SGD

SGD class

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```

Gradient descent (with momentum) optimizer.

Update rule for parameter `w` with gradient `g` when `momentum` is 0:

```
w = w - learning_rate * g
```

Update rule when `momentum` is larger than 0:

```
velocity = momentum * velocity - learning_rate * g
w = w + velocity
```

When `nesterov=True`, this rule becomes:

```
velocity = momentum * velocity - learning_rate * g
w = w + momentum * velocity - learning_rate * g
```

Arguments

- **learning_rate**: A `Tensor`, floating point value, or a schedule that is a `tf.keras.optimizers.schedules.LearningRateSchedule`, or a callable that takes no arguments and returns the actual value to use. The learning rate. Defaults to 0.01.
- **momentum**: float hyperparameter >= 0 that accelerates gradient descent in the relevant direction and dampens oscillations. Defaults to 0, i.e., vanilla gradient descent.
- **nesterov**: boolean. Whether to apply Nesterov momentum. Defaults to `False`.
- **name**: Optional name prefix for the operations created when applying gradients. Defaults to `"SGD"`.
- ****kwargs**: Keyword arguments. Allowed to be one of `"clipnorm"` or `"clipvalue"`. `"clipnorm"` (float) clips gradients by norm; `"clipvalue"` (float) clips gradients by value.

Usage:

```
>>> opt = tf.keras.optimizers.SGD(learning_rate=0.1)
>>> var = tf.Variable(1.0)
>>> loss = lambda: (var ** 2)/2.0         # d(loss)/d(var1) = var1
>>> step_count = opt.minimize(loss, [var]).numpy()
>>> # Step is `- learning_rate * grad`
>>> var.numpy()
0.9
```

```
>>> opt = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
>>> var = tf.Variable(1.0)
>>> val0 = var.value()
>>> loss = lambda: (var ** 2)/2.0         # d(loss)/d(var1) = var1
>>> # First step is `- learning_rate * grad`
>>> step_count = opt.minimize(loss, [var]).numpy()
>>> val1 = var.value()
>>> (val0 - val1).numpy()
0.1
>>> # On later steps, step-size increases because of momentum
>>> step_count = opt.minimize(loss, [var]).numpy()
>>> val2 = var.value()
>>> (val1 - val2).numpy()
0.18
```

Reference

- For `nesterov=True`, See Sutskever et al., 2013.