Chex Quick Reference

Core Purpose & Philosophy

- **Goal:** Enhance reliability, debugging, and testing for JAX applications.
- **Why Needed:** Standard Python `assert` can be unreliable or optimized away within JAX transformations (`jit`, `vmap`) due to tracing with abstract values (Tracers).
- **Chex Advantage:** Assertions work reliably on both Tracers (during tracing/compilation) and concrete values (at runtime). Provides clear, JAX-specific error messages. Acts as executable documentation.

Key Assertion Functions (Instrumentation)

- **`chex.assert_shape(x, expected_shape)`**: Verifies array shape. `expected_shape` can use integers, `None` (variable dim), `...` (any number of leading/trailing dims). *Essential for JAX.*
- **`chex.assert_type(x, expected_type)`**: Checks array `dtype` (e.g., `jnp.float32`).
- **`chex.assert_rank(x, expected_rank)`**: Checks the number of dimensions (rank) of an array.
- **`chex.assert_axis_dimension(x, axis, expected_size)`**: Checks the size of a *specific* dimension. Useful in NNX `__call__`.
- **`chex.assert_scalar(x)`**: Checks if `x` is a scalar (shape `()`, rank 0).
- **`chex.assert_trees_all_close(tree1, tree2, **kwargs)`**: Compares nested JAX PyTrees (e.g., model params, states) for approximate equality.
- **`chex.assert_trees_all_equal(tree1, tree2)`**: Compares PyTrees for exact equality.
- **`chex.assert_tree_all_finite(tree)`**: Checks for `NaN`/`Inf` values in all numerical arrays within a PyTree. *Crucial for debugging numerical stability (activations, gradients).*

Usage: Place these assertions directly within JAX functions (including those decorated with `@jax.jit`, `@jax.vmap`) and Flax `nnx.Module` methods (`__call__`).

Debugging Value-Based Logic Inside Transformations

- **Problem:** Standard Python `if`/`assert` based on *tensor values* (e.g., `if jnp.all(x > 0):`, `assert jnp.sum(x) > 0`) often fails during JAX tracing (`jit`) because the value isn't known abstractly.
- **Solution:**
    - Apply *before* JAX transformations (`@chex.chexify`, then `@jax.jit`).
    - **How it Works:** Lets JAX trace normally for compilation, but runs the original Python logic again at *runtime* with concrete values, allowing value-based `assert` or `if` statements to execute.
    - Enables checks like `assert jnp.all(...)`, `assert jnp.any(jnp.isnan(...))`, or calling `chex.assert_tree_all_finite` inside transformed functions.
    - Requires `chex.block_until_chexify_assertions_complete()` after the function call typically.
- **Caveats:**
    - **Debugging Tool:** Primarily for temporary checks during development/testing.
    - **Performance Overhead:** Significantly slower due to potential double execution. Remove for production.
    - **Colab:** Currently does not work in Colab environments.

Debugging Performance: Detecting Recompilation

- **Problem:** `@jax.jit` recompiles functions if input structures (shapes, dtypes, static args) change. Frequent recompilation is slow and often indicates an issue (e.g., unstable shapes, mishandled static args).
- **Solution:**
    - Apply to a function (often alongside `@jax.jit`).
    - Monitors how many times the function is traced (compiled).
    - Raises `AssertionError` if the trace count exceeds `n`.
    - Use `chex.clear_trace_counter()` before tests if needed.
- **When to Use:**
    - During development/testing on key jitted functions (training steps, model passes).
    - Debugging performance issues to pinpoint unexpected recompilations.

- **Benefits:** Catches performance regressions, improves understanding of `static_argnums`/JIT behavior.
- **Note:** Typically remove or disable in production code.

Integration with JAX / Flax NNX

- **Core JAX:** Use assertions directly inside functions, works seamlessly with `@jax.jit`, `@jax.vmap`.
- **Flax NNX (**
    - **Method:** Primary location for `chex.assert_shape`, `_type`, `_rank`, `_axis_dimension` on inputs, intermediate activations, and outputs. Use `chex.assert_tree_all_finite` on activations.
    - **Training/Evaluation Loops:** Use assertions to check input/target batch shapes (`chex.assert_shape`, `chex.assert_equal_shape`), model output shapes vs targets, and gradient validity (`chex.assert_tree_all_finite(grads)`).

Other Mentioned Chex Utilities

- **Dataclasses:** JAX-friendly dataclass implementation.
- **Warnings:** Utilities for adding common warnings (e.g., deprecation).
- **Test Variants**: Run tests under different conditions (e.g., `with_jit=True/False`).
- **Fakes:** Utilities to simulate environments (e.g., fake multi-device).

More Information

- JAX AI Stack - https://jaxstack.ai
- Chex - https://chex.readthedocs.io
- JAX - https://jax.dev
- Flax - https://flax.readthedocs.io