel values filled in will be of the value 'replace'. The located where the mask will be applied is randomly chosen uniformly over the whole images.

**Value**

An image Tensor.

**Raises**

InvalidArgumentError: if mask\_size can't be divisible by 2.

---

img\_dense\_image\_warp *Dense image warp*

---

### Description

Image warping using per-pixel flow vectors.

### Usage

```
img_dense_image_warp(image, flow, name = NULL)
```

### Arguments

image	4-D float Tensor with shape [batch, height, width, channels].
flow	A 4-D float Tensor with shape [batch, height, width, 2].
name	A name for the operation (optional).

### Details

Apply a non-linear warp to the image, where the warp is specified by a dense flow field of offset vectors that define the correspondences of pixel values in the output image back to locations in the source image. Specifically, the pixel value at output[b, j, i, c] is images[b, j - flow[b, j, i, 0], i - flow[b, j, i, 1], c]. The locations specified by this formula do not necessarily map to an int index. Therefore, the pixel value is obtained by bilinear interpolation of the 4 nearest pixels around (b, j - flow[b, j, i, 0], i - flow[b, j, i, 1]). For locations outside of the image, we use the nearest pixel values at the image boundary.

### Value

A 4-D float 'Tensor' with shape '[batch, height, width, channels]' and same type as input image.

### Raises

ValueError: if height < 2 or width < 2 or the inputs have the wrong number of dimensions.

### Note

Note that image and flow can be of type tf\$half, tf\$float32, or tf\$float64, and do not necessarily have to be the same type.

### Examples

```
## Not run:  
flow_shape = list(1L, as.integer(input_img$shape[[2]]), as.integer(input_img$shape[[3]]), 2L)  
init_flows = tf$random$normal(flow_shape) * 2.0  
dense_img_warp = img_dense_image_warp(input_img, init_flows)  
dense_img_warp = tf$squeeze(dense_img_warp, 0)
```

```
## End(Not run)
```

---

img_equalize	<i>Equalize</i>
--------------	-----------------

---

### Description

Equalize image(s)

### Usage

```
img_equalize(image, data_format = "channels_last", name = NULL)
```

### Arguments

image	A tensor of shape (num_images, num_rows, num_columns, num_channels) (NHWC), or (num_images, num_channels, num_rows, num_columns) (NCHW), or (num_rows, num_columns, num_channels) (HWC), or (num_channels, num_rows, num_columns) (CHW), or (num_rows, num_columns) (HW). The rank must be statically known (the shape is not TensorShape(None)).
data_format	Either 'channels_first' or 'channels_last'
name	The name of the op. Returns: Image(s) with the same type and shape as 'images', equalized.

### Value

Image(s) with the same type and shape as 'images', equalized.

### Examples

```
## Not run:  
img_equalize(img)  
  
## End(Not run)
```

---

img\_euclidean\_dist\_transform  
*Euclidean dist transform*

---

### Description

Applies euclidean distance transform(s) to the image(s).

### Usage

```
img_euclidean_dist_transform(images, dtype = tf$float32, name = NULL)
```

### Arguments

images	A tensor of shape (num_images, num_rows, num_columns, 1) (NHWC), or (num_rows, num_columns, 1) (HWC) or (num_rows, num_columns) (HW).
dtype	DType of the output tensor.
name	The name of the op.

### Value

Image(s) with the type 'dtype' and same shape as 'images', with the transform applied. If a tensor of all ones is given as input, the output tensor will be filled with the max value of the 'dtype'.

### Raises

TypeError: If 'image' is not tf.uint8, or 'dtype' is not floating point. ValueError: If 'image' more than one channel, or 'image' is not of rank between 2 and 4.

### Examples

```
## Not run:
img_path = tf$keras$utils$get_file('tensorflow.png', 'https://tensorflow.org/images/tf_logo.png')
img_raw = tf$io$read_file(img_path)
img = tf$io$decode_png(img_raw)
img = tf$image$convert_image_dtype(img, tf$float32)
img = tf$image$resize(img, c(500L, 500L))
bw_img = 1.0 - tf$image$rgb_to_grayscale(img)
gray = tf$image$convert_image_dtype(bw_img, tf$uint8)
gray = tf$expand_dims(gray, 0L)
eucid = img_euclidean_dist_transform(gray)
eucid = tf$squeeze(eucid, c(0, -1))

## End(Not run)
```

---

img\_flat\_transforms\_to\_matrices  
*Flat transforms to matrices*

---

**Description**

Converts projective transforms to affine matrices.

**Usage**

```
img_flat_transforms_to_matrices(transforms, name = NULL)
```

**Arguments**

transforms      Vector of length 8, or batches of transforms with shape '(N, 8)'.  
name             The name for the op.

**Details**

Note that the output matrices map output coordinates to input coordinates. For the forward transformation matrix, call 'tf.linalg\$inv' on the result.

**Value**

3D tensor of matrices with shape '(N, 3, 3)'. The output matrices map the *output coordinates* (in homogeneous coordinates) of each transform to the corresponding *input coordinates*.

**Raises**

ValueError: If 'transforms' have an invalid shape.

---

img\_from\_4D                      *From 4D image*

---

**Description**

Convert back to an image with 'ndims' rank.

**Usage**

```
img_from_4D(image, ndims)
```

**Arguments**

image             4D tensor.  
ndims             The original rank of the image.

**Value**

'ndims'-D tensor with the same type.

---

img_get_ndims	<i>Get ndims</i>
---------------	------------------

---

**Description**

Print dimensions

**Usage**

```
img_get_ndims(image)
```

**Arguments**

image	image
-------	-------

**Value**

dimensions of the image

---

img_interpolate_bilinear	<i>Interpolate bilinear</i>
--------------------------	-----------------------------

---

**Description**

Similar to Matlab's interp2 function.

**Usage**

```
img_interpolate_bilinear(grid, query_points, indexing = "ij", name = NULL)
```

**Arguments**

grid	a 4-D float Tensor of shape [batch, height, width, channels].
query_points	a 3-D float Tensor of N points with shape [batch, N, 2].
indexing	whether the query points are specified as row and column (ij), or Cartesian coordinates (xy).
name	a name for the operation (optional).

**Details**

Finds values for query points on a grid using bilinear interpolation.

**Value**

values: a 3-D ‘Tensor‘ with shape ‘[batch, N, channels]‘

**Raises**

ValueError: if the indexing mode is invalid, or if the shape of the inputs invalid.

---

img\_interpolate\_spline  
*Interpolate spline*

---

**Description**

Interpolate signal using polyharmonic interpolation.

**Usage**

```
img_interpolate_spline(
    train_points,
    train_values,
    query_points,
    order,
    regularization_weight = 0,
    name = "interpolate_spline"
)
```

**Arguments**

train_points	‘[batch_size, n, d]‘ float ‘Tensor‘ of n d-dimensional locations. These do not need to be regularly-spaced.
train_values	‘[batch_size, n, k]‘ float ‘Tensor‘ of n c-dimensional values evaluated at train_points.
query_points	‘[batch_size, m, d]‘ ‘Tensor‘ of m d-dimensional locations where we will output the interpolant’s values.
order	order of the interpolation. Common values are 1 for ‘ $\phi(r) = r$ ’, 2 for ‘ $\phi(r) = r^2 * \log(r)$ ’ (thin-plate spline), or 3 for ‘ $\phi(r) = r^3$ ‘.
regularization_weight	weight placed on the regularization term. This will depend substantially on the problem, and it should always be tuned. For many problems, it is reasonable to use no regularization. If using a non-zero value, we recommend a small value like 0.001.
name	name prefix for ops created by this function



**Details**

The interpolant has the form  $f(x) = \sum_{i=1}^n w_i \phi(\|x - c_i\|) + v^T x + b$ . This is a sum of two terms: (1) a weighted sum of radial basis function (RBF) terms, with the centers  $(c_1, \dots, c_n)$ , and (2) a linear term with a bias. The  $(c_i)$  vectors are 'training' points. In the code,  $b$  is absorbed into  $v$  by appending 1 as a final dimension to  $x$ . The coefficients  $w$  and  $v$  are estimated such that the interpolant exactly fits the value of the function at the  $(c_i)$  points, the vector  $w$  is orthogonal to each  $(c_i)$ , and the vector  $w$  sums to 0. With these constraints, the coefficients can be obtained by solving a linear system. ' $\phi$ ' is an RBF, parametrized by an interpolation order. Using `order=2` produces the well-known thin-plate spline. We also provide the option to perform regularized interpolation. Here, the interpolant is selected to trade off between the squared loss on the training data and a certain measure of its curvature ([details](https://en.wikipedia.org/wiki/Polyharmonic\_spline)). Using a regularization weight greater than zero has the effect that the interpolant will no longer exactly fit the training data. However, it may be less vulnerable to overfitting, particularly for high-order interpolation. Note the interpolation procedure is differentiable with respect to all inputs besides the order parameter. We support dynamically-shaped inputs, where `batch_size`, `n`, and `m` are NULL at graph construction time. However, `d` and `k` must be known.

**Value**

'[`b`, `m`, `k`]' float 'Tensor' of query values. We use `train_points` and `train_values` to perform polyharmonic interpolation. The query values are the values of the interpolant evaluated at the locations specified in `query_points`.

**This is a sum of two terms**

(1) a weighted sum of radial basis function: (RBF) terms, with the centers  $(c_1, \dots, c_n)$ , and (2) a linear term with a bias. The  $(c_i)$  vectors are 'training' points. In the code,  $b$  is absorbed into  $v$  by appending 1 as a final dimension to  $x$ . The coefficients  $w$  and  $v$  are estimated such that the interpolant exactly fits the value of the function at the  $(c_i)$  points, the vector  $w$  is orthogonal to each  $(c_i)$ , and the vector  $w$  sums to 0. With these constraints, the coefficients can be obtained by solving a linear system.

---



*Matrices to flat transforms*

---

**Description**

Converts affine matrices to projective transforms.

**Usage**

```
img_matrices_to_flat_transforms(transform_matrices, name = NULL)
```

**Arguments**

transform_matrices	One or more affine transformation matrices, for the reverse transformation in homogeneous coordinates. Shape 'c(3, 3)' or 'c(N, 3, 3)'.
name	The name for the op.

**Details**

Note that we expect matrices that map output coordinates to input coordinates. To convert forward transformation matrices, call 'tf.linalg\$inv' on the matrices and use the result here.

**Value**

2D tensor of flat transforms with shape '(N, 8)', which may be passed into 'transform' op.

**Raises**

ValueError: If 'transform\_matrices' have an invalid shape.

---

img\_mean\_filter2d      *Mean filter2d*

---

**Description**

Perform mean filtering on image(s).

**Usage**

```
img_mean_filter2d(
  image,
  filter_shape = list(3, 3),
  padding = "REFLECT",
  constant_values = 0,
  name = NULL
)
```

**Arguments**

image	Either a 2-D Tensor of shape [height, width], a 3-D Tensor of shape [height, width, channels], or a 4-D Tensor of shape [batch_size, height, width, channels].
filter_shape	An integer or tuple/list of 2 integers, specifying the height and width of the 2-D mean filter. Can be a single integer to specify the same value for all spatial dimensions.
padding	A string, one of "REFLECT", "CONSTANT", or "SYMMETRIC". The type of padding algorithm to use, which is compatible with mode argument in tf.pad. For more details, please refer to <a href="https://www.tensorflow.org/api_docs/python/tf/pad">https://www.tensorflow.org/api_docs/python/tf/pad</a> .

constant\_values      A scalar, the pad value to use in "CONSTANT" padding mode.

name                  A name for this operation (optional).

**Value**

3-D or 4-D ‘Tensor’ of the same dtype as input.

**Raises**

ValueError: If ‘image’ is not 2, 3 or 4-dimensional, if ‘padding’ is other than "REFLECT", "CONSTANT" or "SYMMETRIC", or if ‘filter\_shape’ is invalid.

---

img\_median\_filter2d      *Median filter2d*

---

**Description**

Perform median filtering on image(s).

**Usage**

```
img_median_filter2d(
    image,
    filter_shape = list(3, 3),
    padding = "REFLECT",
    constant_values = 0,
    name = NULL
)
```

**Arguments**

image                  Either a 2-D Tensor of shape [height, width], a 3-D Tensor of shape [height, width, channels], or a 4-D Tensor of shape [batch\_size, height, width, channels].

filter\_shape          An integer or tuple/list of 2 integers, specifying the height and width of the 2-D median filter. Can be a single integer to specify the same value for all spatial dimensions.

padding                A string, one of "REFLECT", "CONSTANT", or "SYMMETRIC". The type of padding algorithm to use, which is compatible with mode argument in tf.pad. For more details, please refer to [https://www.tensorflow.org/api\\_docs/python/tf/pad](https://www.tensorflow.org/api_docs/python/tf/pad).

constant\_values      A scalar, the pad value to use in "CONSTANT" padding mode.

name                  A name for this operation (optional)

**Value**

3-D or 4-D ‘Tensor’ of the same dtype as input.

**Raises**

ValueError: If 'image' is not 2, 3 or 4-dimensional, if 'padding' is other than "REFLECT", "CONSTANT" or "SYMMETRIC", or if 'filter\_shape' is invalid.

---

img\_random\_cutout      *Random cutout*

---

**Description**

Apply cutout (<https://arxiv.org/abs/1708.04552>) to images.

**Usage**

```
img_random_cutout(
    images,
    mask_size,
    constant_values = 0,
    seed = NULL,
    data_format = "channels_last"
)
```

**Arguments**

images	A tensor of shape (batch_size, height, width, channels) (NHWC), (batch_size, channels, height, width)(NCHW).
mask_size	Specifies how big the zero mask that will be generated is that is applied to the images. The mask will be of size (mask_height x mask_width). Note: mask_size should be divisible by 2.
constant_values	What pixel value to fill in the images in the area that has the cutout mask applied to it.
seed	An integer. Used in combination with 'tf\$random\$set_seed' to create a reproducible sequence of tensors across multiple calls.
data_format	A string, one of 'channels_last' (default) or 'channels_first'. The ordering of the dimensions in the inputs. 'channels_last' corresponds to inputs with shape '(batch_size, ..., channels)' while 'channels_first' corresponds to inputs with shape '(batch_size, channels, ...)'.

