jax.lax package

Contents

`jax.lax` is a library of primitives operations that underpins libraries such as `jax.numpy`. Transformation rules, such as JVP and batching rules, are typically defined as transformations on `jax.lax` primitives.

Many of the primitives are thin wrappers around equivalent XLA operations, described by the XLA operation semantics documentation. In a few cases JAX diverges from XLA, usually to ensure that the set of operations is closed under the operation of JVP and transpose rules.

Where possible, prefer to use libraries such as `jax.numpy` instead of using `jax.lax` directly. The `jax.numpy` API follows NumPy, and is therefore more stable and less likely to change than the `jax.lax` API.

Operators

abs(x)	Elementwise absolute value: $\|x\|$.
add(x, y)	Elementwise addition: $x + y$.
acos(x)	Elementwise arc cosine: $\mathrm{acos}(x)$.
approx_max_k(operand, k[, ...])	Returns max `k` values and their indices of the `operand` in an approximate manner.
approx_min_k(operand, k[, ...])	Returns min `k` values and their indices of the `operand` in an approximate manner.
argmax(operand, axis, index_dtype)	Computes the index of the maximum element along `axis`.
argmin(operand, axis, index_dtype)	Computes the index of the minimum element along `axis`.
asin(x)	Elementwise arc sine: $\mathrm{asin}(x)$.
atan(x)	Elementwise arc tangent: $\mathrm{atan}(x)$.
atan2(x, y)	Elementwise arc tangent of two variables: $\mathrm{atan}(\frac{x}{y})$.
batch_matmul(lhs, rhs[, precision])	Batch matrix multiplication.
bessel_i0e(x)	Exponentially scaled modified Bessel function of order 0: $\mathrm{i0e}(x) = e^{-\|x\|}\mathrm{i0}(x)$
bessel_i1e(x)	Exponentially scaled modified Bessel function of order 1: $\mathrm{i1e}(x) = e^{-\|x\|}\mathrm{i1}(x)$
betainc(a, b, x)	Elementwise regularized incomplete beta integral.
bitcast_convert_type(operand, new_dtype)	Elementwise bitcast.
bitwise_not(x)	Elementwise NOT: $\neg x$.
bitwise_and(x, y)	Elementwise AND: $x \wedge y$.
bitwise_or(x, y)	Elementwise OR: $x \vee y$.
bitwise_xor(x, y)	Elementwise exclusive OR: $x \oplus y$.
population_count(x)	Elementwise popcount, count the number of set bits in each element.
broadcast(operand, sizes)	Broadcasts an array, adding new leading dimensions
broadcasted_iota(dtype, shape, dimension)	Convenience wrapper around `iota`.
broadcast_in_dim(operand, shape, ...)	Wraps XLA's BroadcastInDim operator.
cbrt(x)	Elementwise cube root: $\sqrt[3]{x}$.
ceil(x)	Elementwise ceiling: $\lceil x \rceil$.

