tf.layers.batch_normalization

```
batch_normalization(
    inputs,
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer=tf.zeros_initializer(),
    gamma_initializer=tf.ones_initializer(),
    moving_mean_initializer=tf.zeros_initializer(),
    moving_variance_initializer=tf.ones_initializer(),
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    training=False,
    trainable=True,
    name=None,
    reuse=None,
    renorm=False,
    renorm_clipping=None,
    renorm_momentum=0.99,
    fused=None
)
```

Defined in `tensorflow/python/layers/normalization.py`.

See the guide: Reading data > Multiple input pipelines

Functional interface for the batch normalization layer.

Reference: http://arxiv.org/abs/1502.03167

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"

Sergey Ioffe, Christian Szegedy

Note: when training, the moving_mean and moving_variance need to be updated. By default the update ops are placed in `tf.GraphKeys.UPDATE_OPS`, so they need to be added as a dependency to the `train_op`. For example:

```
update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
  train_op = optimizer.minimize(loss)
```

Arguments:

- `inputs`: Tensor input.
- `axis`: Integer, the axis that should be normalized (typically the features axis). For instance, after a `Convolution2D` layer with `data_format="channels_first"`, set `axis=1` in `BatchNormalization`.
- `momentum`: Momentum for the moving average.

- `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. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling can be done by the next layer.
- `beta_initializer`: Initializer for the beta weight.
- `gamma_initializer`: Initializer for the gamma weight.
- `moving_mean_initializer`: Initializer for the moving mean.
- `moving_variance_initializer`: Initializer for the moving variance.
- `beta_regularizer`: Optional regularizer for the beta weight.
- `gamma_regularizer`: Optional regularizer for the gamma weight.
- `beta_constraint`: An optional projection function to be applied to the `beta` weight after being updated by an `Optimizer` (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected variable and must return the projected variable (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.
- `gamma_constraint`: An optional projection function to be applied to the `gamma` weight after being updated by an `Optimizer`.
- `training`: Either a Python boolean, or a TensorFlow boolean scalar tensor (e.g. a placeholder). Whether to return the output in training mode (normalized with statistics of the current batch) or in inference mode (normalized with moving statistics). **NOTE**: make sure to set this parameter correctly, or else your training/inference will not work properly.
- `trainable`: Boolean, if `True` also add variables to the graph collection `GraphKeys.TRAINABLE_VARIABLES` (see tf.Variable).
- `name`: String, the name of the layer.
- `reuse`: Boolean, whether to reuse the weights of a previous layer by the same name.
- `renorm`: Whether to use Batch Renormalization (https://arxiv.org/abs/1702.03275). This adds extra variables during training. The inference is the same for either value of this parameter.
- `renorm_clipping`: A dictionary that may map keys 'rmax', 'rmin', 'dmax' to scalar `Tensors` used to clip the renorm correction. The correction `(r, d)` is used as `corrected_value = normalized_value * r + d`, with `r` clipped to [rmin, rmax], and `d` to [-dmax, dmax]. Missing rmax, rmin, dmax are set to inf, 0, inf, respectively.
- `renorm_momentum`: Momentum used to update the moving means and standard deviations with renorm. Unlike `momentum`, this affects training and should be neither too small (which would add noise) nor too large (which would give stale estimates). Note that `momentum` is still applied to get the means and variances for inference.
- `fused`: if `True`, use a faster, fused implementation if possible. If `None`, use the system recommended implementation.

Returns:

Output tensor.

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**