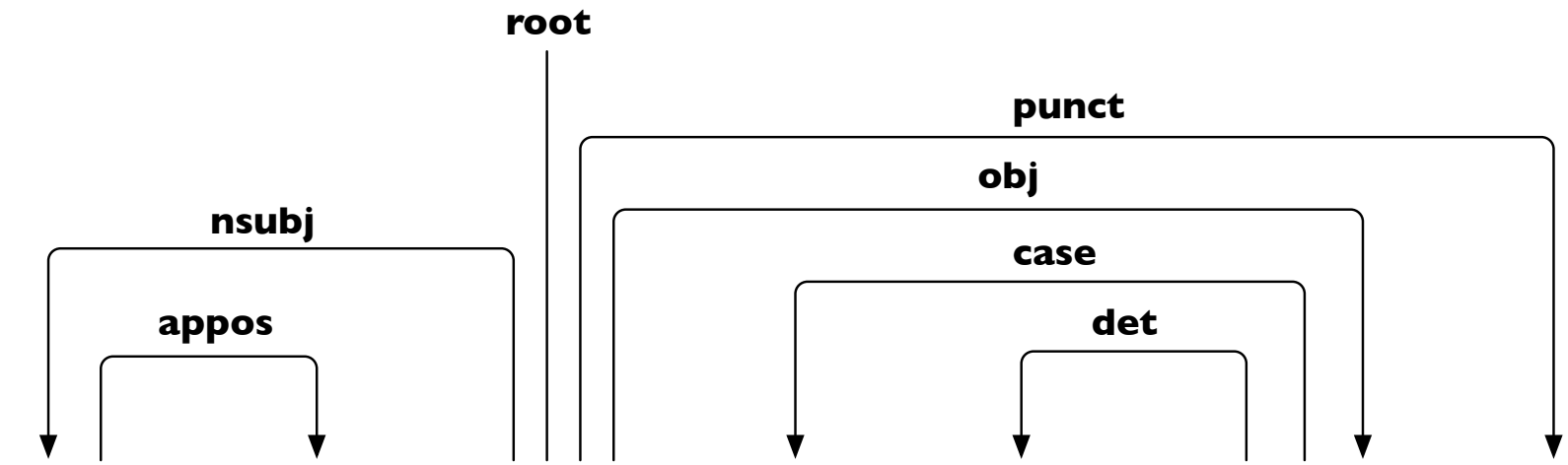
Fast and accurate dependency parsing for Dutch and German

Daniël de Kok[1] and Patricia Beer[2]

[1] TensorDot ✉ daniel@tensordot.com 🗣 @danieldekok
[2] University of Tübingen ✉ patricia.beer@uni-tuebingen.de 🗣 @patafis



	NOUN	PROPN	VERB	ADP	DET	NOUN	PUNCT
	Kapitän	Eilts	laborieren	an	ein	Zehenbruch	.
Number:	Sing	Sing	Sing		Sing	Sing	
Case:	Nom	Nom		Dat	Dat	Dat	
Gender:	Masc	Masc			Masc	Masc	
Person:			3				
Tense:			Pres				
Mood:			Ind				
VerbForm:			Fin				
Definite:					Ind		
PronType:					Art		

