

Knowledge Distillation

Wei Xu

(some slides from Greg Durrett)

Two Common LM Objectives

- ▶ Masked Language Modeling (e.g., BERT):
$$\mathcal{L}_{\text{MLM}}(\theta) = - \sum_{t \in M} \log p_{\theta}(x_t \mid \tilde{\mathbf{x}})$$

corrupted input sequence
(e.g., with [MASK] tokens)

- ▶ Autoregressive Language Models (e.g., GPT), i.e., next-token prediction:

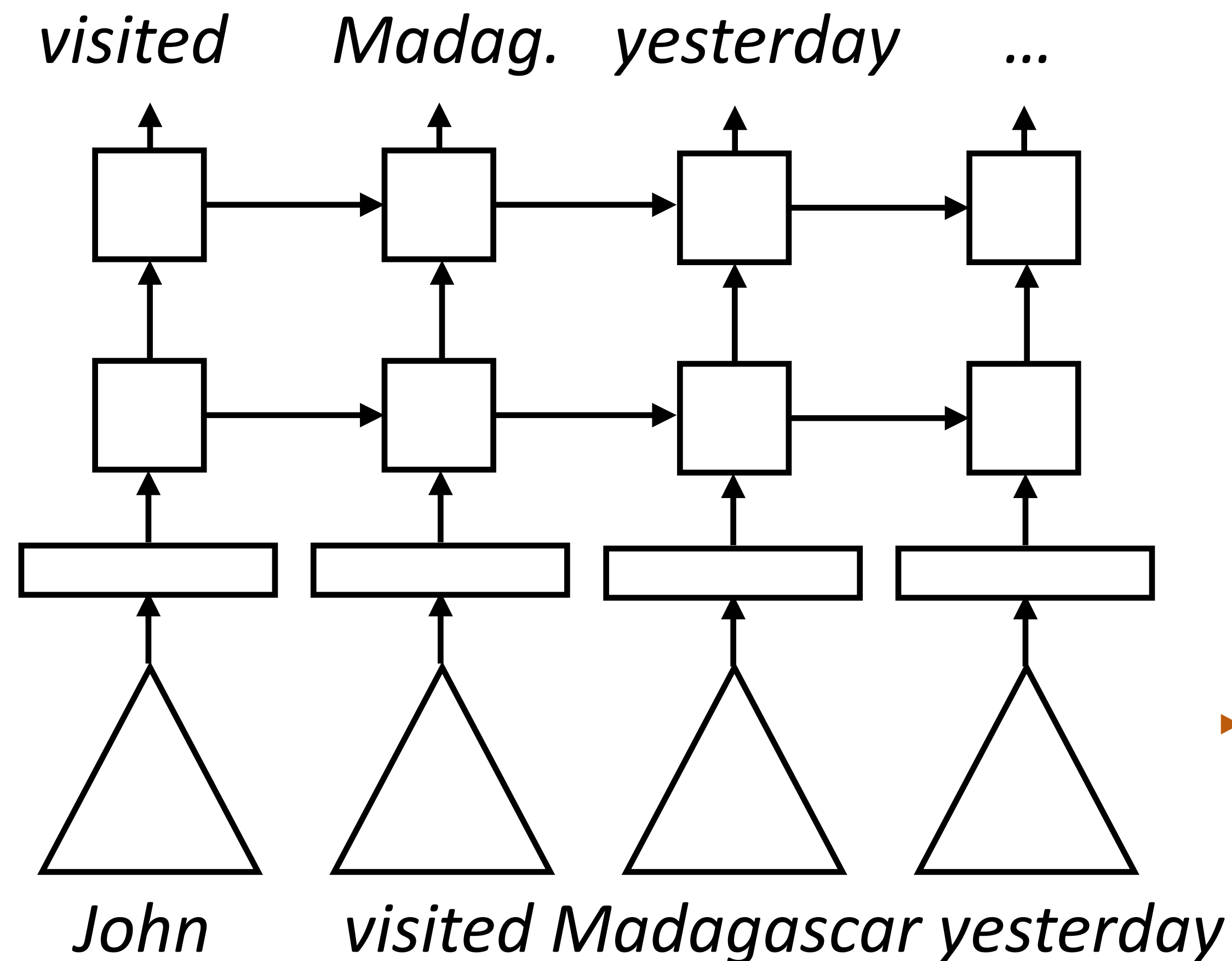
$$\mathcal{L}_{\text{NTP}}(\theta) = - \sum_{t=1}^T \log p_{\theta}(x_t \mid \mathbf{x}_{<t})$$

$x_1, x_2, \dots, x_{t-1}$

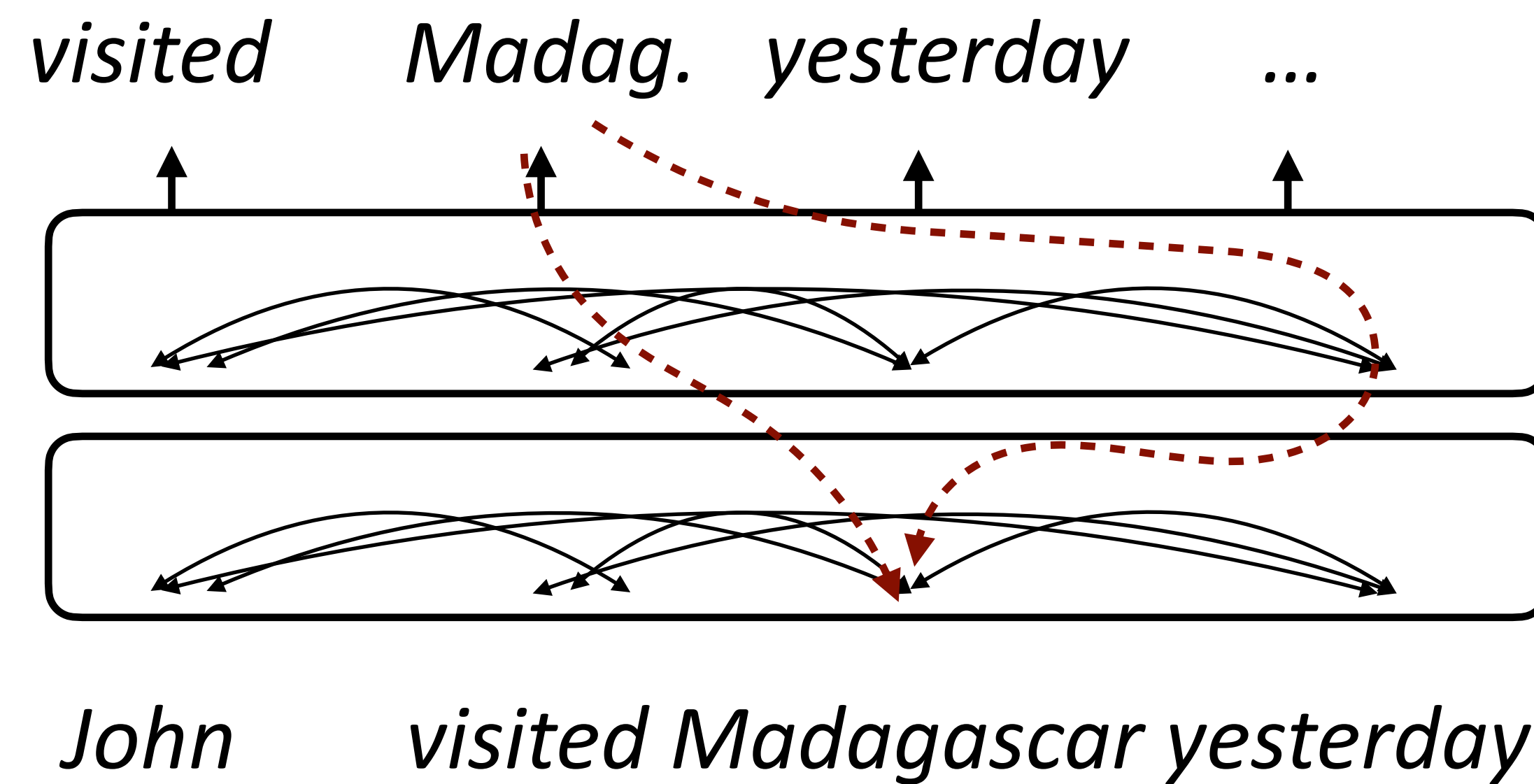
(Recap) BERT

- How to learn a “deeply bidirectional” model? What happens if we just replace an LSTM with a transformer?

