Compression techniques of neural networks applied on NMT

Martin Geletka, 456576

Outline

- NMT (results english <-> czech)
  - State of NMT
  - Tools used
  - Results and future work
- Compression techniques
  - Quantization
  - Distillation
  - MobileBERT
  - Lottery ticket hypothesis

Neural Machine Translation

History

History

Ruled based systems

1950 - 1990

Statistical Machine

Translation

1990 - 2010

Neural Machine

Translation

2010 -

*for more info Stanford CS224N course

Used tools / libraries

Tokenization: https://github.com/google/sentencepiece

Training and additional scripts: https://github.com/pytorch/fairseq

Evaluation: https://github.com/alvations/sacremoses

Logging: https://wandb.ai/site

Used data

Data:

- TRAINING:
  - czeng_train
- VALIDATION:
  - newstest 2018
  - tedtalk (first 10k sentences)
- TEST
  - newstest 2019
  - czeng test
  - tedtalk

Used hyper params

Architecture:

- Using Wasmani 2017 transformer
- 6 layers encoder, 6 layers decoder
- 16 heads per layer

Tokenization:

- BPE algorithm implemented in sentencepiece

Approach with 16-bit training

Original paper: https://arxiv.org/abs/1806.00187

model	# gpu	bsz	cumul	BLEU	updates	tkn/sec	time	speedup
Vaswani et al. (2017)	8×P100	25k	1	26.4	300k	~25k	~5,000	–
Our reimplementation	8×V100	25k	1	26.4	192k	54k	1,429	reference
+ 16-bit	8	25k	1	26.7	193k	143k	495	2.9x
+ cumul	8	402k	16	26.7	13.7k	195k	447	3.2x
+ 2x lr	8	402k	16	26.5	9.6k	196k	311	4.6x
+ 5k tkn/gpu	8	365k	10	26.5	10.3k	202k	294	4.9x
16 nodes (from +2x lr)	128	402k	1	26.5	9.5k	1.53M	37	38.6x
+ overlap comm+bwd	128	402k	1	26.5	9.7k	1.82M	32	44.7x

Comparison with other

- Compare locally with sacremose but also submitted to Euro Matrix
- Cs -> En
  - Newstests 2019: http://matrix.statmt.org/matrix/systems_list/1866
- En -> Cs
  - Newstest 2017: http://matrix.statmt.org/matrix/systems_list/1867
  - Newstest 2019: http://matrix.statmt.org/matrix/systems_list/1896

Future ‘base’ experiments

English→Czech system	BLEU cased	BLEU uncased	chrF2 cased
Nematus (Sennrich et al., 2016b)	22.80	23.29	0.5059
T2T (Popel and Bojar, 2018)	23.84	24.40	0.5164
our mixed backtranslation	24.85 (+1.01)	25.33	0.5267
our concat backtranslation	25.77 (+0.92)	26.29	0.5352
+ higher quality backtranslation	26.60 (+0.83)	27.10	0.5410
+ CZ/nonCZ tuning	**26.81** (+0.21)	**27.30**	**0.5431**

Table 2: Automatic evaluation on (English→Czech) `newstest2017`. The three scores in parenthesis show BLEU difference relative to the previous line.

https://www.aclweb.org/anthology/W18-6424.pdf

Compression techniques

Quantization

Quantization as post processing

- 2011 - Used and tested already in pre transformer era
- achieves a good compression rate with the additional benefit of accelerating inference on supporting hardware.
- But the errors made by these approximations accumulate in the computations operated during the forward pass, inducing a significant drop in performance

*Improving the speed of neural networks on CPUs

Quantization as post processing

- 2011 - Used and tested already in pre transformer era
- achieves a good compression rate with the additional benefit of accelerating inference on supporting hardware.
- But the errors made by these approximations accumulate in the computations operated during the forward pass, inducing a significant drop in performance
- Solution:
  - Quantized Aware Training

