Model Compression

Advanced NLP: Summer 2023

Anoop Sarkar

Compressing Large Language Models

Reduce memory and compute costs

- There are many different ways to solve the compression problem:

  1. **Distillation**: train a small lightweight student model on the output of a large teacher model.

  2. **Pruning**: Use an importance criterion to prune weights, prune layers, prune attention heads, etc.

  3. **Reduce precision** of the weights: FP16, int8, etc.

  4. Low rank **factorization** of weight matrices.

  5. **Weight sharing** (ALBERT).

https://doi.org/10.1162/tacl_a_00413

Distillation

Distilling the Knowledge in a Neural Network

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, NIPS 2014 DL workshop

https://arxiv.org/abs/1503.02531

See also: Bucila, Caruana, and Niculescu-Mizil. Model compression. In KDD, 2006.

Large Language Models

Are they necessary?

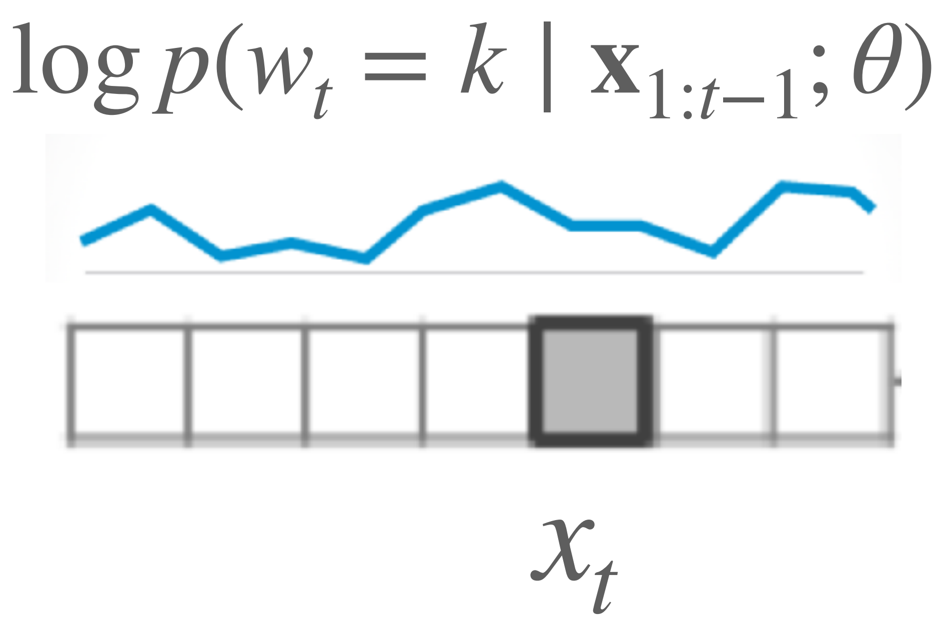
- Scaling to larger language models has led to improved zero-shot and few-shot accuracy on many NLP tasks.
- All modern deep learning models are heavily over-parameterized compared to the dataset size they train on.
- Smaller models by themselves do not give the same accuracy.
- Deployment of LLMs is challenging from a compute cost perspective.
- "Distill" a student model by training it on the output of a "teacher" model (a LLM).

Standard setup

for training a language model

- Minimize the log likelihood loss for prediction.
- Find parameters θ to minimize loss $\mathscr{L}$:

$$\mathscr{L} = -\sum_{t}\sum_{k\in\mathscr{V}} \delta(x_t = k)\log p(w_t = k \mid \mathbf{x}_{1:t-1}; \theta)$$



w_t is the softmax over the vocabulary $\mathscr{V}$

x_t is the ground truth target token; $\mathbf{x}$ is the sentence

$\delta(p) = 1$ if p is true and 0 otherwise

Knowledge Distillation

Can be used for pre-training or fine-tuning

- Train a larger teacher model (massive LLM models) to get a **teacher** distribution over outputs $q(\cdot)$ with parameters θ_T
- Train a smaller **student** model $p(\cdot)$ to mimic the teacher
- The student model has parameters $\theta << \theta_T$

Word level distillation

for training a language model

- Teacher distribution: $q(w_t \mid \mathbf{x}_{1:t-1}; \theta_T)$
- **Standard** loss:

$$\mathscr{L} = -\sum_t \sum_{k \in \mathscr{V}} \delta(x_t = k) \log p(w_t = k \mid \mathbf{x}_{1:t-1}; \theta)$$

- **Distillation** loss (uses **cross entropy** between *p* and *q*):

$$\mathscr{L}_D = -\sum_t \sum_{k \in \mathscr{V}} q(w_t = k \mid \mathbf{x}_{1:t-1}; \theta_T) \log p(w_t = k \mid \mathbf{x}_{1:t-1}; \theta)$$

