jax.jit

Contents

- `jit()`

jax.jit(*fun*, */*, ***, *in_shardings=UnspecifiedValue*, *out_shardings=UnspecifiedValue*, *static_argnums=None*, *static_argnames=None*, *donate_argnums=None*, *donate_argnames=None*, *keep_unused=False*, *device=None*, *backend=None*, *inline=False*, *abstracted_axes=None*, *compiler_options=None*) [source]

Sets up `fun` for just-in-time compilation with XLA.

Parameters:

- **fun** (*Callable*) –

  Function to be jitted. `fun` should be a pure function.
  The arguments and return value of `fun` should be arrays, scalar, or (nested) standard Python containers (tuple/list/dict) thereof. Positional arguments indicated by `static_argnums` can be any hashable type. Static arguments are included as part of a compilation cache key, which is why hash and equality operators must be defined. JAX keeps a weak reference to `fun` for use as a compilation cache key, so the object `fun` must be weakly-referenceable.
- **in_shardings** (*Any*) – optional, a `Sharding` or pytree with `Sharding` leaves and structure that is a tree prefix of the positional arguments tuple to `fun`. If provided, the positional arguments passed to `fun` must have shardings that are compatible with `in_shardings` or an error is raised, and the compiled computation has input shardings corresponding to `in_shardings`. If not provided, the compiled computation's input shardings are inferred from argument shardings.
- **out_shardings** (*Any*) – op `ding` or pytree with `Sharding` leaves and structure that is a tree prefix of the output of `fun`. If provided, it has the same effect as applying `jax.lax.with_sharding_constraint()` to the output of `fun`.
- **static_argnums** (*int* | *Sequence[int]* | *None*) –

  optional, an int or collection of ints that specify which positi
  treat as static (trace- and compile-time constant).

Static arguments should be hashable, meaning both `__hash__` and `__eq__` are implemented, and immutable. Otherwise, they can be arbitrary Python objects. Calling the jitted function with different values for these constants will trigger recompilation. Arguments that are not array-like or containers thereof must be marked as static.
If neither `static_argnums` nor `static_argnames` is provided, no arguments are treated as static. If `static_argnums` is not provided but `static_argnames` is, or vice versa, JAX uses `inspect.signature(fun)` to find any positional arguments that correspond to `static_argnames` (or vice versa). If both `static_argnums` and `static_argnames` are provided, `inspect.signature` is not used, and only actual parameters listed in either `static_argnums` or `static_argnames` will be treated as static.

- **static_argnames** (*str* | *Iterable[str]* | *None*) – optional, a string or collection of strings specifying which named arguments to treat as static (compile-time constant). See the comment on `static_argnums` for details. If not provided but `static_argnums` is set, the default is based on calling `inspect.signature(fun)` to find corresponding named arguments.
- **donate_argnums** (*int* | *Sequence[int]* | *None*) –

  optional, collection of integers to specify which positional argument buffers can be overwritten by the computation and marked deleted in the caller. It is safe to donate argument buffers if you no longer need them once the computation has started. In some cases XLA can make use of donated buffers to reduce the amount of memory needed to perform a computation, for example recycling one of your input buffers to store a result. You should not reuse buffers that you donate to a computation; JAX will raise an error if you try to. By default, no argument buffers are donated.
  If neither `donate_argnums` nor `donate_argnames` is provided, no arguments are donated. If `donate_argnums` is not provided but `donate_argnames` is, or vice versa, JAX uses `inspect.signature(fun)` to find any positional arguments that correspond to `donate_argnames` (or vice versa). If both `donate_argnums` and `donate_argnames` are provided, `inspect.signature` is not used, and only actual parameters listed in either `donate_argnums` or `donate_argnames` will be donated.
  For more details on buffer donation see the FAQ.
- **donate_argnames** (*str* | *Iterable[str]* | *None*) – optional, a string or collection of strings specifying which named arguments are donated to the computation. See the comment on `donate_argnums` for details. If not provide `donate_argnums` is set, the default is based on calling `inspe` to find corresponding named arguments.

- **keep_unused** (*bool*) – optional boolean. If *False* (the default), arguments that JAX determines to be unused by *fun may* be dropped from resulting compiled XLA executables. Such arguments will not be transferred to the device nor provided to the underlying executable. If *True*, unused arguments will not be pruned.
- **device** (*xc.Device | None*) – This is an experimental feature and the API is likely to change. Optional, the Device the jitted function will run on. (Available devices can be retrieved via `jax.devices()`.) The default is inherited from XLA's DeviceAssignment logic and is usually to use `jax.devices()[0]`.
- **backend** (*str | None*) – This is an experimental feature and the API is likely to change. Optional, a string representing the XLA backend: `'cpu'`, `'gpu'`, or `'tpu'`.
- **inline** (*bool*) – Optional boolean. Specify whether this function should be inlined into enclosing jaxprs. Default False.
- **abstracted_axes** (*Any | None*)
- **compiler_options** (*dict[str, Any] | None*)

Returns:

A wrapped version of `fun`, set up for just-in-time compilation.

Return type:

pjit.JitWrapped

Examples

In the following example, `selu` can be compiled into a single fused kernel by XLA:

```
>>> import jax
>>>
>>> @jax.jit
... def selu(x, alpha=1.67, lmbda=1.05):
...   return lmbda * jax.numpy.where(x > 0, x, alpha * jax.numpy.exp(x) - alpha
>>>
>>> key = jax.random.key(0)
>>> x = jax.random.normal(key, (10,))
>>> print(selu(x))
[-0.54485  0.27744 -0.29255 -0.91421 -0.62452 -0.24748
 -0.85743 -0.78232  0.76827  0.59566 ]
```

To pass arguments such as `static_argnames` when decorating a function, a common pattern is to use `functools.partial()`:

```
>>> from functools import partial
>>>
>>> @partial(jax.jit, static_argnames=['n'])
```

latest

```
... def g(x, n):
...   for i in range(n):
...     x = x ** 2
...   return x
>>>
>>> g(jnp.arange(4), 3)
Array([   0,    1,  256, 6561], dtype=int32)
```

Previous
jax.transfer_guard

Next
jax.disable_jit