ETH zürich

STen: An Interface for Efficient Sparsity in PyTorch

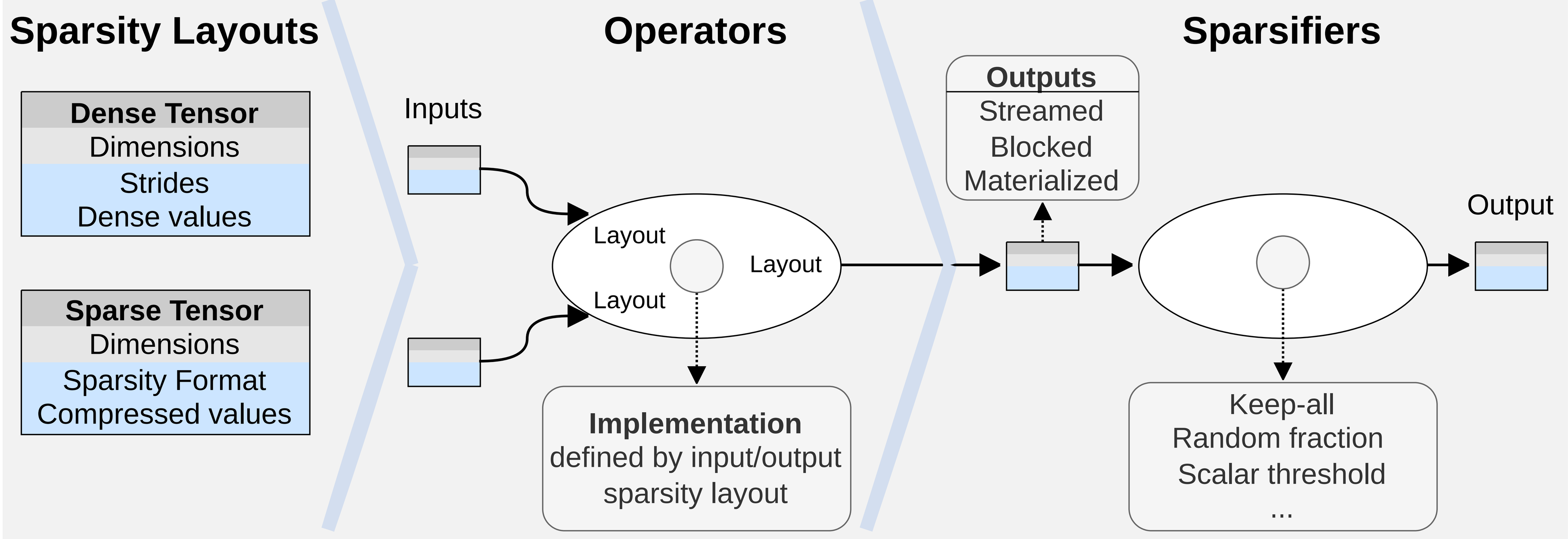
Andrei Ivanov, Nikoli Dryden, Torsten Hoefler
Department of Computer Science, ETH Zurich

State of the sparsity in PyTorch

- Plain COO – **slow** fine-grained n-dimensional tensors
- Hybrid COO – fast **blocked** n-dimensional tensors
- CSR – fast fine-grained **two**-dimensional tensors

Sparse operators: ~3% of all operators (not even convolution)
torch.autograd support: ~0.2% of all operators
No general pipeline for sparsity: no custom formats, no re-sparsifying in runtime, no control over sparsity in training.

Our programming model



Construct sparse model

```
import sten
sparse_add = sten.sparsified_op(
    orig_op=torch.add,
    out_fmt=[
        (fwd_inline_sparsifier, fwd_temp_format,
         fwd_external_sparsifier, fwd_out_format)],
    grad_out_fmt=[
        (bwd_inline_sparsifier, bwd_temp_format,
         bwd_external_sparsifier, bwd_out_format)])
```

