

## HOW USABILITY IMPROVES PERFORMANCE IN PYTORCH

PRESENTED BY ZACHARY DEVITO Ú

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

Ú

IN MY OWN OPINION

## Why was PyTorch successful?

## Laser-focused on usability

### **C** LASER FOCUSED ON USABILITY

### Eager mode by default

#### A graph is created on the fly



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



Ü

+

# Ċ

### LASER FOCUSED ON USABILITY

### < 24 hour response time on GitHub issues and forums

<> Code	() Issues 5k+ 11 Pull requests 2.5k () Actions	Projects 15 🛄 Wiki 🕕 Security	Insights					
	Pinned issues		6					
	Image: [RFC] DataLoader architecture updates         ×           and TarDataset implemen         #49440 opened on Dec 15, 2020 by VitalyFedyunin	IVI.8.1] Release Tracker #63572 opened 23 days ago by seemethere	× [POLL][RFC] Can we reti Process Multi-Device Mc #47012 opened on Oct 28, 2020 by n	<sup>(</sup> ) PyTorch		Sign Up	🛓 Log In	Q =
	Open      50  Filters → Q is:issue is:open	🕐 Open 🖓 33	<ul> <li>Open □ 12</li> <li>Labels 241 ♀ Milestone</li> </ul>	all categories  all  Latest Top Categories Topic		Replies	Views	Activit
	□ ① <b>5,938 Open</b> ✓ 13,475 Closed	Author - L	abel • Projects • Milestones •	Inverse of tensor.fold	0	1	16	13m
	O Accept objects withfloat wherever regular float s are accepted feature module: half triaged #54983 opened yesterday by kosiokarchev			By using relu function my model is giving the outputs according to the hidden layer size. if have 6 or 7 hidden layers its giving 6 or 7 outputs but my output size is 1. Why relu() is not giving the one output?		0	4	19m
	(Image: Constant of the second s	bolic Shape Inference oncall: jit		Reducer Buckets message with 4 GPUs     distributed	H 🗿 🥪	2	594	38m
	<ul> <li>① cudnn_convolution_add_relu failed when con module: cudnn triaged</li> </ul>	v3d is used and allow_tf32 is enabled (cuda10, cu	udnn7)	RuntimeError: mat1 and mat2 shapes cannot be multiplied	<b>(S)</b> (A)	2	15	1h
	#54980 opened yesterday by desertfire	iailure (fx) oncall: quantization		Fused 4-bit quantization support	6	0	18	1h
	#54979 opened yesterday by H-Huang  CUBLAS_STATUS_EXECUTION_FAILED error on torch >= 1.8.0 and CUDA 11.1 (module: cuda) (needs reproduction) (riaged) #54975 opened yesterday by msbaines		eds reproduction	Statements are executed out of order	2	0	6	1h
				Upsampling odd pixel numbers vision	۸ 📽	1	17	2h
	O Build from course with USE VIII KAN faile (requires C++20 instead of C++11) (module build) (related			AttributeError: 'Conv2d' object has no attribute 'planes'	۵ ک	4	30	2h
				Create Dataloader that samples even number of data from each class	0	2	19	2h
				Beginner:Query Regarding Confusion matrix first argument with pytorch data class	<b>P</b>	4	55	3h
				What are the main reasons for receiving RuntimeError: stack expects a non-empty Tensor error for torch.stack?	.ist	0	12	3h

### LASER FOCUSED ON USABILITY

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

Enomorroule	Throughput (higher is better)							
Framework	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF		
Chainer	$778 \pm 15$	N/A	$219 \pm 1$	N/A	N/A	N/A		
CNTK	$845\pm8$	$84\pm3$	$210 \pm 1$	N/A	N/A	N/A		
MXNet	$1554 \pm 22$	$113 \pm 1$	$218\pm2$	$444 \pm 2$	N/A	N/A		
PaddlePaddle	$933 \pm 123$	$112 \pm 2$	$192 \pm 4$	$557 \pm 24$	N/A	N/A		
TensorFlow	$1422\pm27$	$66 \pm 2$	$200 \pm 1$	$216 \pm 15$	$9631\pm1.3\%$	$4.8e6 \pm 2.9\%$		
PyTorch	$1547\pm316$	$\pmb{119} \pm 1$	$212\pm2$	$463\pm17$	$\textbf{15512} \pm 4.8\%$	$\textbf{5.4e6} \pm 3.4\%$		