**Details**

This operation applies a (mask\_height x mask\_width) mask of zeros to a random location within 'img'. The pixel values filled in will be of the value 'replace'. The located where the mask will be applied is randomly chosen uniformly over the whole images.

**Value**

An image Tensor.

**Raises**

InvalidArgumentError: if mask\_size can't be divisible by 2.

---

img\_random\_hsv\_in\_yiq *Random hsv in yiq*

---

**Description**

Adjust hue, saturation, value of an RGB image randomly in YIQ color

**Usage**

```
img_random_hsv_in_yiq(
    image,
    max_delta_hue = 0,
    lower_saturation = 1,
    upper_saturation = 1,
    lower_value = 1,
    upper_value = 1,
    seed = NULL,
    name = NULL
)
```

**Arguments**

image	RGB image or images. Size of the last dimension must be 3.
max_delta_hue	float. Maximum value for the random delta_hue. Passing 0 disables adjusting hue.
lower_saturation	float. Lower bound for the random scale_saturation.
upper_saturation	float. Upper bound for the random scale_saturation.
lower_value	float. Lower bound for the random scale_value.
upper_value	float. Upper bound for the random scale_value.
seed	An operation-specific seed. It will be used in conjunction with the graph-level seed to determine the real seeds that will be used in this operation. Please see the documentation of set_random_seed for its interaction with the graph-level random seed.
name	A name for this operation (optional).

**Details**

space. Equivalent to 'adjust\_yiq\_hsv()' but uses a 'delta\_h' randomly picked in the interval '[-max\_delta\_hue, max\_delta\_hue]', a 'scale\_saturation' randomly picked in the interval '[lower\_saturation, upper\_saturation]', and a 'scale\_value' randomly picked in the interval '[lower\_saturation, upper\_saturation]'.

**Value**

3-D float tensor of shape '[height, width, channels]'.

**Raises**

ValueError: if 'max\_delta', 'lower\_saturation', 'upper\_saturation', 'lower\_value', or 'upper\_value' is invalid.

**Examples**

```
## Not run:
delta = 0.5
lower_saturation = 0.1
upper_saturation = 0.9
lower_value = 0.2
upper_value = 0.8
rand_hsvinyiq = img_random_hsv_in_yiq(img, delta,
lower_saturation, upper_saturation,
lower_value, upper_value)
)

## End(Not run)
```

---

img\_resampler

*Resampler*


---

**Description**

Resamples input data at user defined coordinates.

**Usage**

```
img_resampler(data, warp, name = NULL)
```

**Arguments**

data	Tensor of shape [batch_size, data_height, data_width, data_num_channels] containing 2D data that will be resampled.
warp	Tensor of minimum rank 2 containing the coordinates at which resampling will be performed. Since only bilinear interpolation is currently supported, the last dimension of the warp tensor must be 2, representing the (x, y) coordinate where x is the index for width and y is the index for height.
name	Optional name of the op.

**Details**

The resampler currently only supports bilinear interpolation of 2D data.

**Value**

Tensor of resampled values from 'data'. The output tensor shape is determined by the shape of the warp tensor. For example, if 'data' is of shape '[batch\_size, data\_height, data\_width, data\_num\_channels]' and warp of shape '[batch\_size, dim\_0, ... , dim\_n, 2]' the output will be of shape '[batch\_size, dim\_0, ... , dim\_n, data\_num\_channels]'.

**Raises**

ImportError: if the wrapper generated during compilation is not present when the function is called.

---

img_rotate	<i>Rotate</i>
------------	---------------

---

**Description**

Rotate image(s) counterclockwise by the passed angle(s) in radians.

**Usage**

```
img_rotate(images, angles, interpolation = "NEAREST", name = NULL)
```

**Arguments**

images	A tensor of shape (num_images, num_rows, num_columns, num_channels) (NHWC), (num_rows, num_columns, num_channels) (HWC), or (num_rows, num_columns) (HW).
angles	A scalar angle to rotate all images by, or (if images has rank 4) a vector of length num_images, with an angle for each image in the batch.
interpolation	Interpolation mode. Supported values: "NEAREST", "BILINEAR".
name	The name of the op.

**Value**

Image(s) with the same type and shape as 'images', rotated by the given angle(s). Empty space due to the rotation will be filled with zeros.

**Raises**

TypeError: If 'image' is an invalid type.

---

img_sharpness	<i>Sharpness</i>
---------------	------------------

---

**Description**

Change sharpness of image(s)

**Usage**

```
img_sharpness(image, factor)
```

**Arguments**

image	an image
factor	A floating point value or Tensor above 0.0.

**Value**

Image(s) with the same type and shape as 'images', sharper.

---

img_shear_x	<i>Shear x-axis</i>
-------------	---------------------

---

**Description**

Perform shear operation on an image (x-axis)

**Usage**

```
img_shear_x(image, level, replace)
```

**Arguments**

image	A 3D image Tensor.
level	A float denoting shear element along y-axis
replace	A one or three value 1D tensor to fill empty pixels.

**Value**

Transformed image along X or Y axis, with space outside image filled with replace.



---

img_shear_y	<i>Shear y-axis</i>
-------------	---------------------

---

**Description**

Perform shear operation on an image (y-axis)

**Usage**

```
img_shear_y(image, level, replace)
```

**Arguments**

image	A 3D image Tensor.
level	A float denoting shear element along x-axis
replace	A one or three value 1D tensor to fill empty pixels.

**Value**

Transformed image along X or Y axis, with space outside image filled with replace.

---

img_sparse_image_warp	<i>Sparse image warp</i>
-----------------------	--------------------------

---

**Description**

Image warping using correspondences between sparse control points.

**Usage**

```
img_sparse_image_warp(  
    image,  
    source_control_point_locations,  
    dest_control_point_locations,  
    interpolation_order = 2,  
    regularization_weight = 0,  
    num_boundary_points = 0,  
    name = "sparse_image_warp"  
)
```

**Arguments**

image	'[batch, height, width, channels]' float 'Tensor'
source_control_point_locations	'[batch, num_control_points, 2]' float 'Tensor'
dest_control_point_locations	'[batch, num_control_points, 2]' float 'Tensor'
interpolation_order	polynomial order used by the spline interpolation
regularization_weight	weight on smoothness regularizer in interpolation
num_boundary_points	How many zero-flow boundary points to include at each image edge. Usage: num_boundary_points=0: don't add zero-flow points num_boundary_points=1: 4 corners of the image num_boundary_points=2: 4 corners and one in the middle of each edge (8 points total) num_boundary_points=n: 4 corners and n-1 along each edge
name	A name for the operation (optional).

**Details**

Apply a non-linear warp to the image, where the warp is specified by the source and destination locations of a (potentially small) number of control points. First, we use a polyharmonic spline ('tf\$contrib\$image\$interpolate\_spline') to interpolate the displacements between the corresponding control points to a dense flow field. Then, we warp the image using this dense flow field ('tf\$contrib\$image\$dense\_image\_warp'). Let  $t$  index our control points. For regularization\_weight=0, we have: warped\_image[b, dest\_control\_point\_locations[b, t, 0], dest\_control\_point\_locations[b, t, 1], :] = image[b, source\_control\_point\_locations[b, t, 0], source\_control\_point\_locations[b, t, 1], :]. For regularization\_weight > 0, this condition is met approximately, since regularized interpolation trades off smoothness of the interpolant vs. reconstruction of the interpolant at the control points. See 'tf\$contrib\$image\$interpolate\_spline' for further documentation of the interpolation\_order and regularization\_weight arguments.

**Value**

warped\_image: '[batch, height, width, channels]' float 'Tensor' with same type as input image.  
 flow\_field: '[batch, height, width, 2]' float 'Tensor' containing the dense flow field produced by the interpolation.

---

img\_to\_4D

*To 4D image*


---

**Description**

Convert 2/3/4D image to 4D image.

**Usage**

```
img_to_4D(image)
```

**Arguments**

image            2/3/4D tensor.

**Value**

4D tensor with the same type.

**Examples**

```
## Not run:
img_to_4D(img)

## End(Not run)
```

---

img_transform	<i>Transform</i>
---------------	------------------

---

**Description**

Applies the given transform(s) to the image(s).

**Usage**

```
img_transform(
  images,
  transforms,
  interpolation = "NEAREST",
  output_shape = NULL,
  name = NULL
)
```

**Arguments**

images            A tensor of shape (num\_images, num\_rows, num\_columns, num\_channels) (NHWC), (num\_rows, num\_columns, num\_channels) (HWC), or (num\_rows, num\_columns) (HW).

transforms        Projective transform matrix/matrices. A vector of length 8 or tensor of size N x 8. If one row of transforms is [a0, a1, a2, b0, b1, b2, c0, c1], then it maps the output point (x, y) to a transformed input point (x', y') = ((a0 x + a1 y + a2) / k, (b0 x + b1 y + b2) / k), where k = c0 x + c1 y + 1. The transforms are inverted compared to the transform mapping input points to output points. Note that gradients are not backpropagated into transformation parameters.

interpolation	Interpolation mode. Supported values: "NEAREST", "BILINEAR".
output_shape	Output dimension after the transform, [height, width]. If NULL, output is the same size as input image.
name	The name of the op.

**Value**

Image(s) with the same type and shape as 'images', with the given transform(s) applied. Transformed coordinates outside of the input image will be filled with zeros.

**Raises**

TypeError: If 'image' is an invalid type. ValueError: If output shape is not 1-D int32 Tensor.

**Examples**

```
## Not run:
transform = img_transform(img, c(1.0, 1.0, -250, 0.0, 1.0, 0.0, 0.0, 0.0))

## End(Not run)
```

---

img_translate	<i>Translate</i>
---------------	------------------

---

**Description**

Translate image(s) by the passed vectors(s).

**Usage**

```
img_translate(images, translations, interpolation = "NEAREST", name = NULL)
```

**Arguments**

images	A tensor of shape (num_images, num_rows, num_columns, num_channels) (NHWC), (num_rows, num_columns, num_channels) (HWC), or (num_rows, num_columns) (HW). The rank must be statically known (the shape is not TensorShape(None)).
translations	A vector representing [dx, dy] or (if images has rank 4) a matrix of length num_images, with a [dx, dy] vector for each image in the batch.
interpolation	Interpolation mode. Supported values: "NEAREST", "BILINEAR".
name	The name of the op.

**Value**

Image(s) with the same type and shape as 'images', translated by the given vector(s). Empty space due to the translation will be filled with zeros.

**Raises**

TypeError: If 'images' is an invalid type.

---

<code>img_translate_xy</code>	<i>Translate xy dims</i>
-------------------------------	--------------------------

---

**Description**

Translates image in X or Y dimension.

**Usage**

```
img_translate_xy(image, translate_to, replace)
```

**Arguments**

- `image`            A 3D image Tensor.
- `translate_to`    A 1D tensor to translate [x, y]
- `replace`          A one or three value 1D tensor to fill empty pixels.

**Value**

Translated image along X or Y axis, with space outside image filled with replace. Raises: ValueError: if axis is neither 0 nor 1.

**Raises**

ValueError: if axis is neither 0 nor 1.

---

img\_translations\_to\_projective\_transforms  
*Translations to projective transforms*

---

**Description**

Returns projective transform(s) for the given translation(s).

**Usage**

```
img_translations_to_projective_transforms(translations, name = NULL)
```

**Arguments**

translations	A 2-element list representing [dx, dy] or a matrix of 2-element lists representing [dx, dy] to translate for each image (for a batch of images). The rank must be statically known (the shape is not 'TensorShape(NULL)').
name	The name of the op.

**Value**

A tensor of shape c(num\_images, 8) projective transforms which can be given to 'img\_transform'.

---

img\_unwrap                      *Uwrap*

---

**Description**

Unwraps an image produced by wrap.

**Usage**

```
img_unwrap(image, replace)
```

**Arguments**

image	image
replace	a one or three value 1D tensor to fill empty pixels.

**Details**

Where there is a 0 in the last channel for every spatial position, the rest of the three channels in that spatial dimension are grayed (set to 128). Operations like translate and shear on a wrapped Tensor will leave 0s in empty locations. Some transformations look at the intensity of values to do preprocessing, and we want these empty pixels to assume the 'average' value, rather than pure black.

**Value**

a 3D image Tensor with 3 channels.

---

img_wrap	<i>Wrap</i>
----------	-------------

---

**Description**

wrap an image array

**Usage**

```
img_wrap(image)
```

**Arguments**

image            a 3D Image Tensor with 4 channels.

**Value**

'image' with an extra channel set to all 1s.

---

install_tfaddons	<i>Install TensorFlow SIG Addons</i>
------------------	--------------------------------------

---

**Description**

This function is used to install the 'TensorFlow SIG Addons' python module

**Usage**

```
install_tfaddons(version = NULL, ..., restart_session = TRUE)
```

**Arguments**

version            for specific version of 'TensorFlow SIG Addons', e.g. "0.10.0"  
 ...                other arguments passed to [reticulate::py\_install()].  
 restart\_session   Restart R session after installing (note this will only occur within RStudio).

**Value**

a python module 'tensorflow\_addons'

---

layer\_activation\_gelu *Gaussian Error Linear Unit*

---

**Description**

Gaussian Error Linear Unit

**Usage**

```
layer_activation_gelu(object, approximate = TRUE, ...)
```

**Arguments**

object	Model or layer object
approximate	(bool) Whether to apply approximation
...	additional parameters to pass

**Details**

A smoother version of ReLU generally used in the BERT or BERT architecture based models.  
Original paper: <https://arxiv.org/abs/1606.08415>

**Value**

A tensor

**Note**

Input shape: Arbitrary. Use the keyword argument 'input\_shape' (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape: Same shape as the input.

---

layer\_correlation\_cost  
*Correlation Cost Layer.*

---

**Description**

Correlation Cost Layer.



**Usage**

```

layer_correlation_cost(
    object,
    kernel_size,
    max_displacement,
    stride_1,
    stride_2,
    pad,
    data_format,
    ...
)

```

**Arguments**

object	Model or layer object
kernel_size	An integer specifying the height and width of the patch used to compute the per-patch costs.
max_displacement	An integer specifying the maximum search radius for each position.
stride_1	An integer specifying the stride length in the input.
stride_2	An integer specifying the stride length in the patch.
pad	An integer specifying the paddings in height and width.
data_format	Specifies the data format. Possible values are: "channels_last" float [batch, height, width, channels] "channels_first" float [batch, channels, height, width] Defaults to "channels_last".
...	additional parameters to pass

**Details**

This layer implements the correlation operation from FlowNet Learning Optical Flow with Convolutional Networks (Fischer et al.): <https://arxiv.org/abs/1504.06>

**Value**

A tensor

---

layer\_filter\_response\_normalization  
*FilterResponseNormalization*

---

**Description**

Filter response normalization layer.

