HOW USABILITY IMPROVES PERFORMANCE IN PYTORCH

PRESENTED BY
ZACHARY DEVITO

IN MY OWN OPINION

Why was PyTorch successful?

- Performance? No, original design allowed 20% slowdown for a better API
- Innovative new algorithm? No, adopted autograd model popular in Chainer, DyNet, and autograd package

IN MY OWN OPINION

Why was PyTorch successful?

Laser-focused on usability

Eager mode by default

A graph is created on the fly

W_h h W_x x

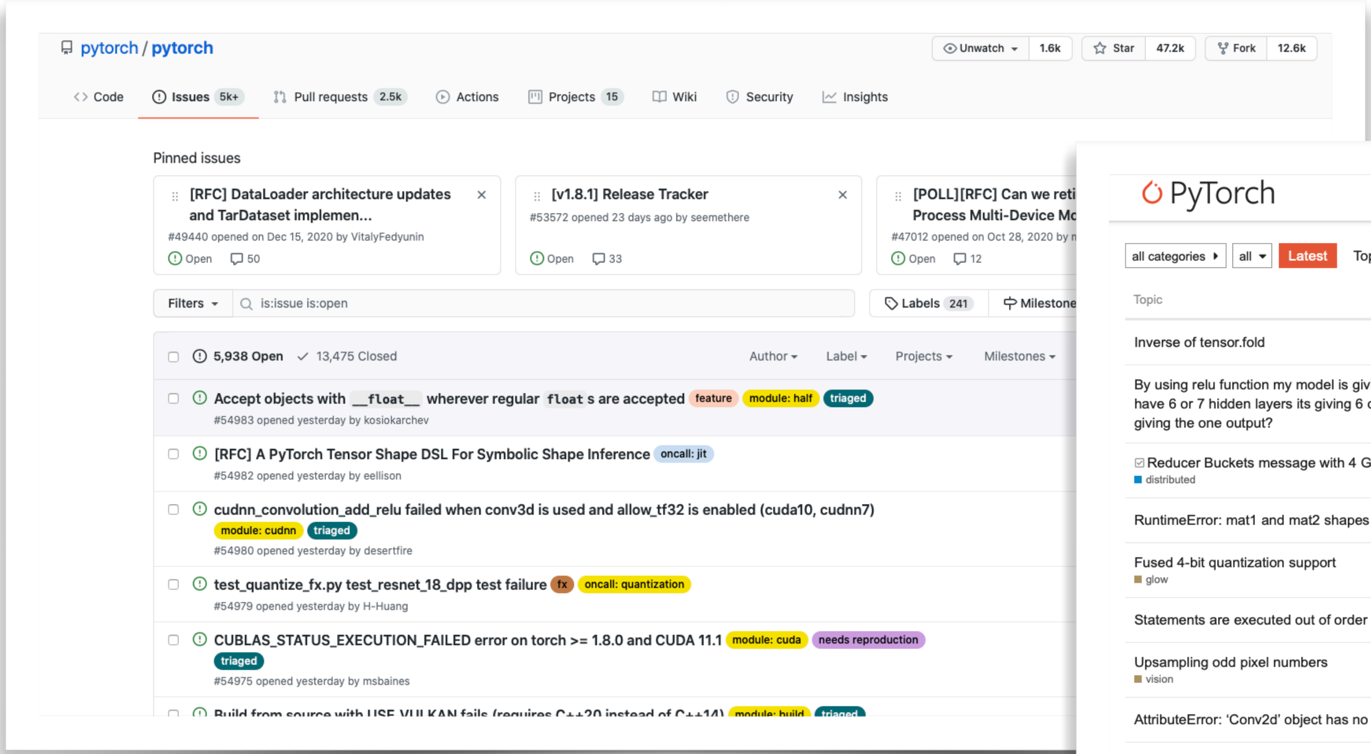
```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

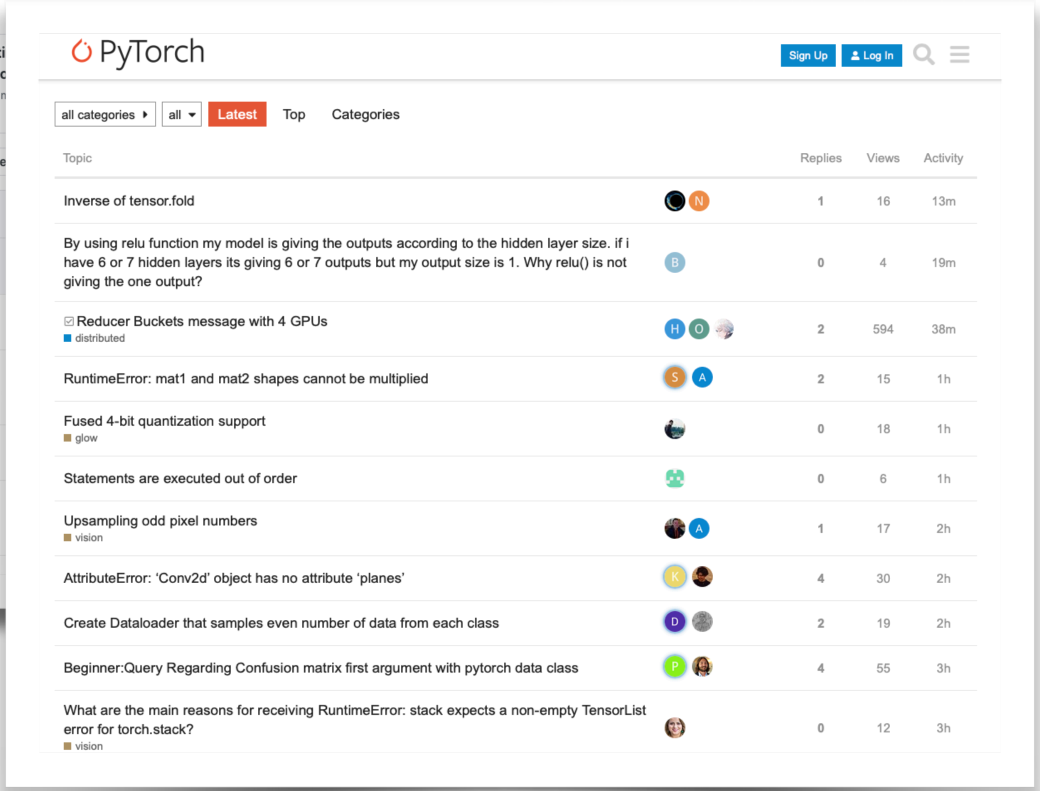
