tinygrad Public

master 5 Tags Go to file t Add file Code About

chenyuxyz 3,482 Commits

.github/wor...	onnx Einsum, Cu...	2 hours ago
disassembl...	start Qualcomm ...	last month
docs	fix fuzz_linearizer...	last week
examples	cifar move Global...	3 days ago
extra	onnx Einsum, Cu...	2 hours ago
openpilot	remove numpy fr...	2 weeks ago
test	onnx Einsum, Cu...	2 hours ago
tinygrad	hip launch speed ...	19 hours ago
.editorconfig	Revert "update e...	6 months ago
.gitignore	No extra vars call ...	2 weeks ago
.pre-comm...	remove numpy fr...	2 weeks ago
.pylintrc	ruff checks the m...	last month
.tokeignore	Add a quick start ...	8 months ago
LICENSE	Updated LICENS...	9 months ago
README.md	complex PRs will ...	last week
mypy.ini	back to 6.54GB f...	2 months ago
push_pypi....	push pypi	4 years ago
ruff.toml	ruff checks the m...	last month
run_multib...	convert $@ to "$@...	6 months ago

You like pytorch? You like micrograd? You love tinygrad! ❤️

Readme
MIT license
Activity
Custom properties
22.1k stars
272 watching
2.9k forks
Report repository

Releases 5

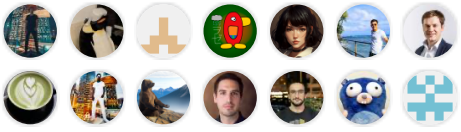
tinygrad 0.8.0 Latest
2 weeks ago

+ 4 releases

Packages

No packages published

Contributors 269



+ 255 contributors

Languages

Python 79.0% C 17.9%
C++ 1.4% Objective-C++ 0.9%

setup.py	Remove webgpu, ...	3 weeks ago
strip_white...	strip whitespace	7 months ago
sz.py	fix some long line...	3 weeks ago

Shell 0.4% Assembly 0.1%
Other 0.3%

README MIT license

tinygrad: For something between PyTorch and karpathy/micrograd. Maintained by tiny corp.

Homepage | Documentation | Examples | Showcase | Discord

This may not be the best deep learning framework, but it is a deep learning framework.

Due to its extreme simplicity, it aims to be the easiest framework to add new accelerators to, with support for both inference and training. If XLA is CISC, tinygrad is RISC.

tinygrad is still alpha software, but we raised some money to make it good. Someday, we will tape out chips.

Features

LLaMA and Stable Diffusion

tinygrad can run LLaMA and Stable Diffusion!

Laziness

Try a matmul. See how, despite the style, it is fused into one kernel with the power of laziness.

```
DEBUG=3 python3 -c "from tinygrad import Tensor;
N = 1024; a, b = Tensor.rand(N, N), Tensor.rand(N, N);
c = (a.reshape(N, 1, N) * b.T.reshape(1, N, N)).sum(axis=2);
print((c.numpy() - (a.numpy() @ b.numpy())).mean())"
```

And we can change `DEBUG` to `4` to see the generated code.

Neural networks

As it turns out, 90% of what you need for neural networks are a decent autograd/tensor library. Throw in an optimizer, a data loader, and some compute, and you have all you need.

```
from tinygrad import Tensor, nn

class LinearNet:
  def __init__(self):
    self.l1 = Tensor.kaiming_uniform(784, 128)
    self.l2 = Tensor.kaiming_uniform(128, 10)
  def __call__(self, x:Tensor) -> Tensor:
    return x.flatten(1).dot(self.l1).relu().dot(self.l2)

model = LinearNet()
optim = nn.optim.Adam([model.l1, model.l2], lr=0.001)

x, y = Tensor.rand(4, 1, 28, 28), Tensor([2,4,3,7])  # replace with real mnist dataloader

for i in range(10):
  optim.zero_grad()
  loss = model(x).sparse_categorical_crossentropy(y).backward()
  optim.step()
  print(i, loss.item())
```

See examples/beautiful_mnist.py for the full version that gets 98% in ~5 seconds

Accelerators

tinygrad already supports numerous accelerators, including:

- [x] CPU
- [x] GPU (OpenCL)
- [x] C Code (Clang)
- [x] LLVM
- [x] METAL
- [x] CUDA
- [x] PyTorch
- [x] HIP

And it is easy to add more! Your accelerator of choice only needs to support a total of ~25 low level ops. More

And it is easy to add more! Your accelerator of choice only needs to support a total of ~25 low level ops. More information can be found in the documentation for adding new accelerators.