Model architecture

Encoder: stock transformer architecture

Sequence labeling

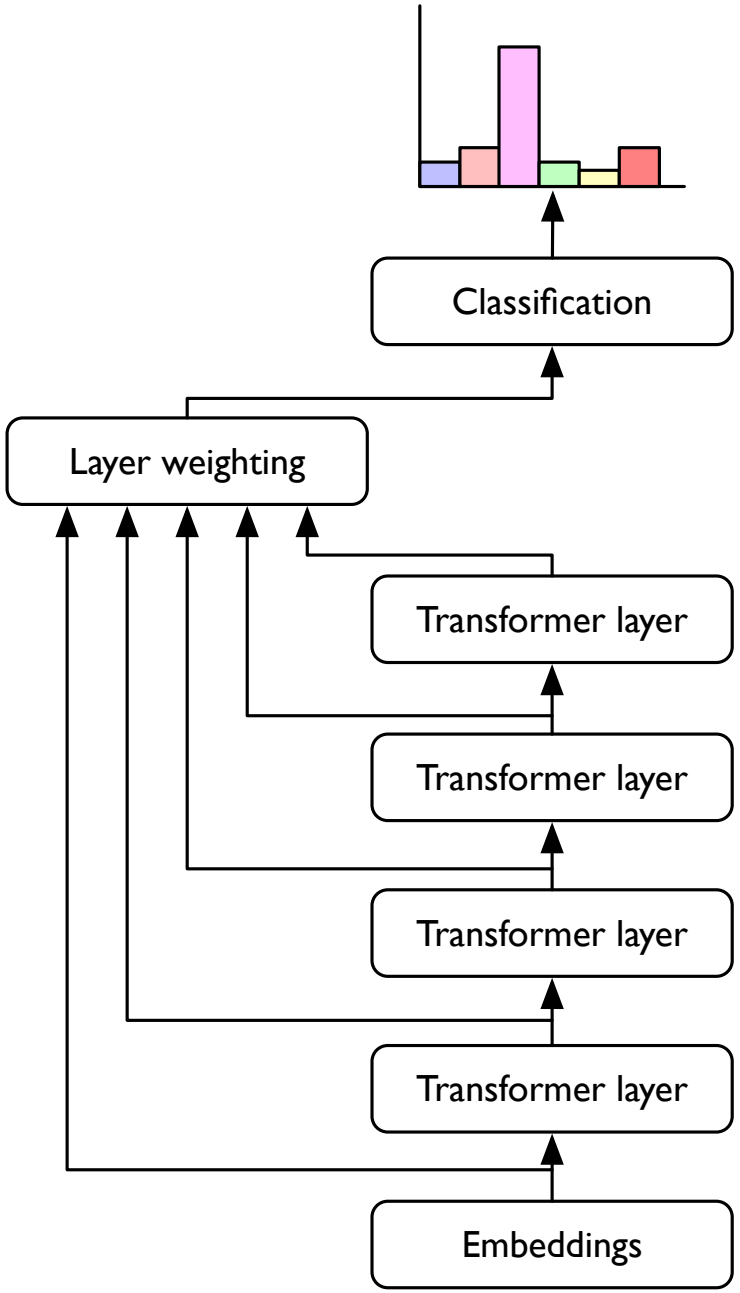
- Per-task weighting of transformer layer outputs (Peters et al, 2018, Kondratyuk & Straka, 2019).
- Feed-forward layer + softmax classifier.

Dependency parsing

- Task-specific weighting of transformer layer outputs.
- Biaffine parsing layer (Dozat & Manning, 2016) with MST decoding.

Training

Finetuning of the XLM-R base (multi-lingual) model (Conneau et al, 2020), using initial encoder freezing, learning rate warmup, weight decay, and label smoothing.



Finetuned models

Model	UD POS	Morphology	Lemma	LAS
Finetuned XLM-R – Dutch	98.90	98.87	99.03	94.37
Finetuned XLM-R – German	99.54	98.38	99.34	96.59

Finetuning on:

- **Dutch:** Lassy Small (Van Noord et al., 2013, Bouma & Van Noord, 2017)
- **German:** TüBa-D/Z (Teljohann et al., 2005, Çöltekin et al., 2017)

Random shuffle, 70% train, 10% dev, 20% validation.