**Usage**

```
layer_filter_response_normalization(
    object,
    epsilon = 1e-06,
    axis = c(1, 2),
    beta_initializer = "zeros",
    gamma_initializer = "ones",
    beta_regularizer = NULL,
    gamma_regularizer = NULL,
    beta_constraint = NULL,
    gamma_constraint = NULL,
    learned_epsilon = FALSE,
    learned_epsilon_constraint = NULL,
    name = NULL
)
```

**Arguments**

<code>object</code>	Model or layer object
<code>epsilon</code>	Small positive float value added to variance to avoid dividing by zero.
<code>axis</code>	List of axes that should be normalized. This should represent the spatial dimensions.
<code>beta_initializer</code>	Initializer for the beta weight.
<code>gamma_initializer</code>	Initializer for the gamma weight.
<code>beta_regularizer</code>	Optional regularizer for the beta weight.
<code>gamma_regularizer</code>	Optional regularizer for the gamma weight.
<code>beta_constraint</code>	Optional constraint for the beta weight.
<code>gamma_constraint</code>	Optional constraint for the gamma weight.
<code>learned_epsilon</code>	(bool) Whether to add another learnable epsilon parameter or not.
<code>learned_epsilon_constraint</code>	<code>learned_epsilon_constraint</code>
<code>name</code>	Optional name for the layer

**Details**

Filter Response Normalization (FRN), a normalization method that enables models trained with per-channel normalization to achieve high accuracy. It performs better than all other normalization techniques for small batches and is par with Batch Normalization for bigger batch sizes.

**Value**

A tensor

**Note**

Input shape Arbitrary. Use the keyword argument 'input\_shape' (list of integers, does not include the samples axis) when using this layer as the first layer in a model. This layer, as of now, works on a 4-D tensor where the tensor should have the shape [N X H X W X C] TODO: Add support for NCHW data format and FC layers. Output shape Same shape as input. References - [Filter Response Normalization Layer: Eliminating Batch Dependence in the training of Deep Neural Networks] (<https://arxiv.org/abs/1911.09737>)

---

layer\_group\_normalization

*Group normalization layer*

---

**Description**

Group normalization layer

**Usage**

```
layer_group_normalization(
    object,
    groups = 2,
    axis = -1,
    epsilon = 0.001,
    center = TRUE,
    scale = TRUE,
    beta_initializer = "zeros",
    gamma_initializer = "ones",
    beta_regularizer = NULL,
    gamma_regularizer = NULL,
    beta_constraint = NULL,
    gamma_constraint = NULL,
    ...
)
```

**Arguments**

object	Model or layer object
groups	Integer, the number of groups for Group Normalization. Can be in the range [1, N] where N is the input dimension. The input dimension must be divisible by the number of groups.
axis	Integer, the axis that should be normalized.
epsilon	Small float added to variance to avoid dividing by zero.

<code>center</code>	If TRUE, add offset of beta to normalized tensor. If False, beta is ignored.
<code>scale</code>	If TRUE, multiply by gamma. If False, gamma is not used.
<code>beta_initializer</code>	Initializer for the beta weight.
<code>gamma_initializer</code>	Initializer for the gamma weight.
<code>beta_regularizer</code>	Optional regularizer for the beta weight.
<code>gamma_regularizer</code>	Optional regularizer for the gamma weight.
<code>beta_constraint</code>	Optional constraint for the beta weight.
<code>gamma_constraint</code>	Optional constraint for the gamma weight.
<code>...</code>	additional parameters to pass

**Details**

Group Normalization divides the channels into groups and computes within each group the mean and variance for normalization. Empirically, its accuracy is more stable than batch norm in a wide range of small batch sizes, if learning rate is adjusted linearly with batch sizes. Relation to Layer Normalization: If the number of groups is set to 1, then this operation becomes identical to Layer Normalization. Relation to Instance Normalization: If the number of groups is set to the input dimension (number of groups is equal to number of channels), then this operation becomes identical to Instance Normalization.

**Value**

A tensor

---

`layer_instance_normalization`  
*Instance normalization layer*

---

**Description**

Instance normalization layer

**Usage**

```
layer_instance_normalization(
    object,
    groups = 2,
    axis = -1,
    epsilon = 0.001,
```

```

        center = TRUE,
        scale = TRUE,
        beta_initializer = "zeros",
        gamma_initializer = "ones",
        beta_regularizer = NULL,
        gamma_regularizer = NULL,
        beta_constraint = NULL,
        gamma_constraint = NULL,
        ...
    )

```

### Arguments

object	Model or layer object
groups	Integer, the number of groups for Group Normalization. Can be in the range [1, N] where N is the input dimension. The input dimension must be divisible by the number of groups.
axis	Integer, the axis that should be normalized.
epsilon	Small float added to variance to avoid dividing by zero.
center	If TRUE, add offset of 'beta' to normalized tensor. If FALSE, 'beta' is ignored.
scale	If TRUE, multiply by 'gamma'. If FALSE, 'gamma' is not used.
beta_initializer	Initializer for the beta weight.
gamma_initializer	Initializer for the gamma weight.
beta_regularizer	Optional regularizer for the beta weight.
gamma_regularizer	Optional regularizer for the gamma weight.
beta_constraint	Optional constraint for the beta weight.
gamma_constraint	Optional constraint for the gamma weight.
...	additional parameters to pass

### Details

Instance Normalization is an specific case of "GroupNormalizationsince" it normalizes all features of one channel. The Groupsize is equal to the channel size. Empirically, its accuracy is more stable than batch norm in a wide range of small batch sizes, if learning rate is adjusted linearly with batch sizes.

### Value

A tensor

**References**

[Instance Normalization: The Missing Ingredient for Fast Stylization](<https://arxiv.org/abs/1607.08022>)

---

layer_maxout	<i>Maxout layer</i>
--------------	---------------------

---

**Description**

Maxout layer

**Usage**

```
layer_maxout(object, num_units, axis = -1, ...)
```

**Arguments**

object	Model or layer object
num_units	Specifies how many features will remain after maxout in the axis dimension (usually channel). This must be a factor of number of features.
axis	The dimension where max pooling will be performed. Default is the last dimension.
...	additional parameters to pass

**Details**

"Maxout Networks" Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio. <https://arxiv.org/abs/1302.4389> Usually the operation is performed in the filter/channel dimension. This can also be used after Dense layers to reduce number of features.

**Value**

A tensor

---

layer_multi_head_attention	<i>Keras-based multi head attention layer</i>
----------------------------	---

---

**Description**

MultiHead Attention layer.

**Usage**

```

layer_multi_head_attention(
    object,
    head_size,
    num_heads,
    output_size = NULL,
    dropout = 0,
    use_projection_bias = TRUE,
    return_attn_coef = FALSE,
    kernel_initializer = "glorot_uniform",
    kernel_regularizer = NULL,
    kernel_constraint = NULL,
    bias_initializer = "zeros",
    bias_regularizer = NULL,
    bias_constraint = NULL,
    ...
)

```

**Arguments**

object	Model or layer object
head_size	int, dimensionality of the ‘query’, ‘key’ and ‘value’ tensors after the linear transformation.
num_heads	int, number of attention heads.
output_size	int, dimensionality of the output space, if ‘NULL’ then the input dimension of ‘value’ or ‘key’ will be used, default ‘NULL’.
dropout	float, ‘rate’ parameter for the dropout layer that is applied to attention after softmax, default ‘0’.
use_projection_bias	bool, whether to use a bias term after the linear output projection.
return_attn_coef	bool, if ‘TRUE’, return the attention coefficients as an additional output argument.
kernel_initializer	initializer, initializer for the kernel weights.
kernel_regularizer	regularizer, regularizer for the kernel weights.
kernel_constraint	constraint, constraint for the kernel weights.
bias_initializer	initializer, initializer for the bias weights.
bias_regularizer	regularizer, regularizer for the bias weights.
bias_constraint	constraint, constraint for the bias weights.
...	additional parameters to pass

**Details**

Defines the MultiHead Attention operation as defined in [Attention Is All You Need](https://arxiv.org/abs/1706.03762) which takes in a 'query', 'key' and 'value' tensors returns the dot-product attention between them.

**Value**

A tensor

**Examples**

```
## Not run:

mha = layer_multi_head_attention(head_size=128, num_heads=128)
query = tf$random$uniform(list(32L, 20L, 200L)) # (batch_size, query_elements, query_depth)
key = tf$random$uniform(list(32L, 15L, 300L)) # (batch_size, key_elements, key_depth)
value = tf$random$uniform(list(32L, 15L, 400L)) # (batch_size, key_elements, value_depth)
attention = mha(list(query, key, value)) # (batch_size, query_elements, value_depth)

# If `value` is not given then internally `value = key` will be used:
mha = layer_multi_head_attention(head_size=128, num_heads=128)
query = tf$random$uniform(list(32L, 20L, 200L)) # (batch_size, query_elements, query_depth)
key = tf$random$uniform(list(32L, 15L, 300L)) # (batch_size, key_elements, key_depth)
attention = mha(list(query, key)) # (batch_size, query_elements, value_depth)

## End(Not run)
```

---

layer\_nas\_cell

*Neural Architecture Search (NAS) recurrent network cell.*

---

**Description**

Neural Architecture Search (NAS) recurrent network cell.

**Usage**

```
layer_nas_cell(
  object,
  units,
  projection = NULL,
  use_bias = FALSE,
  kernel_initializer = "glorot_uniform",
  recurrent_initializer = "glorot_uniform",
  projection_initializer = "glorot_uniform",
  bias_initializer = "zeros",
  ...
)
```



**Arguments**

object	Model or layer object
units	int, The number of units in the NAS cell.
projection	(optional) int, The output dimensionality for the projection matrices. If None, no projection is performed.
use_bias	(optional) bool, If 'TRUE' then use biases within the cell. This is 'FALSE' by default.
kernel_initializer	Initializer for kernel weight.
recurrent_initializer	Initializer for recurrent kernel weight.
projection_initializer	Initializer for projection weight, used when projection is not 'NULL'.
bias_initializer	Initializer for bias, used when 'use_bias' is 'TRUE'.
...	Additional keyword arguments.

**Details**

This implements the recurrent cell from the paper: <https://arxiv.org/abs/1611.01578> Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning" Proc. ICLR 2017. The class uses an optional projection layer.

**Value**

A tensor

---

layer\_norm\_lstm\_cell *LSTM cell with layer normalization and recurrent dropout.*

---

**Description**

LSTM cell with layer normalization and recurrent dropout.

**Usage**

```
layer_norm_lstm_cell(
    object,
    units,
    activation = "tanh",
    recurrent_activation = "sigmoid",
    use_bias = TRUE,
    kernel_initializer = "glorot_uniform",
    recurrent_initializer = "orthogonal",
    bias_initializer = "zeros",
```

```

unit_forget_bias = TRUE,
kernel_regularizer = NULL,
recurrent_regularizer = NULL,
bias_regularizer = NULL,
kernel_constraint = NULL,
recurrent_constraint = NULL,
bias_constraint = NULL,
dropout = 0,
recurrent_dropout = 0,
norm_gamma_initializer = "ones",
norm_beta_initializer = "zeros",
norm_epsilon = 0.001,
...
)

```

### Arguments

<code>object</code>	Model or layer object
<code>units</code>	Positive integer, dimensionality of the output space.
<code>activation</code>	Activation function to use. Default: hyperbolic tangent ('tanh'). If you pass 'NULL', no activation is applied (ie. "linear" activation: 'a(x) = x').
<code>recurrent_activation</code>	Activation function to use for the recurrent step. Default: sigmoid ('sigmoid'). If you pass 'NULL', no activation is applied (ie. "linear" activation: 'a(x) = x').
<code>use_bias</code>	Boolean, whether the layer uses a bias vector.
<code>kernel_initializer</code>	Initializer for the 'kernel' weights matrix, used for the linear transformation of the inputs.
<code>recurrent_initializer</code>	Initializer for the 'recurrent_kernel' weights matrix, used for the linear transformation of the recurrent state.
<code>bias_initializer</code>	Initializer for the bias vector.
<code>unit_forget_bias</code>	Boolean. If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force 'bias_initializer="zeros"'. This is recommended in [Jozefowicz et al.]( <a href="http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf">http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf</a> )
<code>kernel_regularizer</code>	Regularizer function applied to the 'kernel' weights matrix.
<code>recurrent_regularizer</code>	Regularizer function applied to the 'recurrent_kernel' weights matrix.
<code>bias_regularizer</code>	Regularizer function applied to the bias vector.
<code>kernel_constraint</code>	Constraint function applied to the 'kernel' weights matrix.
<code>recurrent_constraint</code>	Constraint function applied to the 'recurrent_kernel' weights matrix.

bias_constraint	Constraint function applied to the bias vector.
dropout	Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
recurrent_dropout	Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.
norm_gamma_initializer	Initializer for the layer normalization gain initial value.
norm_beta_initializer	Initializer for the layer normalization shift initial value.
norm_epsilon	Float, the epsilon value for normalization layers.
...	List, the other keyword arguments for layer creation.

### Details

This class adds layer normalization and recurrent dropout to a LSTM unit. Layer normalization implementation is based on: <https://arxiv.org/abs/1607.06450>. "Layer Normalization" Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton and is applied before the internal nonlinearities. Recurrent dropout is based on: <https://arxiv.org/abs/1603.05118> "Recurrent Dropout without Memory Loss" Stanislaw Semeniuta, Aliaksei Severyn, Erhardt Barth.