$\delta(p) = 1$ if p is true and 0 otherwise

No Knowledge Distillation

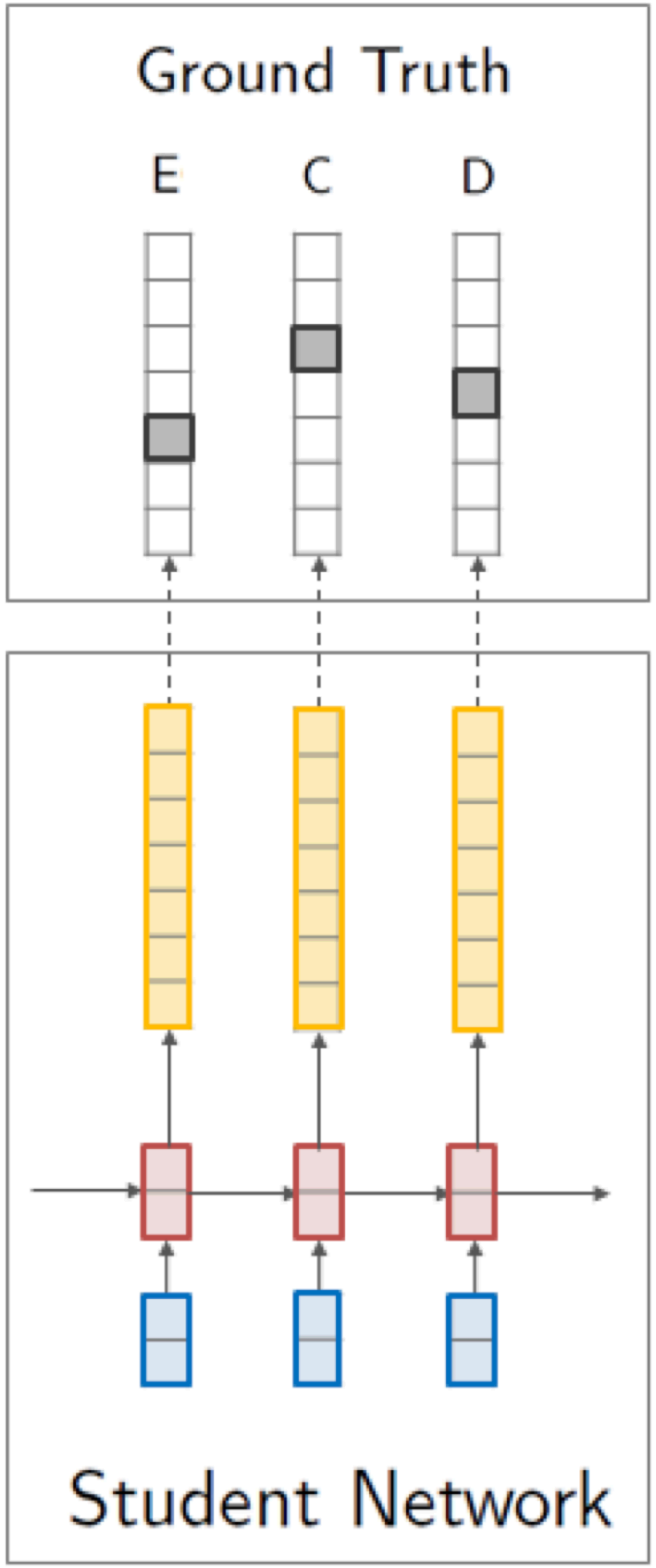


Fig from https://nlp.seas.harvard.edu/slides/emnlp16_seqkd.pdf

Word Level Knowledge Distillation

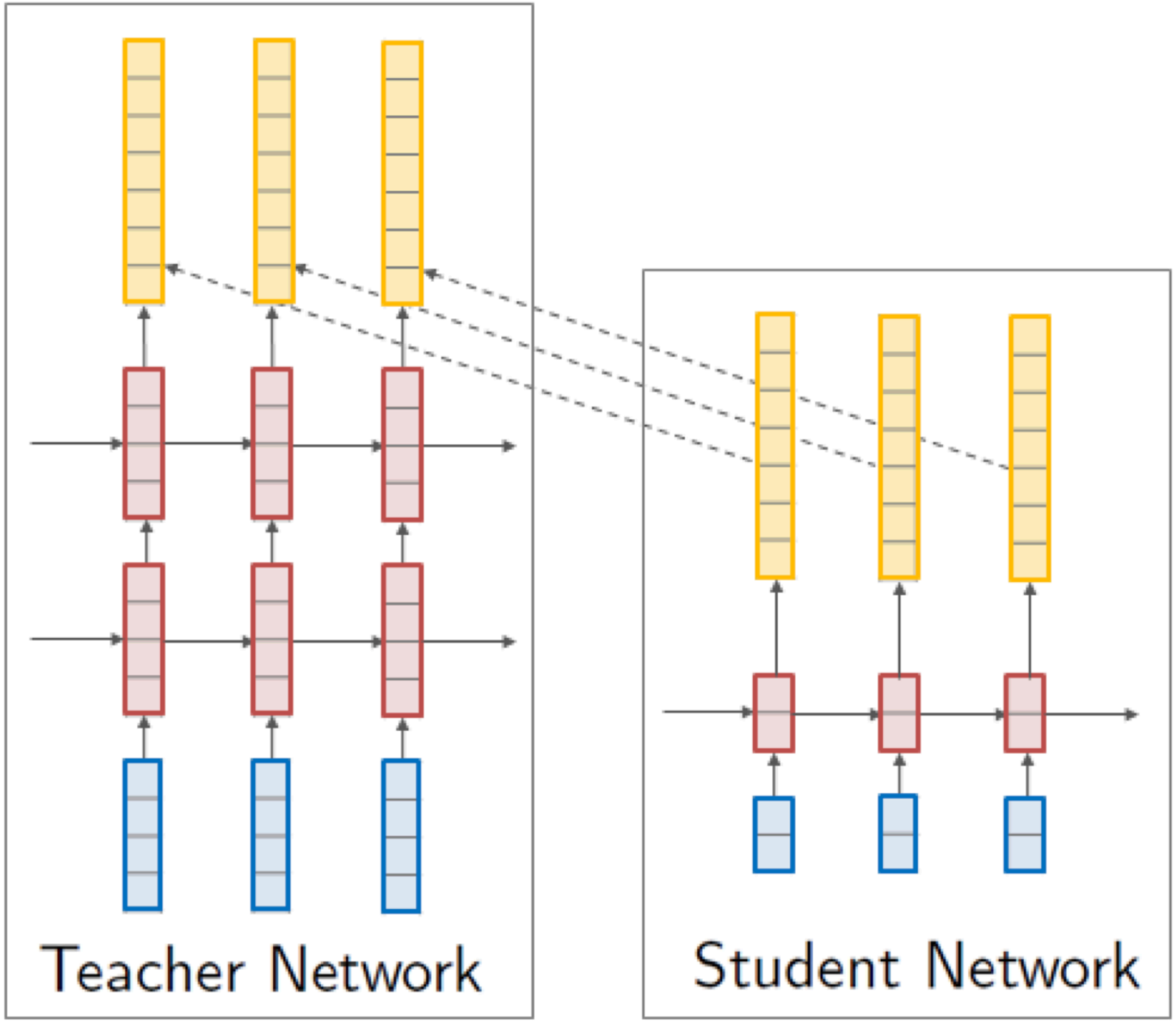


Fig from https://nlp.seas.harvard.edu/slides/emnlp16_seqkd.pdf

Combine standard loss and distillation loss

$$\mathscr{L}_C = \alpha \mathscr{L} + (1 - \alpha)\mathscr{L}_D$$

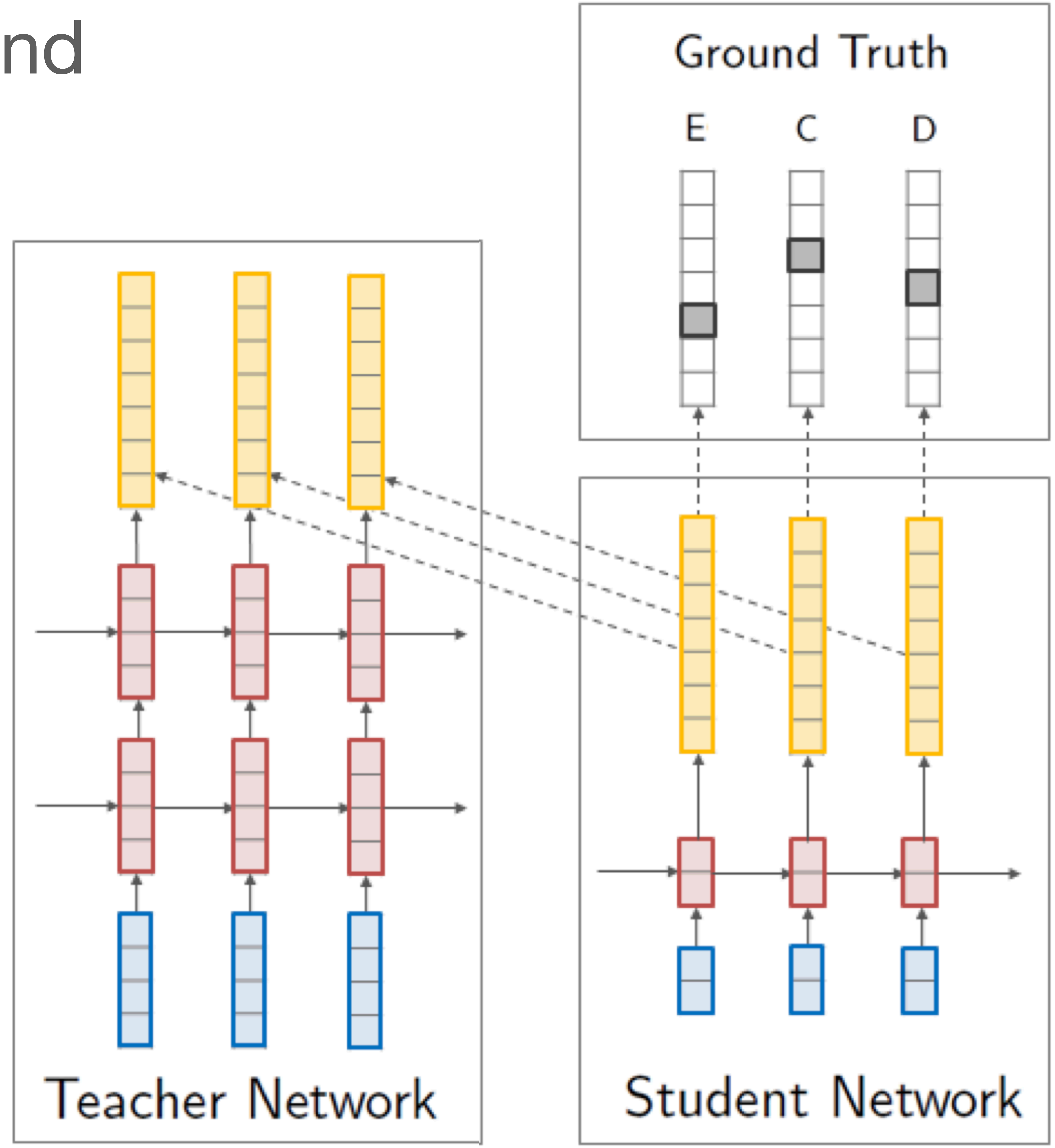


Fig from https://nlp.seas.harvard.edu/slides/emnlp16_seqkd.pdf

Soft targets

- Standard method to compute $p(w_t = k \mid \mathbf{x}_{1:t-1}; \theta)$

$$p_k = \frac{\exp(z_k)}{\sum_i \exp(z_i)}$$

z_i are the logits used to compute the softmax

- Divide the logits by a **temperature** parameter to get a softer distribution

$$p_k = \frac{\exp(\frac{z_k}{T})}{\sum_i \exp(\frac{z_i}{T})}$$

Soft targets

- Gradient wrt z_k

$$\frac{\partial \mathcal{L}_D}{\partial z_k} = \frac{1}{T}(q_k - p_k) \ = \frac{1}{T}\left(\frac{\exp(\frac{z_k}{T})}{\sum_i \exp(\frac{z_i}{T})} - \frac{\exp(\frac{v_k}{T})}{\sum_i \exp(\frac{v_i}{T})}\right) \ \approx \frac{1}{T^2}(z_k - v_k)$$

z_i, v_i are the logits used to compute the softmax for the teacher and student respectively

assuming the logits are zero-meaned, i.e. $\sum_i z_i = 0$ and $\sum_i v_i = 0$

Soft targets

Fig from Hinton's "Dark Knowledge" talk slides

cow	dog	cat	car	
