Module: tf.nn

Defined in `tensorflow/python/ops/nn.py`.

Neural network support.

See the Neural Network guide.

Modules

`rnn_cell` module: Module for constructing RNN Cells.

Functions

`all_candidate_sampler(...)`: Generate the set of all classes.

`atrous_conv2d(...)`: Atrous convolution (a.k.a. convolution with holes or dilated convolution).

`atrous_conv2d_transpose(...)`: The transpose of `atrous_conv2d`.

`avg_pool(...)`: Performs the average pooling on the input.

`avg_pool3d(...)`: Performs 3D average pooling on the input.

`batch_norm_with_global_normalization(...)`: Batch normalization.

`batch_normalization(...)`: Batch normalization.

`bias_add(...)`: Adds `bias` to `value`.

`bidirectional_dynamic_rnn(...)`: Creates a dynamic version of bidirectional recurrent neural network.

`compute_accidental_hits(...)`: Compute the position ids in `sampled_candidates` matching `true_classes`.

`conv1d(...)`: Computes a 1-D convolution given 3-D input and filter tensors.

`conv2d(...)`: Computes a 2-D convolution given 4-D `input` and `filter` tensors.

`conv2d_backprop_filter(...)`: Computes the gradients of convolution with respect to the filter.

`conv2d_backprop_input(...)`: Computes the gradients of convolution with respect to the input.

`conv2d_transpose(...)`: The transpose of `conv2d`.

`conv3d(...)`: Computes a 3-D convolution given 5-D `input` and `filter` tensors.

`conv3d_backprop_filter_v2(...)`: Computes the gradients of 3-D convolution with respect to the filter.

`conv3d_transpose(...)`: The transpose of `conv3d`.

`convolution(...)` : Computes sums of N-D convolutions (actually cross-correlation).

`crelu(...)` : Computes Concatenated ReLU.

`ctc_beam_search_decoder(...)` : Performs beam search decoding on the logits given in input.

`ctc_greedy_decoder(...)` : Performs greedy decoding on the logits given in input (best path).

`ctc_loss(...)` : Computes the CTC (Connectionist Temporal Classification) Loss.

`depthwise_conv2d(...)` : Depthwise 2-D convolution.

`depthwise_conv2d_native(...)` : Computes a 2-D depthwise convolution given 4-D `input` and `filter` tensors.

`depthwise_conv2d_native_backprop_filter(...)` : Computes the gradients of depthwise convolution with respect to the filter.

`depthwise_conv2d_native_backprop_input(...)` : Computes the gradients of depthwise convolution with respect to the input.

`dilation2d(...)` : Computes the grayscale dilation of 4-D `input` and 3-D `filter` tensors.

`dropout(...)` : Computes dropout.

`dynamic_rnn(...)` : Creates a recurrent neural network specified by RNNCell `cell`.

`elu(...)` : Computes exponential linear: `exp(features) - 1` if < 0, `features` otherwise.

`embedding_lookup(...)` : Looks up `ids` in a list of embedding tensors.

`embedding_lookup_sparse(...)` : Computes embeddings for the given ids and weights.

`erosion2d(...)` : Computes the grayscale erosion of 4-D `value` and 3-D `kernel` tensors.

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

`fractional_avg_pool(...)` : Performs fractional average pooling on the input.

`fractional_max_pool(...)` : Performs fractional max pooling on the input.

`fused_batch_norm(...)` : Batch normalization.

`in_top_k(...)` : Says whether the targets are in the top `K` predictions.

`l2_loss(...)` : L2 Loss.

`l2_normalize(...)` : Normalizes along dimension `dim` using an L2 norm.

`leaky_relu(...)` : Compute the Leaky ReLU activation function.

`learned_unigram_candidate_sampler(...)` : Samples a set of classes from a distribution learned during training.

`local_response_normalization(...)` : Local Response Normalization.

`log_poisson_loss(...)` : Computes log Poisson loss given `log_input`.

`log_softmax(...)` : Computes log softmax activations.

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

`lrn(...)` : Local Response Normalization.

`max_pool(...)` : Performs the max pooling on the input.

`max_pool3d(...)` : Performs 3D max pooling on the input.

`max_pool_with_argmax(...)` : Performs max pooling on the input and outputs both max values and indices.

`moments(...)` : Calculate the mean and variance of `x` .

`nce_loss(...)` : Computes and returns the noise-contrastive estimation training loss.

`normalize_moments(...)` : Calculate the mean and variance of based on the sufficient statistics.

`pool(...)` : Performs an N-D pooling operation.

`quantized_avg_pool(...)` : Produces the average pool of the input tensor for quantized types.

`quantized_conv2d(...)` : Computes a 2D convolution given quantized 4D input and filter tensors.

`quantized_max_pool(...)` : Produces the max pool of the input tensor for quantized types.

`quantized_relu_x(...)` : Computes Quantized Rectified Linear X: `min(max(features, 0), max_value)`

`raw_rnn(...)` : Creates an `RNN` specified by RNNCell `cell` and loop function `loop_fn` .

`relu(...)` : Computes rectified linear: `max(features, 0)` .

`relu6(...)` : Computes Rectified Linear 6: `min(max(features, 0), 6)` .

`relu_layer(...)` : Computes Relu(x * weight + biases).

`sampled_softmax_loss(...)` : Computes and returns the sampled softmax training loss.

`selu(...)` : Computes scaled exponential linear: `scale * alpha * (exp(features) - 1)`

`separable_conv2d(...)` : 2-D convolution with separable filters.

`sigmoid(...)` : Computes sigmoid of `x` element-wise.

`sigmoid_cross_entropy_with_logits(...)` : Computes sigmoid cross entropy given `logits` .

`softmax(...)` : Computes softmax activations.

`softmax_cross_entropy_with_logits(...)` : Computes softmax cross entropy between `logits` and `labels` .

`softplus(...)` : Computes softplus: `log(exp(features) + 1)` .

`softsign(...)` : Computes softsign: `features / (abs(features) + 1)` .

`sparse_softmax_cross_entropy_with_logits(...)` : Computes sparse softmax cross entropy between `logits` and `labels` .

`static_bidirectional_rnn(...)` : Creates a bidirectional recurrent neural network.

`static_rnn(...)` : Creates a recurrent neural network specified by RNNCell `cell` .

`static_state_saving_rnn(...)` : RNN that accepts a state saver for time-truncated RNN calculation.

`sufficient_statistics(...)` : Calculate the sufficient statistics for the mean and variance of `x` .

`tanh(...)` : Computes hyperbolic tangent of `x` element-wise.

`top_k(...)` : Finds values and indices of the `k` largest entries for the last dimension.

`uniform_candidate_sampler(...)` : Samples a set of classes using a uniform base distribution.

`weighted_cross_entropy_with_logits(...)` : Computes a weighted cross entropy.

`weighted_moments(...)` : Returns the frequency-weighted mean and variance of `x` .

`with_space_to_batch(...)` : Performs `op` on the space-to-batch representation of `input` .

`xw_plus_b(...)` : Computes matmul(x, weights) + biases.

`zero_fraction(...)` : Returns the fraction of zeros in `value` .