ELMo



BERT



- Can do this by “one-sided” Transformer (masked self-attention), but “two-sided” Transformer encoder can cheat

(Recap) Masked Language Modeling

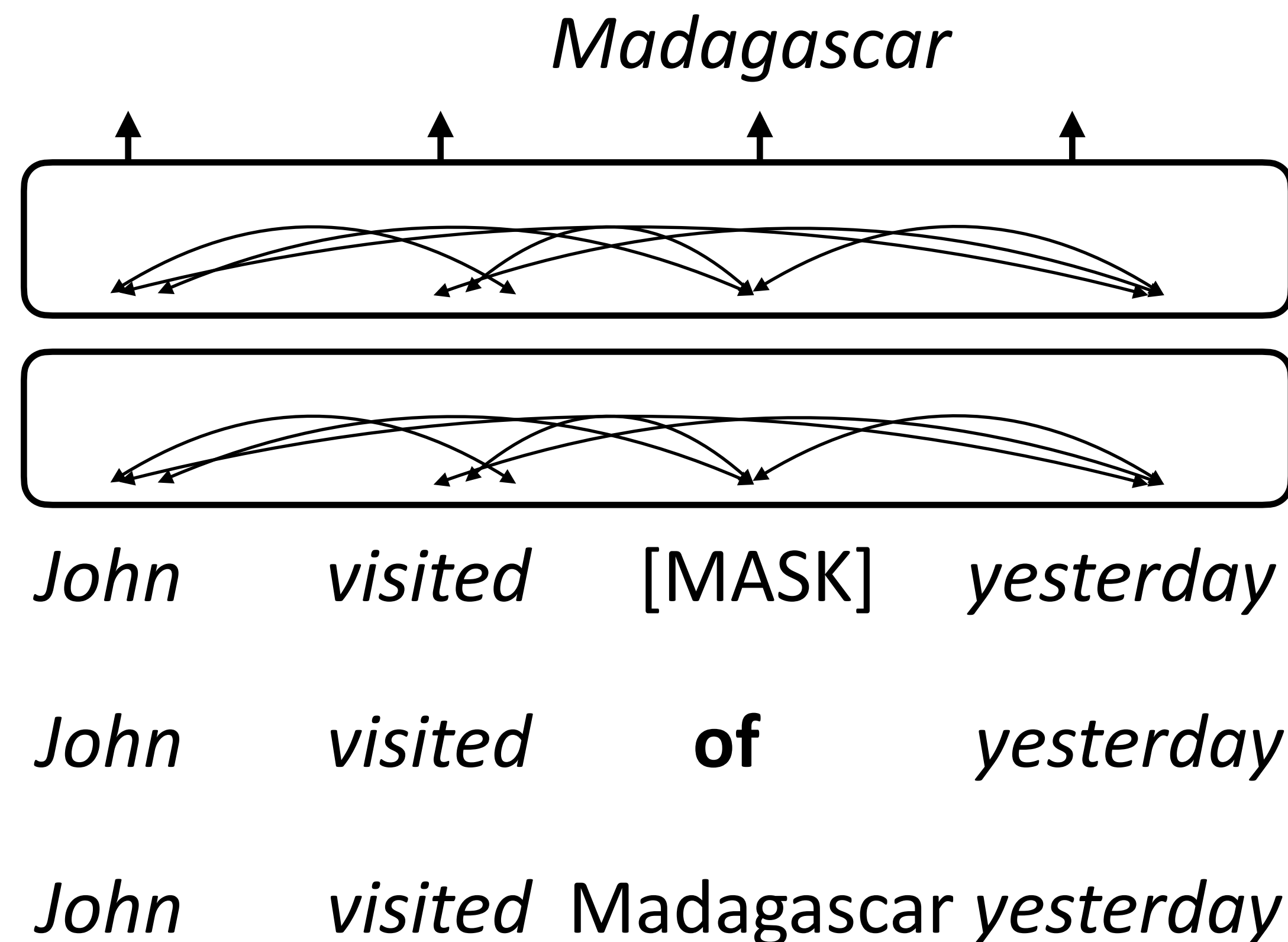
- ▶ How to prevent cheating? Next word prediction fundamentally doesn't work for bidirectional models, instead do *masked language modeling*

- ▶ BERT formula: take a chunk of text, predict 15% of the tokens

- ▶ For 80% (of the 15%), replace the input token with [MASK]