Sparsify existing dense model

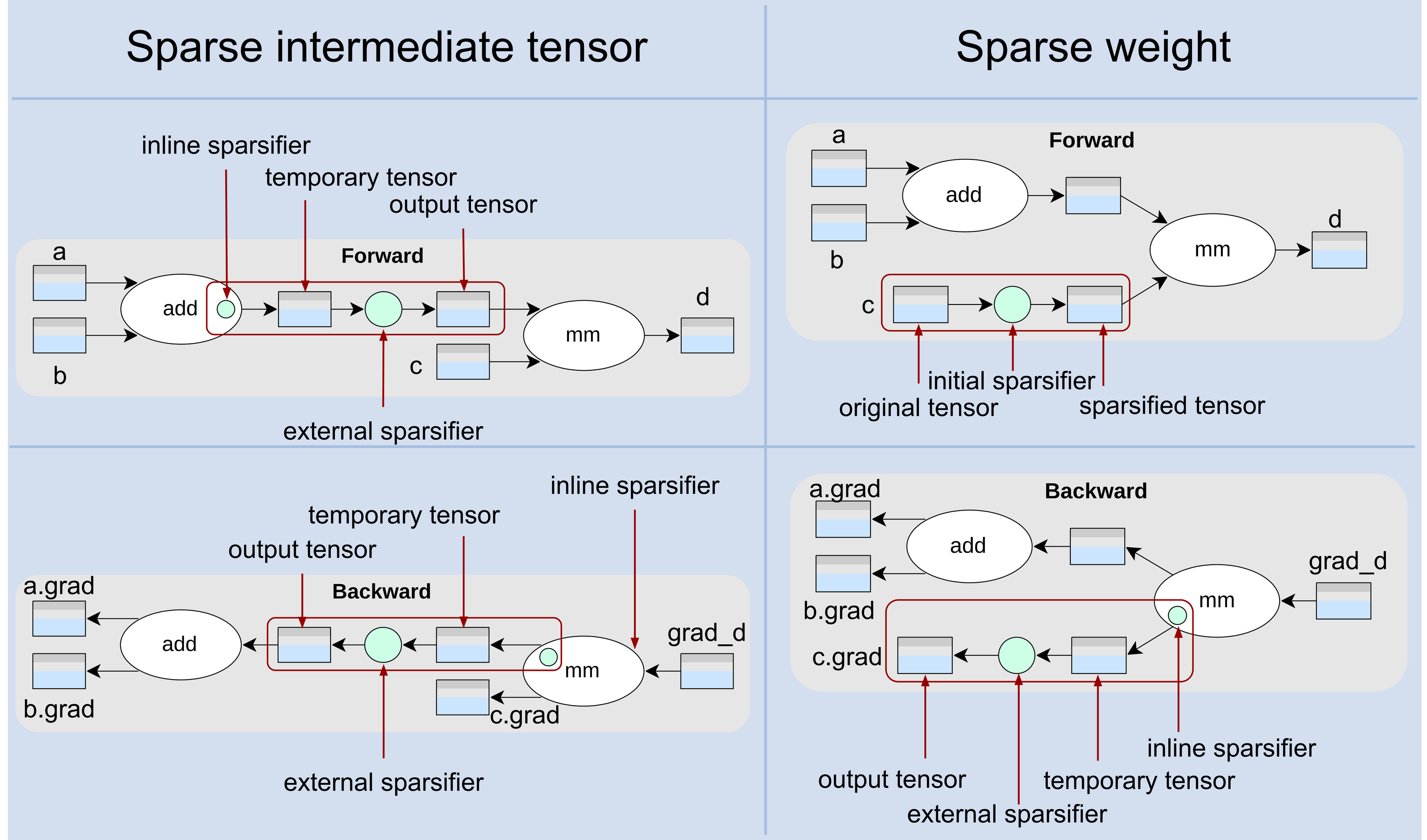
```
import sten
sb = sten.SparsityBuilder(model)
sb.set_weight('attention.self.query.weight',
    initial_sparsifier=
        sten.ScalarFractionSparsifier(0.9),
    out_format=sten.CsrTensor,
)
sparse_model = sb.get_sparse_model()
output = sparse_model(input)
```

Sparsifiers

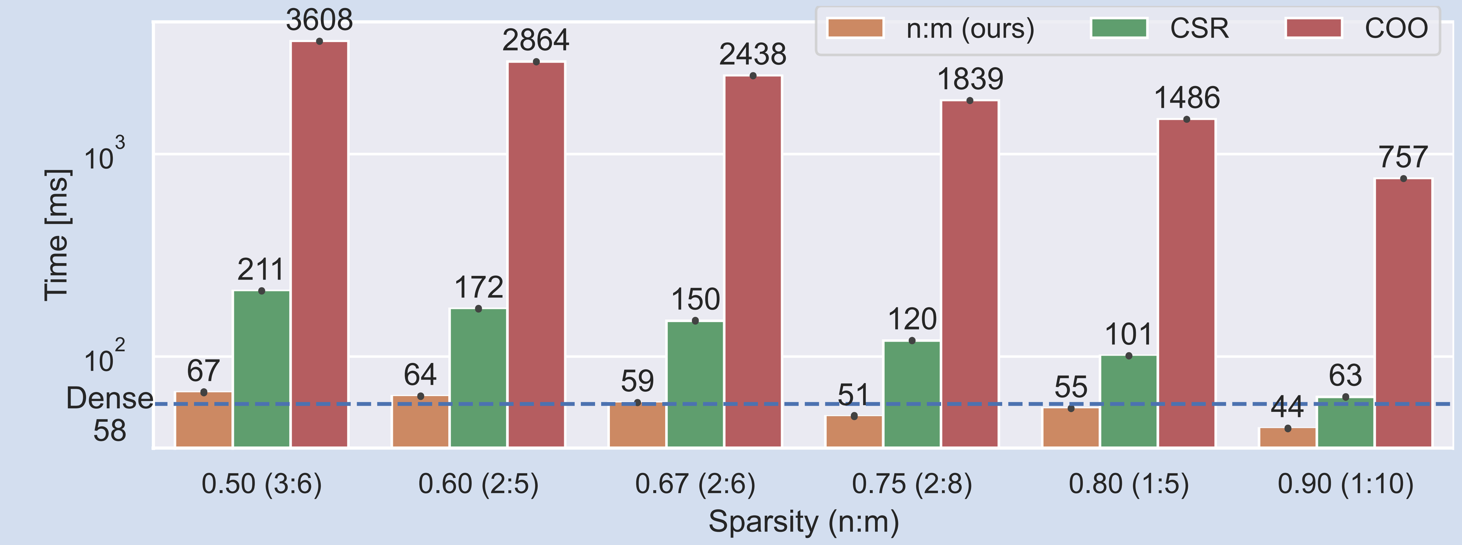
Sparsifier types and examples, the number of passes over a tensor made, their memory requirements (*nnz* total nonzeros, block size *b* when blocking), and sparsifier type. Some complex weight sparsifiers could be implemented more efficiently than with materialization.