0	1	0	0	Hard Target

cow	dog	cat	car	
10^{-6}	.9	.1	10^{-9}	Teacher distribution

cow	dog	cat	car	
.05	.3	.2	.005	Softened distribution

"Softened outputs reveal the **dark knowledge** in the teacher distribution"

Soft targets from BERT output distribution

```
Input: ['[CLS]', 'i', 'think', 'this', 'is', 'the', 'beginning', 'of', 'a', 'beautiful', '[MASK]', '.', '[SEP]']
Rank 0  - Token: day                - Prob: 0.21348
Rank 1  - Token: life               - Prob: 0.18380
Rank 2  - Token: future             - Prob: 0.06267
Rank 3  - Token: story              - Prob: 0.05854
Rank 4  - Token: world              - Prob: 0.04935
Rank 5  - Token: era                - Prob: 0.04555
Rank 6  - Token: time               - Prob: 0.03210
Rank 7  - Token: year               - Prob: 0.01722
Rank 8  - Token: history            - Prob: 0.01663
Rank 9  - Token: summer             - Prob: 0.01335
Rank 10 - Token: adventure          - Prob: 0.01233
Rank 11 - Token: dream              - Prob: 0.01209
Rank 12 - Token: moment             - Prob: 0.01129
Rank 13 - Token: night              - Prob: 0.01084
Rank 14 - Token: beginning          - Prob: 0.00937
Rank 15 - Token: season             - Prob: 0.00664
Rank 16 - Token: journey            - Prob: 0.00621
Rank 17 - Token: period             - Prob: 0.00553
Rank 18 - Token: relationship       - Prob: 0.00517
Rank 19 - Token: thing              - Prob: 0.00508
```