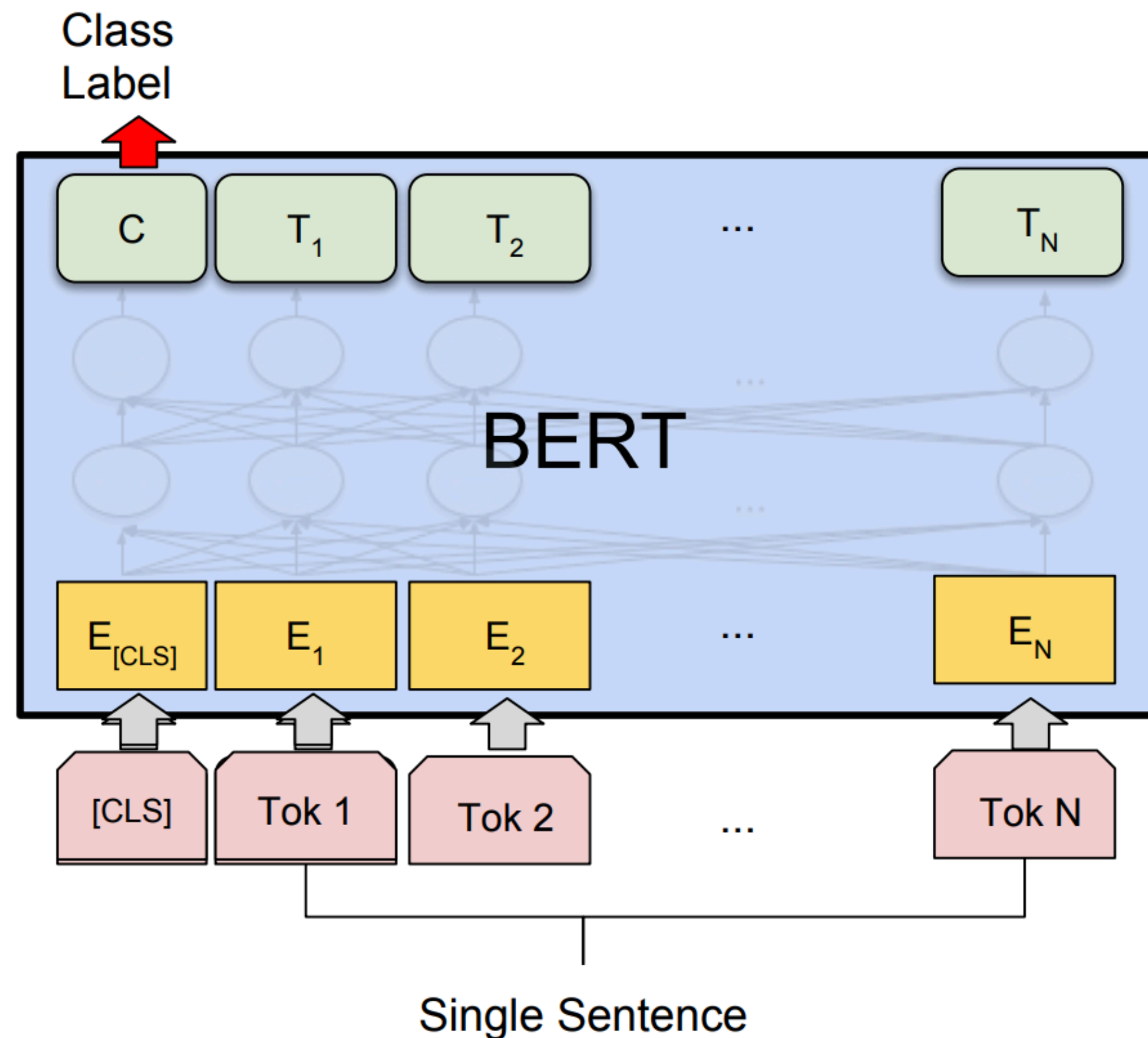
- ▶ For 10%, replace w/random

- ▶ For 10%, keep same



(Recap) Fine-tuning BERT

- Fine-tune for 1-3 epochs, small learning rate (e.g. $2e-5$ - $5e-5$)



(b) Single Sentence Classification Tasks:
SST-2, CoLA

- Large changes to weights up here (particularly in last layer to route the right information to [CLS])
- Smaller changes to weights lower down in the transformer
- Small LR and short fine-tuning schedule mean weights don't change much
- More complex “triangular learning rate” schemes exist

(Recap) BERT Fine-tuning Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	91.1	94.9	60.5	86.5	89.3	70.1	81.9

- ▶ Huge improvements over prior work (even compared to ELMo)
- ▶ Effective at “sentence pair” tasks: textual entailment (does sentence A imply sentence B), paraphrase detection

Devlin et al. (2018)

Other Encoder-only LMs

- ▶ XLM-RoBERTa (Conneau et al., 2020) for 100 languages
- ▶ mDeBERTa (He et al., 2021) for 100 languages
- ▶ MosaicBERT (Portes et al., 2023; Nussbaum et al., 2024)
- ▶ ModernBERT (Warner et al., 2024)
- ▶ EuroBERT (Boizard et al., 2025) for 15 languages
- ▶ NeoBERT (Le Breton et al., 2025)
- ▶ mmBERT (Marone et al., 2025) for 1800 languages
- ▶ ? Jev (TypeSafe AI, 2026)

mmBERT

- Multi-stage training with different number of languages per stage

Category	Dataset	Pre-training		Mid-training		Decay Phase	
		Tokens (B)	%	Tokens (B)	%	Tokens (B)	%
Code	Code (ProLong)	–	–	–	–	2.8	2.7
Code	Starcoder	100.6	5.1	17.2	2.9	0.5	0.5
Crawl	DCLM	600.0	30.2	10.0	1.7	–	–
Crawl	DCLM (Dolmino)	–	–	40.0	6.7	2.0	2.0
Crawl	FineWeb2	1196.6	60.2	506.7	84.3	78.5	76.0
Instruction	Tulu Flan	15.3	0.8	3.1	0.5	1.0	1.0
Math	Dolmino Math	11.2	0.6	4.3	0.7	0.5	0.5
Reference	Books	4.3	0.2	3.9	0.7	2.2	2.1
Reference	Textbooks (ProLong)	–	–	–	–	3.1	3.0
Reference	Wikipedia (MegaWika)	4.7	0.2	1.2	0.2	9.5	9.2
Scientific	Arxiv	27.8	1.4	5.4	0.9	3.3	3.2
Scientific	PeS2o	8.4	0.4	3.2	0.5	–	–
Social	StackExchange	18.6	0.9	3.0	0.5	–	–
Social	StackExchange (Dolmino)	1.4	0.1	2.8	0.5	–	–
Total		1989.0	100.0	600.8	100.0	103.3	100.0

mmBERT

- Pre-trained on 3T tokens of multilingual text in over 1800 languages

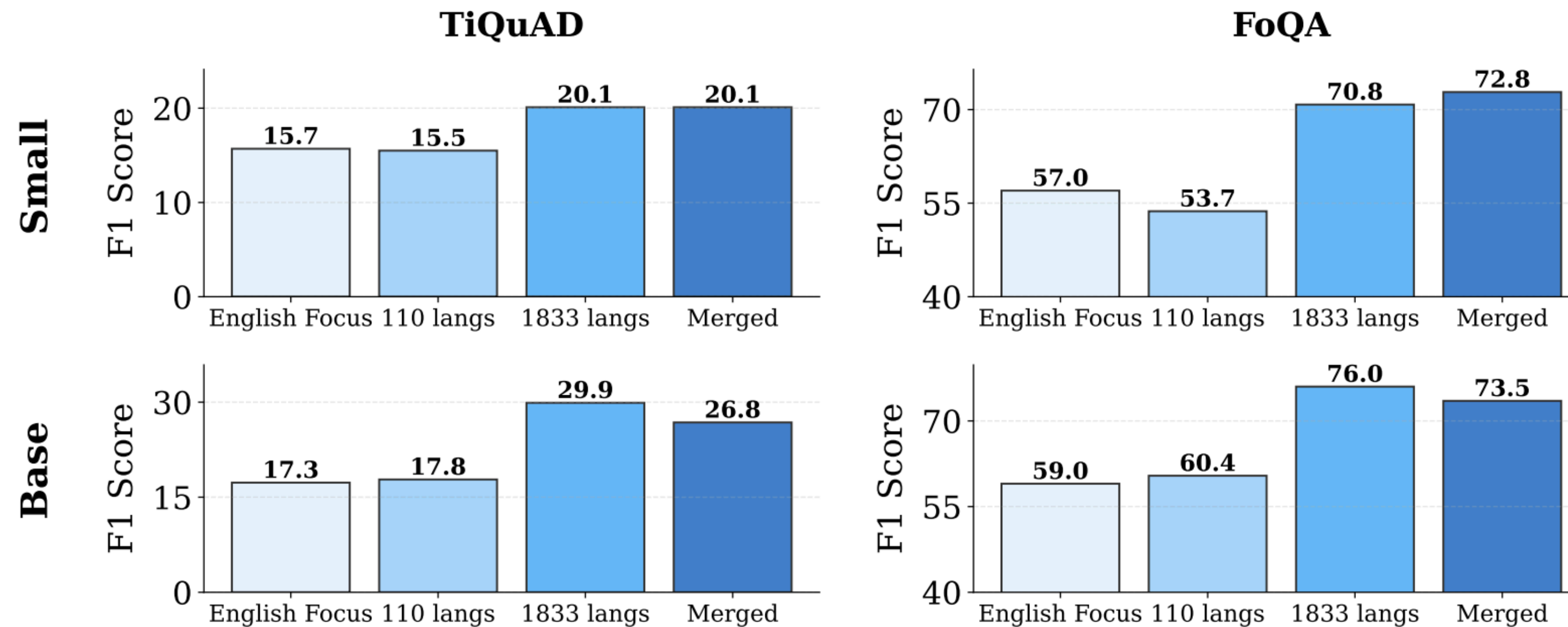


Figure 2: Performance of models using different decay phases on two languages (Tigray and Faroese) only added during the decay phase. We see that MMBERT with the 1833 language decay phase shows rapid performance improvements despite only having the models in the last 100B tokens of training. The final MMBERT models shows improvements by merging together checkpoints.