### Value

A tensor

---

layer\_poincare\_normalize

*Project into the Poincare ball with norm  $\leq 1.0 - \epsilon$*

---

### Description

Project into the Poincare ball with norm  $\leq 1.0 - \epsilon$

### Usage

```
layer_poincare_normalize(object, axis = 1, epsilon = 1e-05, ...)
```

### Arguments

object	Model or layer object
axis	Axis along which to normalize. A scalar or a vector of integers.
epsilon	A small deviation from the edge of the unit sphere for numerical stability.
...	additional parameters to pass

**Details**

[https://en.wikipedia.org/wiki/Poincare\\_ball\\_model](https://en.wikipedia.org/wiki/Poincare_ball_model) Used in Poincare Embeddings for Learning Hierarchical Representations Maximilian Nickel, Douwe Kiela <https://arxiv.org/pdf/1705.08039.pdf>  
For a 1-D tensor with axis = 0, computes

**Value**

A tensor

---

layer_sparsemax	<i>Sparsemax activation function</i>
-----------------	--------------------------------------

---

**Description**

Sparsemax activation function

**Usage**

```
layer_sparsemax(object, axis = -1, ...)
```

**Arguments**

object	Model or layer object
axis	Integer, axis along which the sparsemax normalization is applied.
...	additional parameters to pass

**Details**

The output shape is the same as the input shape. <https://arxiv.org/abs/1602.02068>

**Value**

A tensor

**Examples**

```
## Not run:
model = keras_model_sequential() %>%
  layer_conv_2d(filters = 10, kernel_size = c(3,3), input_shape = c(28,28,1),
               activation = activation_gelu) %>%
  layer_sparsemax()

## End(Not run)
```

---

layer\_weight\_normalization  
*Weight Normalization layer*

---

### Description

Weight Normalization layer

### Usage

```
layer_weight_normalization(object, layer, data_init = TRUE, ...)
```

### Arguments

object	Model or layer object
layer	a layer instance.
data_init	If 'TRUE' use data dependent variable initialization
...	additional parameters to pass

### Details

This wrapper reparameterizes a layer by decoupling the weight's magnitude and direction. This speeds up convergence by improving the conditioning of the optimization problem. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks: <https://arxiv.org/abs/1602.07868> Tim Salimans, Diederik P. Kingma (2016) WeightNormalization wrapper works for keras and tf layers.

### Value

A tensor

### Examples

```
## Not run:  
  
model= keras_model_sequential() %>%  
  layer_weight_normalization(  
    layer_conv_2d(filters = 2, kernel_size = 2, activation = 'relu'),  
    input_shape = c(32L, 32L, 3L))  
model  
  
## End(Not run)
```

---

lookahead_mechanism	<i>Lookahead mechanism</i>
---------------------	----------------------------

---

### Description

Lookahead mechanism

### Usage

```
lookahead_mechanism(
    optimizer,
    sync_period = 6,
    slow_step_size = 0.5,
    name = "Lookahead",
    clipnorm = NULL,
    clipvalue = NULL,
    decay = NULL,
    lr = NULL
)
```

### Arguments

optimizer	The original optimizer that will be used to compute and apply the gradients.
sync_period	An integer. The synchronization period of lookahead. Enable lookahead mechanism by setting it with a positive value.
slow_step_size	A floating point value. The ratio for updating the slow weights.
name	Optional name for the operations created when applying gradients. Defaults to "Lookahead".
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

### Details

The mechanism is proposed by Michael R. Zhang et.al in the paper [Lookahead Optimizer: k steps forward, 1 step back](<https://arxiv.org/abs/1907.08610v1>). The optimizer iteratively updates two sets of weights: the search directions for weights are chosen by the inner optimizer, while the "slow weights" are updated each k steps based on the directions of the "fast weights" and the two sets of weights are synchronized. This method improves the learning stability and lowers the variance of its inner optimizer.

**Value**

Optimizer for use with 'keras::compile()'

**Examples**

```
## Not run:

opt = tf.keras.optimizers.SGD(learning_rate)
opt = lookahead_mechanism(opt)

## End(Not run)
```

---

loss_contrastive	<i>Contrastive loss</i>
------------------	-------------------------

---

**Description**

Computes the contrastive loss between 'y\_true' and 'y\_pred'.

**Usage**

```
loss_contrastive(
    margin = 1,
    reduction = tf.keras.losses.Reduction.SUM_OVER_BATCH_SIZE,
    name = "contrastive_loss"
)
```

**Arguments**

margin	Float, margin term in the loss definition. Default value is 1.0.
reduction	(Optional) Type of <code>tf.keras.losses.Reduction</code> to apply. Default value is <code>SUM_OVER_BATCH_SIZE</code> .
name	(Optional) name for the loss.

**Details**

This loss encourages the embedding to be close to each other for the samples of the same label and the embedding to be far apart at least by the margin constant for the samples of different labels. The euclidean distances 'y\_pred' between two embedding matrices 'a' and 'b' with shape [batch\_size, hidden\_size] can be computed as follows: `python # y_pred = '\sqrt' ('\sum_i' (a[:, i] - b[:, i])^2)` `y_pred = tf.linalg.norm(a - b, axis=1)` See: <http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf>

**Value**

contrastive\_loss: 1-D float 'Tensor' with shape [batch\_size].

## Examples

```
## Not run:
keras_model_sequential() %>%
  layer_dense(4, input_shape = c(784)) %>%
  compile(
    optimizer = 'sgd',
    loss=loss_contrastive(),
    metrics='accuracy'
  )

## End(Not run)
```

---

loss\_giou

*Implements the GIoU loss function.*

---

## Description

GIoU loss was first introduced in the [Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression](<https://giou.stanford.edu/GIoU.pdf>). GIoU is an enhancement for models which use IoU in object detection.

## Usage

```
loss_giou(
  mode = "giou",
  reduction = tf$keras$losses$Reduction$AUTO,
  name = "giou_loss"
)
```

## Arguments

mode	one of ['giou', 'iou'], decided to calculate GIoU or IoU loss.
reduction	(Optional) Type of tf\$keras\$losses\$Reduction to apply. Default value is SUM_OVER_BATCH_SIZE.
name	A name for the operation (optional).

## Value

GIoU loss float 'Tensor'.



---

loss_hamming	<i>Hamming loss</i>
--------------	---------------------

---

**Description**

Computes hamming loss.

**Usage**

```
loss_hamming(
    mode,
    name = "hamming_loss",
    threshold = NULL,
    dtype = tf$float32,
    ...
)
```

**Arguments**

mode	multi-class or multi-label
name	(optional) String name of the metric instance.
threshold	Elements of 'y_pred' greater than threshold are converted to be 1, and the rest 0. If threshold is None, the argmax is converted to 1, and the rest 0.
dtype	(optional) Data type of the metric result. Defaults to 'tf\$float32'.
...	additional arguments that are passed on to function 'fn'.

**Details**

Hamming loss is the fraction of wrong labels to the total number of labels. In multi-class classification, hamming loss is calculated as the hamming distance between 'actual' and 'predictions'. In multi-label classification, hamming loss penalizes only the individual labels.

**Value**

hamming loss: float

**Examples**

```
## Not run:

# multi-class hamming loss
hl = loss_hamming(mode='multiclass', threshold=0.6)
actuals = tf$constant(list(as.integer(c(1, 0, 0, 0)),as.integer(c(0, 0, 1, 0)),
                          as.integer(c(0, 0, 0, 1)),as.integer(c(0, 1, 0, 0))),
                      dtype=tf$float32)
predictions = tf$constant(list(c(0.8, 0.1, 0.1, 0),
                               c(0.2, 0, 0.8, 0),
```

```

                                c(0.05, 0.05, 0.1, 0.8),
                                c(1, 0, 0, 0)),
                                dtype=tf$float32)
hl$update_state(actuals, predictions)
paste('Hamming loss: ', hl$result()$numpy()) # 0.25
# multi-label hamming loss
hl = loss_hamming(mode='multilabel', threshold=0.8)
actuals = tf$constant(list(as.integer(c(1, 0, 1, 0)),as.integer(c(0, 1, 0, 1)),
                           as.integer(c(0, 0, 0,1))), dtype=tf$int32)
predictions = tf$constant(list(c(0.82, 0.5, 0.90, 0),
                               c(0, 1, 0.4, 0.98),
                               c(0.89, 0.79, 0, 0.3)),
                           dtype=tf$float32)
hl$update_state(actuals, predictions)
paste('Hamming loss: ', hl$result()$numpy()) # 0.16666667

## End(Not run)

```

---

loss\_lifted\_struct      *Lifted structured loss*

---

### Description

Computes the lifted structured loss.

### Usage

```
loss_lifted_struct(margin = 1, name = NULL, ...)
```

### Arguments

margin	Float, margin term in the loss definition.
name	Optional name for the op.
...	additional parameters to pass

### Details

The loss encourages the positive distances (between a pair of embeddings with the same labels) to be smaller than any negative distances (between a pair of embeddings with different labels) in the mini-batch in a way that is differentiable with respect to the embedding vectors. See: <https://arxiv.org/abs/1511.06452>

### Value

lifted\_loss: tf\$float32 scalar.

---

loss_npairs	<i>Npairs loss</i>
-------------	--------------------

---

**Description**

Computes the npairs loss between 'y\_true' and 'y\_pred'.

**Usage**

```
loss_npairs(name = "npairs_loss")
```

**Arguments**

name                    Optional name for the op.

**Details**

Npairs loss expects paired data where a pair is composed of samples from the same labels and each pairs in the minibatch have different labels. The loss takes each row of the pair-wise similarity matrix, 'y\_pred', as logits and the remapped multi-class labels, 'y\_true', as labels. The similarity matrix 'y\_pred' between two embedding matrices 'a' and 'b' with shape '[batch\_size, hidden\_size]' can be computed as follows: `“ # y_pred = a * b^T y_pred = tf.matmul(a, b, transpose_a=FALSE, transpose_b=TRUE) “` See: [http://www.nec-labs.com/uploads/images/Department-Images/MediaAnalytics/papers/nips16\\_n](http://www.nec-labs.com/uploads/images/Department-Images/MediaAnalytics/papers/nips16_n)

**Value**

npairs\_loss: float scalar.

---

loss_npairs_multilabel	<i>Npairs multilabel loss</i>
------------------------	-------------------------------

---

**Description**

Computes the npairs loss between multilabel data 'y\_true' and 'y\_pred'.

**Usage**

```
loss_npairs_multilabel(name = "npairs_multilabel_loss")
```

**Arguments**

name                    Optional name for the op.

**Details**

Npairs loss expects paired data where a pair is composed of samples from the same labels and each pairs in the minibatch have different labels. The loss takes each row of the pair-wise similarity matrix, 'y\_pred', as logits and the remapped multi-class labels, 'y\_true', as labels. To deal with multilabel inputs, the count of label intersection is computed as follows: “‘ L<sub>i,j</sub> = | set\_of\_labels\_for(i) ∩ set\_of\_labels\_for(j) | “‘ Each row of the count based label matrix is further normalized so that each row sums to one. 'y\_true' should be a binary indicator for classes. That is, if 'y\_true[i, j] = 1', then 'i'th sample is in 'j'th class; if 'y\_true[i, j] = 0', then 'i'th sample is not in 'j'th class. The similarity matrix 'y\_pred' between two embedding matrices 'a' and 'b' with shape '[batch\_size, hidden\_size]' can be computed as follows: “‘ # y\_pred = a \* b^T y\_pred = tf.matmul(a, b, transpose\_a=FALSE, transpose\_b=TRUE) “‘

**Value**

npairs\_multilabel\_loss: float scalar.

**See**

[http://www.nec-labs.com/uploads/images/Department-Images/MediaAnalytics/papers/nips16\\_npaiometriclearning.pdf](http://www.nec-labs.com/uploads/images/Department-Images/MediaAnalytics/papers/nips16_npaiometriclearning.pdf)

---

loss\_pinball

*Pinball loss*

---

**Description**

Computes the pinball loss between 'y\_true' and 'y\_pred'.

**Usage**

```
loss_pinball(
    tau = 0.5,
    reduction = tf.keras.losses.Reduction.AUTO,
    name = "pinball_loss"
)
```

**Arguments**

tau	(Optional) Float in [0, 1] or a tensor taking values in [0, 1] and shape = [d0,..., dn]. It defines the slope of the pinball loss. In the context of quantile regression, the value of tau determines the conditional quantile level. When tau = 0.5, this amounts to l1 regression, an estimator of the conditional median (0.5 quantile).
reduction	(Optional) Type of tf.keras.losses.Reduction to apply to loss. Default value is AUTO. AUTO indicates that the reduction option will be determined by the usage context. For almost all cases this defaults to SUM_OVER_BATCH_SIZE. When used with tf.distribute.Strategy, outside of built-in training loops such as tf.keras compile and fit, using AUTO or SUM_OVER_BATCH_SIZE will raise an error. Please see <a href="https://www.tensorflow.org/alpha/tutorials/distribute/training_loops">https://www.tensorflow.org/alpha/tutorials/distribute/training_loops</a> for more details on this.

name Optional name for the op.

### Details

$\text{loss} = \text{maximum}(\tau * (y_{\text{true}} - y_{\text{pred}}), (\tau - 1) * (y_{\text{true}} - y_{\text{pred}}))$  In the context of regression this, loss yields an estimator of the tau conditional quantile. See: [https://en.wikipedia.org/wiki/Quantile\\_regression](https://en.wikipedia.org/wiki/Quantile_regression)  
 Usage: `python loss = pinball_loss([0., 0., 1., 1.], [1., 1., 1., 0.], tau=.1) # loss = max(0.1 * (y_true - y_pred), (0.1 - 1) * (y_true - y_pred)) # = (0.9 + 0.9 + 0 + 0.1) / 4 print('Loss: ', loss.numpy()) # Loss: 0.475`

### Value

pinball\_loss: 1-D float 'Tensor' with shape [batch\_size].

pinball\_loss: 1-D float 'Tensor' with shape [batch\_size].

### Usage

```
python_loss = pinball_loss([0., 0., 1., 1.], [1., 1., 1., 0.], tau=.1)
```

### References

- [https://en.wikipedia.org/wiki/Quantile\\_regression](https://en.wikipedia.org/wiki/Quantile_regression) - [https://projecteuclid.org/download/pdfview\\_1/euclid.bj/1297173840](https://projecteuclid.org/download/pdfview_1/euclid.bj/1297173840)

### Examples

```
## Not run:
keras_model_sequential() %>%
  layer_dense(4, input_shape = c(784)) %>%
  compile(
    optimizer = 'sgd',
    loss=loss_pinball(),
    metrics='accuracy'
  )

## End(Not run)
```

---

loss_sequence	<i>Weighted cross-entropy loss for a sequence of logits.</i>
---------------	--

---

### Description

Weighted cross-entropy loss for a sequence of logits.