Q8BERT - Quantization Schema

$$\text{Quantize}(x|S^x, M) := \text{Clamp}\left(\lfloor x \times S^x \rceil, -M, M\right),$$
$$\text{Clamp}(x, a, b) = \min\left(\max(x, a), b\right)$$

$$M = 2^{b-1} - 1$$

$$S^x = \frac{M}{\text{EMA}\left(\max\left(|x|\right)\right)} \qquad S^W = \frac{M}{\max\left(|W|\right)}$$

https://arxiv.org/pdf/1910.06188.pdf

Q8BERT: Results

Dataset	Metric	BERT baseline accuracy (STD)	QAT BERT 8bit (STD)	DQ BERT 8bit (STD)
CoLA	Matthew's corr.	**58.48** (1.54)	**58.48** (1.32)	56.74 (0.61)
MRPC	F1	**90** (0.23)	89.56 (0.18)	87.88 (2.03)
MRPC-Large	F1	90.86 (0.55)	**90.9** (0.29)	88.18 (2.19)
QNLI	Accuracy	90.3 (0.44)	**90.62** (0.29)	89.34 (0.61)
QNLI-Large	Accuracy	91.66 (0.15)	**91.74** (0.36)	88.38 (2.22)
QQP	F1	87.84 (0.19)	**87.96** (0.35)	84.98 (0.97)
RTE	Accuracy	**69.7** (1.5)	68.78 (3.52)	63.32 (4.58)
SST-2	Accuracy	**92.36** (0.59)	92.24 (0.27)	91.04 (0.43)
STS-B	Pearson corr.	**89.62** (0.31)	89.04 (0.17)	87.66 (0.41)
STS-B-Large	Pearson corr.	**90.34** (0.21)	90.12 (0.13)	83.04 (5.71)
SQuADv1.1	F1	**88.46** (0.15)	87.74 (0.15)	80.02 (2.38)

QUANTIZATION NOISE FOR EXTREME MODEL COMPRESSION

- Traditional vector quantization = split the matrix W into its p columns and learn a codebook on the resulting p vectors.
- Product Quantization splits each column into m subvectors and learns the same codebook for each of the resulting m × p subvectors.
- Iterative PQ = quantize layers sequentially from the lowest to the highest, and finetune the upper layers as the lower layers are quantized
- Then combining fixed-point with product quantization

https://arxiv.org/pdf/2004.07320.pdf

QUANTIZATION NOISE - Training

- Select just subset of block and apply quantization
- When selecting all blocks = QAT
- Advantage of selecting only subset = unbiased gradients continue to flow via blocks unaffected by the noise

https://arxiv.org/pdf/2004.07320.pdf

QUANTIZATION NOISE - Results

Quantization Scheme	**Language Modeling** 16-layer Transformer Wikitext-103			**Image Classification** EfficientNet-B3 ImageNet-1k		
	Size	Compression	PPL	Size	Compression	Top-1
Uncompressed model	942	× 1	18.3	46.7	× 1	81.5
`int4` quantization	118	× 8	39.4	5.8	× 8	45.3
- trained with QAT	118	× 8	34.1	5.8	× 8	59.4
- trained with Quant-Noise	118	× 8	**21.8**	5.8	× 8	**67.8**
`int8` quantization	236	× 4	19.6	11.7	× 4	80.7
- trained with QAT	236	× 4	21.0	11.7	× 4	80.8
- trained with Quant-Noise	236	× 4	**18.7**	11.7	× 4	**80.9**
iPQ	38	× 25	25.2	3.3	× 14	79.0
- trained with QAT	38	× 25	41.2	3.3	× 14	55.7
- trained with Quant-Noise	38	× 25	**20.7**	3.3	× 14	**80.0**
iPQ & `int8` + Quant-Noise	38	× 25	21.1	3.1	× 15	79.8

Distillation

Knowledge distillation

- a compression technique in which the student model is trained to reproduce the behaviour of the teacher model

- Training loss
  - Distillation loss
  - Cosine Embedding loss
  - Original training loss (f.e. e masked language modeling loss)

$$L_{ce} = \sum_i t_i * \log(s_i)$$

https://arxiv.org/pdf/1910.01108.pdf

Distill BERT - student

- **Architecture**
  - token-type embeddings and the pooler are removed while the number of layers is reduced by a factor of 2.