+

Bindings for SOTA algorithms: CUDNN, BLAS, Intel MKL

LASER FOCUSED ON USABILITY

< 24 hour response time on GitHub issues and forums





At the time, competitors gained little with graph-mode but did reduce usability

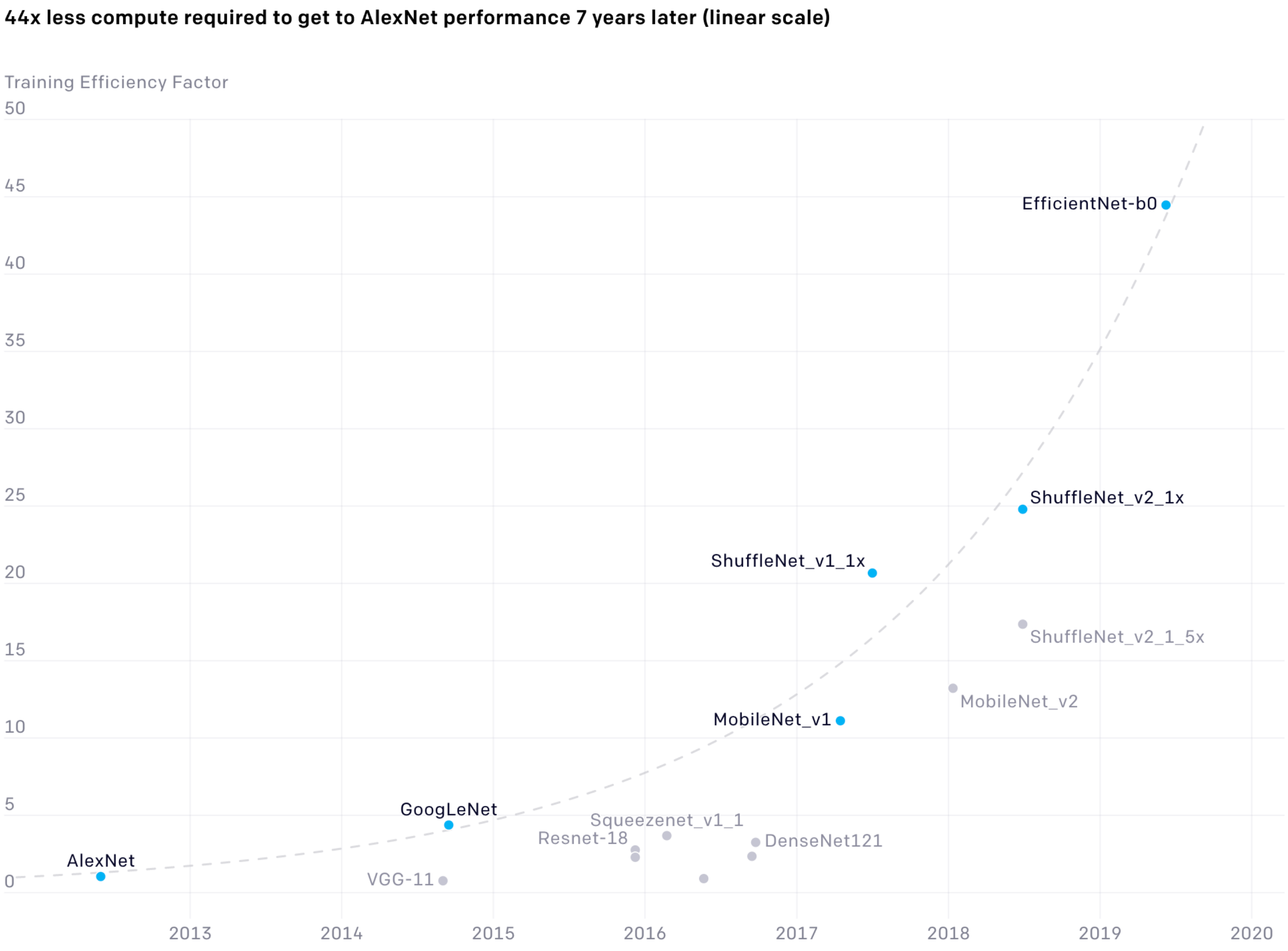
Framework	*Throughput (higher is better)*					
	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
Chainer	778 ± 15	N/A	$\mathbf{219} \pm 1$	N/A	N/A	N/A
CNTK	845 ± 8	84 ± 3	210 ± 1	N/A	N/A	N/A
MXNet	$\mathbf{1554} \pm 22$	113 ± 1	218 ± 2	444 ± 2	N/A	N/A
PaddlePaddle	933 ± 123	112 ± 2	192 ± 4	$\mathbf{557} \pm 24$	N/A	N/A