Compressing BERT

ALBERT

- ▶ A Lite BERT (18x fewer parameters, 1.7x faster training than BERT)
- ▶ Factorized embedding matrix to save parameters, model context-independent words with fewer parameters

Ordinarily $|V| \times H$ — $|V|$ is 30k-90k, H is >1000

-8	-2	-6	6
-4	-1	-3	3
28	7	21	-21
24	6	18	-18

 $=$

-2
-1
7
6

 $\times$

4	1	3	-3
---	---	---	----

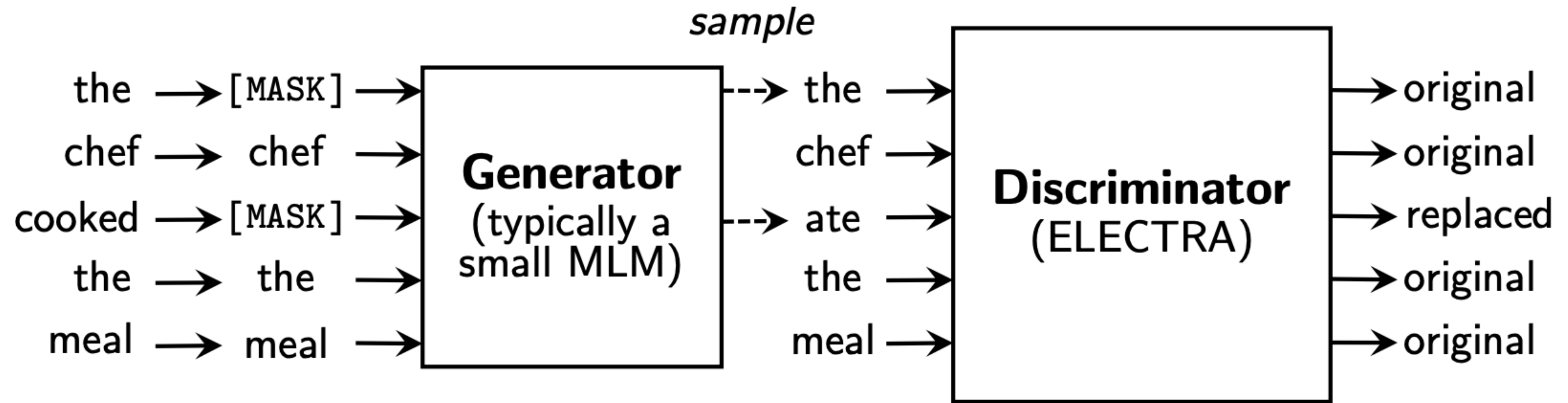
Factor into two matrices with a low-rank approximation

Now: $|V| \times E$ and $E \times H$ — E is 128 in their implementation

- ▶ Additional cross-layer parameter sharing

Lan et al. (2020)

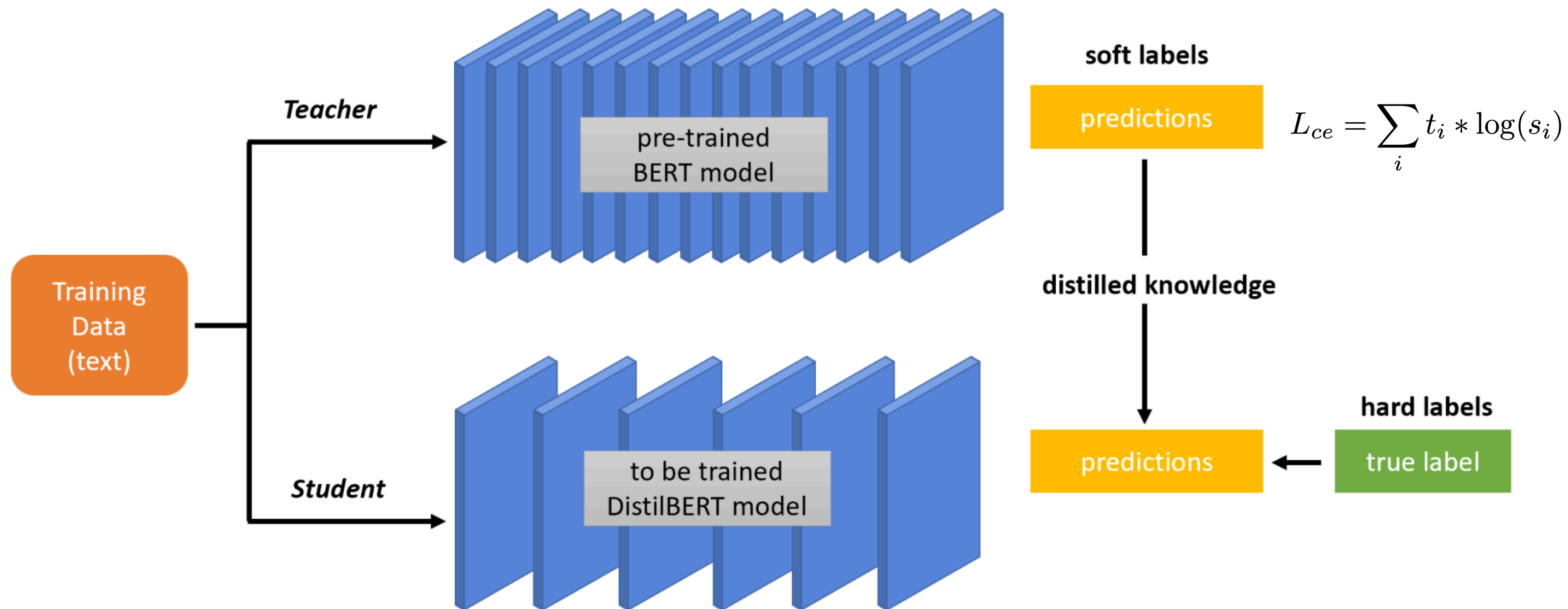
ELECTRA



- ▶ No need to necessarily have a generative model (predicting words)
- ▶ This objective is more computationally efficient (trains faster) than the standard BERT objective