### Usage

```
loss_sequence(...)
```

**Arguments**

... A list of parameters

**Value**

None

---

loss\_sigmoid\_focal\_crossentropy  
*Sigmoid focal crossentropy loss*

---

**Description**

Sigmoid focal crossentropy loss

**Usage**

```
loss_sigmoid_focal_crossentropy(
    from_logits = FALSE,
    alpha = 0.25,
    gamma = 2,
    reduction = tf$keras$losses$Reduction$NONE,
    name = "sigmoid_focal_crossentropy"
)
```

**Arguments**

`from_logits` If logits are provided then convert the predictions into probabilities

`alpha` balancing factor.

`gamma` modulating factor.

`reduction` (Optional) Type of `tf$keras$losses$Reduction` to apply. Default value is `SUM_OVER_BATCH_SIZE`.

`name` (Optional) name for the loss.

**Value**

Weighted loss float 'Tensor'. If 'reduction' is 'NONE', this has the same shape as 'y\_true'; otherwise, it is scalar.

**Examples**

```
## Not run:
keras_model_sequential() %>%
  layer_dense(4, input_shape = c(784)) %>%
  compile(
    optimizer = 'sgd',
```

```

        loss=loss_sigmoid_focal_crossentropy(),
        metrics='accuracy'
    )

    ## End(Not run)

```

---

loss_sparsemax	<i>Sparsemax loss</i>
----------------	-----------------------

---

### Description

Sparsemax loss function [1].

### Usage

```

loss_sparsemax(
  from_logits = TRUE,
  reduction = tf$keras$losses$Reduction$SUM_OVER_BATCH_SIZE,
  name = "sparsemax_loss"
)

```

### Arguments

from_logits	Whether y_pred is expected to be a logits tensor. Default is True, meaning y_pred is the logits.
reduction	(Optional) Type of tf\$keras\$losses\$Reduction to apply to loss. Default value is SUM_OVER_BATCH_SIZE.
name	Optional name for the op.

### Details

Computes the generalized multi-label classification loss for the sparsemax function. The implementation is a reformulation of the original loss function such that it uses the sparsemax probability output instead of the internal au variable. However, the output is identical to the original loss function. [1]: <https://arxiv.org/abs/1602.02068>

### Value

A ‘Tensor’. Has the same type as ‘logits’.

---

loss_triplet_hard	<i>Triplet hard loss</i>
-------------------	--------------------------

---

**Description**

Computes the triplet loss with hard negative and hard positive mining.

**Usage**

```
loss_triplet_hard(margin = 1, soft = FALSE, name = NULL, ...)
```

**Arguments**

margin	Float, margin term in the loss definition. Default value is 1.0.
soft	Boolean, if set, use the soft margin version. Default value is False.
name	Optional name for the op.
...	additional arguments to pass

**Value**

triplet\_loss: float scalar with dtype of y\_pred.

**Examples**

```
## Not run:
model = keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = 2, padding='same', input_shape=c(28,28,1)) %>%
  layer_max_pooling_2d(pool_size=2) %>%
  layer_flatten() %>%
  layer_dense(256, activation= NULL) %>%
  layer_lambda(f = function(x) tf$math$l2_normalize(x, axis = 1L))

model %>% compile(
  optimizer = optimizer_lazy_adam(),
  # apply triplet semihard loss
  loss = loss_triplet_hard())

## End(Not run)
```



---

loss\_triplet\_semihard *Triplet semihard loss*

---

### Description

Computes the triplet loss with semi-hard negative mining.

### Usage

```
loss_triplet_semihard(margin = 1, name = NULL, ...)
```

### Arguments

margin	Float, margin term in the loss definition. Default value is 1.0.
name	Optional name for the op.
...	additional arguments to pass

### Value

triplet\_loss: float scalar with dtype of y\_pred.

### Examples

```
## Not run:
model = keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = 2, padding='same', input_shape=c(28,28,1)) %>%
  layer_max_pooling_2d(pool_size=2) %>%
  layer_flatten() %>%
  layer_dense(256, activation= NULL) %>%
  layer_lambda(f = function(x) tf$math$l2_normalize(x, axis = 1L))

model %>% compile(
  optimizer = optimizer_lazy_adam(),
  # apply triplet semihard loss
  loss = loss_triplet_semihard())

## End(Not run)
```

---

metrics_f1score	<i>F1Score</i>
-----------------	----------------

---

**Description**

Computes F-1 Score.

**Usage**

```
metrics_f1score(
    num_classes,
    average = NULL,
    threshold = NULL,
    name = "f1_score",
    dtype = tf$float32
)
```

**Arguments**

num_classes	Number of unique classes in the dataset.
average	Type of averaging to be performed on data. Acceptable values are NULL, micro, macro and weighted. Default value is NULL. - None: Scores for each class are returned - micro: True positivies, false positives and false negatives are computed globally. - macro: True positivies, false positives and - false negatives are computed for each class and their unweighted mean is returned. - weighted: Metrics are computed for each class and returns the mean weighted by the number of true instances in each class.
threshold	Elements of y_pred above threshold are considered to be 1, and the rest 0. If threshold is NULL, the argmax is converted to 1, and the rest 0.
name	(optional) String name of the metric instance.
dtype	(optional) Data type of the metric result. Defaults to 'tf\$float32'.

**Details**

It is the harmonic mean of precision and recall. Output range is [0, 1]. Works for both multi-class and multi-label classification.  $F-1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$

**Value**

F-1 Score: float

**Raises**

ValueError: If the 'average' has values other than [NULL, micro, macro, weighted].

**Examples**

```

## Not run:
model = keras_model_sequential() %>%
  layer_dense(units = 10, input_shape = ncol(iris) - 1, activation = activation_list) %>%
  layer_dense(units = 3)

model %>% compile(loss = 'categorical_crossentropy',
                 optimizer = optimizer_radam(),
                 metrics = metrics_f1score(3))

## End(Not run)

```

---

metric\_cohen\_kappa      *Computes Kappa score between two raters*

---

**Description**

Computes Kappa score between two raters

**Usage**

```

metric_cohen_kappa(
  num_classes,
  name = "cohen_kappa",
  weightage = NULL,
  sparse_labels = FALSE,
  regression = FALSE,
  dtype = NULL
)

```

**Arguments**

num_classes	Number of unique classes in your dataset.
name	(optional) String name of the metric instance
weightage	(optional) Weighting to be considered for calculating kappa statistics. A valid value is one of [None, 'linear', 'quadratic']. Defaults to 'NULL'
sparse_labels	(bool) Valid only for multi-class scenario. If True, ground truth labels are expected to be integers and not one-hot encoded
regression	(bool) If set, that means the problem is being treated as a regression problem where you are regressing the predictions. <b>Note:</b> If you are regressing for the values, the the output layer should contain a single unit.
dtype	(optional) Data type of the metric result. Defaults to 'NULL'

### Details

The score lies in the range [-1, 1]. A score of -1 represents complete disagreement between two raters whereas a score of 1 represents complete agreement between the two raters. A score of 0 means agreement by chance.

### Value

Input tensor or list of input tensors.

### Examples

```
## Not run:
model = keras_model_sequential() %>%
  layer_dense(units = 10, input_shape = ncol(iris) - 1, activation = activation_list) %>%
  layer_dense(units = 3)

model %>% compile(loss = 'categorical_crossentropy',
                 optimizer = optimizer_radam(),
                 metrics = metric_cohen_kappa(3))

## End(Not run)
```

---

metric_fbetascore	<i>FBetaScore</i>
-------------------	-------------------

---

### Description

Computes F-Beta score.

### Usage

```
metric_fbetascore(
  num_classes,
  average = NULL,
  beta = 1,
  threshold = NULL,
  name = "fbeta_score",
  dtype = tf$float32,
  ...
)
```

**Arguments**

num_classes	Number of unique classes in the dataset.
average	Type of averaging to be performed on data. Acceptable values are None, micro, macro and weighted. Default value is NULL. micro, macro and weighted. Default value is NULL. - None: Scores for each class are returned - micro: True positivies, false positives and false negatives are computed globally. - macro: True positivies, false positives and - false negatives are computed for each class and their unweighted mean is returned. - weighted: Metrics are computed for each class and returns the mean weighted by the number of true instances in each class.-
beta	Determines the weight of precision and recall in harmonic mean. Determines the weight given to the precision and recall. Default value is 1.
threshold	Elements of y_pred greater than threshold are converted to be 1, and the rest 0. If threshold is None, the argmax is converted to 1, and the rest 0.
name	(optional) String name of the metric instance.
dtype	(optional) Data type of the metric result. Defaults to 'tf\$float32'.
...	additional parameters to pass

**Details**

It is the weighted harmonic mean of precision and recall. Output range is [0, 1]. Works for both multi-class and multi-label classification.  $F\text{-Beta} = (1 + \text{beta}^2) * (\text{prec} * \text{recall}) / ((\text{beta}^2 * \text{prec}) + \text{recall})$

**Value**

F-Beta Score: float

**Raises**

ValueError: If the 'average' has values other than [NULL, micro, macro, weighted].

---

metric\_hamming\_distance

*Hamming distance*

---

**Description**

Computes hamming distance.

**Usage**

```
metric_hamming_distance(actuals, predictions)
```

**Arguments**

actuals	actual value
predictions	predicted value

**Details**

Hamming distance is for comparing two binary strings. It is the number of bit positions in which two bits are different.

**Value**

hamming distance: float

**Examples**

```
## Not run:

actuals = tf$constant(as.integer(c(1, 1, 0, 0, 1, 0, 1, 0, 0, 1)), dtype=tf$int32)
predictions = tf$constant(as.integer(c(1, 0, 0, 0, 1, 0, 0, 1, 0, 1)), dtype=tf$int32)
result = metric_hamming_distance(actuals, predictions)
paste('Hamming distance: ', result$numpy())

## End(Not run)
```

---

metric\_mcc

*MatthewsCorrelationCoefficient*

---

**Description**

Computes the Matthews Correlation Coefficient.

**Usage**

```
metric_mcc(
  num_classes = NULL,
  name = "MatthewsCorrelationCoefficient",
  dtype = tf$float32
)
```

**Arguments**

num_classes	Number of unique classes in the dataset.
name	(Optional) String name of the metric instance.
dtype	(Optional) Data type of the metric result. Defaults to 'tf\$float32'.

**Details**

The statistic is also known as the phi coefficient. The Matthews correlation coefficient (MCC) is used in machine learning as a measure of the quality of binary and multiclass classifications. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The correlation coefficient value of MCC is between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. The statistic is also known as the phi coefficient.  $MCC = (TP * TN) - (FP * FN) / ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))^{(1/2)}$  Usage:

**Value**

Matthews correlation coefficient: float

**Examples**

```
## Not run:

actuals = tf$constant(list(1, 1, 1, 0), dtype=tf$float32)
preds = tf$constant(list(1,0,1,1), dtype=tf$float32)
# Matthews correlation coefficient
mcc = metric_mcc(num_classes=1)
mcc$update_state(actuals, preds)
paste('Matthews correlation coefficient is:', mcc$result())$numpy())
# Matthews correlation coefficient is : -0.3333334

## End(Not run)
```

---

```
metric_multilabel_confusion_matrix
      MultiLabelConfusionMatrix
```

---

**Description**

Computes Multi-label confusion matrix.

**Usage**

```
metric_multilabel_confusion_matrix(
  num_classes,
  name = "Multilabel_confusion_matrix",
  dtype = tf$int32
)
```

**Arguments**

num_classes	Number of unique classes in the dataset.
name	(Optional) String name of the metric instance.
dtype	(Optional) Data type of the metric result. Defaults to 'tf\$int32'.

**Details**

Class-wise confusion matrix is computed for the evaluation of classification. If multi-class input is provided, it will be treated as multilabel data. Consider classification problem with two classes (i.e num\_classes=2). Resultant matrix 'M' will be in the shape of (num\_classes, 2, 2). Every class 'i' has a dedicated 2\*2 matrix that contains: - true negatives for class i in M(0,0) - false positives for class i in M(0,1) - false negatives for class i in M(1,0) - true positives for class i in M(1,1) “python # multilabel confusion matrix y\_true = tf\$constant(list(as.integer(c(1, 0, 1)), as.integer(c(0, 1, 0))), dtype=tf\$int32) y\_pred = tf\$constant(list(as.integer(c(1, 0, 0)), as.integer(c(0, 1, 1))), dtype=tf\$int32) output = metric\_multilabel\_confusion\_matrix(num\_classes=3) output\$update\_state(y\_true, y\_pred) paste('Confusion matrix:', output\$result()) # Confusion matrix: [[[1 0] [0 1]] [[1 0] [0 1]] [[0 1] [1 0]]] # if multiclass input is provided y\_true = tf\$constant(list(as.integer(c(1, 0, 0)), as.integer(c(0, 1, 0))), dtype=tf\$int32) y\_pred = tf\$constant(list(as.integer(c(1, 0, 0)), as.integer(c(0, 0, 1))), dtype=tf\$int32) output = metric\_multilabel\_confusion\_matrix(num\_classes=3) output\$update\_state(y\_true, y\_pred) paste('Confusion matrix:', output\$result()) # Confusion matrix: [[[1 0] [0 1]] [[1 0] [1 0]] [[1 1] [0 0]]] “

**Value**

MultiLabelConfusionMatrix: float

---

metric_rsquare	<i>RSquare This is also called as coefficient of determination. It tells how close are data to the fitted regression line. Highest score can be 1.0 and it indicates that the predictors perfectly accounts for variation in the target. Score 0.0 indicates that the predictors do not account for variation in the target. It can also be negative if the model is worse.</i>
----------------	---

---

**Description**

RSquare

This is also called as coefficient of determination. It tells how close are data to the fitted regression line. Highest score can be 1.0 and it indicates that the predictors perfectly accounts for variation in the target. Score 0.0 indicates that the predictors do not account for variation in the target. It can also be negative if the model is worse.