clamp(min, x, max)	Elementwise clamp.
collapse(operand, start_dimension, ...)	Collapses dimensions of an array into a single dimension.
complex(x, y)	Elementwise make complex number: $x + jy$.
concatenate(operands, dimension)	Concatenates a sequence of arrays along *dimension*.
conj(x)	Elementwise complex conjugate function: $\overline{x}$.
conv(lhs, rhs, window_strides, padding[, ...])	Convenience wrapper around *conv_general_dilated*.
convert_element_type(operand, new_dtype)	Elementwise cast.
conv_dimension_numbers(lhs_shape, rhs_shape, ...)	Converts convolution *dimension_numbers* to a *ConvDimensionNumbers*.
conv_general_dilated(lhs, rhs, ...[, ...])	General n-dimensional convolution operator, with optional dilation.
conv_general_dilated_local(lhs, rhs, ...[, ...])	General n-dimensional unshared convolution operator with optional dilation.
conv_general_dilated_patches(lhs, ...[, ...])	Extract patches subject to the receptive field of *conv_general_dilated*.
conv_with_general_padding(lhs, rhs, ...[, ...])	Convenience wrapper around *conv_general_dilated*.
conv_transpose(lhs, rhs, strides, padding[, ...])	Convenience wrapper for calculating the N-d convolution "transpose".
cos(x)	Elementwise cosine: $\cos(x)$.
cosh(x)	Elementwise hyperbolic cosine: $\cosh(x)$.
cummax(operand[, axis, reverse])	Computes a cumulative maximum along *axis*.
cummin(operand[, axis, reverse])	Computes a cumulative minimum along *axis*.
cumprod(operand[, axis, reverse])	Computes a cumulative product along *axis*.
cumsum(operand[, axis, reverse])	Computes a cumulative sum along *axis*.
digamma(x)	Elementwise digamma: $\psi(x)$.
div(x, y)	Elementwise division: $\frac{x}{y}$.
dot(lhs, rhs[, precision, ...])	Vector/vector, matrix/vector, and matrix/matrix multiplication.
dot_general(lhs, rhs, dimension_numbers[, ...])	More general contraction operator.

dynamic_index_in_dim(operand, index[, axis, ...])	Convenience wrapper around dynamic_slice to perform int indexing.
dynamic_slice(operand, start_indices, ...)	Wraps XLA's DynamicSlice operator.
dynamic_slice_in_dim(operand, start_index, ...)	Convenience wrapper around dynamic_slice applying to one dimension.
dynamic_update_slice(operand, update, ...)	Wraps XLA's DynamicUpdateSlice operator.
dynamic_update_index_in_dim(operand, update, ...)	Convenience wrapper around `dynamic_update_slice()` to update a slice of size 1 in a single `axis`.
dynamic_update_slice_in_dim(operand, update, ...)	Convenience wrapper around `dynamic_update_slice()` to update a slice in a single `axis`.
eq(x, y)	Elementwise equals: $x = y$.
erf(x)	Elementwise error function: $\mathrm{erf}(x)$.
erfc(x)	Elementwise complementary error function: $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$.
erf_inv(x)	Elementwise inverse error function: $\mathrm{erf}^{-1}(x)$.
exp(x)	Elementwise exponential: e^x.
expand_dims(array, dimensions)	Insert any number of size 1 dimensions into an array.
expm1(x)	Elementwise $e^x - 1$.
fft(x, fft_type, fft_lengths)	**param fft_type:**
floor(x)	Elementwise floor: $\lfloor x \rfloor$.
full(shape, fill_value[, dtype])	Returns an array of *shape* filled with *fill_value*.
full_like(x, fill_value[, dtype, shape])	Create a full array like np.full based on the example array *x*.
gather(operand, start_indices, ...[, ...])	Gather operator.
ge(x, y)	Elementwise greater-than-or-equals: $x \geq y$.
gt(x, y)	Elementwise greater-than: $x > y$.
igamma(a, x)	Elementwise regularized incomplete gamma function.
igammac(a, x)	Elementwise complementary regularized incomplete gamma function.