Fig from https://medium.com/huggingface/distilbert-8cf3380435b5

Distillation training step

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Optimizer

KD_loss = nn.KLDivLoss(reduction='batchmean')

def kd_step(teacher: nn.Module,
            student: nn.Module,
            temperature: float,
            inputs: torch.tensor,
            optimizer: Optimizer):
    teacher.eval()
    student.train()

    with torch.no_grad():
        logits_t = teacher(inputs=inputs)
    logits_s = student(inputs=inputs)

    loss = KD_loss(input=F.log_softmax(logits_s/temperature, dim=-1),
                   target=F.softmax(logits_t/temperature, dim=-1))

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Fig from https://medium.com/huggingface/distilbert-8cf3380435b5

Soft targets and combined loss

$$\mathscr{L}_C = \alpha \mathscr{L} + (1 - \alpha)\mathscr{L}_D$$

Temperature is set to 1 for Ground Truth

Higher temperature used for Teacher Distribution

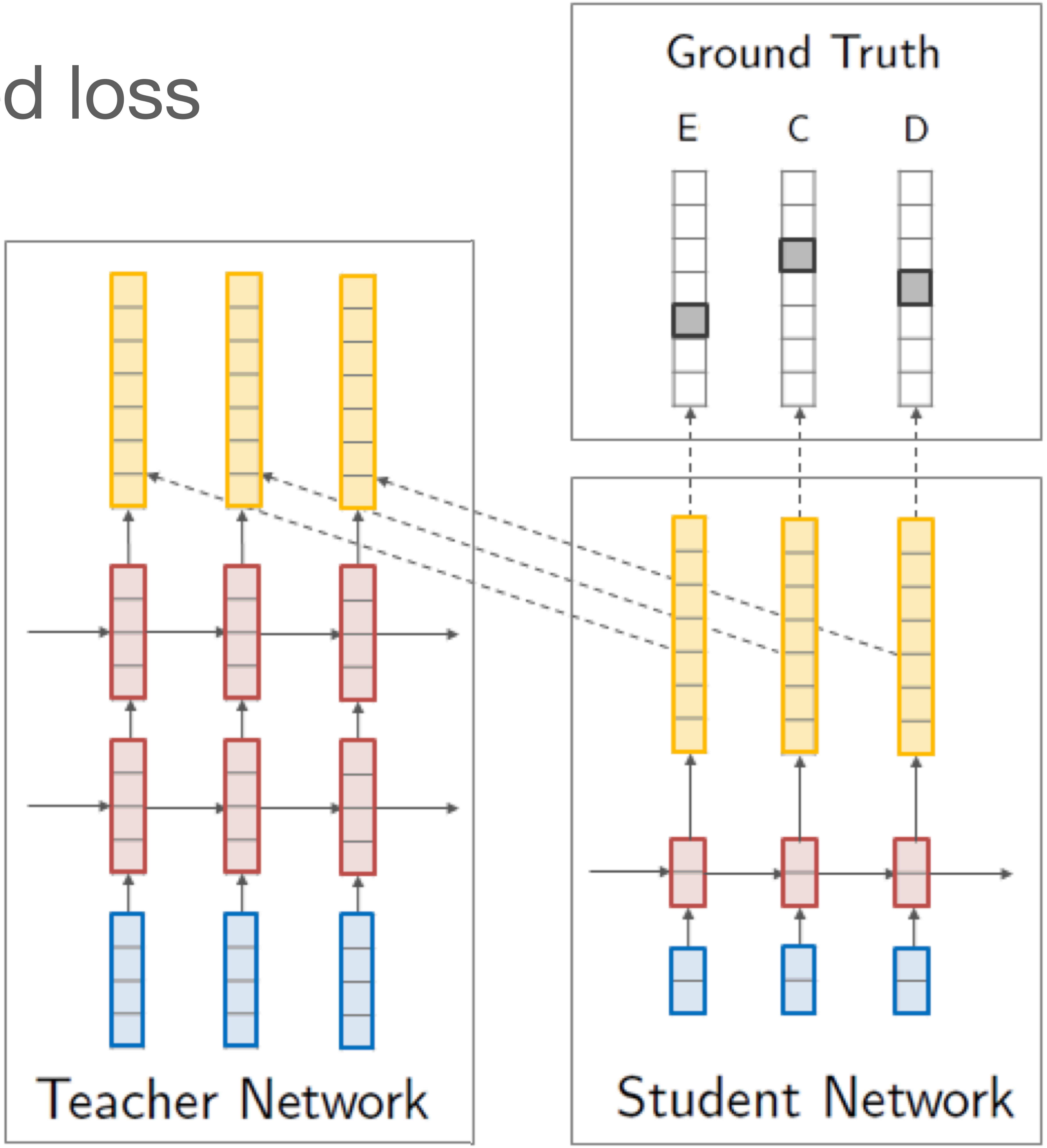
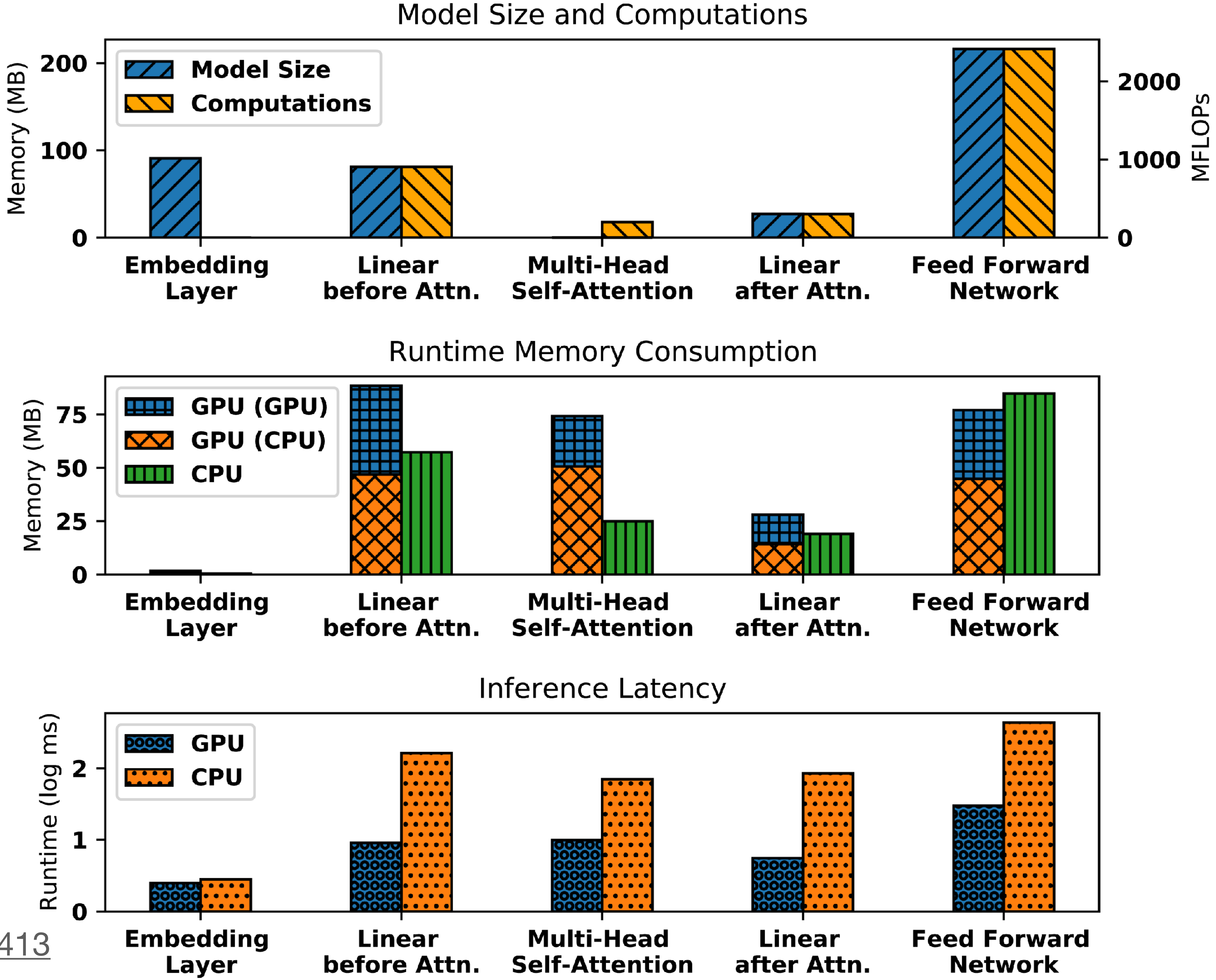