**Usage**

```
metric_rsquare(
  name = "r_square",
  dtype = tf$float32,
  ...,
  multioutput = "uniform_average"
)
```

**Arguments**

name	(Optional) String name of the metric instance.
dtype	(Optional) Data type of the metric result. Defaults to 'tf\$float32'.
...	additional arguments to pass
multioutput	one of the following: "raw_values", "uniform_average", "variance_weighted"

**Value**

r squared score: float

**Examples**

```
## Not run:

actuals = tf$constant(c(1, 4, 3), dtype=tf$float32)
preds = tf$constant(c(2, 4, 4), dtype=tf$float32)
result = metric_rsquare()
result$update_state(actuals, preds)
paste('R^2 score is: ', r1$result()$numpy()) # 0.57142866

## End(Not run)
```

---

optimizer\_conditional\_gradient  
*Conditional Gradient*

---

**Description**

Conditional Gradient

**Usage**

```
optimizer_conditional_gradient(
  learning_rate,
  lambda_,
  epsilon = 1e-07,
```

```

    use_locking = FALSE,
    name = "ConditionalGradient",
    clipnorm = NULL,
    clipvalue = NULL,
    decay = NULL,
    lr = NULL
)

```

### Arguments

learning_rate	A Tensor or a floating point value, or a schedule that is a <code>tf.keras.optimizers.schedules.LearningRateSchedule</code> . The learning rate.
lambda_	A Tensor or a floating point value. The constraint.
epsilon	A Tensor or a floating point value. A small constant for numerical stability when handling the case of norm of gradient to be zero.
use_locking	If True, use locks for update operations.
name	Optional name prefix for the operations created when applying gradients. Defaults to 'ConditionalGradient'.
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

### Value

Optimizer for use with `'keras::compile()'`

---

`optimizer_decay_adamw` *Optimizer that implements the Adam algorithm with weight decay*

---

### Description

This is an implementation of the AdamW optimizer described in "Decoupled Weight Decay Regularization" by Loshchilov & Hutter (<https://arxiv.org/abs/1711.05101>) ([pdf])(<https://arxiv.org/pdf/1711.05101.pdf>). It computes the update step of `tf.keras.optimizers.Adam` and additionally decays the variable. Note that this is different from adding L2 regularization on the variables to the loss: it regularizes variables with large gradients more than L2 regularization would, which was shown to yield better training loss and generalization error in the paper above.

**Usage**

```
optimizer_decay_adamw(
    weight_decay,
    learning_rate = 0.001,
    beta_1 = 0.9,
    beta_2 = 0.999,
    epsilon = 1e-07,
    amsgrad = FALSE,
    name = "AdamW",
    clipnorm = NULL,
    clipvalue = NULL,
    decay = NULL,
    lr = NULL
)
```

**Arguments**

<code>weight_decay</code>	A Tensor or a floating point value. The weight decay.
<code>learning_rate</code>	A Tensor or a floating point value. The learning rate.
<code>beta_1</code>	A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
<code>beta_2</code>	A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
<code>epsilon</code>	A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper.
<code>amsgrad</code>	boolean. Whether to apply AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and beyond".
<code>name</code>	Optional name for the operations created when applying
<code>clipnorm</code>	is clip gradients by norm.
<code>clipvalue</code>	is clip gradients by value.
<code>decay</code>	is included for backward compatibility to allow time inverse decay of learning rate.
<code>lr</code>	is included for backward compatibility, recommended to use <code>learning_rate</code> instead.

**Value**

Optimizer for use with `'keras::compile()'`

**Examples**

```
## Not run:
step = tf$Variable(0L, trainable = FALSE)
```

```

schedule = tf$optimizers$schedules$PiecewiseConstantDecay(list(c(10000, 15000)),
list(c(1e-0, 1e-1, 1e-2)))
lr = 1e-1 * schedule(step)
wd = lambda: 1e-4 * schedule(step)

## End(Not run)

```

---

optimizer\_decay\_sgdw *Optimizer that implements the Momentum algorithm with weight\_decay*

---

### Description

This is an implementation of the SGDw optimizer described in "Decoupled Weight Decay Regularization" by Loshchilov & Hutter (<https://arxiv.org/abs/1711.05101>) ([pdf])(<https://arxiv.org/pdf/1711.05101.pdf>). It computes the update step of `tf.keras.optimizers.SGD` and additionally decays the variable. Note that this is different from adding L2 regularization on the variables to the loss. Decoupling the weight decay from other hyperparameters (in particular the learning rate) simplifies hyperparameter search. For further information see the documentation of the SGD Optimizer.

### Usage

```

optimizer_decay_sgdw(
  weight_decay,
  learning_rate = 0.001,
  momentum = 0,
  nesterov = FALSE,
  name = "SGDW",
  clipnorm = NULL,
  clipvalue = NULL,
  decay = NULL,
  lr = NULL
)

```

### Arguments

<code>weight_decay</code>	weight decay rate.
<code>learning_rate</code>	float hyperparameter $\geq 0$ . Learning rate.
<code>momentum</code>	float hyperparameter $\geq 0$ that accelerates SGD in the relevant direction and dampens oscillations.
<code>nesterov</code>	boolean. Whether to apply Nesterov momentum.
<code>name</code>	Optional name prefix for the operations created when applying gradients. Defaults to 'SGD'.
<code>clipnorm</code>	is clip gradients by norm.

clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

**Value**

Optimizer for use with 'keras::compile()'

**Examples**

```
## Not run:

step = tf$Variable(0L, trainable = FALSE)
schedule = tf$optimizers$schedules$PiecewiseConstantDecay(list(c(10000, 15000)),
list(c(1e-0, 1e-1, 1e-2)))
lr = 1e-1 * schedule(step)
wd = lambda: 1e-4 * schedule(step)

## End(Not run)
```

---

optimizer_lamb	<i>Layer-wise Adaptive Moments</i>
----------------	------------------------------------

---

**Description**

Layer-wise Adaptive Moments

**Usage**

```
optimizer_lamb(
  learning_rate = 0.001,
  beta_1 = 0.9,
  beta_2 = 0.999,
  epsilon = 1e-06,
  weight_decay_rate = 0,
  exclude_from_weight_decay = NULL,
  exclude_from_layer_adaptation = NULL,
  name = "LAMB",
  clipnorm = NULL,
  clipvalue = NULL,
  decay = NULL,
  lr = NULL
)
```

**Arguments**

learning_rate	A 'Tensor' or a floating point value. or a schedule that is a 'tf.keras.optimizers.schedules.LearningRateSchedule'. The learning rate.
beta_1	A 'float' value or a constant 'float' tensor. The exponential decay rate for the 1st moment estimates.
beta_2	A 'float' value or a constant 'float' tensor. The exponential decay rate for the 2nd moment estimates.
epsilon	A small constant for numerical stability.
weight_decay_rate	weight decay rate.
exclude_from_weight_decay	List of regex patterns of variables excluded from weight decay. Variables whose name contain a substring matching the pattern will be excluded.
exclude_from_layer_adaptation	List of regex patterns of variables excluded from layer adaptation. Variables whose name contain a substring matching the pattern will be excluded.
name	Optional name for the operations created when applying gradients. Defaults to "LAMB".
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

**Value**

Optimizer for use with 'keras::compile()'

**Examples**

```
## Not run:
keras_model_sequential() %>%
  layer_dense(32, input_shape = c(784)) %>%
  compile(
    optimizer = optimizer_lamb(),
    loss='binary_crossentropy',
    metrics='accuracy'
  )

## End(Not run)
```

---

optimizer\_lazy\_adam    *Lazy Adam*

---

## Description

Lazy Adam

## Usage

```
optimizer_lazy_adam(
    learning_rate = 0.001,
    beta_1 = 0.9,
    beta_2 = 0.999,
    epsilon = 1e-07,
    amsgrad = FALSE,
    name = "LazyAdam",
    clipnorm = NULL,
    clipvalue = NULL,
    decay = NULL,
    lr = NULL
)
```

## Arguments

learning_rate	A Tensor or a floating point value. or a schedule that is a <code>tf.keras.optimizers.schedules.LearningRateSchedule</code> . The learning rate.
beta_1	A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
beta_2	A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
epsilon	A small constant for numerical stability. This epsilon is "epsilon hat" in Adam: A Method for Stochastic Optimization. Kingma et al., 2014 (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper.
amsgrad	boolean. Whether to apply AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and beyond". Note that this argument is currently not supported and the argument can only be False.
name	Optional name for the operations created when applying gradients. Defaults to "LazyAdam".
clipnorm	is clip gradients by norm;
clipvalue	is clip gradients by value,
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

**Value**

Optimizer for use with 'keras::compile()'

**Examples**

```
## Not run:
keras_model_sequential() %>%
  layer_dense(32, input_shape = c(784)) %>%
  compile(
    optimizer = optimizer_lazy_adam(),
    loss='binary_crossentropy',
    metrics='accuracy'
  )

## End(Not run)
```

---

optimizer\_moving\_average

*Moving Average*

---

**Description**

Moving Average

**Usage**

```
optimizer_moving_average(
  optimizer,
  sequential_update = TRUE,
  average_decay = 0.99,
  num_updates = NULL,
  name = "MovingAverage",
  clipnorm = NULL,
  clipvalue = NULL,
  decay = NULL,
  lr = NULL
)
```

**Arguments**

**optimizer** str or `tf.keras.optimizers.Optimizer` that will be used to compute and apply gradients.

**sequential\_update** Bool. If False, will compute the moving average at the same time as the model is updated, potentially doing benign data races. If True, will update the moving average after gradient updates.



average_decay	float. Decay to use to maintain the moving averages of trained variables.
num_updates	Optional count of the number of updates applied to variables.
name	Optional name for the operations created when applying gradients. Defaults to "MovingAverage".
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

### Details

Optimizer that computes a moving average of the variables. Empirically it has been found that using the moving average of the trained parameters of a deep network is better than using its trained parameters directly. This optimizer allows you to compute this moving average and swap the variables at save time so that any code outside of the training loop will use by default the average values instead of the original ones.

### Value

Optimizer for use with 'keras::compile()'

### Examples

```
## Not run:

opt = tf$keras$optimizers$SGD(learning_rate)
opt = moving_average(opt)

## End(Not run)
```

---

optimizer_novograd	<i>NovoGrad</i>
--------------------	-----------------

---

### Description

NovoGrad

**Usage**

```
optimizer_novograd(
    learning_rate = 0.001,
    beta_1 = 0.9,
    beta_2 = 0.999,
    epsilon = 1e-07,
    weight_decay = 0,
    grad_averaging = FALSE,
    amsgrad = FALSE,
    name = "NovoGrad",
    clipnorm = NULL,
    clipvalue = NULL,
    decay = NULL,
    lr = NULL
)
```

**Arguments**

learning_rate	A ‘Tensor‘ or a floating point value. or a schedule that is a ‘tf\$keras\$optimizers\$schedules\$LearningRateScheduler’.
beta_1	The learning rate.
beta_2	A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
epsilon	A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
weight_decay	A small constant for numerical stability.
grad_averaging	A floating point value. Weight decay for each param.
amsgrad	determines whether to use Adam style exponential moving averaging for the first order moments.
name	boolean. Whether to apply AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and beyond"
clipnorm	Optional name for the operations created when applying gradients. Defaults to "NovoGrad".
clipvalue	is clip gradients by norm.
decay	is clip gradients by value.
lr	is included for backward compatibility to allow time inverse decay of learning rate.
	is included for backward compatibility, recommended to use learning_rate instead.

**Value**

Optimizer for use with ‘keras::compile()’

**Examples**

```

## Not run:
keras_model_sequential() %>%
  layer_dense(32, input_shape = c(784)) %>%
  compile(
    optimizer = optimizer_novograd(),
    loss='binary_crossentropy',
    metrics='accuracy'
  )

## End(Not run)

```

---

optimizer_radam	<i>Rectified Adam (a.k.a. RAdam)</i>
-----------------	--------------------------------------

---

**Description**

Rectified Adam (a.k.a. RAdam)

**Usage**

```

optimizer_radam(
  learning_rate = 0.001,
  beta_1 = 0.9,
  beta_2 = 0.999,
  epsilon = 1e-07,
  weight_decay = 0,
  amsgrad = FALSE,
  sma_threshold = 5,
  total_steps = 0,
  warmup_proportion = 0.1,
  min_lr = 0,
  name = "RectifiedAdam",
  clipnorm = NULL,
  clipvalue = NULL,
  decay = NULL,
  lr = NULL
)

```

**Arguments**

learning_rate	A ‘Tensor‘ or a floating point value. or a schedule that is a ‘tf\$keras\$optimizers\$schedules\$LearningRateScheduler’.
beta_1	A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.

beta_2	A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
epsilon	A small constant for numerical stability.
weight_decay	A floating point value. Weight decay for each param.
amsgrad	boolean. Whether to apply AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and beyond".
sma_threshold	A float value. The threshold for simple mean average.
total_steps	An integer. Total number of training steps. Enable warmup by setting a positive value.
warmup_proportion	A floating point value. The proportion of increasing steps.
min_lr	A floating point value. Minimum learning rate after warmup.
name	Optional name for the operations created when applying gradients. Defaults to "RectifiedAdam".
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

**Value**

Optimizer for use with 'keras::compile()'

---

optimizer_swa	<i>Stochastic Weight Averaging</i>
---------------	------------------------------------

---

**Description**

Stochastic Weight Averaging

**Usage**

```
optimizer_swa(
    optimizer,
    start_averaging = 0,
    average_period = 10,
    name = "SWA",
    sequential_update = TRUE,
    clipnorm = NULL,
    clipvalue = NULL,
    decay = NULL,
    lr = NULL
)
```

**Arguments**

optimizer	The original optimizer that will be used to compute and apply the gradients.
start_averaging	An integer. Threshold to start averaging using SWA. Averaging only occurs at start_averaging iters, must be $\geq 0$ . If start_averaging = m, the first snapshot will be taken after the mth application of gradients (where the first iteration is iteration 0).
average_period	An integer. The synchronization period of SWA. The averaging occurs every average_period steps. Averaging period needs to be $\geq 1$ .
name	Optional name for the operations created when applying gradients. Defaults to 'SWA'.
sequential_update	Bool. If FALSE, will compute the moving average at the same time as the model is updated, potentially doing benign data races. If True, will update the moving average after gradient updates
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.
decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

**Details**

The Stochastic Weight Averaging mechanism was proposed by Pavel Izmailov et. al in the paper [Averaging Weights Leads to Wider Optima and Better Generalization](<https://arxiv.org/abs/1803.05407>). The optimizer implements averaging of multiple points along the trajectory of SGD. The optimizer expects an inner optimizer which will be used to apply the gradients to the variables and itself computes a running average of the variables every k steps (which generally corresponds to the end of a cycle when a cyclic learning rate is employed). We also allow the specification of the number of steps averaging should first happen after. Let's say, we want averaging to happen every k steps after the first m steps. After step m we'd take a snapshot of the variables and then average the weights appropriately at step m + k, m + 2k and so on. The assign\_average\_vars function can be called at the end of training to obtain the averaged\_weights from the optimizer.