Model size and performance

XLM-R base hyperparameters

- 250,000 piece vocabulary
- 12 hidden layers
- 768 hidden units
- 3072 hidden units in the positionwise feed-forward layer

Issues

The finetuned XLM-R models are large:

- Dutch: 1087 MiB
- German: 1104 MiB

The finetuned XLM-R models are slow (AMD Ryzen 3700X):

- Dutch: 76 sentences/second
- German: 71 sentences/second

Model complexity

Component	Time complexity	Space complexity	XLM-R size (MiB)
Piece embeddings	$\mathcal{O}(n)$	$\mathcal{O}(ve)$	732
Self-attention	$\mathcal{O}(nh^2 + n^2h)$	$\mathcal{O}(lh^2)$	108
Positionwise feed-forward	$\mathcal{O}(nhi)$	$\mathcal{O}(lhi)$	216

- $n =$ sentence length in pieces
- $v =$ vocabulary size
- $h =$ hidden layer size
- $l =$ number of hidden layers
- $e =$ embedding size, standard transformer: $e = h$
- $i =$ inner size of positionwise feed-forward layers

Monolingual vocabulary

Component	Time complexity	Space complexity	XLM-R size (MiB)	After changes (MiB)	Size reduction
Piece embeddings	$\mathcal{O}(n)$	$\mathcal{O}(vh)$	732	**88**	8.3x
Self-attention	$\mathcal{O}(nh^2 + n^2h)$	$\mathcal{O}(lh^2)$	108	108	
Positionwise feed-forward	$\mathcal{O}(nhi)$	$\mathcal{O}(lhi)$	216	216	

Change #1: replace the multi-lingual vocabulary ($v = 250{,}000$) by a mono-lingual vocabulary ($v = 30{,}000$):

- 8.3x reduction in vocabulary size.

Hidden layer size

Component	Time complexity	Space complexity	Monolingual vocab (MiB)	After changes (MiB)	Size reduction
Piece embeddings	$\mathcal{O}(n)$	$\mathcal{O}(vh)$	88	**44**	2x
Self-attention	$\mathcal{O}(nh^2 + n^2h)$	$\mathcal{O}(lh^2)$	108	**27**	4x
Positionwise feed-forward	$\mathcal{O}(nhi)$	$\mathcal{O}(lhi)$	216	**108**	2x

Change #2: Use a hidden layer size $h = 384$ rather than $h = 768$:

- Given a parameter budget, more layers are preferable over larger layer size (Turc et al., 2019).
- We found that decreasing the inner size i leads to larger accuracy losses.

Layer sharing and smaller embeddings

Component	Time complexity	Space complexity	Monolingual vocal + hidden size (MiB)	After changes (MiB)	Size reduction
Piece embeddings	$\mathcal{O}(n)$	$\mathcal{O}(vh)$	44	**15**	3x
Self-attention	$\mathcal{O}(nh^2 + n^2h)$	$\mathcal{O}(lh^2)$	27	**14**	2x
Positionwise feed-forward	$\mathcal{O}(nhi)$	$\mathcal{O}(lhi)$	108	**54**	2x

Change #3 (optional): Use $l = 12, h = 384$, but:

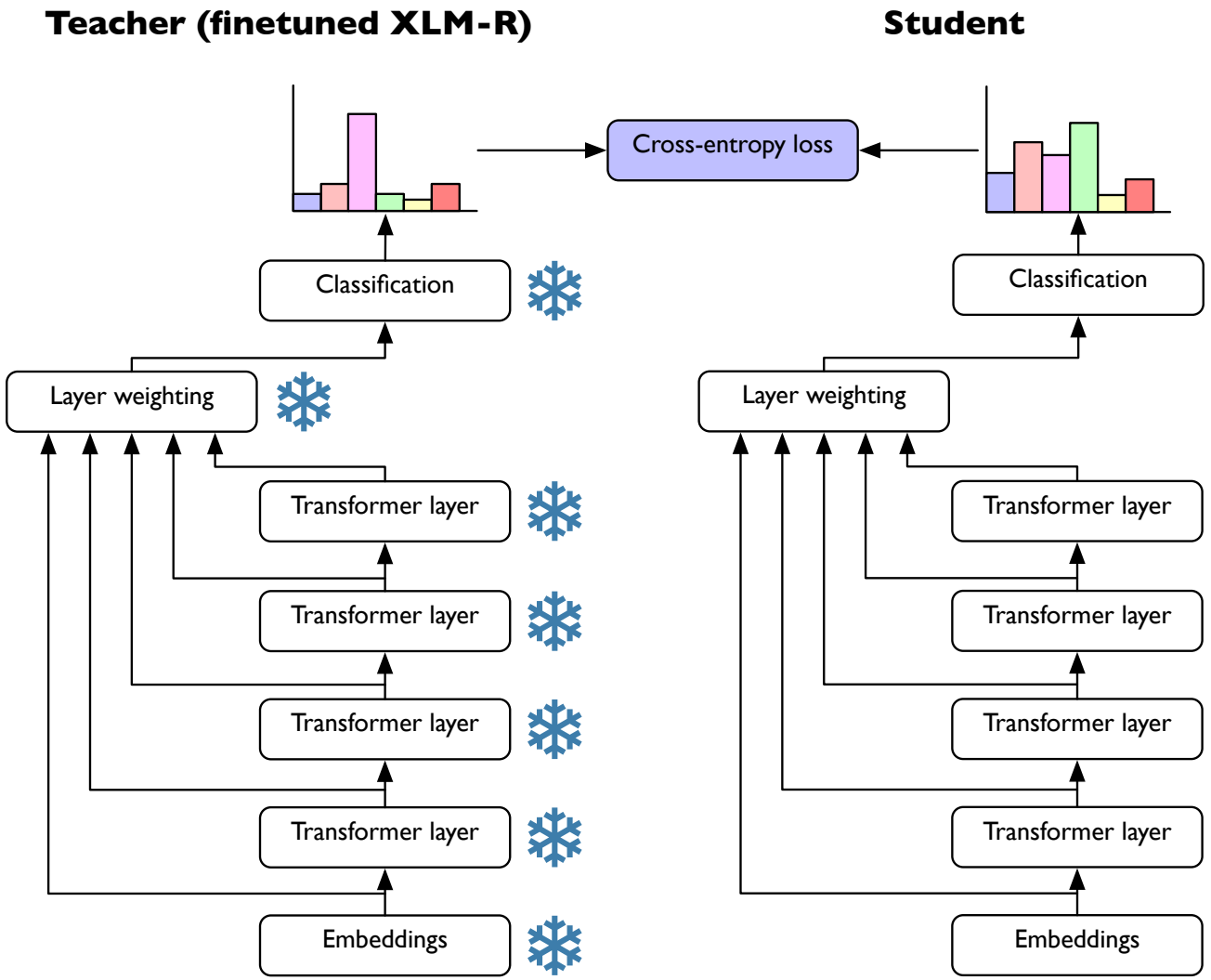
- Let every 2 layers share weigths (6 layer groups, $g = 6$).
- Use an embedding dimensionality $e = 128$.
- ALBERT architecture (Lan et al., 2020)