DistilBERT

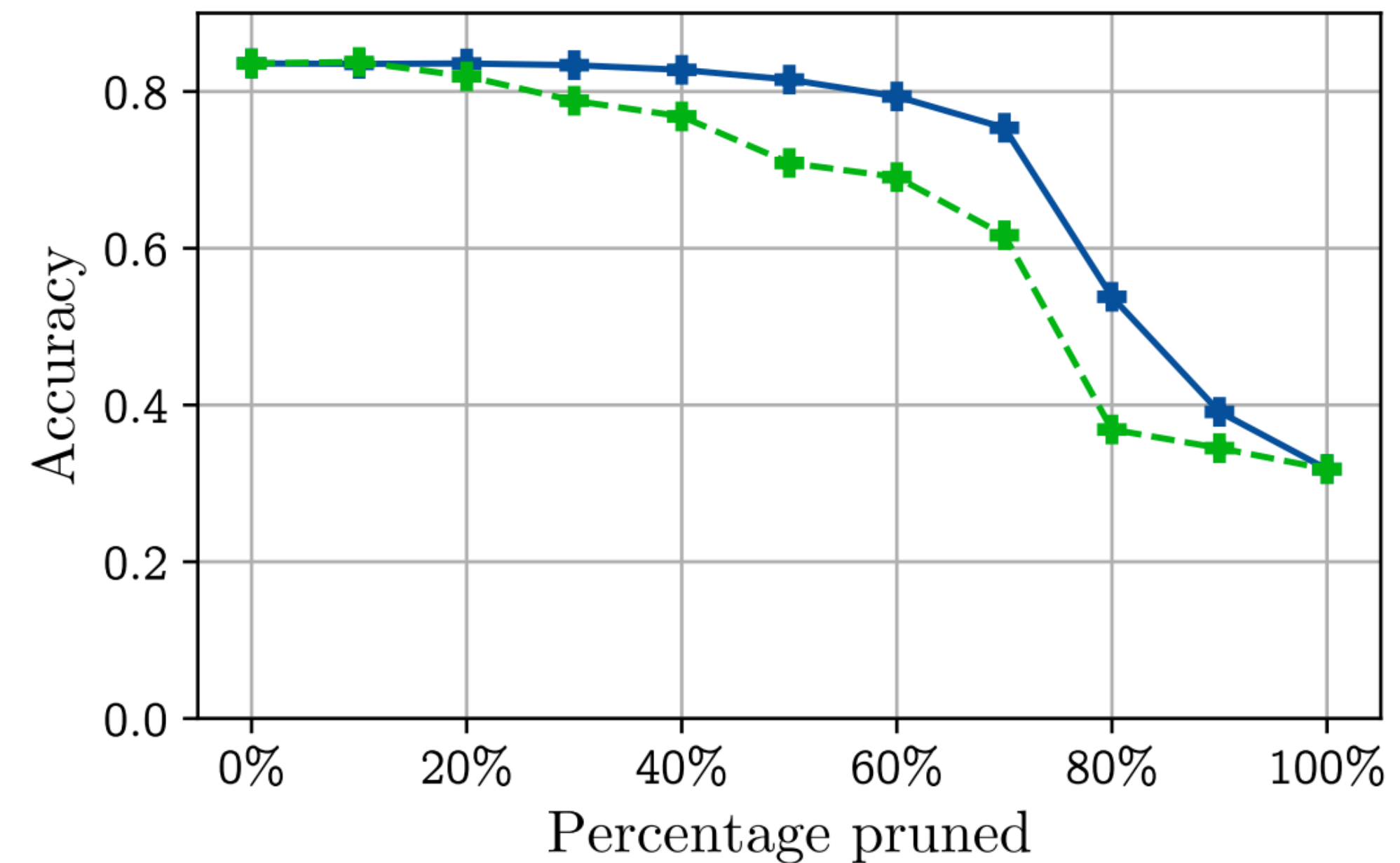
- DistilBERT is pretrained by knowledge distillation to create a smaller model with faster inference and requires less compute to train.



Sanh et al. (2019)

Compressing BERT

- ▶ Remove 60+% of BERT's heads post-training with minimal drop in performance
- ▶ DistilBERT (Sanh et al., 2019): nearly as good with half the parameters of BERT (via knowledge distillation)



(b) Evolution of accuracy on the MultiNLI-matched validation set when heads are pruned from BERT according to I_h (solid blue) and accuracy difference (dashed green).

Knowledge Distillation

Knowledge Distillation

arXiv:1503.02531v1 [stat.ML] 9 Mar 2015

Distilling the Knowledge in a Neural Network

Geoffrey Hinton^{*}
Google Inc.
Mountain View
geoffhinton@google.com

Oriol Vinyals[†]
Google Inc.
Mountain View
vinyals@google.com

Jeff Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

A very simple way to improve the performance of almost any machine learning algorithm is to train many different models on the same data and then to average their predictions [3]. Unfortunately, making predictions using a whole ensemble of models is cumbersome and may be too computationally expensive to allow deployment to a large number of users, especially if the individual models are large neural nets. Caruana and his collaborators [1] have shown that it is possible to compress the knowledge in an ensemble into a single model which is much easier to deploy and we develop this approach further using a different compression technique. We achieve some surprising results on MNIST and we show that we can significantly improve the acoustic model of a heavily used commercial system by distilling the knowledge in an ensemble of models into a single model. We also introduce a new type of ensemble composed of one or more full models and many specialist models which learn to distinguish fine-grained classes that the full models confuse. Unlike a mixture of experts, these specialist models can be trained rapidly and in parallel.

1 Introduction

Many insects have a larval form that is optimized for extracting energy and nutrients from the environment and a completely different adult form that is optimized for the very different requirements of traveling and reproduction. In large-scale machine learning, we typically use very similar models for the training stage and the deployment stage despite their very different requirements: For tasks like speech and object recognition, training must extract structure from very large, highly redundant datasets but it does not need to operate in real time and it can use a huge amount of computation. Deployment to a large number of users, however, has much more stringent requirements on latency and computational resources. The analogy with insects suggests that we should be willing to train very cumbersome models if that makes it easier to extract structure from the data. The cumbersome model could be an ensemble of separately trained models or a single very large model trained with a very strong regularizer such as dropout [9]. Once the cumbersome model has been trained, we can then use a different kind of training, which we call “distillation” to transfer the knowledge from the cumbersome model to a small model that is more suitable for deployment. A version of this strategy has already been pioneered by Rich Caruana and his collaborators [1]. In their important paper they demonstrate convincingly that the knowledge acquired by a large ensemble of models can be transferred to a single small model.

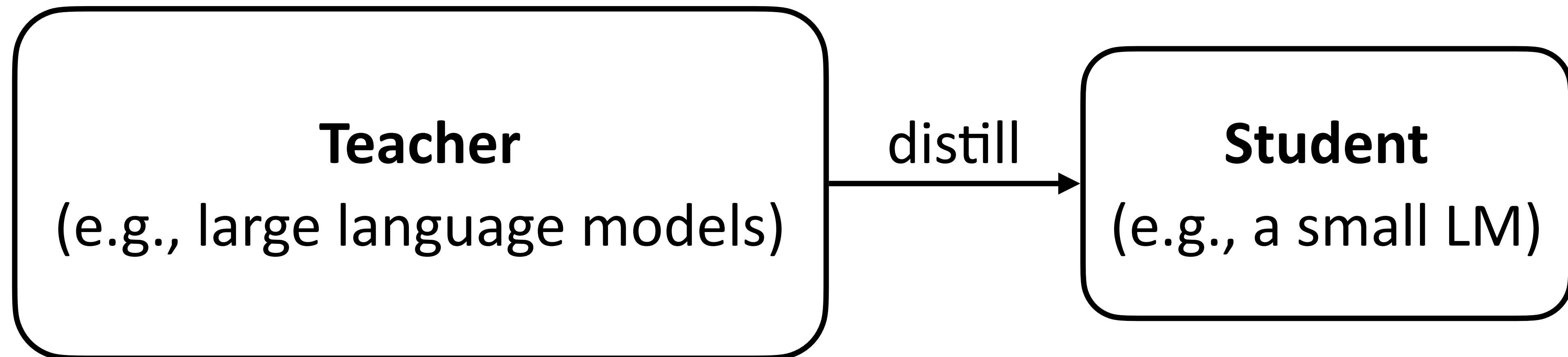
A conceptual block that may have prevented more investigation of this very promising approach is that we tend to identify the knowledge in a trained model with the learned parameter values and this makes it hard to see how we can change the form of the model but keep the same knowledge. A more abstract view of the knowledge, that frees it from any particular instantiation, is that it is a learned

^{*}Also affiliated with the University of Toronto and the Canadian Institute for Advanced Research.
[†]Equal contribution.

Hinton et al. (2015)

Knowledge Distillation

- ▶ Compress a large and complex model (the teacher model) into a smaller and simpler model (the student model)



Knowledge Distillation

- Formally, NNs typically use a softmax function to convert logits z_i into class probabilities:

$$P(z_i, T) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

- One classic way of knowledge distillation is to align the class probability distribution from teacher and student networks.