**Value**

Optimizer for use with 'keras::compile()'

**Examples**

```
## Not run:
opt = tf.keras.optimizers.SGD(learning_rate)
opt = optimizer_swa(opt, start_averaging=m, average_perizer=k)

## End(Not run)
```

---

optimizer_yogi	<i>Yogi</i>
----------------	-------------

---

### Description

Yogi

### Usage

```
optimizer_yogi(
  learning_rate = 0.01,
  beta1 = 0.9,
  beta2 = 0.999,
  epsilon = 0.001,
  l1_regularization_strength = 0,
  l2_regularization_strength = 0,
  initial_accumulator_value = 1e-06,
  activation = "sign",
  name = "Yogi",
  clipnorm = NULL,
  clipvalue = NULL,
  decay = NULL,
  lr = NULL
)
```

### Arguments

learning_rate	A Tensor or a floating point value. The learning rate.
beta1	A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
beta2	A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
epsilon	A constant trading off adaptivity and noise.
l1_regularization_strength	A float value, must be greater than or equal to zero.
l2_regularization_strength	A float value, must be greater than or equal to zero.
initial_accumulator_value	The starting value for accumulators. Only positive values are allowed.
activation	Use hard sign or soft tanh to determin sign.
name	Optional name for the operations created when applying gradients. Defaults to "Yogi".
clipnorm	is clip gradients by norm.
clipvalue	is clip gradients by value.

decay	is included for backward compatibility to allow time inverse decay of learning rate.
lr	is included for backward compatibility, recommended to use learning_rate instead.

**Value**

Optimizer for use with `'keras::compile()'`

---

parse_time	<i>Parse time</i>
------------	-------------------

---

**Description**

Parse an input string according to the provided format string into a

**Usage**

```
parse_time(time_string, time_format, output_unit)
```

**Arguments**

time_string	The input time string to be parsed.
time_format	The time format.
output_unit	The output unit of the parsed unix time. Can only be SECOND, MILLISECOND, MICROSECOND, NANOSECOND.

**Details**

Unix time. Parse an input string according to the provided format string into a Unix time, the number of seconds / milliseconds / microseconds / nanoseconds elapsed since January 1, 1970 UTC. Uses `strftime()`-like formatting options, with the same extensions as `FormatTime()`, but with the exceptions that

characters as it can, so the matching data should always be terminated with a non-numeric. consumes exactly four characters, including any sign. Unspecified fields are taken from the default date and time of ... "1970-01-01 00:00:00.0 +0000" For example, parsing a string of "15:45" ( Unix time that represents "1970-01-01 15:45:00.0 +0000". Note that `ParseTime` only heeds the fields year, month, day, hour, minute, (fractional) second, and UTC offset. Other fields, like `weekday` ( ignored in the conversion. Date and time fields that are out-of-range will be treated as errors rather than normalizing them like `'absl::CivilSecond'` does. For example, it is an error to parse the date "Oct 32, 2013" because 32 is out of range. A leap second of ":60" is normalized to ":00" of the following minute with fractional seconds discarded. The following table shows how the given seconds and subseconds will be parsed: "59.x" -> 59.x // exact "60.x" -> 00.0 // normalized "00.x" -> 00.x // exact

**Value**

the number of seconds / milliseconds / microseconds / nanoseconds elapsed since January 1, 1970 UTC.

**Raises**

ValueError: If 'output\_unit' is not a valid value, if parsing 'time\_string' according to 'time\_format' failed.

---

register_all	<i>Register all</i>
--------------	---------------------

---

**Description**

Register TensorFlow Addons' objects in TensorFlow global dictionaries.

**Usage**

```
register_all(keras_objects = TRUE, custom_kernels = TRUE)
```

**Arguments**

**keras\_objects** boolean, 'TRUE' by default. If 'TRUE', register all Keras objects with 'tf.keras.utils.register\_keras\_serializable'. If set to FALSE, doesn't register any Keras objects of Addons in TensorFlow.

**custom\_kernels** boolean, 'TRUE' by default. If 'TRUE', loads all custom kernels of TensorFlow Addons with 'tf.load\_op\_library("path/to/so/file.so")'. Loading the SO files register them automatically. If 'FALSE' doesn't load and register the shared objects files. Not that it might be useful to turn it off if your installation of Addons doesn't work well with custom ops.

**Details**

When loading a Keras model that has a TF Addons' function, it is needed for this function to be known by the Keras deserialization process. There are two ways to do this, either do `tf.keras.models.load_model("my_model.tf", custom_objects=list("LAMB": tfaddons::optimizer_lamb))` or you can do: `python register_all() tf.keras.models.load_model("my_model.tf")`. If the model contains custom ops (compiled ops) of TensorFlow Addons, and the graph is loaded with 'tf.saved\_model.load', then custom ops need to be registered before to avoid an error of the type: `tensorflow.python.framework.errors_impl.NotFoundError: Op type not registered '...' in binary running on ... Make sure the Op and Kernel are registered in the binary running in this process.` In this case, the only way to make sure that the ops are registered is to call this function: `register_all() tf.saved_model.load("my_model.tf")`. Note that you can call this function multiple times in the same process, it only has an effect the first time. Afterward, it's just a no-op.

**Value**

None



---

register\_custom\_kernels  
*Register custom kernels*

---

**Description**

Register custom kernels

**Usage**

register\_custom\_kernels(...)

**Arguments**

... parameters to pass

**Value**

None

---

register\_keras\_objects  
*Register keras objects*

---

**Description**

Register keras objects

**Usage**

register\_keras\_objects(...)

**Arguments**

... parameters to pass

**Value**

None

---

safe_cumprod	<i>Safe cumprod</i>
--------------	---------------------

---

**Description**

Computes cumprod of x in logspace using cumsum to avoid underflow.

**Usage**

```
safe_cumprod(x, ...)
```

**Arguments**

x	Tensor to take the cumulative product of.
...	Passed on to cumsum; these are identical to those in cumprod

**Details**

The cumprod function and its gradient can result in numerical instabilities when its argument has very small and/or zero values. As long as the argument is all positive, we can instead compute the cumulative product as  $\exp(\text{cumsum}(\log(x)))$ . This function can be called identically to `tf$cumprod`.

**Value**

Cumulative product of x.

---

sampler	<i>Sampler</i>
---------	----------------

---

**Description**

Interface for implementing sampling in seq2seq decoders.

**Usage**

```
sampler(...)
```

**Arguments**

...	parametr to pass batch_size, initialize, next_inputs, sample, sample_ids_dtype, sample_ids_shape
-----	--

**Value**

None

---

sampler_custom	<i>Base abstract class that allows the user to customize sampling.</i>
----------------	--

---

**Description**

Base abstract class that allows the user to customize sampling.

**Usage**

```
sampler_custom(
    initialize_fn,
    sample_fn,
    next_inputs_fn,
    sample_ids_shape = NULL,
    sample_ids_dtype = NULL
)
```

**Arguments**

`initialize_fn` callable that returns (finished, next\_inputs) for the first iteration.

`sample_fn` callable that takes (time, outputs, state) and emits tensor sample\_ids.

`next_inputs_fn` callable that takes (time, outputs, state, sample\_ids) and emits (finished, next\_inputs, next\_state).

`sample_ids_shape` Either a list of integers, or a 1-D Tensor of type int32, the shape of each value in the sample\_ids batch. Defaults to a scalar.

`sample_ids_dtype` The dtype of the sample\_ids tensor. Defaults to int32.

**Value**

None

---

sampler_greedy_embedding	<i>Greedy Embedding Sampler</i>
--------------------------	---------------------------------

---

**Description**

A sampler for use during inference.

**Usage**

```
sampler_greedy_embedding(embedding_fn = NULL)
```

**Arguments**

`embedding_fn` A optional callable that takes a vector tensor of ids (argmax ids), or the params argument for `embedding_lookup`. The returned tensor will be passed to the decoder input. Default to use `tf.nn.embedding_lookup`.

**Details**

Uses the argmax of the output (treated as logits) and passes the result through an embedding layer to get the next input.

**Value**

None

---

`sampler_inference`      *Inference Sampler*

---

**Description**

Inference Sampler

**Usage**

```
sampler_inference(
    sample_fn,
    sample_shape,
    sample_dtype = tf.int32,
    end_fn,
    next_inputs_fn = NULL,
    ...
)
```

**Arguments**

`sample_fn` A callable that takes outputs and emits tensor `sample_ids`.

`sample_shape` Either a list of integers, or a 1-D Tensor of type `int32`, the shape of the each sample in the batch returned by `sample_fn`.

`sample_dtype` the dtype of the sample returned by `sample_fn`.

`end_fn` A callable that takes `sample_ids` and emits a bool vector shaped `[batch_size]` indicating whether each sample is an end token.

`next_inputs_fn` (Optional) A callable that takes `sample_ids` and returns the next batch of inputs. If not provided, `sample_ids` is used as the next batch of inputs.

... A list that contains other common arguments for layer creation.

**Details**

A helper to use during inference with a custom sampling function.

**Value**

None

---

sampler\_sample\_embedding  
*Sample Embedding Sampler*

---

**Description**

A sampler for use during inference.

**Usage**

```
sampler_sample_embedding(  
    embedding_fn = NULL,  
    softmax_temperature = NULL,  
    seed = NULL  
)
```

**Arguments**

`embedding_fn` (Optional) A callable that takes a vector tensor of ids (argmax ids), or the params argument for `embedding_lookup`. The returned tensor will be passed to the decoder input.

`softmax_temperature` (Optional) float32 scalar, value to divide the logits by before computing the softmax. Larger values (above 1.0) result in more random samples, while smaller values push the sampling distribution towards the argmax. Must be strictly greater than 0. Defaults to 1.0.

`seed` (Optional) The sampling seed.

**Details**

Uses sampling (from a distribution) instead of argmax and passes the result through an embedding layer to get the next input.

**Value**

None

---

sampler\_scheduled\_embedding\_training

*A training sampler that adds scheduled sampling*

---

### Description

A training sampler that adds scheduled sampling

### Usage

```
sampler_scheduled_embedding_training(
    sampling_probability,
    embedding_fn = NULL,
    time_major = FALSE,
    seed = NULL,
    scheduling_seed = NULL
)
```

### Arguments

sampling_probability	A float32 0-D or 1-D tensor: the probability of sampling categorically from the output ids instead of reading directly from the inputs.
embedding_fn	A callable that takes a vector tensor of ids (argmax ids), or the params argument for embedding_lookup.
time_major	bool. Whether the tensors in inputs are time major. If 'FALSE' (default), they are assumed to be batch major.
seed	The sampling seed.
scheduling_seed	The schedule decision rule sampling seed.

### Value

Returns -1s for sample\_ids where no sampling took place; valid sample id values elsewhere.

---

sampler\_scheduled\_output\_training

*Scheduled Output Training Sampler*

---

### Description

A training sampler that adds scheduled sampling directly to outputs.

**Usage**

```
sampler_scheduled_output_training(
    sampling_probability,
    time_major = FALSE,
    seed = NULL,
    next_inputs_fn = NULL
)
```

**Arguments**

`sampling_probability` A float32 scalar tensor: the probability of sampling from the outputs instead of reading directly from the inputs.

`time_major` bool. Whether the tensors in inputs are time major. If False (default), they are assumed to be batch major.

`seed` The sampling seed.

`next_inputs_fn` (Optional) callable to apply to the RNN outputs to create the next input when sampling. If None (default), the RNN outputs will be used as the next inputs.

**Value**

FALSE for `sample_ids` where no sampling took place; TRUE elsewhere.

---

<code>sampler_training</code>	<i>A Sampler for use during training.</i>
-------------------------------	---

---

**Description**

Only reads inputs.

**Usage**

```
sampler_training(time_major = FALSE)
```

**Arguments**

`time_major` bool. Whether the tensors in inputs are time major. If 'FALSE' (default), they are assumed to be batch major.

**Value**

None

---

sample\_bernoulli      *Bernoulli sample*

---

**Description**

Samples from Bernoulli distribution.

**Usage**

```
sample_bernoulli(  
  probs = NULL,  
  logits = NULL,  
  dtype = tf$int32,  
  sample_shape = list(),  
  seed = NULL  
)
```

**Arguments**

probs	probabilities
logits	logits
dtype	the data type
sample_shape	a list/vector of integers
seed	integer, random seed

**Value**

a Tensor

---

sample\_categorical      *Categorical sample*

---

**Description**

Samples from categorical distribution.

