Adam

`Adam` class

```
tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    name="Adam",
    **kwargs
)
```

Optimizer that implements the Adam algorithm.

Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

According to Kingma et al., 2014, the method is "*computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters*".

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 0.001.
- **beta_1**: A float value or a constant float tensor, or a callable that takes no arguments and returns the actual value to use. The exponential decay rate for the 1st moment estimates. Defaults to 0.9.
- **beta_2**: A float value or a constant float tensor, or a callable that takes no arguments and returns the actual value to use, The exponential decay rate for the 2nd moment estimates. Defaults to 0.999.
- **epsilon**: A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper. Defaults to 1e-7.
- **amsgrad**: Boolean. Whether to apply AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and beyond". Defaults to `False`.
- **name**: Optional name for the operations created when applying gradients. Defaults to `"Adam"`.
- ****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.Adam(learning_rate=0.1)
>>> var1 = tf.Variable(10.0)
>>> loss = lambda: (var1 ** 2)/2.0       # d(loss)/d(var1) == var1
>>> step_count = opt.minimize(loss, [var1]).numpy()
>>> # The first step is `-learning_rate*sign(grad)`
>>> var1.numpy()
9.9
```

Reference

- Kingma et al., 2014
  - Reddi et al., 2018 for `amsgrad`.

Notes:

The default value of 1e-7 for epsilon might not be a good default in general. For example, when training an Inception network on ImageNet a current good choice is 1.0 or 0.1. Note that since Adam uses the formulation just before Section 2.1 of the Kingma and Ba paper rather than the formulation in Algorithm 1, the "epsilon" referred to here is "epsilon hat" in the paper.

The sparse implementation of this algorithm (used when the gradient is an IndexedSlices object, typically because of `tf.gather` or an embedding lookup in the forward pass) does apply momentum to variable slices even if they were not used in the forward pass (meaning they have a gradient equal to zero). Momentum decay (beta1) is also applied to the entire momentum accumulator. This means that the sparse behavior is equivalent to the dense behavior (in contrast to some momentum implementations which ignore momentum unless a variable slice was actually used).