imag(x)	Elementwise extract imaginary part: $\mathrm{Im}(x)$.
index_in_dim(operand, index[, axis, keepdims])	Convenience wrapper around slice to perform int indexing.
index_take(src, idxs, axes)	**param src:**
iota(dtype, size)	Wraps XLA's Iota operator.
is_finite(x)	Elementwise $\mathrm{isfinite}$.
le(x, y)	Elementwise less-than-or-equals: $x \leq y$.
lt(x, y)	Elementwise less-than: $x < y$.
lgamma(x)	Elementwise log gamma: $\mathrm{log}(\Gamma(x))$.
log(x)	Elementwise natural logarithm: $\mathrm{log}(x)$.
log1p(x)	Elementwise $\mathrm{log}(1 + x)$.
logistic(x)	Elementwise logistic (sigmoid) function: $\frac{1}{1+e^{-x}}$.
max(x, y)	Elementwise maximum: $\mathrm{max}(x, y)$
min(x, y)	Elementwise minimum: $\mathrm{min}(x, y)$
mul(x, y)	Elementwise multiplication: $x \times y$.
ne(x, y)	Elementwise not-equals: $x \neq y$.
neg(x)	Elementwise negation: $-x$.
nextafter(x1, x2)	Returns the next representable value after *x1* in the direction of *x2*.
pad(operand, padding_value, padding_config)	Applies low, high, and/or interior padding to an array.
pow(x, y)	Elementwise power: x^y.
real(x)	Elementwise extract real part: $\mathrm{Re}(x)$.
reciprocal(x)	Elementwise reciprocal: $\frac{1}{x}$.
reduce(operands, init_values, computation, ...)	Wraps XLA's Reduce operator.
reduce_precision(operand, exponent_bits, ...)	Wraps XLA's ReducePrecision operator.
reduce_window(operand, init_value, ...[, ...])	Wraps XLA's ReduceWindowWithGeneralPadding operator.
reshape(operand, new_sizes[, dimensions])	Wraps XLA's Reshape operator.
rem(x, y)	Elementwise remainder: $x \bmod y$.

`rev`(operand, dimensions)	Wraps XLA's Rev operator.
`round`(x[, rounding_method])	Elementwise round.
`rsqrt`(x)	Elementwise reciprocal square root: $\frac{1}{\sqrt{x}}$.
`scatter`(operand, scatter_indices, updates, ...)	Scatter-update operator.
`scatter_add`(operand, scatter_indices, ...[, ...])	Scatter-add operator.
`scatter_max`(operand, scatter_indices, ...[, ...])	Scatter-max operator.
`scatter_min`(operand, scatter_indices, ...[, ...])	Scatter-min operator.
`scatter_mul`(operand, scatter_indices, ...[, ...])	Scatter-multiply operator.
`select`(pred, on_true, on_false)	Selects between two branches based on a boolean predicate.
`shift_left`(x, y)	Elementwise left shift: $x \ll y$.
`shift_right_arithmetic`(x, y)	Elementwise arithmetic right shift: $x \gg y$.
`shift_right_logical`(x, y)	Elementwise logical right shift: $x \gg y$.
`slice`(operand, start_indices, limit_indices)	Wraps XLA's Slice operator.
`slice_in_dim`(operand, start_index, limit_index)	Convenience wrapper around slice applying to only one dimension.
`sign`(x)	Elementwise sign.
`sin`(x)	Elementwise sine: $\sin(x)$.
`sinh`(x)	Elementwise hyperbolic sine: $\sinh(x)$.
`sort`()	Wraps XLA's Sort operator.
`sort_key_val`(keys, values[, dimension, ...])	Sorts `keys` along `dimension` and applies the same permutation to `values`.
`sqrt`(x)	Elementwise square root: $\sqrt{x}$.
`square`(x)	Elementwise square: x^2.
`squeeze`(array, dimensions)	Squeeze any number of size 1 dimensions from an array.
`sub`(x, y)	Elementwise subtraction: $x - y$.
`tan`(x)	Elementwise tangent: $\tan(x)$.
`tie_in`(x, y)	Deprecated.
`top_k`(operand, k)	Returns top `k` values and their indices along the last axis of `operand`.
`transpose`(operand, permutation)	Wraps XLA's Transpose operator.