**Usage**

```
sample_categorical(  
  logits,  
  dtype = tf$int32,  
  sample_shape = list(),  
  seed = NULL  
)
```



**Arguments**

logits	logits
dtype	dtype
sample_shape	the shape of sample
seed	random seed: integer

**Value**

a Tensor

---

skip_gram_sample	<i>Skip gram sample</i>
------------------	-------------------------

---

**Description**

Generates skip-gram token and label paired Tensors from the input

**Usage**

```
skip_gram_sample(
    input_tensor,
    min_skips = 1,
    max_skips = 5,
    start = 0,
    limit = -1,
    emit_self_as_target = FALSE,
    vocab_freq_table = NULL,
    vocab_min_count = NULL,
    vocab_subsampling = NULL,
    corpus_size = NULL,
    batch_size = NULL,
    batch_capacity = NULL,
    seed = NULL,
    name = NULL
)
```

**Arguments**

input_tensor	A rank-1 'Tensor' from which to generate skip-gram candidates.
min_skips	'int' or scalar 'Tensor' specifying the minimum window size to randomly use for each token. Must be $\geq 0$ and $\leq$ 'max_skips'. If 'min_skips' and 'max_skips' are both 0, the only label outputted will be the token itself when 'emit_self_as_target = TRUE' - or no output otherwise.
max_skips	'int' or scalar 'Tensor' specifying the maximum window size to randomly use for each token. Must be $\geq 0$ .

start	'int' or scalar 'Tensor' specifying the position in 'input_tensor' from which to start generating skip-gram candidates.
limit	'int' or scalar 'Tensor' specifying the maximum number of elements in 'input_tensor' to use in generating skip-gram candidates. -1 means to use the rest of the 'Tensor' after 'start'.
emit_self_as_target	'bool' or scalar 'Tensor' specifying whether to emit each token as a label for itself.
vocab_freq_table	(Optional) A lookup table (subclass of 'lookup.InitializableLookupTableBase') that maps tokens to their raw frequency counts. If specified, any token in 'input_tensor' that is not found in 'vocab_freq_table' will be filtered out before generating skip-gram candidates. While this will typically map to integer raw frequency counts, it could also map to float frequency proportions. 'vocab_min_count' and 'corpus_size' should be in the same units as this.
vocab_min_count	(Optional) 'int', 'float', or scalar 'Tensor' specifying minimum frequency threshold (from 'vocab_freq_table') for a token to be kept in 'input_tensor'. If this is specified, 'vocab_freq_table' must also be specified - and they should both be in the same units.
vocab_subsampling	(Optional) 'float' specifying frequency proportion threshold for tokens from 'input_tensor'. Tokens that occur more frequently (based on the ratio of the token's 'vocab_freq_table' value to the 'corpus_size') will be randomly down-sampled. Reasonable starting values may be around 1e-3 or 1e-5. If this is specified, both 'vocab_freq_table' and 'corpus_size' must also be specified. See Eq. 5 in <a href="http://arxiv.org/abs/1310.4546">http://arxiv.org/abs/1310.4546</a> for more details.
corpus_size	(Optional) 'int', 'float', or scalar 'Tensor' specifying the total number of tokens in the corpus (e.g., sum of all the frequency counts of 'vocab_freq_table'). Used with 'vocab_subsampling' for down-sampling frequently occurring tokens. If this is specified, 'vocab_freq_table' and 'vocab_subsampling' must also be specified.
batch_size	(Optional) 'int' specifying batch size of returned 'Tensors'.
batch_capacity	(Optional) 'int' specifying batch capacity for the queue used for batching returned 'Tensors'. Only has an effect if 'batch_size' > 0. Defaults to 100 * 'batch_size' if not specified.
seed	(Optional) 'int' used to create a random seed for window size and subsampling. See 'set_random_seed' docs for behavior.
name	(Optional) A 'string' name or a name scope for the operations.

### Details

tensor. Generates skip-gram ("token", "label") pairs using each element in the rank-1 'input\_tensor' as a token. The window size used for each token will be randomly selected from the range specified by '[min\_skips, max\_skips]', inclusive. See <https://arxiv.org/abs/1301.3781> for more details about skip-gram. For example, given 'input\_tensor = ["the", "quick", "brown", "fox", "jumps"]', 'min\_skips = 1', 'max\_skips = 2', 'emit\_self\_as\_target = FALSE', the output '(tokens, labels)'

pairs for the token "quick" will be randomly selected from either `(tokens=["quick", "quick"], labels=["the", "brown"])` for 1 skip, or `(tokens=["quick", "quick", "quick"], labels=["the", "brown", "fox"])` for 2 skips. If `emit_self_as_target = TRUE`, each token will also be emitted as a label for itself. From the previous example, the output will be either `(tokens=["quick", "quick", "quick"], labels=["the", "quick", "brown"])` for 1 skip, or `(tokens=["quick", "quick", "quick", "quick"], labels=["the", "quick", "brown", "fox"])` for 2 skips. The same process is repeated for each element of `input_tensor` and concatenated together into the two output rank-1 `Tensors` (one for all the tokens, another for all the labels). If `vocab_freq_table` is specified, tokens in `input_tensor` that are not present in the vocabulary are discarded. Tokens whose frequency counts are below `vocab_min_count` are also discarded. Tokens whose frequency proportions in the corpus exceed `vocab_subsampling` may be randomly down-sampled. See Eq. 5 in <http://arxiv.org/abs/1310.4546> for more details about subsampling. Due to the random window sizes used for each token, the lengths of the outputs are non-deterministic, unless `batch_size` is specified to batch the outputs to always return `Tensors` of length `batch_size`.

### Value

A `list` containing (token, label) `Tensors`. Each output `Tensor` is of rank-1 and has the same type as `input_tensor`. The `Tensors` will be of length `batch_size`; if `batch_size` is not specified, they will be of random length, though they will be in sync with each other as long as they are evaluated together.

### Raises

`ValueError`: If `vocab_freq_table` is not provided, but `vocab_min_count`, `vocab_subsampling`, or `corpus_size` is specified. If `vocab_subsampling` and `corpus_size` are not both present or both absent.

---

skip\_gram\_sample\_with\_text\_vocab  
*Skip gram sample with text vocab*

---

### Description

Skip-gram sampling with a text vocabulary file.

### Usage

```
skip_gram_sample_with_text_vocab(
  input_tensor,
  vocab_freq_file,
  vocab_token_index = 0,
  vocab_token_dtype = tf$string,
  vocab_freq_index = 1,
  vocab_freq_dtype = tf$float64,
  vocab_delimiter = ",",
  vocab_min_count = NULL,
```

```

vocab_subsampling = NULL,
corpus_size = NULL,
min_skips = 1,
max_skips = 5,
start = 0,
limit = -1,
emit_self_as_target = FALSE,
batch_size = NULL,
batch_capacity = NULL,
seed = NULL,
name = NULL
)

```

### Arguments

`input_tensor` A rank-1 ‘Tensor’ from which to generate skip-gram candidates.

`vocab_freq_file` ‘string’ specifying full file path to the text vocab file.

`vocab_token_index` ‘int’ specifying which column in the text vocab file contains the tokens.

`vocab_token_dtype` ‘DType’ specifying the format of the tokens in the text vocab file.

`vocab_freq_index` ‘int’ specifying which column in the text vocab file contains the frequency counts of the tokens.

`vocab_freq_dtype` ‘DType’ specifying the format of the frequency counts in the text vocab file.

`vocab_delimiter` ‘string’ specifying the delimiter used in the text vocab file.

`vocab_min_count` ‘int’, ‘float’, or scalar ‘Tensor’ specifying minimum frequency threshold (from ‘vocab\_freq\_file’) for a token to be kept in ‘input\_tensor’. This should correspond with ‘vocab\_freq\_dtype’.

`vocab_subsampling` (Optional) ‘float’ specifying frequency proportion threshold for tokens from ‘input\_tensor’. Tokens that occur more frequently will be randomly down-sampled. Reasonable starting values may be around 1e-3 or 1e-5. See Eq. 5 in <http://arxiv.org/abs/1310.4546> for more details.

`corpus_size` (Optional) ‘int’, ‘float’, or scalar ‘Tensor’ specifying the total number of tokens in the corpus (e.g., sum of all the frequency counts of ‘vocab\_freq\_file’). Used with ‘vocab\_subsampling’ for down-sampling frequently occurring tokens. If this is specified, ‘vocab\_freq\_file’ and ‘vocab\_subsampling’ must also be specified. If ‘corpus\_size’ is needed but not supplied, then it will be calculated from ‘vocab\_freq\_file’. You might want to supply your own value if you have already eliminated infrequent tokens from your vocabulary files (where frequency



---

tfaddons_version	<i>Version of TensorFlow SIG Addons</i>
------------------	---

---

**Description**

Get the current version of TensorFlow SIG Addons

**Usage**

```
tfaddons_version()
```

**Value**

prints the version.

---

tile_batch	<i>Tile batch</i>
------------	-------------------

---

**Description**

Tile the batch dimension of a (possibly nested structure of) tensor(s)

**Usage**

```
tile_batch(t, multiplier, name = NULL)
```

**Arguments**

t	'Tensor' shaped '[batch_size, ...]'.
multiplier	Python int.
name	Name scope for any created operations.

**Details**

t. For each tensor t in a (possibly nested structure) of tensors, this function takes a tensor t shaped '[batch\_size, s0, s1, ...]' composed of minibatch entries 't[0], ..., t[batch\_size - 1]' and tiles it to have a shape '[batch\_size \* multiplier, s0, s1, ...]' composed of minibatch entries 't[0], t[0], ..., t[1], t[1], ...' where each minibatch entry is repeated 'multiplier' times.

**Value**

A (possibly nested structure of) 'Tensor' shaped '[batch\_size \* multiplier, ...]'.

**Raises**

ValueError: if tensor(s) 't' do not have a statically known rank or the rank is < 1.

---

viterbi_decode	<i>Viterbi decode</i>
----------------	-----------------------

---

**Description**

Decode the highest scoring sequence of tags outside of TensorFlow.

**Usage**

```
viterbi_decode(score, transition_params)
```

**Arguments**

score                   A [seq\_len, num\_tags] matrix of unary potentials.  
transition\_params       A [num\_tags, num\_tags] matrix of binary potentials.

**Details**

This should only be used at test time.

**Value**

viterbi: A [seq\_len] list of integers containing the highest scoring tag indices. viterbi\_score: A float containing the score for the Viterbi sequence.

# Index

activation\_gelu, 5  
activation\_hardshrink, 6  
activation\_light, 7  
activation\_mish, 7  
activation\_rrelu, 8  
activation\_softshrink, 9  
activation\_sparsemax, 10  
activation\_tanhshrink, 10  
attention\_bahdanau, 11  
attention\_bahdanau\_monotonic, 12  
attention\_luong, 13  
attention\_luong\_monotonic, 15  
attention\_monotonic, 16  
attention\_wrapper, 17  
attention\_wrapper\_state, 19

callback\_average\_model\_checkpoint, 20  
callback\_time\_stopping, 21  
callback\_tqdm\_progress\_bar, 22  
crf\_binary\_score, 23  
crf\_decode, 24  
crf\_decode\_backward, 24  
crf\_decode\_forward, 25  
crf\_forward, 25  
crf\_log\_likelihood, 26  
crf\_log\_norm, 27  
crf\_multitag\_sequence\_score, 27  
crf\_sequence\_score, 28  
crf\_unary\_score, 29

decode\_dynamic, 34  
decoder, 29  
decoder\_base, 30  
decoder\_basic, 30  
decoder\_basic\_output, 31  
decoder\_beam\_search, 31  
decoder\_beam\_search\_output, 32  
decoder\_beam\_search\_state, 33  
decoder\_final\_beam\_search\_output, 34

extend\_with\_decoupled\_weight\_decay, 35

gather\_tree, 36  
gather\_tree\_from\_array, 37

hardmax, 38

img\_adjust\_hsv\_in\_yiq, 38  
img\_angles\_to\_projective\_transforms, 39  
img\_blend, 40  
img\_compose\_transforms, 40  
img\_connected\_components, 41  
img\_cutout, 42  
img\_dense\_image\_warp, 43  
img\_equalize, 44  
img\_euclidean\_dist\_transform, 45  
img\_flat\_transforms\_to\_matrices, 46  
img\_from\_4D, 46  
img\_get\_ndims, 47  
img\_interpolate\_bilinear, 47  
img\_interpolate\_spline, 48  
img\_matrices\_to\_flat\_transforms, 49  
img\_mean\_filter2d, 50  
img\_median\_filter2d, 51  
img\_random\_cutout, 52  
img\_random\_hsv\_in\_yiq, 53  
img\_resampler, 54  
img\_rotate, 55  
img\_sharpness, 56  
img\_shear\_x, 56  
img\_shear\_y, 57  
img\_sparse\_image\_warp, 57  
img\_to\_4D, 58  
img\_transform, 59  
img\_translate, 60  
img\_translate\_xy, 61  
img\_translations\_to\_projective\_transforms, 62  
img\_unwrap, 62



img\_wrap, 63  
install\_tfaddons, 63  
layer\_activation\_gelu, 64  
layer\_correlation\_cost, 64  
layer\_filter\_response\_normalization, 65  
layer\_group\_normalization, 67  
layer\_instance\_normalization, 68  
layer\_maxout, 70  
layer\_multi\_head\_attention, 70  
layer\_nas\_cell, 72  
layer\_norm\_lstm\_cell, 73  
layer\_poincare\_normalize, 75  
layer\_sparsemax, 76  
layer\_weight\_normalization, 77  
lookahead\_mechanism, 78  
loss\_contrastive, 79  
loss\_giou, 80  
loss\_hamming, 81  
loss\_lifted\_struct, 82  
loss\_npairs, 83  
loss\_npairs\_multilabel, 83  
loss\_pinball, 84  
loss\_sequence, 85  
loss\_sigmoid\_focal\_crossentropy, 86  
loss\_sparsemax, 87  
loss\_triplet\_hard, 88  
loss\_triplet\_semihard, 89  
metric\_cohen\_kappa, 91  
metric\_fbetascore, 92  
metric\_hamming\_distance, 93  
metric\_mcc, 94  
metric\_multilabel\_confusion\_matrix, 95  
metric\_rsquare, 96  
metrics\_f1score, 90  
optimizer\_conditional\_gradient, 97  
optimizer\_decay\_adamw, 98  
optimizer\_decay\_sgdw, 100  
optimizer\_lamb, 101  
optimizer\_lazy\_adam, 103  
optimizer\_moving\_average, 104  
optimizer\_novograd, 105  
optimizer\_radam, 107  
optimizer\_swa, 108  
optimizer\_yogi, 110  
parse\_time, 111  
register\_all, 112  
register\_custom\_kernels, 113  
register\_keras\_objects, 113  
safe\_cumprod, 114  
sample\_bernoulli, 120  
sample\_categorical, 120  
sampler, 114  
sampler\_custom, 115  
sampler\_greedy\_embedding, 115  
sampler\_inference, 116  
sampler\_sample\_embedding, 117  
sampler\_scheduled\_embedding\_training, 118  
sampler\_scheduled\_output\_training, 118  
sampler\_training, 119  
skip\_gram\_sample, 121  
skip\_gram\_sample\_with\_text\_vocab, 123  
tfaddons\_version, 126  
tile\_batch, 126  
viterbi\_decode, 127