inline-clip_ops.py

Link

> https://github.com/tensorflow/tensorflow/blob/v2.13.0/tensorflow/python/ops/clip_ops.py

---

▼ Code

```
@dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None):
  with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
    t = ops.convert_to_tensor(t, name="t")

    n_element = math_ops.cast(array_ops.size(t), dtypes.float32)
    l2norm_inv = math_ops.rsqrt(
        math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t))))
    tclip = array_ops.identity(
        t * clip_norm * math_ops.minimum(
            l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm),
        name=name)

  return tclip
```

▼ Documentation

```
@dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None):
  """Clips tensor values to a maximum average L2-norm.

  Given a tensor `t`, and a maximum clip value `clip_norm`, this operation
  normalizes `t` so that its average L2-norm is less than or equal to
  `clip_norm`. Specifically, if the average L2-norm is already less than or
  equal to `clip_norm`, then `t` is not modified. If the average L2-norm is
  greater than `clip_norm`, then this operation returns a tensor of the same
  type and shape as `t` with its values set to:

  `t * clip_norm / l2norm_avg(t)`

  In this case, the average L2-norm of the output tensor is `clip_norm`.

  This operation is typically used to clip gradients before applying them with
  an optimizer.

  Args:
    t: A `Tensor`.
    clip_norm: A 0-D (scalar) `Tensor` > 0. A maximum clipping value.
    name: A name for the operation (optional).

  Returns:
    A clipped `Tensor`.
  """
  with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
    t = ops.convert_to_tensor(t, name="t")

    # Calculate L2-norm per element, clip elements by ratio of clip_norm to
    # L2-norm per element
    n_element = math_ops.cast(array_ops.size(t), dtypes.float32)
    l2norm_inv = math_ops.rsqrt(
        math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t))))
    tclip = array_ops.identity(
        t * clip_norm * math_ops.minimum(
            l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm),
        name=name)

  return tclip
```

Prompt

> As a code documentation assistant, you are assigned to document at the in-line level. When in-line comments are needed, your task is to insert comments within the code itself. The output should be the entire code along with the documentation you've added. The code is as follows:
>
> ```
> """
> @dispatch.add_dispatch_support
> def clip_by_average_norm(t, clip_norm, name=None):
>   with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
>     t = ops.convert_to_tensor(t, name="t")
>
>     n_element = math_ops.cast(array_ops.size(t), dtypes.float32)
>     l2norm_inv = math_ops.rsqrt(
>         math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t))))
>     tclip = array_ops.identity(
>         t * clip_norm * math_ops.minimum(
>             l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm),
>         name=name)
> ```

> return tclip
> ”””

Responses

ChatGPT

```

@dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None):
        """
    Clips the tensor `t` by its average L2 norm, ensuring it doesn't exceed `clip_norm`.

    Args:
        t: Input tensor to be clipped.
        clip_norm: The maximum allowable L2 norm for the tensor.
        name: (Optional) Name for the operation.

    Returns:
        tclip: A tensor with values scaled to ensure the L2 norm doesn't exceed `clip_nor
    """
    with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
        t = ops.convert_to_tensor(t, name="t")

        # Calculate the number of elements in the tensor.
        n_element = math_ops.cast(array_ops.size(t), dtypes.float32)

        # Calculate the inverse of the L2 norm.
        l2norm_inv = math_ops.rsqrt(
            math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t))))

        # Scale the tensor to clip its L2 norm.
        tclip = array_ops.identity(
            t * clip_norm * math_ops.minimum(
                l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm),
            name=name)

    return tclip
```

GPT4

```
# Importing necessary decorators for dispatch support
@dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None):
    # Creating a name scope for better graph visualization and debugging
    with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
        # Converting the input to a tensor
        t = ops.convert_to_tensor(t, name="t")

        # Calculating the number of elements in the tensor, converted to float32 for prec
        n_element = math_ops.cast(array_ops.size(t), dtypes.float32)

        # Computing the inverse of L2 norm (root sum square) of the tensor
        l2norm_inv = math_ops.rsqrt(
            math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t))))

        # Clipping the tensor by the given norm. The minimum is taken between the scaled
        # a scaled constant, ensuring the clipping does not exceed the specified clip_nor
        tclip = array_ops.identity(
            t * clip_norm * math_ops.minimum(
                l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm),
            name=name)

    # Returning the clipped tensor
    return tclip
```

Bard

```
"""
@dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None):
    """Clips the tensor `t` by its average norm.

    Args:
        t: The tensor to clip.
        clip_norm: The clipping norm.
        name: The name of the operation.

    Returns:
        A clipped tensor.
    """

    with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
        # Convert the input tensor to a tensor of the appropriate dtype.
        t = ops.convert_to_tensor(t, name="t")

        # Calculate the number of elements in the tensor.
        n_element = math_ops.cast(array_ops.size(t), dtypes.float32)

        # Calculate the inverse of the L2 norm of the tensor.
        l2norm_inv = math_ops.rsqrt(
            math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t)))
        )

        # Clip the tensor by its average norm.
        tclip = array_ops.identity(
            t * clip_norm * math_ops.minimum(l2norm_inv * n_element, constant_op.constant
            name=name,
        )

        return tclip
```

LLama2

```
@dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None): # pylint: disable=unused-argument
"""Clips input tensor by average norm."""
with ops.name_scope(name, "clip_by_average_norm", [t, clip_norm]) as name:
t = ops.convert_to_tensor(t, name="t") # Convert input to tensor

n_element = math_ops.cast(array_ops.size(t), dtypes.float32)  # Calculate number of eleme
l2norm_inv = math_ops.rsqrt(  # Inverse of L2 norm
    math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t)))
)
tclip = array_ops.identity(  # Clip input tensor
    t * clip_norm * math_ops.minimum(
        l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm
    ),
    name=name
)
return tclip # Return clipped tensor
```

StarChat

```
# @dispatch.add_dispatch_support
def clip_by_average_norm(t, clip_norm, name=None):
    """Clips tensor by its L2 norm averaged over all dimensions."""
    # Convert input tensors into Tensors and check their shapes.
    t = ops.convert_to_tensor(t, name="t")

    # Calculate the average L2 norm of each dimension of `t`.
    n_element = math_ops.cast(array_ops.size(t), dtypes.float32)
    l2norm_inv = math_ops.rsqrt(
        math_ops.reduce_sum(t * t, math_ops.range(array_ops.rank(t))))

    # Clip the tensor if its L2 norm exceeds `clip_norm` times the inverse
    # average L2 norm. Otherwise, just return the original tensor unchanged.
    tclip = array_ops.identity(
        t * clip_norm * math_ops.minimum(
            l2norm_inv * n_element, constant_op.constant(1.0) / clip_norm),
        name=name)

    return tclip
```