Control flow operators

`associative_scan`(fn, elems[, reverse, axis])	Performs a scan with an associative binary operation, in parallel.
`cond`(pred, true_fun, false_fun, *operands[, ...])	Conditionally apply `true_fun` or `false_fun`.
`fori_loop`(lower, upper, body_fun, init_val)	Loop from `lower` to `upper` by reduction to `jax.lax.while_loop()`.
`map`(f, xs)	Map a function over leading array axes.
`scan`(f, init, xs[, length, reverse, unroll])	Scan a function over leading array axes while carrying along state.
`switch`(index, branches, *operands[, operand])	Apply exactly one of `branches` given by `index`.
`while_loop`(cond_fun, body_fun, init_val)	Call `body_fun` repeatedly in a loop while `cond_fun` is True.

Custom gradient operators

`stop_gradient`(x)	Stops gradient computation.
`custom_linear_solve`(matvec, b, solve[, ...])	Perform a matrix-free linear solve with implicitly defined gradients.
`custom_root`(f, initial_guess, solve, ...[, ...])	Differentiably solve for a roots of a function.

Parallel operators

Parallelism support is experimental.

all_gather(x, axis_name, *[, ...])	Gather values of x across all replicas.
all_to_all(x, axis_name, split_axis, ...[, ...])	Materialize the mapped axis and map a different axis.
psum(x, axis_name, *[, axis_index_groups])	Compute an all-reduce sum on `x` over the pmapped axis `axis_name`.
pmax(x, axis_name, *[, axis_index_groups])	Compute an all-reduce max on `x` over the pmapped axis `axis_name`.
pmin(x, axis_name, *[, axis_index_groups])	Compute an all-reduce min on `x` over the pmapped axis `axis_name`.
pmean(x, axis_name, *[, axis_index_groups])	Compute an all-reduce mean on `x` over the pmapped axis `axis_name`.
ppermute(x, axis_name, perm)	Perform a collective permutation according to the permutation `perm`.
pshuffle(x, axis_name, perm)	Convenience wrapper of jax.lax.ppermute with alternate permutation encoding
pswapaxes(x, axis_name, axis, *[, ...])	Swap the pmapped axis `axis_name` with the unmapped axis `axis`.
axis_index(axis_name)	Return the index along the mapped axis `axis_name`.

Linear algebra operators (jax.lax.linalg)

cholesky(x, *[, symmetrize_input])	Cholesky decomposition.
eig(x, *[, compute_left_eigenvectors, ...])	Eigendecomposition of a general matrix.
eigh(x, *[, lower, symmetrize_input, ...])	Eigendecomposition of a Hermitian matrix.
hessenberg(a)	Reduces a square matrix to upper Hessenberg form.
lu(x)	LU decomposition with partial pivoting.
householder_product(a, taus)	Product of elementary Householder reflectors.
qdwh(x, *[, is_hermitian, max_iterations, ...])	QR-based dynamically weighted Halley iteration for polar decomposition.
qr(x, *[, full_matrices])	QR decomposition.
schur(x, *[, compute_schur_vectors, ...])	**param x:**
svd()	Singular value decomposition.
triangular_solve(a, b, *[, left_side, ...])	Triangular solve.
tridiagonal(a, *[, lower])	Reduces a symmetric/Hermitian matrix to tridiagonal form.
tridiagonal_solve(dl, d, du, b)	Computes the solution of a tridiagonal linear system.

Argument classes

class jax.lax.**ConvDimensionNumbers**(*lhs_spec: Sequence[int], rhs_spec: Sequence[int], out_spec: Sequence[int]*) [source]

Describes batch, spatial, and feature dimensions of a convolution.

Parameters:
- **lhs_spec** – a tuple of nonnegative integer dimension numbers containing *(batch dimension, feature dimension, spatial dimensions…)*.
- **rhs_spec** – a tuple of nonnegative integer dimension numbers containing *(out feature dimension, in feature dimension, spatial dimensions…)*.
- **out_spec** – a tuple of nonnegative integer dimension numbers containing *(batch dimension, feature dimension, spatial dimensions…)*.

jax.lax.**ConvGeneralDilatedDimensionNumbers**

alias of `Union[None, jax._src.lax.convolution.ConvDimensionNumbers, Tuple[str, str, str]]`

class jax.lax.**GatherDimensionNumbers**(*offset_dims: Tuple[int, ...], collapsed_slice_dims: Tuple[int, ...], start_index_map: Tuple[int, ...]*) [source]

Describes the dimension number arguments to an XLA's Gather operator. See the XLA documentation for more details of what the dimension numbers mean.