- **Initialization**
  - initialize the student from the teacher by taking one layer out of two

Distill BERT - results

- has 40% fewer parameters than BERT and is 60% faster than BERT

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Distill BERT - Ablation study

Table 4: **Ablation study.** Variations are relative to the model trained with triple loss and teacher weights initialization.

Ablation	**Variation on GLUE macro-score**
$\emptyset$ - L_{cos} - L_{mlm}	-2.96
L_{ce} - $\emptyset$ - L_{mlm}	-1.46
L_{ce} - L_{cos} - $\emptyset$	-0.31
Triple loss + random weights initialization	-3.69

Mobile BERT

Architecture

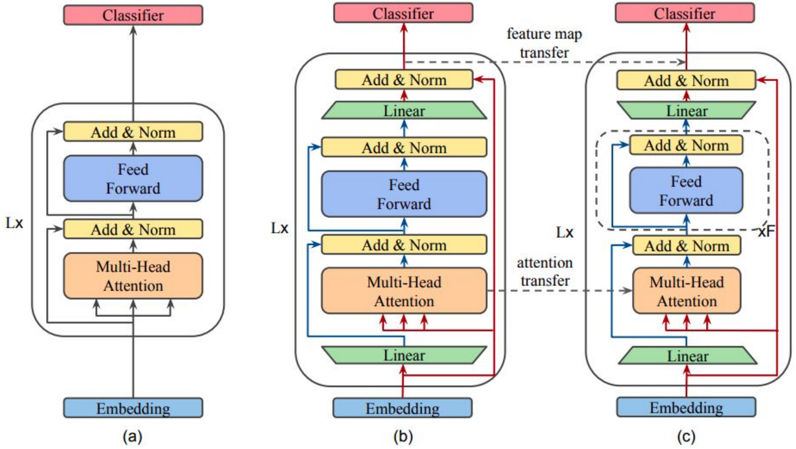


Figure 1: Illustration of three models: (a) BERT; (b) Inverted-Bottleneck BERT (IB-BERT); and (c) MobileBERT. In (b) and (c), red lines denote inter-block flows while blue lines intra-block flows. MobileBERT is trained by layer-to-layer imitating IB-BERT.

https://arxiv.org/abs/2004.02984

Architecture

			$\text{BERT}_{\text{LARGE}}$	$\text{BERT}_{\text{BASE}}$	$\text{IB-BERT}_{\text{LARGE}}$	MobileBERT	$\text{MobileBERT}_{\text{TINY}}$
embedding		$h_{\text{embedding}}$	1024	768	128		
			no-op	no-op	3-convolution		
		h_{inter}	1024	768	512		
body	Linear	h_{input}			512	512	512
		h_{output}			1024	128	128
	MHA	h_{input}	1024	768	512	512	128
		#Head	16	12	4	4	4
		h_{output}	1024	768	1024	128	128
	FFN	h_{input}	1024	768	1024	128	128
		h_{FFN}	4096	3072	4096	512 ($\times 4$)	512 ($\times 2$)
		h_{output}	1024	768	1024	128	128
	Linear	h_{input}			1024	128	128
		h_{output}			512	512	512
			$\times 24$	$\times 12$	$\times 24$	$\times 24$	$\times 24$
#Params			334M	109M	293M	25.3M	15.1M

Table 1: The detailed model settings of a few models. h_{inter}, h_{FFN}, $h_{\text{embedding}}$, #Head and #Params denote the inter-block hidden size (feature map size), FFN intermediate size, embedding table size, the number of heads in multi-head attention, and the number of parameters, respectively.