Fig from https://nlp.seas.harvard.edu/slides/emnlp16_seqkd.pdf

Distillation of BERT models



https://doi.org/10.1162/tacl_a_00413

Distillation of BERT models

https://doi.org/10.1162/tacl_a_00413

DistilBERT

- "*Our student is a small version of BERT in which we removed the token-type embeddings and the pooler (used for the next sentence classification task) and kept the rest of the architecture identical while reducing the numbers of layers by a factor of two.*"
- Why not reduce the hidden size as well?
  - "*In our experiments, the number of layers was the determining factor for the inference time, more than the hidden size.*"
- Using L2 loss instead of cross-entropy loss?
  - "*cross-entropy loss leads to significantly better performance*"
- Initialization is important
  - "*We thus initialize our student ... by taking one layer out of two, leveraging the common hidden size between student and teacher.*"

https://medium.com/huggingface/distilbert-8cf3380435b5

DistilBERT

	Macro Score	CoLA	MNLI	MNLI-MM	MRPC		QNLI	QQP		RTE	SST-2	STS-B		WNLI
		mcc	acc	acc	acc	f1	acc	acc	f1	acc	acc	pearson	spearmanr	acc
GLUE BASELINE (ELMo + BiLSTMs)	68.7	44.1	68.6 (avg)		70.8	82.3	71.1	88.0	84.3	53.4	91.5	70.3	70.5	56.3
BERT base	78.0	55.8	83.7	84.1	86.3	90.5	91.1	90.9	87.7	68.6	92.1	89.0	88.6	43.7
DistilBERT	75.2	42.5	81.6	81.1	82.4	88.3	85.5	90.6	87.7	60.0	92.7	84.5	85.0	55.6

	Nb of parameters (millions)	Inference Time (s)
GLUE BASELINE (ELMo + BiLSTMs)	180	895
BERT base	110	668
DistilBERT	66	410

Distillation of BERT models

Different ways to distill information from a teacher

- Distillation during fine-tuning:
  - On SQuAD 1.1 (QA task) BERT gets **88.5** F1 and DistilBERT gets **85.1**
  - Fine-tuning DistilBERT on the QA task using a fine-tuned BERT model gets **86.2** F1.
- Distillation from Output Logits
- Distillation from Encoder Outputs (distil each layer)
- Distillation from Attention Maps (attn is softmax so can be easily distilled)

Pruning

THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

The Lottery Ticket Hypothesis. *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

https://arxiv.org/abs/1803.03635

Identifying winning tickets

Using iterative pruning

- We are training a large neural network f using training data x
    1. Randomly initialize $f(x; \theta_0)$ where $\theta_0 \in \mathcal{D}_\theta$ (distribution over parameters)
    2. Train the network for j iterations, finding parameters θ_j
    3. Prune p% of the parameters in θ_j with smallest magnitude creating a mask m
    4. Reset remaining parameters to values from θ_0 creating the winning ticket $f(x; m \odot \theta_0)$
- Then repeat: retrain f and prune p% of the parameters iteratively for n rounds

Identifying winning tickets in BERT

Table 2: Performance of subnetworks at the highest sparsity for which IMP finds winning tickets on each task. To account for fluctuations, we consider a subnetwork to be a winning ticket if its performance is within one standard deviation of the unpruned BERT model. Entries with errors are the average across five runs, and errors are the standard deviations. IMP = iterative magnitude pruning; RP = randomly pruning; θ_0 = the pre-trained weights; θ_0' = random weights; θ_0'' = randomly shuffled pre-trained weights.

