tf.train.polynomial_decay

```
polynomial_decay(
    learning_rate,
    global_step,
    decay_steps,
    end_learning_rate=0.0001,
    power=1.0,
    cycle=False,
    name=None
)
```

Defined in `tensorflow/python/training/learning_rate_decay.py`.

See the guide: Training > Decaying the learning rate

Applies a polynomial decay to the learning rate.

It is commonly observed that a monotonically decreasing learning rate, whose degree of change is carefully chosen, results in a better performing model. This function applies a polynomial decay function to a provided initial `learning_rate` to reach an `end_learning_rate` in the given `decay_steps`.

It requires a `global_step` value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

```
global_step = min(global_step, decay_steps)
decayed_learning_rate = (learning_rate - end_learning_rate) *
                        (1 - global_step / decay_steps) ^ (power) +
                        end_learning_rate
```

If `cycle` is True then a multiple of `decay_steps` is used, the first one that is bigger than `global_steps`.

```
decay_steps = decay_steps * ceil(global_step / decay_steps)
decayed_learning_rate = (learning_rate - end_learning_rate) *
                        (1 - global_step / decay_steps) ^ (power) +
                        end_learning_rate
```

Example: decay from 0.1 to 0.01 in 10000 steps using sqrt (i.e. power=0.5):

```
...
global_step = tf.Variable(0, trainable=False)
starter_learning_rate = 0.1
end_learning_rate = 0.01
decay_steps = 10000
learning_rate = tf.train.polynomial_decay(starter_learning_rate, global_step,
                                          decay_steps, end_learning_rate,
                                          power=0.5)
# Passing global_step to minimize() will increment it at each step.
learning_step = (
    tf.train.GradientDescentOptimizer(learning_rate)
    .minimize(...my loss..., global_step=global_step)
)
```

Args:

- `learning_rate`: A scalar `float32` or `float64` `Tensor` or a Python number. The initial learning rate.
- `global_step`: A scalar `int32` or `int64` `Tensor` or a Python number. Global step to use for the decay computation. Must not be negative.
- `decay_steps`: A scalar `int32` or `int64` `Tensor` or a Python number. Must be positive. See the decay computation above.
- `end_learning_rate`: A scalar `float32` or `float64` `Tensor` or a Python number. The minimal end learning rate.
- `power`: A scalar `float32` or `float64` `Tensor` or a Python number. The power of the polynomial. Defaults to linear, 1.0.
- `cycle`: A boolean, whether or not it should cycle beyond decay_steps.
- `name`: String. Optional name of the operation. Defaults to 'PolynomialDecay'.

Returns:

A scalar `Tensor` of the same type as `learning_rate`. The decayed learning rate.

Raises:

- `ValueError`: if `global_step` is not supplied.

English