tf.nn.nce_loss

```
nce_loss(
    weights,
    biases,
    labels,
    inputs,
    num_sampled,
    num_classes,
    num_true=1,
    sampled_values=None,
    remove_accidental_hits=False,
    partition_strategy='mod',
    name='nce_loss'
)
```

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

See the guide: Neural Network > Candidate Sampling

Computes and returns the noise-contrastive estimation training loss.

See Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. Also see our Candidate Sampling Algorithms Reference

A common use case is to use this method for training, and calculate the full sigmoid loss for evaluation or inference. In this case, you must set `partition_strategy="div"` for the two losses to be consistent, as in the following example:

```
if mode == "train":
  loss = tf.nn.nce_loss(
      weights=weights,
      biases=biases,
      labels=labels,
      inputs=inputs,
      ...,
      partition_strategy="div")
elif mode == "eval":
  logits = tf.matmul(inputs, tf.transpose(weights))
  logits = tf.nn.bias_add(logits, biases)
  labels_one_hot = tf.one_hot(labels, n_classes)
  loss = tf.nn.sigmoid_cross_entropy_with_logits(
      labels=labels_one_hot,
      logits=logits)
  loss = tf.reduce_sum(loss, axis=1)
```

★ **Note:** By default this uses a log-uniform (Zipfian) distribution for sampling, so your labels must be sorted in order of decreasing frequency to achieve good results. For more details, see `tf.nn.log_uniform_candidate_sampler`.

★ **Note:** In the case where `num_true` > 1, we assign to each target class the target probability 1 / `num_true` so that the target probabilities sum to 1 per-example.

★ **Note:** It would be useful to allow a variable number of target classes per example. We hope to provide this functionality in a future release. For now, if you have a variable number of target classes, you can pad them out to a constant number by either

repeating them or by padding with an otherwise unused class.

Args:

- `weights`: A `Tensor` of shape `[num_classes, dim]`, or a list of `Tensor` objects whose concatenation along dimension 0 has shape [num_classes, dim]. The (possibly-partitioned) class embeddings.
- `biases`: A `Tensor` of shape `[num_classes]`. The class biases.
- `labels`: A `Tensor` of type `int64` and shape `[batch_size, num_true]`. The target classes.
- `inputs`: A `Tensor` of shape `[batch_size, dim]`. The forward activations of the input network.
- `num_sampled`: An `int`. The number of classes to randomly sample per batch.
- `num_classes`: An `int`. The number of possible classes.
- `num_true`: An `int`. The number of target classes per training example.
- `sampled_values`: a tuple of (`sampled_candidates`, `true_expected_count`, `sampled_expected_count`) returned by a `*_candidate_sampler` function. (if None, we default to `log_uniform_candidate_sampler`)
- `remove_accidental_hits`: A `bool`. Whether to remove "accidental hits" where a sampled class equals one of the target classes. If set to `True`, this is a "Sampled Logistic" loss instead of NCE, and we are learning to generate log-odds instead of log probabilities. See our [Candidate Sampling Algorithms Reference] (https://www.tensorflow.org/extras/candidate_sampling.pdf). Default is False.
- `partition_strategy`: A string specifying the partitioning strategy, relevant if `len(weights) > 1`. Currently `"div"` and `"mod"` are supported. Default is `"mod"`. See `tf.nn.embedding_lookup` for more details.
- `name`: A name for the operation (optional).

Returns:

A `batch_size` 1-D tensor of per-example NCE losses.