Training

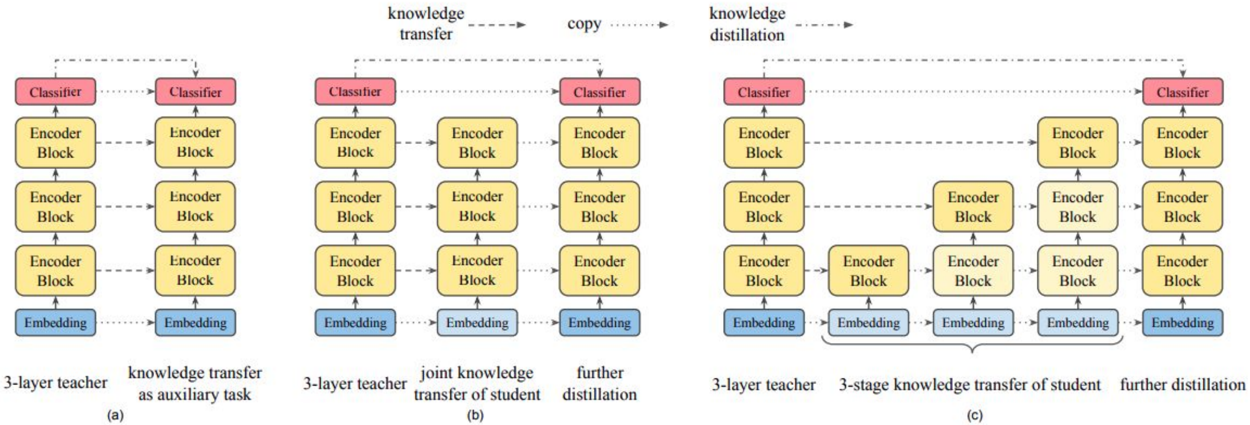


Figure 2: Diagrams of (a) auxiliary knowledge transfer (AKT), (b) joint knowledge transfer (JKT), and (c) progressive knowledge transfer (PKT). Lighter colored blocks represent that they are frozen in that stage.

Results

	#Params	#FLOPS	Latency	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 5.7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	GLUE
ELMo-BiLSTM-Attn	-	-	-	33.6	90.4	84.4	72.3	63.1	74.1/74.5	79.8	58.9	70.0
OpenAI GPT	109M	-	-	47.2	93.1	87.7	84.8	70.1	80.7/80.6	87.2	69.1	76.9
BERT$_{\text{BASE}}$	109M	22.5B	342 ms	**52.1**	**93.5**	**88.9**	**85.8**	71.2	**84.6/83.4**	90.5	66.4	78.3
BERT$_{\text{BASE}}$-6L-PKD*	66.5M	11.3B	-	-	92.0	85.0	-	70.7	81.5/81.0	89.0	65.5	-
BERT$_{\text{BASE}}$-4L-PKD†*	52.2M	7.6B	-	24.8	89.4	82.6	79.8	70.2	79.9/79.3	85.1	62.3	-
BERT$_{\text{BASE}}$-3L-PKD*	45.3M	5.7B	-	-	87.5	80.7	-	68.1	76.7/76.3	84.7	58.2	-
DistilBERT$_{\text{BASE}}$-6L†	62.2M	11.3B	-	-	92.0	85.0		70.7	81.5/81.0	89.0	65.5	-
DistilBERT$_{\text{BASE}}$-4L†	52.2M	7.6B	-	32.8	91.4	82.4	76.1	68.5	78.9/78.0	85.2	54.1	-
TinyBERT*	14.5M	1.2B	-	43.3	92.6	86.4	79.9	**71.3**	82.5/81.8	87.7	62.9	75.4
MobileBERT$_{\text{TINY}}$	15.1M	3.1B	40 ms	46.7	91.7	87.9	80.1	68.9	81.5/81.6	89.5	65.1	75.8
MobileBERT	25.3M	5.7B	62 ms	50.5	92.8	88.8	84.4	70.2	83.3/82.6	90.6	66.2	77.7
MobileBERT w/o OPT	25.3M	5.7B	192 ms	51.1	92.6	88.8	84.8	70.5	84.3/**83.4**	**91.6**	**70.4**	**78.5**

Lottery ticket hypothesis

Idea

- In general the sparser the network, the slower the learning and the lower the eventual test accuracy
- But ….