TensorFlow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	$9631 \pm 1.3\%$	$4.8e6 \pm 2.9\%$
PyTorch	1547 ± 316	$\mathbf{119} \pm 1$	212 ± 2	463 ± 17	$\mathbf{15512} \pm 4.8\%$	$\mathbf{5.4e6} \pm 3.4\%$

Table 1: Training speed for 6 models using 32bit floats. Throughput is measured in images per second for the AlexNet, VGG-19, ResNet-50, and MobileNet models, in tokens per second for the GNMTv2 model, and in samples per second for the NCF model. The fastest speed for each model is shown in bold.

[Paszke et al, NeurIPS 2019, https://arxiv.org/abs/1912.01703]

Productivity vs(?) Performance

Productivity *enables* Performance



[Hernandez and Brown, https://arxiv.org/abs/2005.04305]

The Usability Crisis of Accelerators

Don’t compromise usability for *potential* performance gains.

Empower users to fix any potential performance issues with incrementally increasing effort.

THIS TALK

01

CASE STUDY: FIXED SIZES AND USABILITY

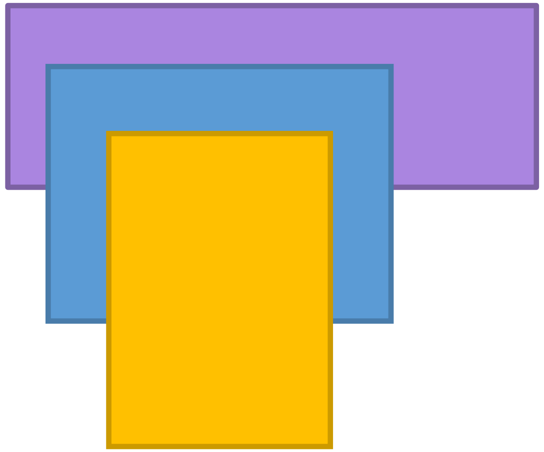
02

UPCOMING TOOLS IN PYTORCH FOR USABILITY

REAL NETWORKS DO NOT ALWAYS HAVE FIXED SIZES

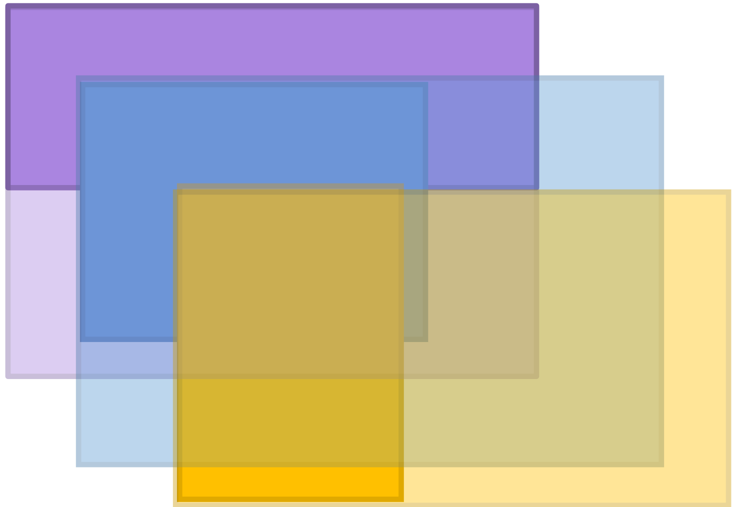
… BUT MANY LIBRARIES DO