Number of hidden layers

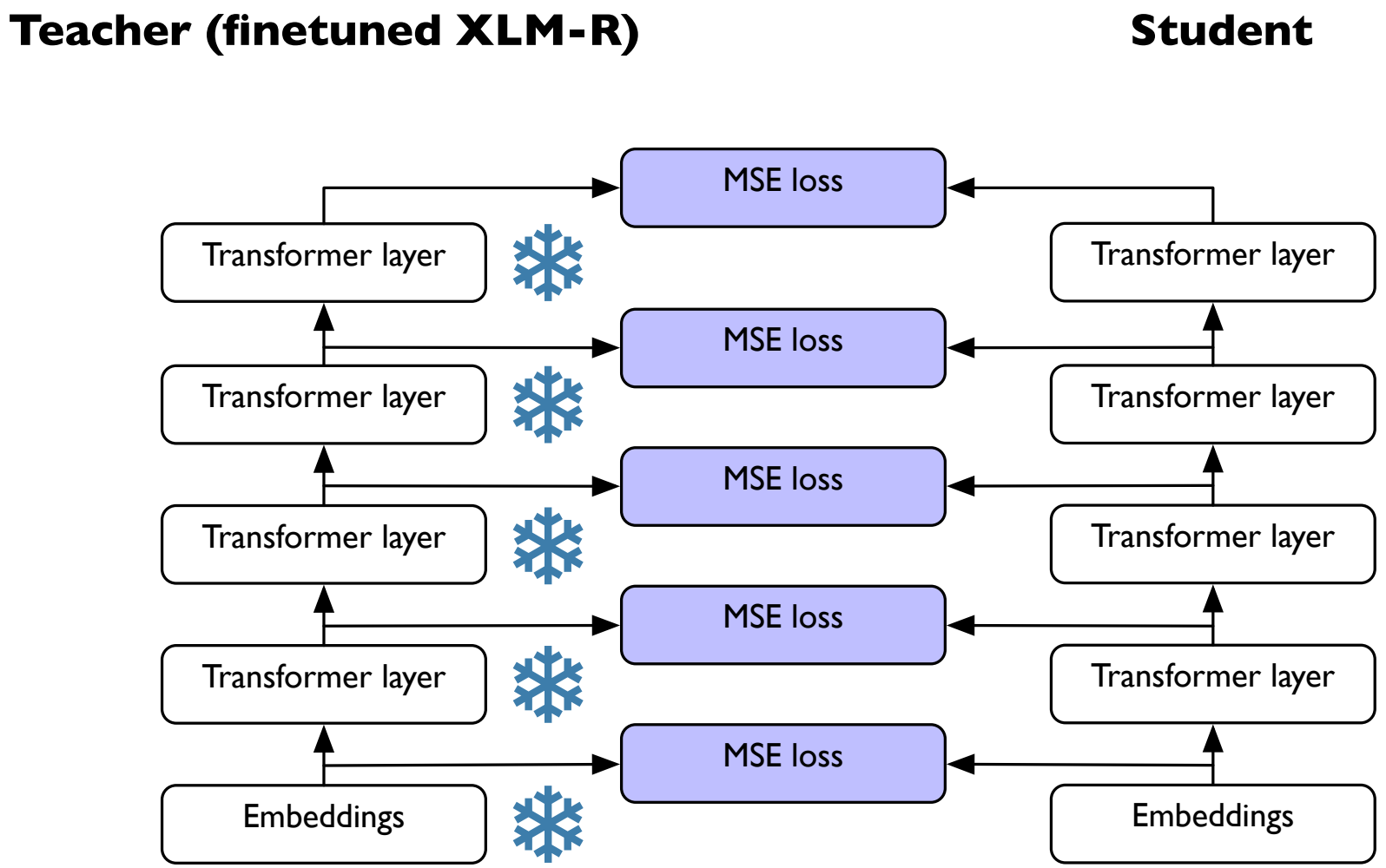
Component	Time complexity	Space complexity	Monolingual vocal + hidden size (MiB)	After changes (MiB)	Size reduction
Piece embeddings	$\mathcal{O}(n)$	$\mathcal{O}(vh)$	44	44	
Self-attention	$\mathcal{O}(nh^2 + n^2h)$	$\mathcal{O}(lh^2)$	27	**14**	2x
Positionwise feed-forward	$\mathcal{O}(nhi)$	$\mathcal{O}(lhi)$	108	**54**	2x

Change #4 (optional): Use $l = 6$ hidden layers rather than $l = 12$.

Model distillation: soft loss (Hinton et al., 2015)



Model distillation: layer loss (Jiao et al., 2020)



Model accuracy (Dutch)

Model	UD POS	Morphology	Lemma	LAS	Size (MiB)
Finetuned XLM-R base	98.90	98.87	99.03	94.37	1087
$h = 384, l = 12, g = 12, e = h$	98.83	98.80	99.03	93.91	200
$h = 384, l = 12, g = 6, e = 128$	98.81	98.76	99.00	93.79	104
$h = 384, l = 6, g = 6, e = h$	98.80	98.79	99.05	93.42	133

- Distillation: one pass over the sentences of Lassy Large (Van Noord et al., 2013).
- Finetuning: Lassy Small

Model accuracy (German)