Idea

- In general the sparser the network, the slower the learning and the lower the eventual test accuracy
- But ....maybe there exist smaller subnetworks which train from the start and learn at least as fast as their larger counterparts while reaching similar test accuracy

LTH - formal definition

- Consider a dense feed-forward neural network $f(x; \theta)$ with initial parameters $\theta = \theta_0 \sim D_\theta$.
- When optimizing with stochastic gradient descent (SGD) on a training set, f reaches minimum validation loss *l* at iteration *j* with test accuracy *a* .
- In addition, consider training $f(x; m . \theta_0)$ with a mask $m \in \{0, 1\}^{|\theta|}$ on its parameters such that its initialization is $m\ \theta_0$. When optimizing with SGD on the same training set (with m fixed), f reaches minimum validation loss *l'* at iteration *j'* with test accuracy *a'*
- The lottery ticket hypothesis predicts that $\exists$ m for which $j' \leq j$ (commensurate training time), $a' \geq a$ (commensurate accuracy), and $||m||_0 << |\theta|$ (fewer parameters).

Identifying winning tickets

- Randomly initialize a neural network with params D_θ
- Train the network for j iterations, arriving at parameters θ_j
- Prune *p%* of the parameters in θ_j, creating a mask *m*
- Reset the remaining parameters to their values in θ_0, creating the winning ticket.
- “Upgrade”:
  - iterative pruning
  - train, prune, and reset the network over n rounds

Results

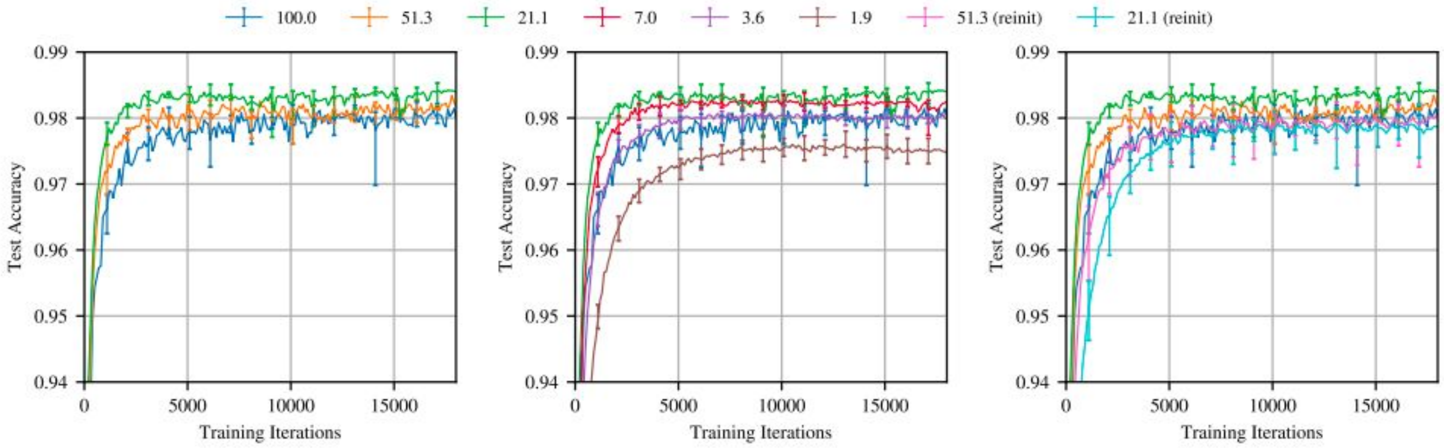
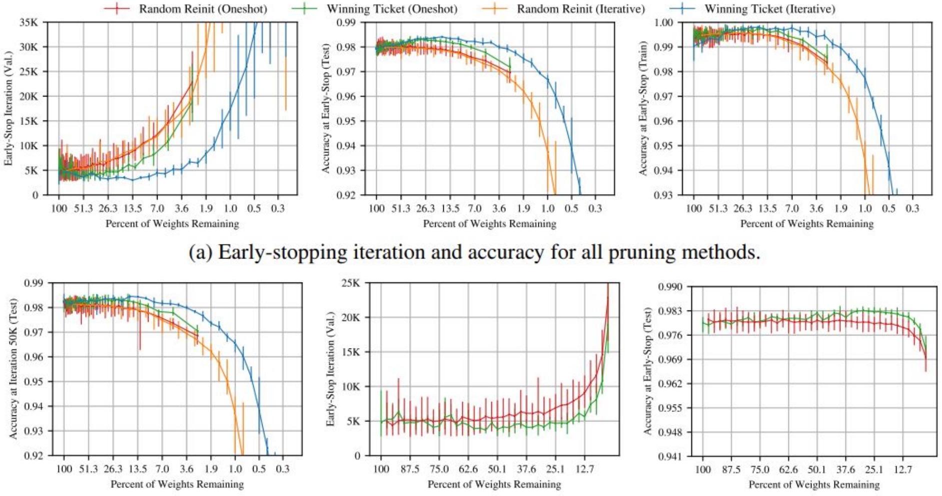