Vision: Images are not the same size, but batches are rectilinear



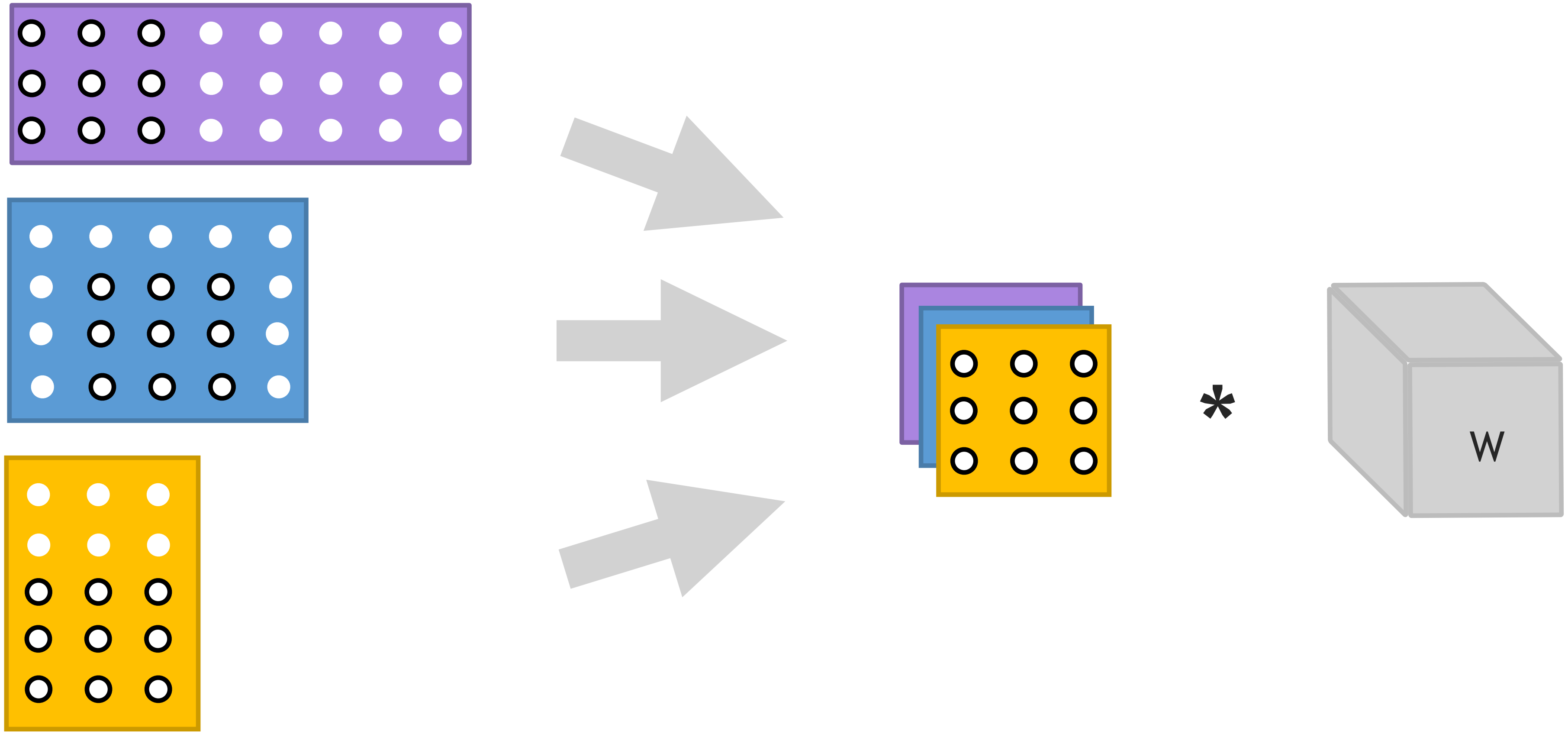
Batch

Scaling - have to check accuracy



Padding - wastes compute

But conv is the same at each pixel



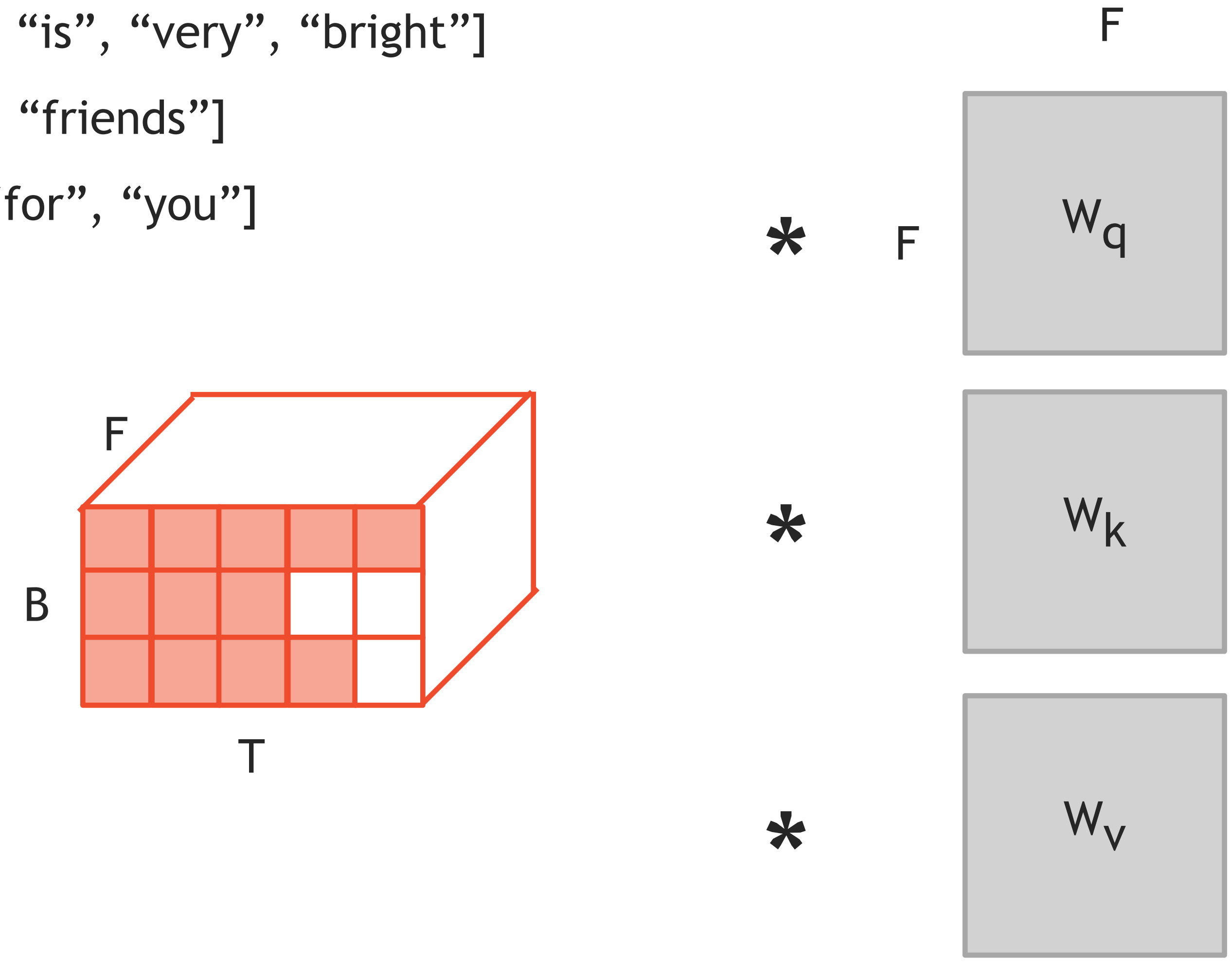
Should the conv primitive have a non-rectilinear batch instead?

Text: Sequences are not the same length

["The", "sun", "is", "very", "bright"]

["Let's", "be", "friends"]

["It", "was", "for", "you"]



Transforms have lots of per-word arithmetic

Effective Transformer: flatten/unflatten for per-word operations