Model	UD POS	Morphology	Lemma	LAS	Size (MiB)
Finetuned XLM-R base	99.54	98.38	99.34	96.59	1104
$h = 384, l = 12, g = 12, e = h$	99.50	98.31	99.31	96.17	140
$h = 384, l = 12, g = 6, e = 128$	99.46	98.23	99.27	95.85	111
$h = 384, l = 6, g = 6, e = h$	99.46	98.20	99.29	95.48	208

- Distillation: one pass over the sentences of the Wikipedia and Taz sections of the TüBa-D/DP (De Kok & Pütz, 2019).
- Finetuning: TüBa-D/Z

Benchmark systems

	AMD Ryzen 3700X	Apple M1 (MacBook Air)
Architecture	x86_64	ARM64
Cores	8	8 (4 Firestorm, 4 Icestorm)
Base clock/boost clock	3.6GHz/4.4GHz	???/3.2GHz
Vector instructions	AVX2 (SIMD)	AMX (hardware matrix multiplication blocks)
L1 cache (per core)	64KiB	320 KiB (Firestorm), 192 KiB (Icestorm)
L2 cache	512KiB per core	12 MiB (Firestorm), 4 MiB (Icestorm)
L3 cache	32MiB (16 GiB per 4 cores)	
DRAM MT/s	3200	4266
Cooling	Active	Passive

Dutch model performance

Model	Ryzen 3700X		M1 sents/s	
	Sents/s	Speedup	Sents/s	Speedup
Finetuned XLM-R base	76	1.0x	101	1.0x
$h = 384, l = 12, g = 12, e = h$	171	2.3x	170	1.7x
$h = 384, l = 12, g = 6, e = 128$	171	2.3x	170	1.7x
$h = 384, l = 6, g = 6, e = h$	304	4.0x	303	3.0x

Benchmark using:

- 4 annotation threads
- Length-sorted batches (minimize sequence length variance within batches)
- Dynamic batching (minimize variance in number of batch computations)
- libtorch, Ryzen: Intel MKL (CPU detection override), M1: Accelerate

Profile AMD Ryzen 3700X

Overhead	Command	Shared Object	Symbol
41.06%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_kernel_0
22.75%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_kernel_0_b0
3.77%	syntaxdot	libtorch_cpu.so	[.] mkl_vml_kernel_sCdfNorm_L9HAynn
3.72%	syntaxdot	libtorch_cpu.so	[.] at::native::(anonymous namespace)::dereference_vec_impl<f
2.69%	syntaxdot	libtorch_cpu.so	[.] c10::function_ref<void (char**, long const*, long, long)>
2.19%	syntaxdot	libtorch_cpu.so	[.] at::native::(anonymous namespace)::multi_row_sum<at::vec2
2.13%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_scopy_right4_ea
2.05%	syntaxdot	libtorch_cpu.so	[.] c10::function_ref<void (char**, long const*, long, long)>
1.89%	syntaxdot	libtorch_cpu.so	[.] at::native::(anonymous namespace)::_vec_softmax_lastdim<f
1.71%	syntaxdot	libtorch_cpu.so	[.] c10::function_ref<void (char**, long const*, long, long)>
1.69%	syntaxdot	libtorch_cpu.so	[.] mkl_vml_kernel_sMul_L9HAynn
1.69%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_scopy_right24_ea
1.36%	syntaxdot	libtorch_cpu.so	[.] at::native::(anonymous namespace)::vectorized_loop<at::na
0.97%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_kernel_nocopy_TN_b0
0.80%	syntaxdot	libtorch_cpu.so	[.] at::native::dim_apply<at::native::(anonymous namespace)::
0.73%	syntaxdot	libtorch_cpu.so	[.] Sleef_finz_expf8_u10avx2
0.71%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_kernel_nocopy_NN_b0
0.64%	syntaxdot	libtorch_cpu.so	[.] mkl_blas_avx2_sgemm_kernel_nocopy_TN_b1
0.56%	syntaxdot	libtorch_cpu.so	[.] std::__heap_select<__gnu_cxx::__normal_iterator<std::pair
0.55%	syntaxdot	libtorch_cpu.so	[.] at::native::(anonymous namespace)::LayerNormKernelImplInt