Sparsifier	Examples	Passes	Memory	Type
Keep-all	Sparse add	1	$\mathcal{O}(1)$	streaming
Random fraction	Dropout	1	$\mathcal{O}(1)$	streaming
Scalar threshold	ReLU	1	$\mathcal{O}(1)$	streaming
Scalar fraction	Magnitude [4]	2	$\mathcal{O}(nnz)$	materializing
Block-wise fraction	Block magnitude [5]	2	$\mathcal{O}(nnz)$	materializing
Per-block fraction	n:m [3]	2	$\mathcal{O}(b)$	blocking
Complex weight sparsifiers	Movement, ℓ_0, etc. [6]	≥ 1	$\mathcal{O}(nnz)$	materializing

Implementation



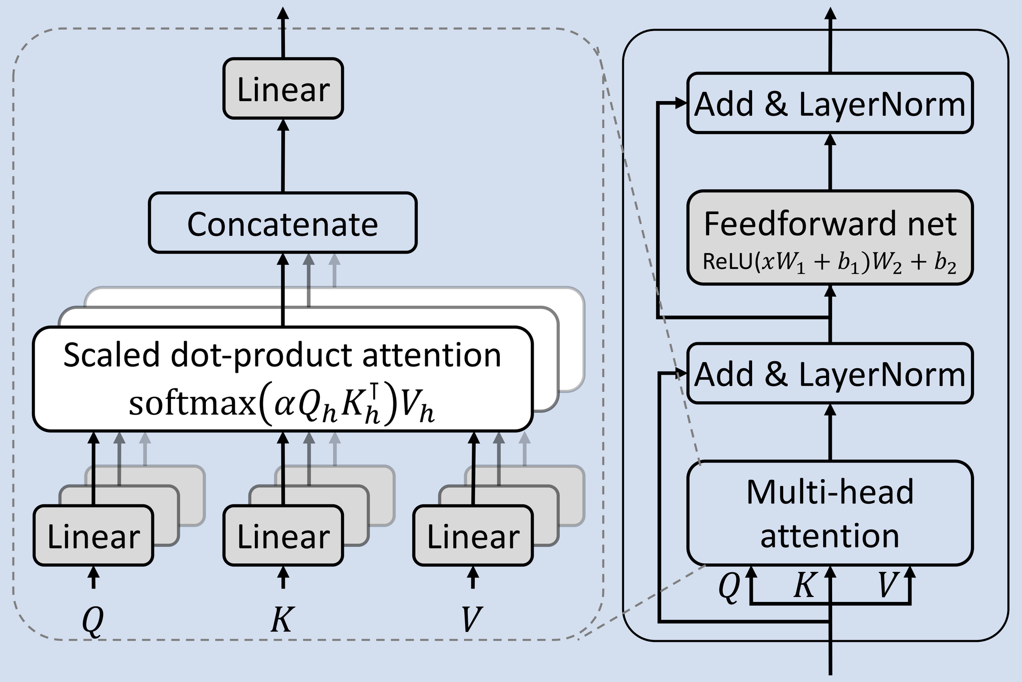
Evaluation



Top: Forward-pass runtime of a single BERT (base, uncased) [1] encoder layer from HuggingFace [2]. We sparsify the weights of feedforward layers and attention projections (except biases; shaded **right**).

Configuration: batch size 8, sequence length 128. CPU: Intel i7–4770.

n:m sparsity format: each group of m elements has n nonzeros [3]



References

1. Jacob Devlin, et al.. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
2. Thomas Wolf, et al. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020.
3. Aojun Zhou, et al. Learning N:M fine-grained structured sparse neural networks from scratch. In International Conference on Learning Representations (ICLR), 2021.
4. Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878, 2017.
5. Hao Li, et al. Pruning filters for efficient convnets. In International Conference on Learning Representations (ICLR), 2017.
6. Torsten Hoefler, et al. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research, 22(241), 2021.

D INFK