```
__global__ void compress_bert_input(
    const __half* from_tensor, const int* mask, const int* prefix_sum,
    __half* to_tensor, int* batch_idx, int* word_idx,
    int batch_size , int seq_len, int hidden_dim) {
  int bid = blockIdx.y;  // batch
  int wid = blockIdx.x;  // word
  int tid = threadIdx.x; //

  /// 1. count pos for from tensor
  int mask_idx  = bid * seq_len + wid;

  if (mask[mask_idx] > 0.5) {
    int valid_idx = prefix_sum[mask_idx];

    /// 2. write batch id and word id for each word
    if (tid == 0) {
      batch_idx[valid_idx] = bid;
      word_idx[valid_idx]  = wid;
    }

    /// 3. copy src data
    half2* src_ptr = (half2*)from_tensor;
    half2* dst_ptr = (half2*)to_tensor;
    int src_idx = mask_idx  * hidden_dim + tid;
    int dst_idx = valid_idx * hidden_dim + tid;
    dst_ptr[dst_idx] = src_ptr[src_idx];
  }
}
```

https://github.com/bytedance/effective_transformer

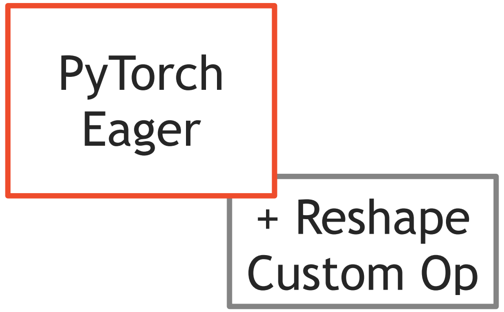
Effective Transformer: flatten/unflatten for per-word operations

Tesla V100, float16, maximum sequence length=32, average serquence length≈20

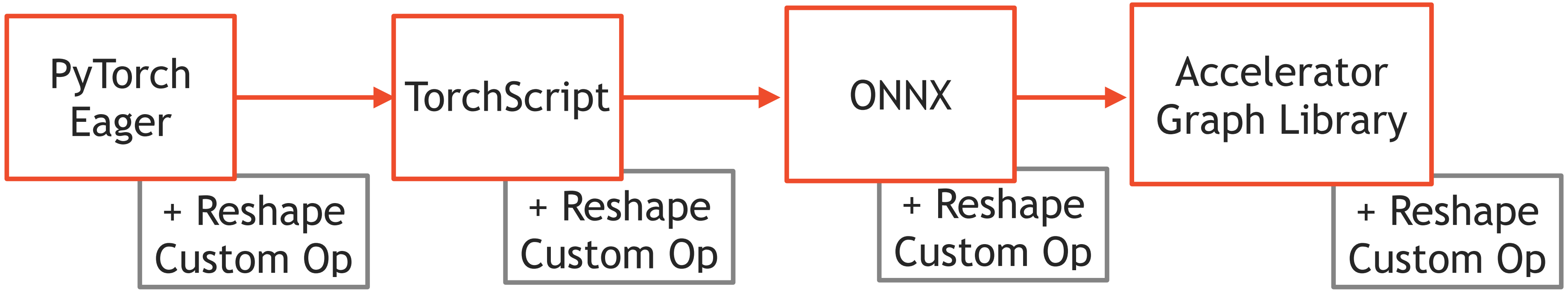
batch_size	XLA (in ms)	Faster Transformer (in ms)	Speedup over XLA	Effective Transformer (in ms)	Speedup over XLA
100	15.08	10.39	1.45	8.75	1.72
200	28.08	19.64	1.43	15.32	1.83
300	41.37	29.65	1.40	22.18	1.86
400	53.65	38.52	1.39	28.31	1.89
500	66.86	48.13	1.39	33.08	2.02
1000	131.46	95.01	1.38	64.34	2.04

https://github.com/bytedance/effective_transformer

It is harder to add custom behavior in deeper stack

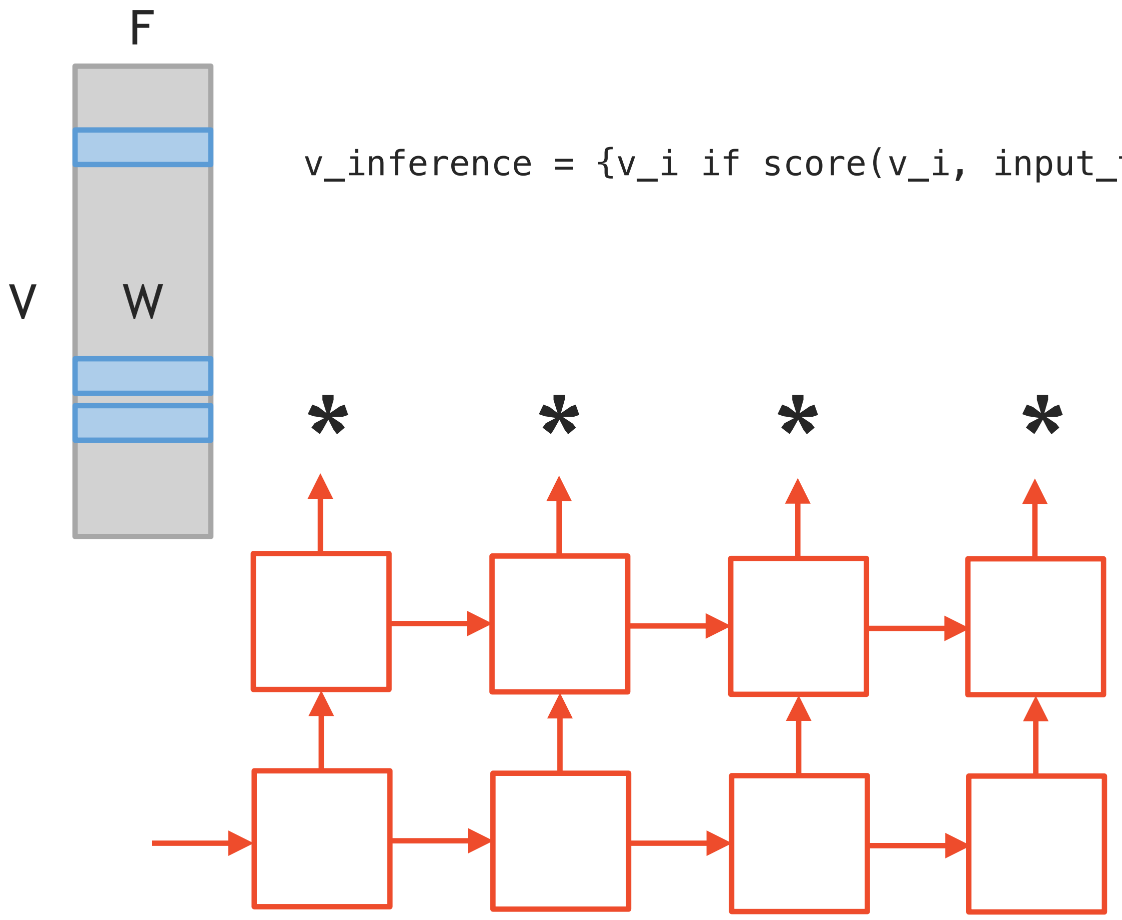


1 custom API to understand



4 custom APIs to understand:
Optimistic case: possible, but can degrade optimization techniques
Pessimistic case: layer doesn’t allow for a the custom op, user is blocked.

Vocab Reduction: varying vocab size