Parameters:
- **offset_dims** – the set of dimensions in the *gather* output that offset into an array sliced from *operand*. Must be a tuple of integers in ascending order, each representing a dimension number of the output.
- **collapsed_slice_dims** – the set of dimensions *i* in *operand* that have *slice_sizes[i] == 1* and that should not have a corresponding dimension in the output of the gather. Must be a tuple of integers in ascending order.
- **start_index_map** – for each dimension in *start_indices*, gives the corresponding dimension in *operand* that is to be sliced. Must be a tuple of integers with size equal to *start_indices.shape[-1]*.

Unlike XLA's *GatherDimensionNumbers* structure, *index_vector_dim* is implicit; there is always an index vector dimension and it must always be the last dimension. To gather scalar indices, add a trailing dimension of size 1.

class jax.lax.**GatherScatterMode**(*value*) [source]

Describes how to handle out-of-bounds indices in a gather or scatter.

Possible values are:

CLIP:

Indices will be clamped to the nearest in-range value, i.e., such that the entire window to be gathered is in-range.

FILL_OR_DROP:

If any part of a gathered window is out of bounds, the entire window that is returned, even those elements that were otherwise in-bounds, will be filled with a constant. If any part of a scattered window is out of bounds, the entire window will be discarded.

PROMISE_IN_BOUNDS:

The user promises that indices are in bounds. No additional checking will be performed. In practice, with the current XLA implementation this means that, out-of-bounds gathers will be clamped but out-of-bounds scatters will be discarded. Gradients will not be correct if indices are out-of-bounds.

class jax.lax.**Precision**(*arg0*) [source]

Precision enum for lax functions

The *precision* argument to JAX functions generally controls the tradeoff between speed and accuracy for array computations on accelerator backends, (i.e. TPU and GPU). Members are:

DEFAULT:

Fastest mode, but least accurate. Performs computations in bfloat16. Aliases: `'default'`, `'fastest'`, `'bfloat16'`.

HIGH:

Slower but more accurate. Performs float32 computations in 3 bfloat16 passes, or using tensorfloat32 where available. Aliases: `'high'`, `'bfloat16_3x'`, `'tensorfloat32'`.

HIGHEST:

Slowest but most accurate. Performs computations in float32 or float64 as applicable. Aliases: `'highest'`, `'float32'`.

class `jax.lax.RoundingMethod`(*value*) [source]

An enumeration.

class `jax.lax.ScatterDimensionNumbers`(*update_window_dims: Sequence[int], inserted_window_dims: Sequence[int], scatter_dims_to_operand_dims: Sequence[int]*) [source]

Describes the dimension number arguments to an XLA's Scatter operator. See the XLA documentation for more details of what the dimension numbers mean.

Parameters:

- **update_window_dims** – the set of dimensions in the *updates* that are window dimensions. Must be a tuple of integers in ascending order, each representing a dimension number.
- **inserted_window_dims** – the set of size 1 window dimensions that must be inserted into the shape of *updates*. Must be a tuple of integers in ascending order, each representing a dimension number of the output. These are the mirror image of *collapsed_slice_dims* in the case of *gather*.
- **scatter_dims_to_operand_dims** – for each dimension in *scatter_indices*, gives the corresponding dimension in *operand*. Must be a sequence of integers with size equal to indices.shape[-1].

Unlike XLA's *ScatterDimensionNumbers* structure, *index_vector_dim* is implicit; there is always an index vector dimension and it must always be the last dimension. To scatter scalar indices, add a trailing dimension of size 1.