Dataset	MNLI	QQP	STS-B	WNLI	QNLI	MRPC	RTE	SST-2	CoLA	SQuAD	MLM
Sparsity	70%	90%	50%	90%	70%	50%	60%	60%	50%	40%	70%
Full $\text{BERT}_{\text{BASE}}$	82.4 ± 0.5	90.2 ± 0.5	88.4 ± 0.3	54.9 ± 1.2	89.1 ± 1.0	85.2 ± 0.1	66.2 ± 3.6	92.1 ± 0.1	54.5 ± 0.4	88.1 ± 0.6	63.5 ± 0.1
$f(x, m_{\text{IMP}} \odot \theta_0)$	82.6 ± 0.2	90.0 ± 0.2	88.2 ± 0.2	54.9 ± 1.2	88.9 ± 0.4	84.9 ± 0.4	66.0 ± 2.4	91.9 ± 0.5	53.8 ± 0.9	87.7 ± 0.5	63.2 ± 0.3
$f(x, m_{\text{RP}} \odot \theta_0)$	67.5	76.3	21.0	53.5	61.9	69.6	56.0	83.1	9.6	31.8	32.3
$f(x, m_{\text{IMP}} \odot \theta_0')$	61.0	77.0	9.2	53.5	60.5	68.4	54.5	80.2	0.0	18.6	14.4
$f(x, m_{\text{IMP}} \odot \theta_0'')$	70.1	79.2	19.6	53.3	62.0	69.6	52.7	82.6	4.0	24.2	42.3