```
v_inference = {v_i if score(v_i, input_tokens) > threshold}
```

CPU decoding up to 10x faster.

[L'Hostis et al., https://arxiv.org/abs/1610.00072]

A surprising amount of dynamic behavior and sizes occur in real world models.

So when is it ok to restrict dynamic behavior to get better performance?

RULES OF THUMB FOR REDUCING USABILITY

— Add restrictions when there are already-realized performance gains. (e.g. 30% slower without using half precision)

— When there are theoretical gains (e.g. if sizes are known we can allocate memory ahead of time, and do layout planing), err on the side of usability.

Why? There are 10x more external than internal developers, and each restriction prohibits them from exploring application-specific optimizations or adds friction

USABILITY IN PYTORCH FOR PRODUCTION

USER STUDY OF TORCHSCRIPT USERS

— Capture the structure of PyTorch programs to do custom transforms

— Create self-contained archives of trained PyTorch programs for transfer learning, or to deploy in a production environment

— Serve models as part of a service (e.g. a language translation server)

— Improve the performance of these models

Can we decouple these uses to make each task easier?

torch.fx capture and transform PyTorch programs directly in Python

```python
class MyModule(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.param = torch.nn.Parameter(torch.rand(3, 4))
    self.linear = torch.nn.Linear(4, 5)

  def forward(self, x):
    return self.linear(x + self.param).clamp(min=0.0, max=1.0)

module = MyModule()

from torch.fx import symbolic_trace, GraphModule
# Symbolic tracing frontend - captures the semantics of the module
symbolic_traced: GraphModule = symbolic_trace(module)


# High-level intermediate representation (IR) - Graph representation
print(symbolic_traced.graph)
"""
graph(x):
    %param : [#users=1] = self.param
    %add_1 : [#users=1] = call_function[target=<built-in function add>](args = (%x, %param), kwargs = {})
    %linear_1 : [#users=1] = call_module[target=linear](args = (%add_1,), kwargs = {})
    %clamp_1 : [#users=1] = call_method[target=clamp](args = (%linear_1,), kwargs = {min: 0.0, max: 1.0})
    return clamp_1
"""
```

OVERVIEW

`torch.fx`

SYMBOLIC TRACING

Construct IR by tracing the execution of a PyTorch model, similar to TF Autograph and JAX's jaxpr tracing.

SIMPLE HIGH-LEVEL IR

4 instruction IR that represents PyTorch programs using the publicly documented PyTorch operators.

PYTHON CODE GENERATION

After you are done working with the IR, you can transform it back to Python code and use it in eager mode, or pass it TorchScript to improve performance

`torch.package` self-contained eager-mode models

```
tacotron2
├── data # tensor data
│   ├── 94163177428688
│   └── ...
├── extern_modules # from `extern` command
├── librosa # mocked out library
│   ├── filters.py
│   └── util
│       └── __init__.py
├── _mock.py
├── model
│   ├── example.pkl # pickled example
│   └── model.pkl # pickled model
├── numpy # mocked out library
│   └── __init__.py
├── scipy
│   ├── io
│   │   └── wavfile.py
│   └── signal
│       └── __init__.py
├── tacotron2 # captured model code
│   ├── audio_processing.py
│   ├── layers.py
│   ├── model.py
│   ├── stft.py
│   └── utils.py
└── version
```

(a) Packaged model structure

```python
from torch.package import PackageExporter

model, example_input = load_tacotron2()

with PackageExporter('tacotron2') as e:
  # Configure how to export the source code
  e.extern(['torch.**'])
  # instead of saving the source code for the
  # torch libraries, let the package link to
  # the libraries of the loading process.

  # Replace these modules with mock
  # implementations. They are not
  # actually used.
  e.mock(['numpy', 'librosa.**', 'scipy.**'])


  e.save_pickle('model', 'model.pkl', model)
  # dependency resolution will walk
  # the pickled objects and find all the
  # required source files