Profile (Apple M1)

Weight		Self		Symbol Name
~~2.66 min~~	~~100.0%~~	~~0 s~~		~~syntaxdot (27420)~~
1.28 min	48.2%	1.28 min		void at::native::(anonymous namespace)::vectorized_loop<at::native::(anonymous namespace)::GeluKernelImpl(at
36.67 s	23.0%	36.67 s		at::native::cpublas::gemm(at::native::cpublas::TransposeType, at::native::cpublas::TransposeType, long long, long
6.96 s	4.3%	6.96 s		void c10::function_ref<void (char**, long long const*, long long, long long)>::callback_fn<auto at::TensorIteratorBa
6.12 s	3.8%	6.12 s		void at::native::(anonymous namespace)::_vec_softmax_lastdim<float>(float*, float*, long long, long long)::'lambd
4.88 s	3.0%	4.88 s		c10::TensorImpl::release_resources() libc10.dylib
4.52 s	2.8%	4.52 s		void c10::function_ref<void (char**, long long const*, long long, long long)>::callback_fn<auto at::TensorIteratorBa
3.71 s	2.3%	3.71 s		void c10::function_ref<void (char**, long long const*, long long, long long)>::callback_fn<auto at::TensorIteratorBa
3.37 s	2.1%	3.37 s		void c10::function_ref<void (char**, long long const*, long long, long long)>::callback_fn<auto at::TensorIteratorBa
2.54 s	1.5%	2.54 s		DYLD-STUB$$erff libtorch_cpu.dylib
2.32 s	1.4%	2.32 s		void at::native::dim_apply<at::native::(anonymous namespace)::topk_kernel(at::Tensor const&, at::Tensor const&,

ReLU

Model	AMD Ryzen 3700X		Apple M1	
	GELU	ReLU	GELU	ReLU
Finetuned XLM-R base	76		101	
$h = 384, l = 12, g = 12, e = h$	171	180	170	245
$h = 384, l = 12, g = 6, e = 128$	171	179	170	245
$h = 384, l = 6, g = 6, e = h$	304	316	303	403

Conclusion & future work

- Model distillation makes it possible to benefit from pre-trained models, allowing for trade-off between:
  - Accuracy
  - Size
  - Speed
- Qualitative analysis: *what* do we lose?
- Other possible speedups and size reductions:
  - Half-precision models (Apple AMX, Intel AVX-512 FP16)
  - Integer-only quantization (Kim et al., 2021)
  - 16 core Neural Engine in the Apple A12/A13/A14/M1
- Explore different hyperparameters (e.g. ALBERT layer sharing vs. smaller embeddings).

Thank you!

Software and models are available at:

https://github.com/tensordot/syntaxdot

Written in Rust 🦀 using libtorch 🔥

GPU performance

Model	AMD Ryzen 3700X		Apple M1		RTX2060 Super	
	GELU	ReLU	GELU	ReLU	GELU	ReLU
Finetuned XLM-R base	76		101		799	
$h = 384, l = 12, g = 12, e = h$	171	180	170	245	1653	1670
$h = 384, l = 12, g = 6, e = 128$	171	179	170	245	1648	1663
$h = 384, l = 6, g = 6, e = h$	304	316	303	403	2732	2741

ReLU accuracy

Model	Nonlinearity	UD POS	Morphology	Lemma	LAS
Finetuned XLM-R base	GELU	98.90	98.87	99.03	94.37
$h = 384, l = 12, g = 12, e = h$	GELU	98.83	98.80	99.03	93.91
	ReLU	98.82	98.79	98.99	93.86
$h = 384, l = 12, g = 6, e = 128$	GELU	98.81	98.76	99.00	93.79
	ReLU	98.80	98.74	98.98	93.67
$h = 384, l = 6, g = 6, e = h$	GELU	98.80	98.79	99.05	93.42
	ReLU	98.81	98.82	99.06	93.37