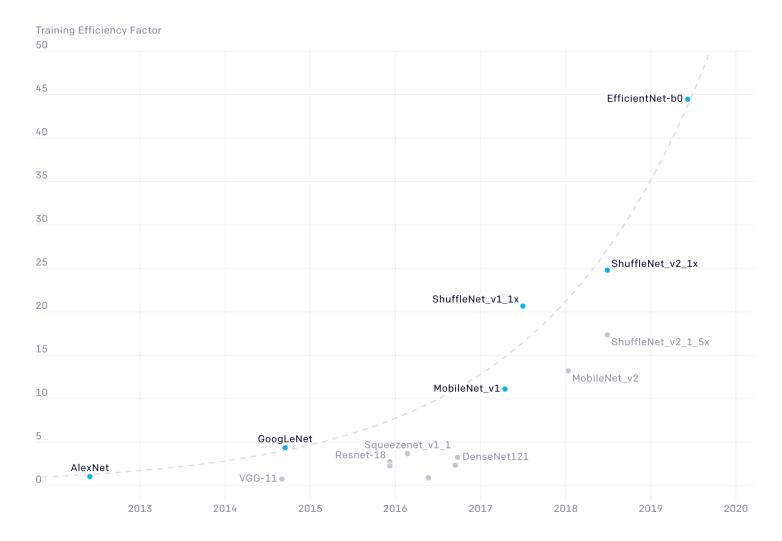
**Table 1:** Training speed for 6 models using 32bit floats. Throughput is measured in images per second for the AlexNet, VGG-19, ResNet-50, and MobileNet models, in tokens per second for the GNMTv2 model, and in samples per second for the NCF model. The fastest speed for each model is shown in bold.

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

# Productivity vs(?) Performance

## Productivity enables Performance

#### 44x less compute required to get to AlexNet performance 7 years later (linear scale)



[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.

Ü

### THIS TALK

01 CASE STUDY: FIXED SIZES AND USABILITY 02 UPCOMING TOOLS IN PYTORCH FOR USABILITY

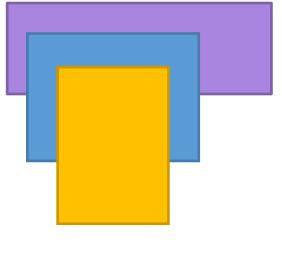


## REAL NETWORKS DO NOT ALWAYS HAVE FIXED SIZES

... BUT MANY LIBRARIES DO



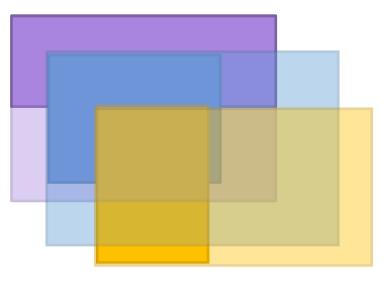
# Vision: Images are not the same size, but batches are rectilinear



