BatchNormalization layer

`BatchNormalization` class

```
tf.keras.layers.BatchNormalization(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer="zeros",
    gamma_initializer="ones",
    moving_mean_initializer="zeros",
    moving_variance_initializer="ones",
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    **kwargs
)
```

Layer that normalizes its inputs.

Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Importantly, batch normalization works differently during training and during inference.

During training (i.e. when using `fit()` or when calling the layer/model with the argument `training=True`), the layer normalizes its output using the mean and standard deviation of the current batch of inputs. That is to say, for each channel being normalized, the layer returns `gamma * (batch - mean(batch)) / sqrt(var(batch) + epsilon) + beta`, where:

- `epsilon` is small constant (configurable as part of the constructor arguments)
- `gamma` is a learned scaling factor (initialized as 1), which can be disabled by passing `scale=False` to the constructor.
- `beta` is a learned offset factor (initialized as 0), which can be disabled by passing `center=False` to the constructor.

During inference (i.e. when using `evaluate()` or `predict()` or when calling the layer/model with the argument `training=False` (which is the default), the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training. That is to say, it returns `gamma * (batch - self.moving_mean) / sqrt(self.moving_var + epsilon) + beta`.

`self.moving_mean` and `self.moving_var` are non-trainable variables that are updated each time the layer in called in training mode, as such:

- `moving_mean = moving_mean * momentum + mean(batch) * (1 - momentum)`
- `moving_var = moving_var * momentum + var(batch) * (1 - momentum)`

As such, the layer will only normalize its inputs during inference *after having been trained on data that has similar statistics as the inference data*.

Arguments

- **axis**: Integer, the axis that should be normalized (typically the features axis). For instance, after a `Conv2D` 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 will 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**: Optional constraint for the beta weight.

- **gamma_constraint**: Optional constraint for the gamma weight.

Call arguments

- **inputs**: Input tensor (of any rank).
- **training**: Python boolean indicating whether the layer should behave in training mode or in inference mode.
    - `training=True`: The layer will normalize its inputs using the mean and variance of the current batch of inputs.
    - `training=False`: The layer will normalize its inputs using the mean and variance of its moving statistics, learned during training.

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 input.

Reference

- Ioffe and Szegedy, 2015.

About setting `layer.trainable = False` on a `BatchNormalization` layer:

The meaning of setting `layer.trainable = False` is to freeze the layer, i.e. its internal state will not change during training: its trainable weights will not be updated during `fit()` or `train_on_batch()`, and its state updates will not be run.

Usually, this does not necessarily mean that the layer is run in inference mode (which is normally controlled by the `training` argument that can be passed when calling a layer). "Frozen state" and "inference mode" are two separate concepts.

However, in the case of the `BatchNormalization` layer, **setting `trainable = False` on the layer means that the layer will be subsequently run in inference mode** (meaning that it will use the moving mean and the moving variance to normalize the current batch, rather than using the mean and variance of the current batch).

This behavior has been introduced in TensorFlow 2.0, in order to enable `layer.trainable = False` to produce the most commonly expected behavior in the convnet fine-tuning use case.

Note that: - Setting `trainable` on an model containing other layers will recursively set the `trainable` value of all inner layers. - If the value of the `trainable` attribute is changed after calling `compile()` on a model, the new value doesn't take effect for this model until `compile()` is called again.
