Accelerated machine-learning research via composable function transformations in Python

https://github.com/google/jax

https://www.youtube.com/watch?v=z-WSrQDXkuM
https://colab.research.google.com/drive/1ii4J5CIy8FL0sBgvOIHbJoumpoV0kXIA?usp=sharing

Fast

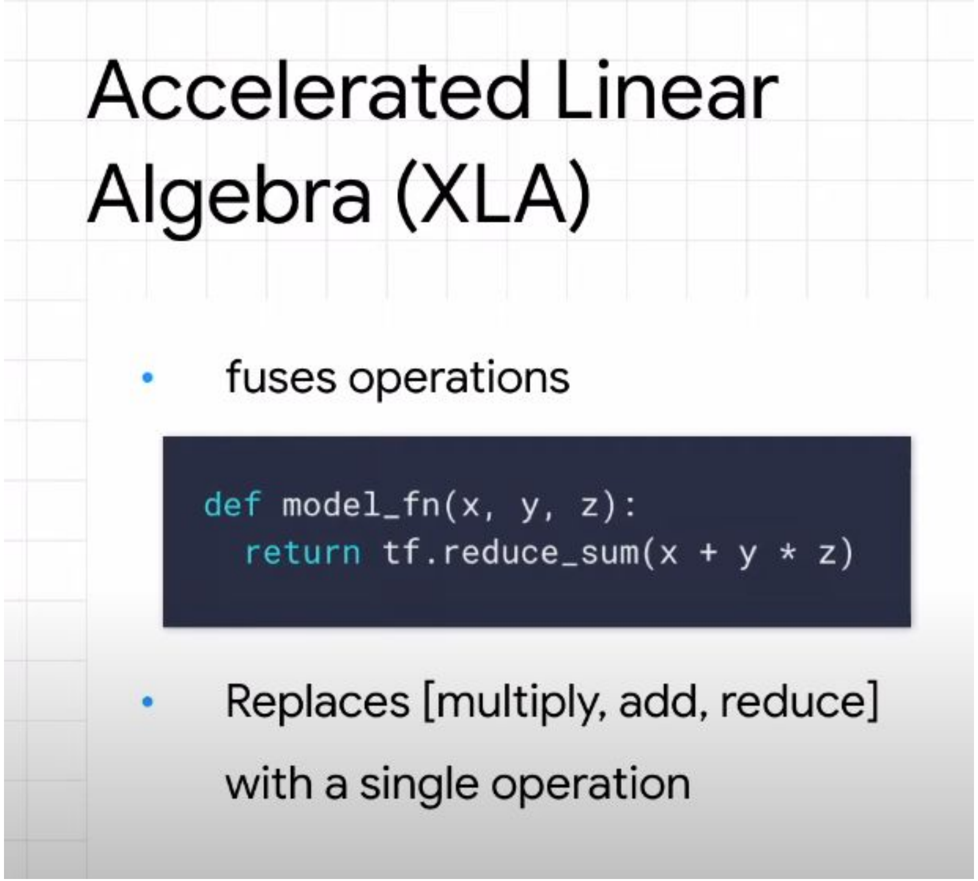
NumPy

Stable & loved

+

- ✓ GPU/TPU
- ✓ compilation
- ✓ autodiff
- ✓ parallelization

=



Smart compiler under the hood

JAX uses XLA to compile and run your NumPy programs on GPUs and TPUs. Compilation happens under the hood by default, with library calls getting just-in-time compiled and executed.

But JAX also lets you just-in-time compile your own Python functions into XLA-optimized kernels using a one-function API, `jit`.

JIT: your func -> XLA optimized for free

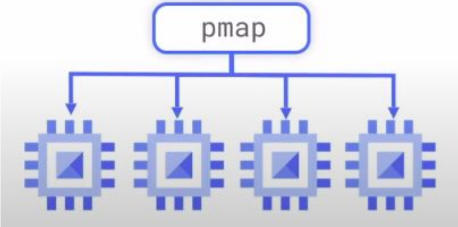
- Use `jit` for "just-in-time" compilation of custom functions

```
@jit
def update(params, x, y):
  grads = grad(loss)(params, x, y)
  return [(w - step_size * dw, b - step_size * db)
          for (w, b), (dw, db) in zip(params, grads)]
```

✅ GPU/TPU ✅ autodiff

✅ compilation ✅ parallelization

- Autodiff (`jax.grad`): Efficient any-order gradients w.r.t any variables
- JIT compilation (`jax.jit`): Trace any function ⟶ fused accelerator ops
- Vectorization (`jax.vmap`): Automatically batch code written for individual samples
- Parallelization (`jax.pmap`): Automatically parallelize code across multiple accelerators (i
  e.g. for TPU pods)



What does this function do?

```python
def f(x):
  return x + 2
```

What does this function do?

```
def f(x):
  return x + 2
```

```
class EspressoDelegator(object):

  def __add__(self, num_espressos):
    subprocess.popen(["ssh", ...])
```

Thinking about JIT compilation.

What does this function do?

```
def f(x:ShapedArray[float32, (2, 2)]):
  return x + 2
```

Python's dynamism means a function can do almost *anything* depending on it's input.

Python function → JAX Intermediate Representation

```python
from jax import lax

def log2(x):
  ln_x = lax.log(x)
  ln_2 = lax.log(2)
  return ln_x / ln_2
```

```
{ lambda  ;  ; a.
  let b = log a
      c = div b 0.693147
  in [c] }
```

```python
x = np.array(...)
y = jit(log2)(x)
```

No compilation yet!

How does this compare to Numba?

`jax.jit` uses tracers to observe how a function operates on abstract inputs, and compiles via XLA.

`numba.jit` transpiles python bytecode directly to LLVM for compilation.

<u>jax.jit does less, but was made for DL and has huge support by researchers</u>

FLAX

https://github.com/google/flax/

- **Neural network API** (`flax.linen`): Dense, Conv, {Batch|Layer|Group} Norm, Attention, Pooling, {LSTM|GRU} Cell, Dropout
- **Optimizers** (`flax.optim`): SGD, Momentum, Adam, LARS, Adagrad, LAMB, RMSprop
- **Utilities and patterns**: replicated training, serialization and checkpointing, metrics, prefetching on device
- **Educational examples** that work out of the box: MNIST, LSTM seq2seq, Graph Neural Networks, Sequence Tagging
- **Fast, tuned large-scale end-to-end examples**: CIFAR10, ResNet on ImageNet, Transformer LM1b

```python
def onehot(sequence, vocab_size):
  """One-hot encode a single sequence of integers."""
  return jnp.array(
      sequence[:, np.newaxis] == jnp.arange(vocab_size), dtype=jnp.float32)
```

Спасибо за внимание!