Batch

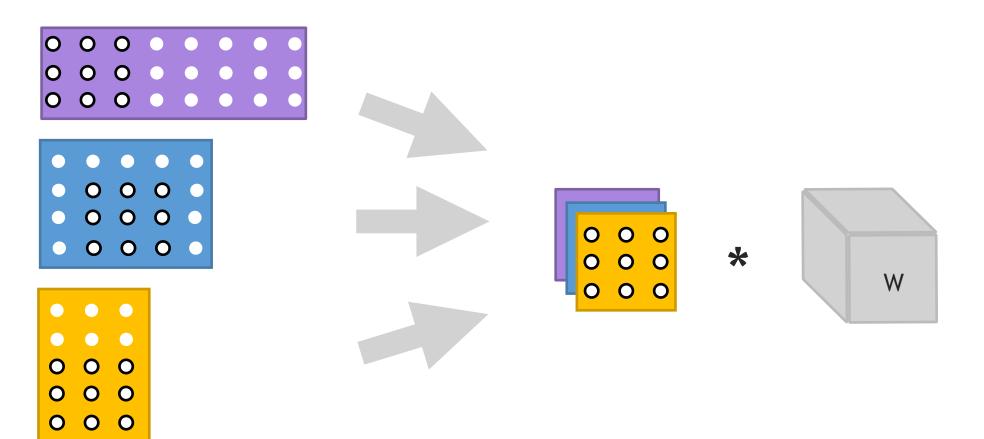


Scaling - have to check accuracy



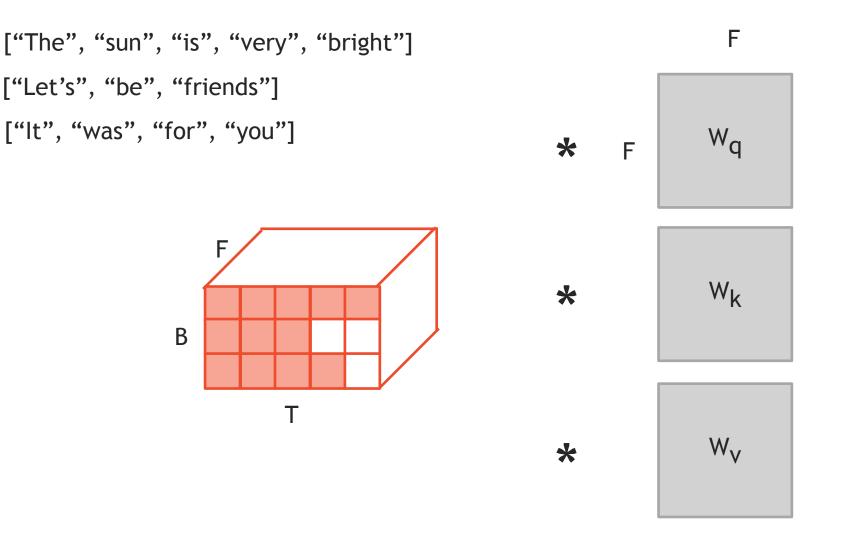
Padding - wastes compute

O But conv is the same at each pixel



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

## O Text: Sequences are not the same length



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

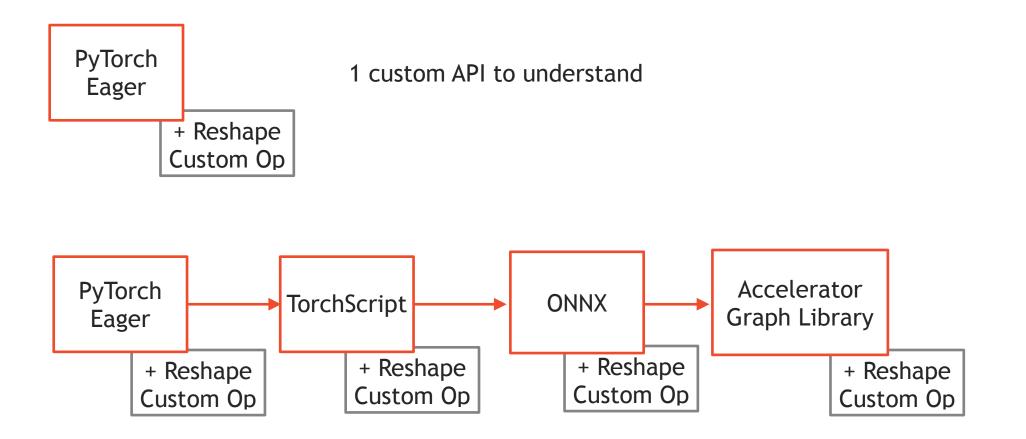
# Effective Transformer: flatten/unflatten for per-word operations