https://arxiv.org/abs/2007.12223

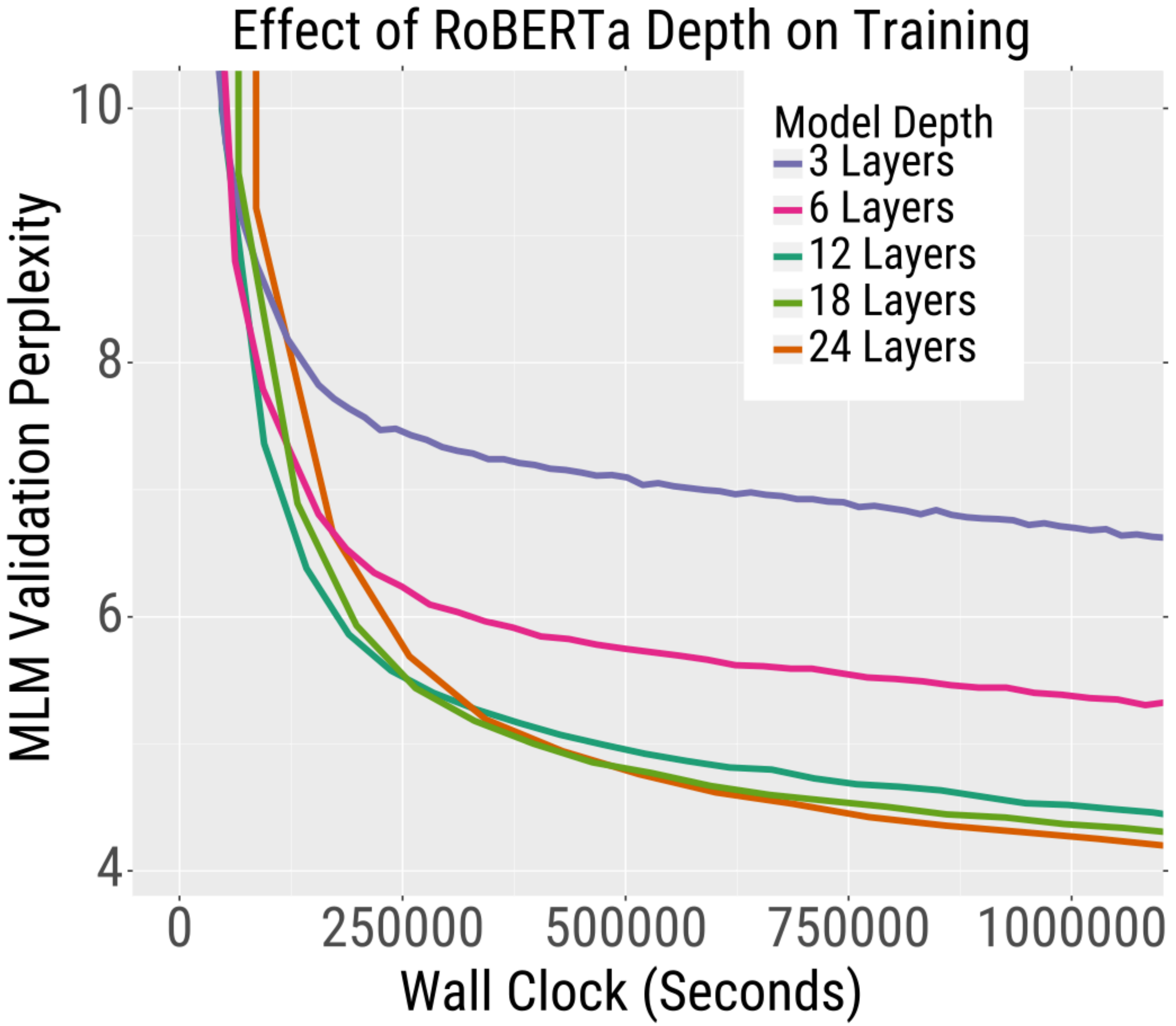
Identifying winning tickets in BERT

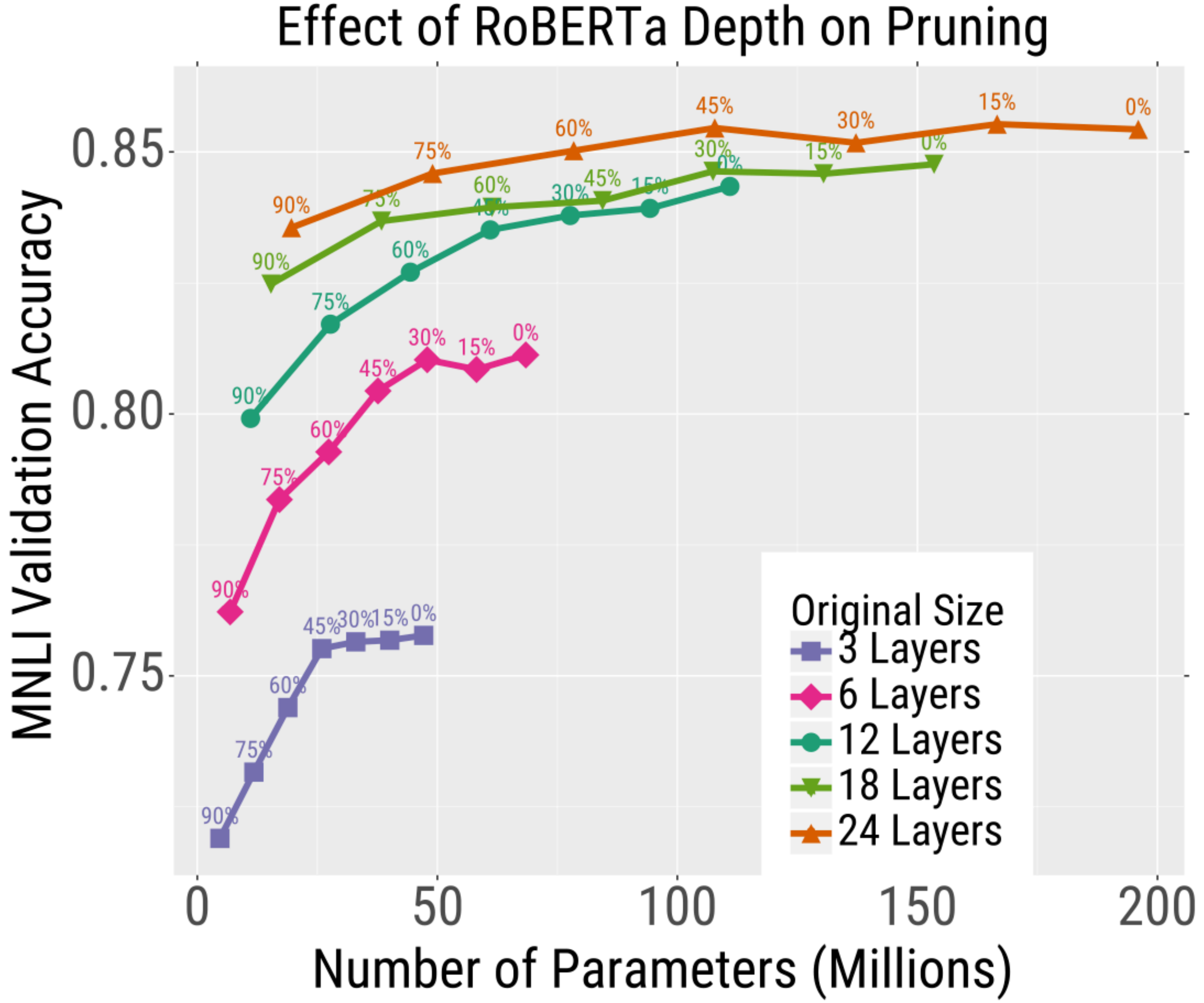
Table 3: Performance of subnetworks found using IMP with rewinding to the steps in the left column and standard pruning (where subnetworks are trained using the final weights from the end of training).

Dataset	MNLI	QQP	STS-B	WNLI	QNLI	MRPC	RTE	SST-2	CoLA	SQuAD	MLM
Sparsity	70%	90%	50%	90%	70%	50%	60%	60%	50%	40%	70%
Full $\text{BERT}_{\text{BASE}}$	82.39	90.19	88.44	54.93	89.14	85.23	66.16	92.12	54.51	88.06	63.48
Rewind 0% (i.e., θ_0)	82.45	89.20	88.12	54.93	88.05	84.07	66.06	91.74	52.05	87.74	63.07
Rewind 5%	82.99	88.98	88.05	54.93	88.85	83.82	62.09	92.43	53.38	87.78	63.18
Rewind 10%	82.93	89.08	88.11	54.93	89.02	84.07	62.09	92.66	52.61	87.77	63.49
Rewind 20%	83.08	89.21	88.28	55.75	88.87	85.78	61.73	92.89	52.02	87.36	63.82
Rewind 50%	82.94	89.54	88.41	53.32	88.72	85.54	62.45	92.66	52.20	87.26	64.21
Standard Pruning	82.11	89.97	88.51	52.82	89.88	85.78	62.95	90.02	52.00	87.12	63.77