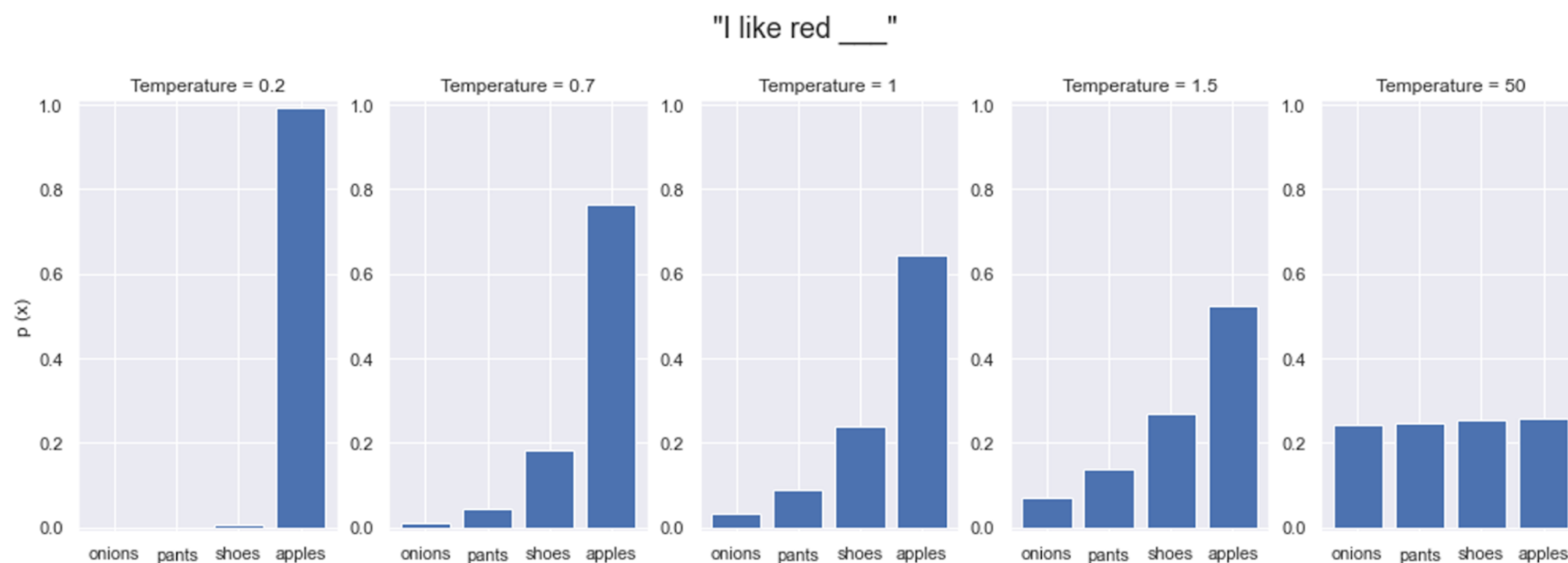
Softmax Temperature

- Formally, NNs typically use a softmax function to convert logits z_i into class probabilities:

$$P(z_i, T) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

temperature to smooth softmax distribution

index of output space $\mathcal{Y}$



- $T = 1$: Standard Softmax

Knowledge Distillation

- Formally, NNs typically use a softmax function to convert logits z_i into class probabilities:

$$P(z_i, T) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

- One classic way of knowledge distillation is to align the class probability distribution from teacher and student networks.

Knowledge Distillation

- Formally, NNs typically use a softmax function to convert logits z_i into class probabilities:

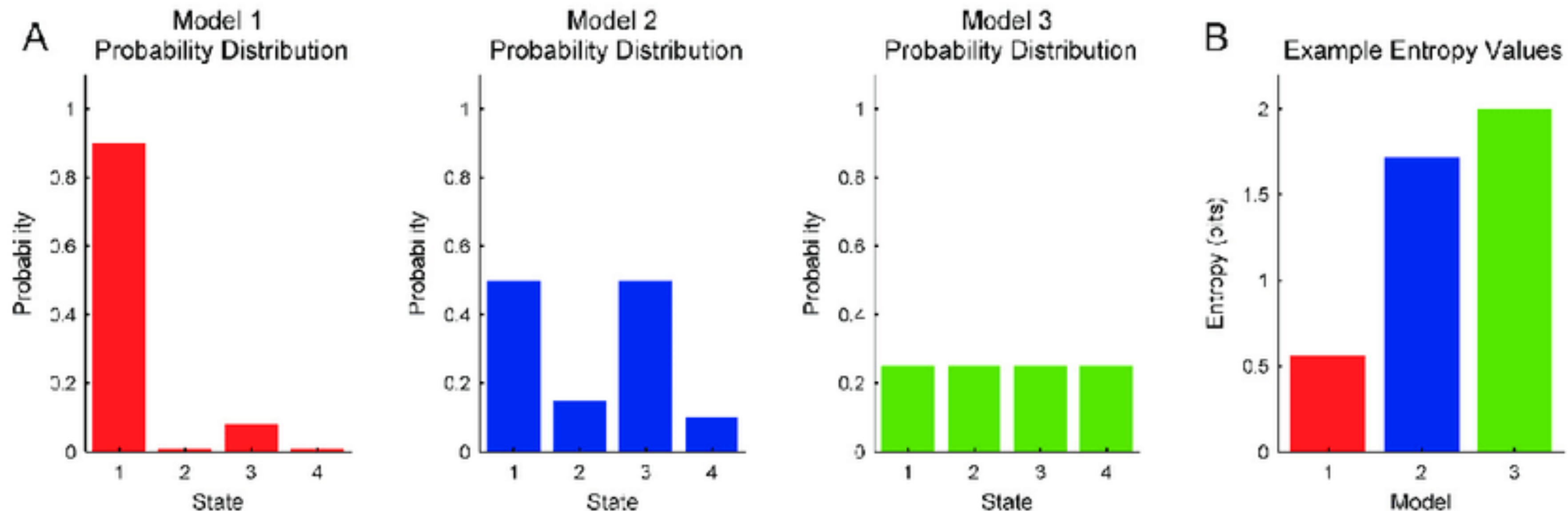
$$P(z_i, T) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

- One classic way of knowledge distillation is to align the class probability distribution from teacher and student networks.
- More advanced KD include aligning intermediate weights, gradients of attention maps, sparsity patterns after ReLU activation, etc.

Entropy

- Entropy measures the inherent randomness or uncertainty of a single probability distribution

$$H(p) = - \sum_{i=1}^n p_i \log p_i$$



Cross-Entropy

- ▶ Cross Entropy (CE) is an extension of the concept of entropy, when two different probability distributions are present

$$H(p, q) = - \sum_{i=1}^n p_i \log q_i$$

$$= - 1.00 * \log(0.21) = - \log(0.21) \approx 1.56$$

p_1	1.00
p_2	0.00
p_3	0.00

q_1	0.21
q_2	0.77
q_3	0.02

(Recap) Multiclass Logistic Regression

$$P_w(y|x) = \frac{\exp(w^\top f(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(w^\top f(x, y'))}$$