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

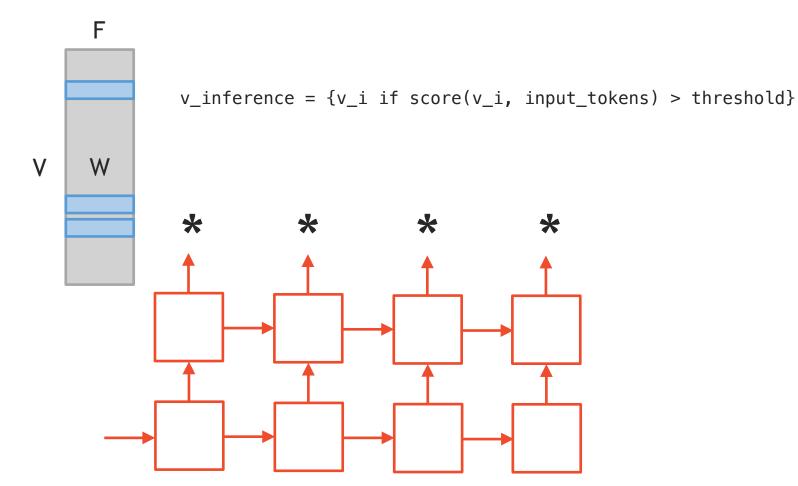
# <sup>()</sup> It is harder to add custom behavior in deeper stack



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.

## **O** Vocab Reduction: varying vocab size



CPU decoding up to 10x faster.

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

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

Ü

## 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?

## **O** torch.fx capture and transform PyTorch programs directly in 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
```

### $\mathsf{O} \mathsf{V} \mathsf{E} \mathsf{R} \mathsf{V} \mathsf{I} \mathsf{E} \mathsf{W}$

## 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

(a) Packaged model structure

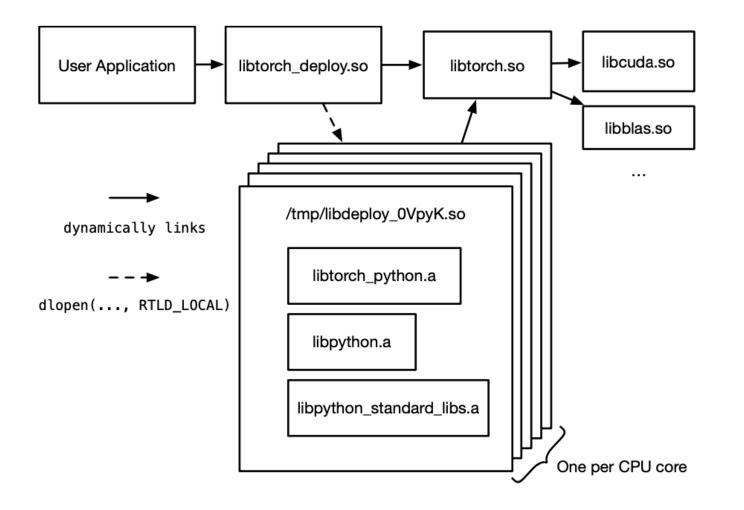
from torch.package import PackageExporter

```
model. example_input = load_tacotron2()
tacotron2
 — data # tensor data
     — 94163177428688
                                          with PackageExporter('tacotron2') as e:
    └─ ···
                                            # Configure how to export the source code
   extern_modules # from `extern` command
                                            e.extern(['torch.**'])
   librosa # mocked out library
                                            # instead of saving the source code for the
    — filters.py
    —— util
                                            # torch libraries, let the package link to
        ____init__.py
                                            # the libraries of the loading process.
   mock.py
   model
     — example.pkl # pickled example
                                            # Replace these modules with mock
    model.pkl # pickled model
                                            # implementations. They are not
   numpy # mocked out library
                                            # actually used.
    └── __init__.py
                                            e.mock(['numpy', 'librosa.**', 'scipy.**'])
                                                                                            from torch.package import PackageImporter
   scipy
       io
        └── wavfile.pv
                                                                                            i = PackageImporter('tacotron2')
       signal
                                            e.save_pickle('model', 'model.pkl', model)
         — ___init__.py
                                                                                            # code for the model is loaded from the model_file
                                            # dependency resolution will walk
   tacotron2 # captured model code
                                            # the pickled objects and find all the
                                                                                            # rather than the normal import system, except
       audio_processing.py
                                            # required source files
                                                                                            # where packages are marked as extern.
       layers.py
                                                                                            model = i.load_pickle('model', 'model.pkl')
       model.py
      - stft.py
                                            # also save example tensor for testing
                                                                                            example_inputs = i.load_pickle('model', 'eg.pkl')
       utils.py
                                            e.save_pickle('model', 'eg.pkl',
                                                                                            # test the model
   version
                                                          example_input)
                                                                                            model(*example_inputs)
```

(b) Model export

(c) Model import

## **O** 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.

## Ċ

### 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