  # also save example tensor for testing
  e.save_pickle('model', 'eg.pkl',
                example_input)
```

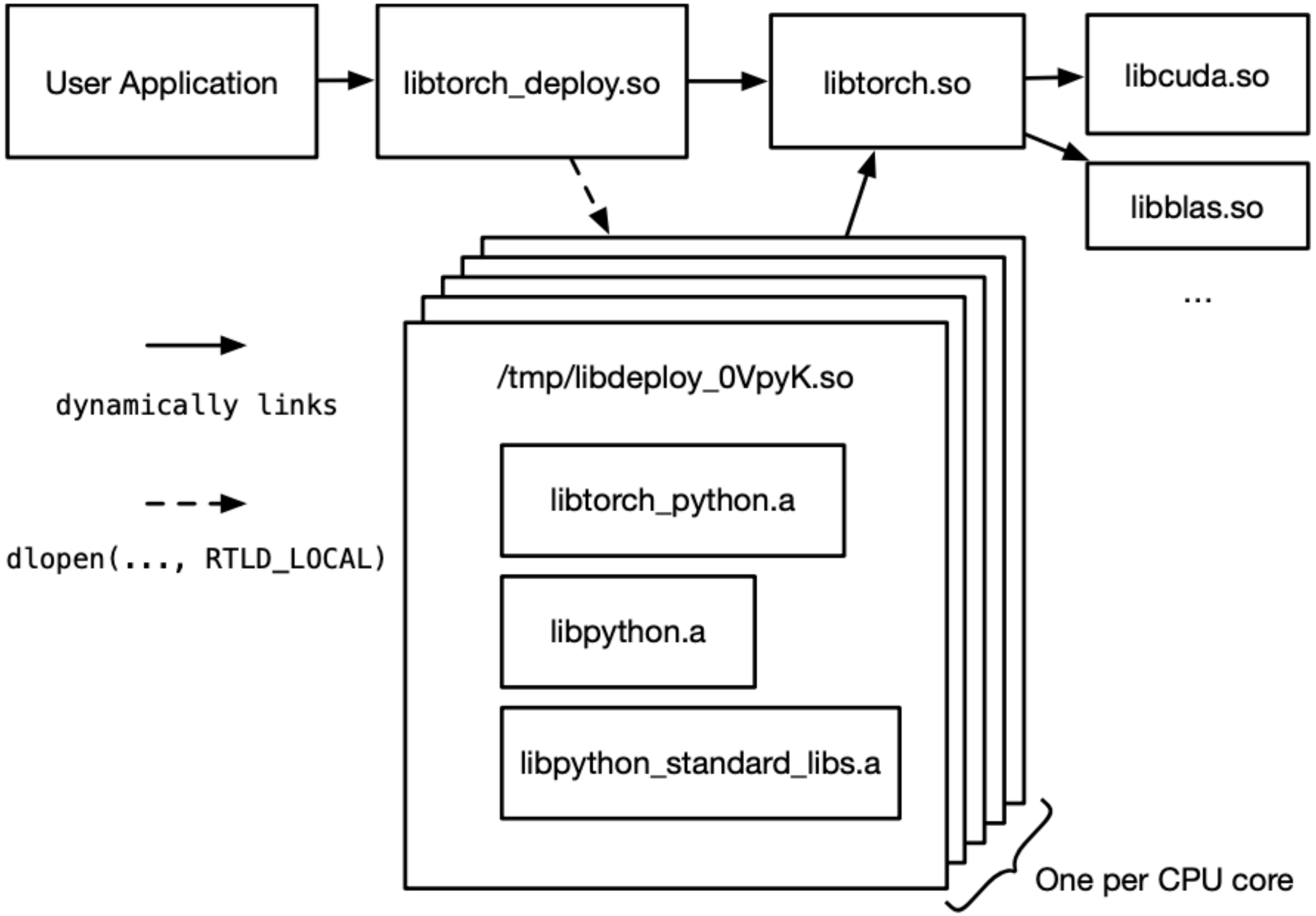
(b) Model export

```python
from torch.package import PackageImporter

i = PackageImporter('tacotron2')

# code for the model is loaded from the model_file
# rather than the normal import system, except
# where packages are marked as extern.
model = i.load_pickle('model', 'model.pkl')
example_inputs = i.load_pickle('model', 'eg.pkl')
# test the model
model(*example_inputs)
```

(c) Model import

[https://arxiv.org/abs/2104.00254]

`torch::deploy` a native library for running packaged models



[https://arxiv.org/abs/2104.00254]

PICK THE RIGHT TOOLS FOR YOUR PROBLEM

— Capture the structure of PyTorch programs: torch.fx

— Create self-contained archives of trained PyTorch programs: torch.package

— Serve models as part of a service: torch::deploy

— Improve the performance of these models: TorchScript

… AND ADD YOUR OWN TOOLS TO THE ECOSYSTEM

— e.g use torch.package to save a PyTorch model that uses TVM to construct custom operators

— e.g use torch.fx to extract a PyTorch program, and write a transformer to run it on new accelerator hardware.

Keep framework users in the loop about hardware performance, empower them to fix performance issues with incrementally increasing effort.

As part of PyTorch, we are trying to build tools to increase usability and lower the friction of getting models into production. Let us know how we can help.

RESOURCES

- torch.fx https://pytorch.org/docs/stable/fx.html
- torch.package and deploy: https://arxiv.org/abs/2104.00254
- TorchScript: https://pytorch.org/docs/stable/jit.html

SPECIAL THANKS TO PYTORCH CONTRIBUTORS WORKING ON OUR USABILITY PROJECTS: JASON ANSEL, WILL CONSTABLE, HORACE HE, JAMES REED, MICHAEL SUO, ANSLEY USSERY, AILING ZHANG