Installation

The current recommended way to install tinygrad is from source.

From source

```
git clone https://github.com/tinygrad/tinygrad.git
cd tinygrad
python3 -m pip install -e .
```

Direct (master)

```
python3 -m pip install git+https://github.com/tinygrad/tinygrad.git
```

Documentation

Documentation along with a quick start guide can be found in the docs/ directory.

Quick example comparing to PyTorch

```
from tinygrad import Tensor

x = Tensor.eye(3, requires_grad=True)
y = Tensor([[2.0,0,-2.0]], requires_grad=True)
z = y.matmul(x).sum()
z.backward()

print(x.grad.numpy())  # dz/dx
print(y.grad.numpy())  # dz/dy
```

The same thing but in PyTorch:

```
import torch

x = torch.eye(3, requires_grad=True)
y = torch.tensor([[2.0,0,-2.0]], requires_grad=True)
z = y.matmul(x).sum()
z.backward()

print(x.grad.numpy())  # dz/dx
print(y.grad.numpy())  # dz/dy
```

Contributing

Contributing

There has been a lot of interest in tinygrad lately. Following these guidelines will help your PR get accepted.

We'll start with what will get your PR closed with a pointer to this section:

- No code golf! While low line count is a guiding light of this project, anything that remotely looks like code golf will be closed. The true goal is reducing complexity and increasing readability, and deleting `\n` s does nothing to help with that.
- All docs and whitespace changes will be closed unless you are a well-known contributor. The people writing the docs should be those who know the codebase the absolute best. People who have not demonstrated that shouldn't be messing with docs. Whitespace changes are both useless *and* carry a risk of introducing bugs.
- Anything you claim is a "speedup" must be benchmarked. In general, the goal is simplicity, so even if your PR makes things marginally faster, you have to consider the tradeoff with maintainablity and readablity.
- In general, the code outside the core `tinygrad/` folder is not well tested, so unless the current code there is broken, you shouldn't be changing it.
- If your PR looks "complex", is a big diff, or adds lots of lines, it won't be reviewed or merged. Consider breaking it up into smaller PRs that are individually clear wins. A common pattern I see is prerequisite refactors before adding new functionality. If you can (cleanly) refactor to the point that the feature is a 3 line change, this is great, and something easy for us to review.

Now, what we want:

- Bug fixes (with a regression test) are great! This library isn't 1.0 yet, so if you stumble upon a bug, fix it, write a test, and submit a PR, this is valuable work.
- Solving bounties! tinygrad offers cash bounties for certain improvements to the library. All new code should be high quality and well tested.
- Features. However, if you are adding a feature, consider the line tradeoff. If it's 3 lines, there's less of a bar of usefulness it has to meet over something that's 30 or 300 lines. All features must have regression tests. In general with no other constraints, your feature's API should match torch or numpy.
- Refactors that are clear wins. In general, if your refactor isn't a clear win it will be closed. But some refactors are amazing! Think about readability in a deep core sense. A whitespace change or moving a few functions around is useless, but if you realize that two 100 line functions can actually use the same 110 line function with arguments while also improving readability, this is a big win.
- Tests/fuzzers. If you can add tests that are non brittle, they are welcome. We have some fuzzers in here too, and there's a plethora of bugs that can be found with them and by improving them. Finding bugs, even writing broken tests (that should pass) with `@unittest.expectedFailure` is great. This is how we make progress.
- Dead code removal from core `tinygrad/` folder. We don't care about the code in extra, but removing dead code from the core library is great. Less for new people to read and be confused by.

Running tests

You should install the pre-commit hooks with `pre-commit install`. This will run the linter, mypy, and a subset of the tests on every commit.

For more examples on how to run the full test suite please refer to the CI workflow

For more examples on how to run the full test suite please refer to the CI workflow.

Some examples of running tests locally:

```
python3 -m pip install -e '.[testing]'  # install extra deps for testing
python3 test/test_ops.py                # just the ops tests
python3 -m pytest test/                 # whole test suite
```