sum over output
space to normalize

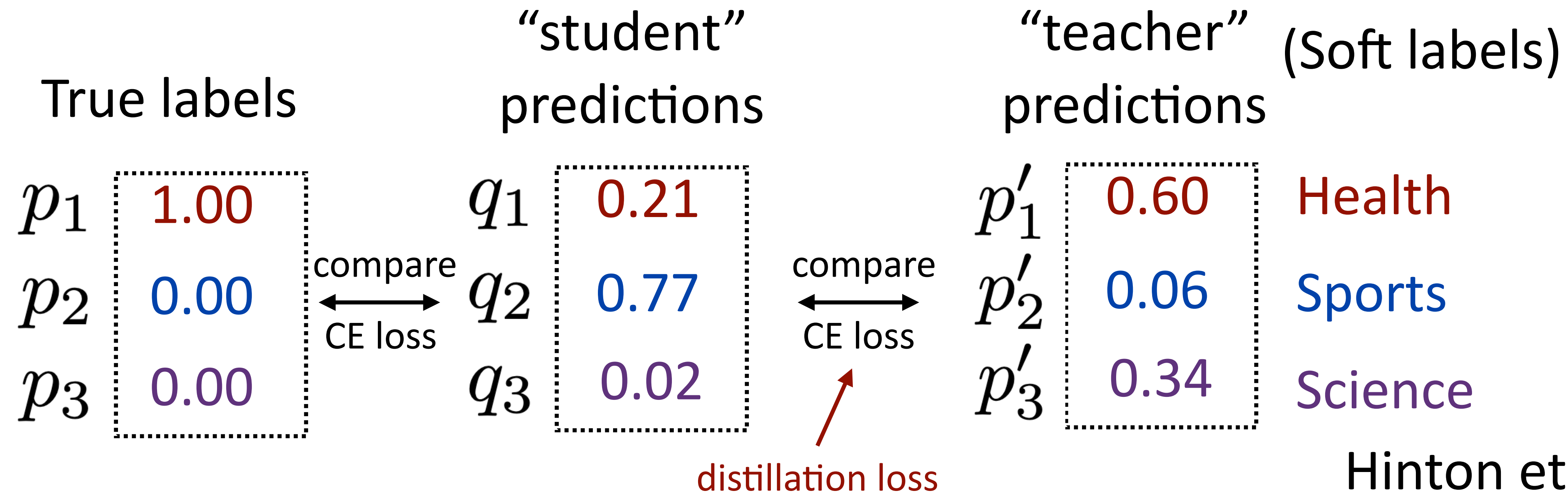
i.e. minimize negative log likelihood
or **cross-entropy loss**

► Training: maximize $\mathcal{L}(x, y) = \sum_{j=1}^m \log P(y_j^* | x_j)$

index of
data points (j)

Distillation Loss

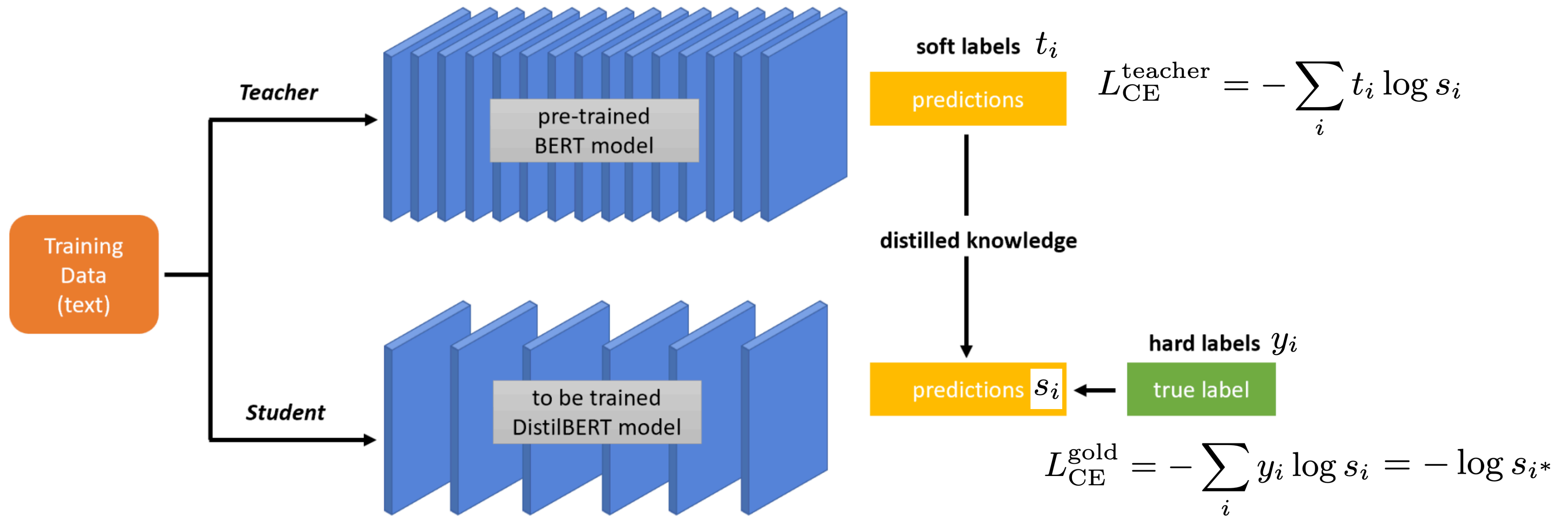
- ▶ Train student to mimic teacher's predicted distributions
- ▶ The relative probabilities of incorrect answers tell us a lot about how the large ("teacher") model tends to generalize (e.g., mixing up "Health/Science" is a smaller error than "Health/Sports").



Hinton et al. (2015)

DistilBERT

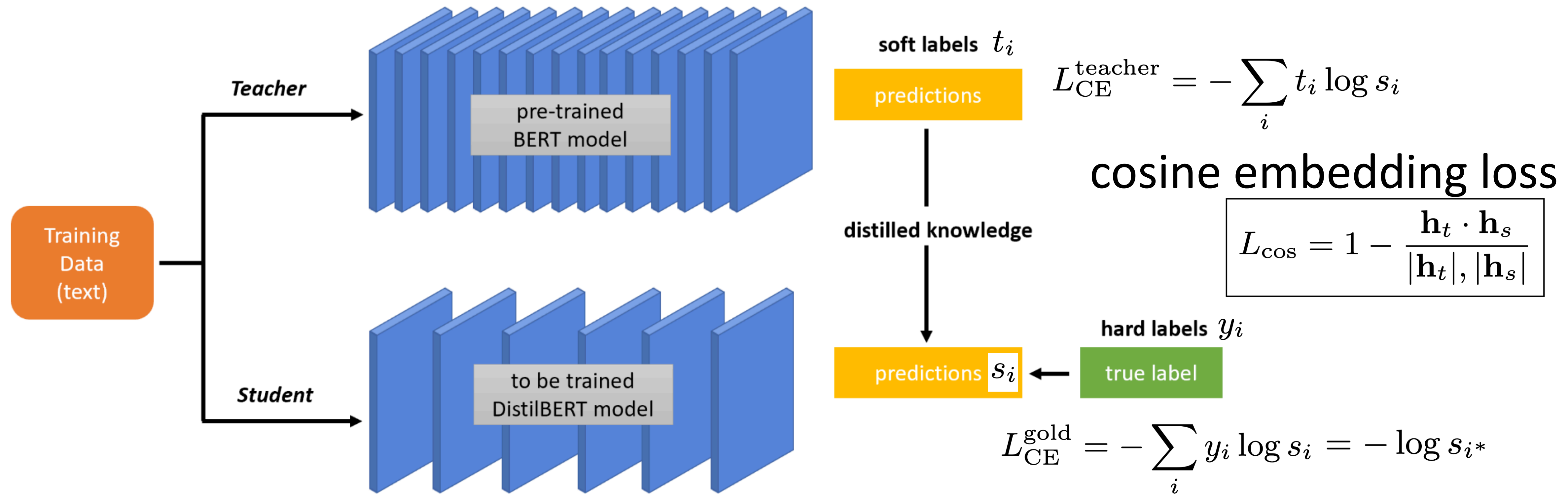
- DistilBERT is pretrained by knowledge distillation to create a smaller model with faster inference and requires less compute to train.



Sanh et al. (2019)

DistilBERT

- DistilBERT is pretrained by knowledge distillation to create a smaller model with faster inference and requires less compute to train.