Figure 3: Test accuracy on Lenet (iterative pruning) as training proceeds. Each curve is the average of five trials. Labels are P_m—the fraction of weights remaining in the network after pruning. Error bars are the minimum and maximum of any trial.

Iterative vs One shot pruning



(a) Early-stopping iteration and accuracy for all pruning methods.

(b) Accuracy at end of training.

(c) Early-stopping iteration and accuracy for one-shot pruning.

Figure 4: Early-stopping iteration and accuracy of Lenet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values. At iteration 50,000, training accuracy $\approx 100\%$ for $P_m \geq 2\%$ for iterative winning tickets (see Appendix D, Figure 12).

Iterative vs One shot pruning

- Iterative pruning extracts smaller winning tickets, but their are costly to find
- One-shot pruning makes it possible to identify winning tickets without this repeated training
- One-shot indeed can find winning tickets, but iteratively-pruned winning tickets learn faster and reach higher test accuracy at smaller network sizes

Lottery tickets and NMT

- Tools used
  - Fairseq
  - Checkpoint averaging
  - Testing on newstests 2014
- Used 2 models Transformer Base and Transformer Base & Transformer Big
- In contrast with original paper:
  - Used late rewinding

https://arxiv.org/pdf/1906.02768.pdf

Lottery tickets and NMT - results

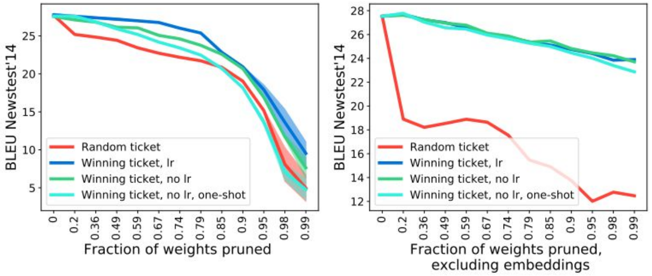


Figure 2: Winning ticket initialization performance for Transformer Base models trained on machine translation.

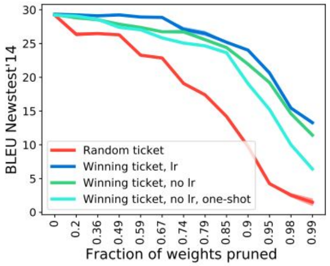


Figure 3: Winning ticket initialization performance for Transformer Big models trained on machine translation.

Questions?