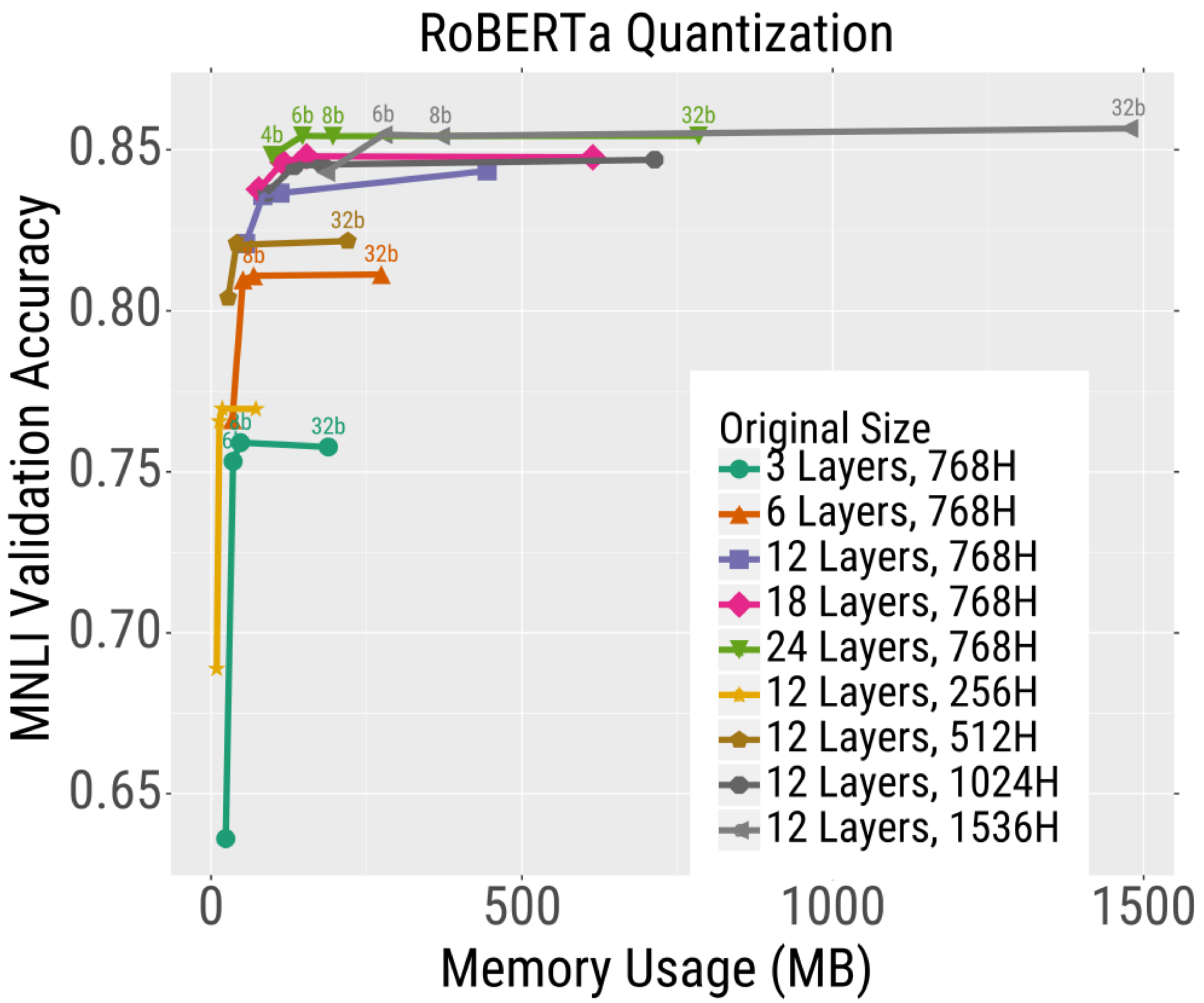
https://arxiv.org/abs/2007.12223

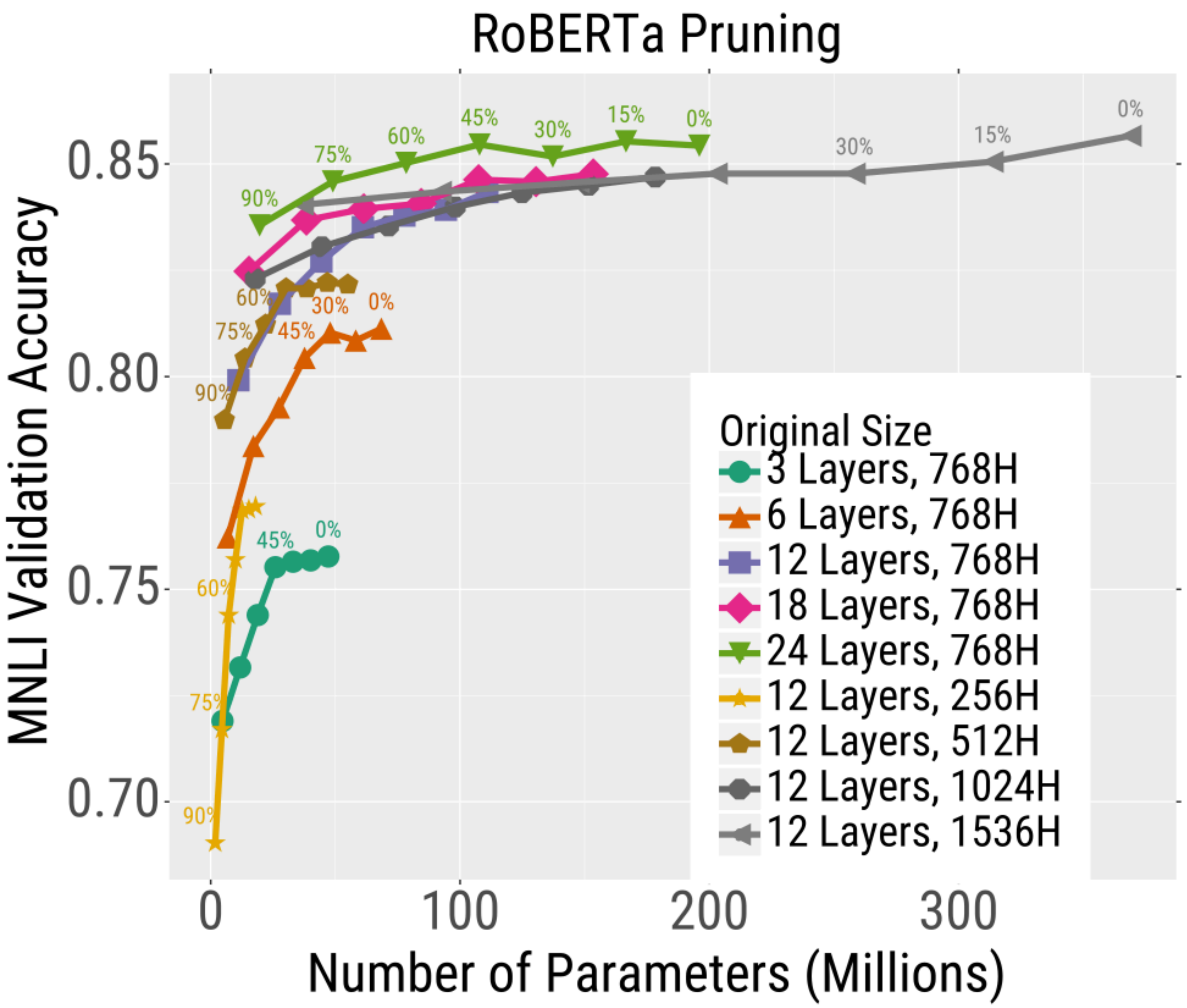
Also see: https://aclanthology.org/2020.emnlp-main.259/





https://arxiv.org/abs/2002.11794





https://arxiv.org/abs/2002.11794