Sanh et al. (2019)

Kullback–Leibler (KL) Divergence

- ▶ KL Divergence from p to q is defined as:

$$D_{\text{KL}}(p||q) = \sum_{i=1}^n p_i \log \frac{p_i}{q_i} = H(p, q) - H(p)$$

- ▶ If p is fixed (e.g., target distribution), then $H(p)$ is a constant, minimizing cross-entropy $H(p, q)$ is equivalent to minimizing the KL divergence

Knowledge Distillation of LMs

- Token-level knowledge distillation

$$\mathcal{L}_{\text{token-KD}} = - \sum_{t=1}^L \sum_{v \in \mathcal{V}} \underset{\text{teacher}}{p_T(v \mid \mathbf{y}_{<t}, \mathbf{x})} \log \underset{\text{student}}{p_{\theta}(v \mid \mathbf{y}_{<t}, \mathbf{x})}$$

At each decoding step t , the student matches the teacher's next-token (or masked-token) prediction.

Knowledge Distillation of LMs

- Token-level knowledge distillation

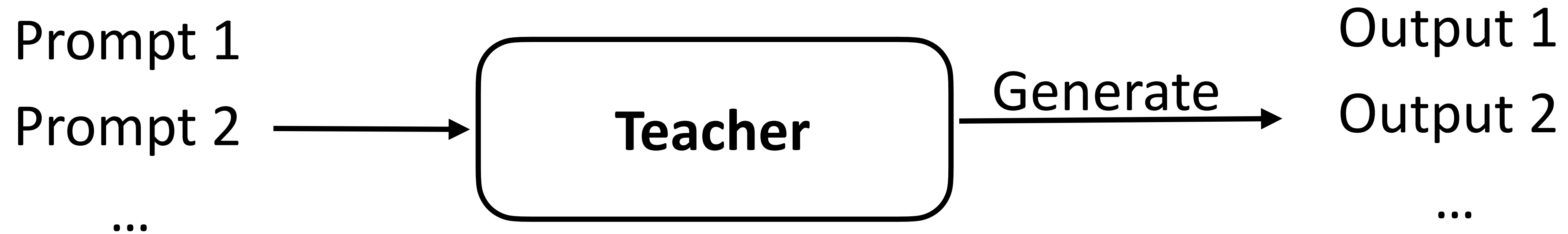
$$\mathcal{L}_{\text{token-KD}} = - \sum_{t=1}^L \sum_{v \in \mathcal{V}} \underset{\substack{\uparrow \\ \text{teacher}}}{p_T(v \mid \mathbf{y}_{<t}, \mathbf{x})} \log \underset{\substack{\uparrow \\ \text{student}}}{p_\theta(v \mid \mathbf{y}_{<t}, \mathbf{x})}$$

At each decoding step t , the student matches the teacher's next-token (or masked-token) prediction. This is equivalent to:

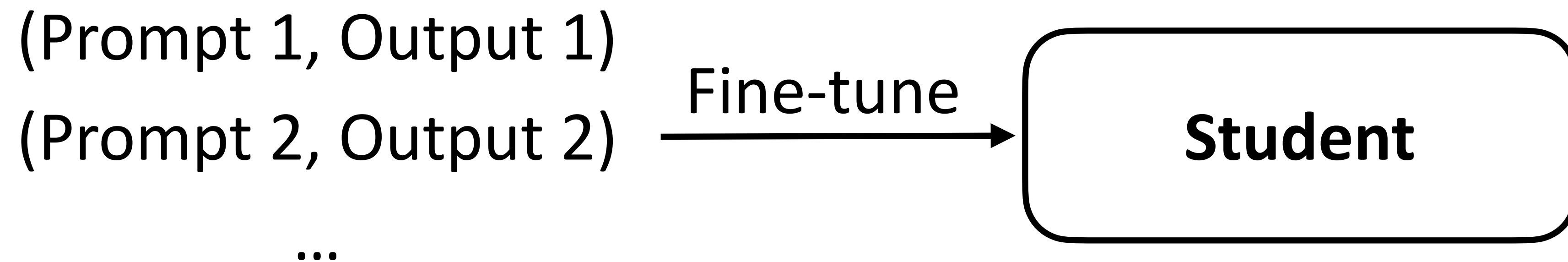
$$\mathcal{L}_{\text{token-KD}} = \sum_{t=1}^L \text{KL}(p_T(\cdot \mid \mathbf{y}_{<t}, \mathbf{x}) \parallel p_\theta(\cdot \mid \mathbf{y}_{<t}, \mathbf{x})) + \text{const.}$$

Sequence-level Knowledge Distillation

- ▶ Generate with a teacher model



- ▶ Student model fine-tunes on the data generated by the teacher model



Some Examples

Distilled Model Evaluation	AIME 2024 pass@1	AIME 2024 cons@64	MATH-500 pass@1	GPQA Diamond	LiveCode Bench	CodeForces rating
GPT-4o-0513	9,3	13,4	74,6	49,9	32,9	759
Claude-3.5-Sonnet-1022	16,0	26,7	78,3	65,0	38,9	717
o1-mini	63,6	80,0	90,0	60,0	53,8	1820
QwQ-32B-Preview	44,0	60,0	90,6	54,5	41,9	1316
DeepSeek-R1-Distill-Qwen-1.5B	28,9	52,7	83,9	33,8	16,9	954
DeepSeek-R1-Distill-Qwen-7B	55,5	83,3	92,8	49,1	37,6	1189
DeepSeek-R1-Distill-Qwen-14B	69,7	80,0	93,9	59,1	53,1	1481
DeepSeek-R1-Distill-Qwen-32B	72,6	83,3	94,3	62,1	57,2	1691
DeepSeek-R1-Distill-Llama-8B	50,4	80,0	89,1	49,0	39,6	1205
DeepSeek-R1-Distill-Llama-70B	70,0	86,7	94,5	65,2	57,5	1633

Sequence-level Knowledge Distillation

- ▶ Generate with a teacher model

$$\hat{\mathbf{y}}_T = \arg \max_{\mathbf{y}} p_T(\mathbf{y} \mid \mathbf{x})$$

- ▶ Student model fine-tunes on the data generated by the teacher model

$$\mathcal{L}_{\text{seq-KD}} = -\log p_{\theta}(\hat{\mathbf{y}}_T \mid \mathbf{x}) = -\sum_{t=1}^{|\hat{\mathbf{y}}_T|} \log p_{\theta}(\hat{\mathbf{y}}_{T,t} \mid \hat{\mathbf{y}}_{T,<t}, x)$$

Treat $\hat{\mathbf{y}}_T$ as a pseudo-label and perform maximum likelihood training on it.
Kim & Rush (2016)

Knowledge Distillation of LMs

- ▶ Token-level knowledge distillation: “soft” targets (token prob. distribution)

$$p_T(\cdot \mid \mathbf{y}_{<t}, \mathbf{x}) \longrightarrow p_\theta(\cdot \mid \mathbf{y}_{<t}, \mathbf{x})$$

match the teacher's probability distribution over the vocabulary at each token position
by minimizing $\text{KL}(p_T(\cdot \mid \mathbf{y}_{<t}, \mathbf{x}) \parallel p_\theta(\cdot \mid \mathbf{y}_{<t}, \mathbf{x}))$

- ▶ Sequence-level knowledge distillation: “hard” pseudo-label sequences

$$\hat{\mathbf{y}}_T \sim p_T(\mathbf{y} \mid \mathbf{x}) \longrightarrow \max_{\theta} p_\theta(\hat{\mathbf{y}}_T \mid \mathbf{x})$$

minimize $\text{KL}(p_T(\mathbf{y} \mid \mathbf{x}) \parallel p_\theta(\mathbf{y} \mid \mathbf{x}))$ over all possible sequences is intractable;
instead, train the student on teacher-generated sequences as pseudo-labels (approximating the distribution).

Takeaways

- ▶ Transformers + lots of data + self-supervision seems to do very well
- ▶ Knowledge distillation: Train a smaller model to imitate a stronger model
- ▶ Next: GPT/GPT-2/GPT-3, etc.